А. Д. ЗАКРЕВСКИЙ

АЛГОРИТМЫ СИНТЕЗА ДИСКРЕТНЫХ АВТОМАТОВ

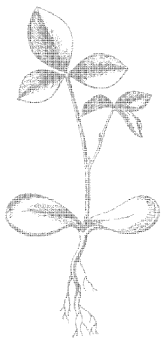
А. А. ЗАКРЕВСКИЙ • АЛГОРИТМЫ СИНТЕЗА ДИСКРЕТНЫХ АВТОМАТОВ

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ТЕХНИЧЕСКОЙ КИБЕРНЕТИКИ

ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1971

А. Д. ЗАКРЕВСКИЙ

АЛГОРИТМЫ СИНТЕЗА ДИСКРЕТНЫХ АВТОМАТОВ



ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1971

6ф 6.5
З
УДК 62·50

Алгоритмы синтеза дискретных автоматов. Закревский А. Д., Изд-во «Наука», Главная редакция физико-математической литературы, М., 1971, 512 стр.

Книга посвящена проблеме автоматизации синтеза дискретных автоматов, решаемой на базе системы автоматического программирования, в основу которой положен алгоритмический язык ЛЯПАС (логический язык представления алгоритмов синтеза).

Книга содержит детальное и достаточно популярное описание языка ЛЯПАС. Излагается методика программирования в языке ЛЯПАС, а также вопросы отладки составленных программ и экспериментально-статистического определения их эффективности, благодаря чему данная книга может служить первым учебным пособием по программированию в языке ЛЯПАС. Основная по объему часть книги занята изложением алгоритмов решения некоторых типичных для теории синтеза задач: решение систем логических уравнений, минимизация булевых функций, нахождение кратчайших покрытий булевых матриц и др.

Табл. 33. Илл. 57. Библ. 50 назв.

3-3-14
130-71

ОГЛАВЛЕНИЕ

Предисловие 9

Глава 1. Язык ЛЯПАС 19

§ 1. Булевы векторы и действия над ними 19

Булев вектор (19). Теоретико-множественная интерпретация (19). Булева матрица бинарного отношения (20). Числовая интерпретация (22). Логические операции (23). Модульные арифметические операции (25). Характеристические операции (29). Операции сдвига (30).

§ 2. Операнды языка ЛЯПАС 31

Машинное представление булевых векторов (31). Выбор размерности булевых векторов (32). Переменные (33). Основные комплексы (33). Индексы (34). Константы (34). Специальные комплексы (36). Символика типов операндов (37).

§ 3. Линейные программы 38

Л-программа (38). Вычислительные операции (38). Простейший пример линейной программы (40). Операции присвоения значений (40). Примеры арифметических программ (44). Примеры логических программ (45).

§ 4. Циклические программы 51

Предложения (51). Операции перехода (52). Программа вычисления значения полинома (54). Программа упорядочения элементов комплекса (56). Реализация программы (58). Операция ухода с возвращением (58). Операция перебора единиц (59). Программа транспонирования булевой матрицы (60). Представление случайных процессов (61).

§ 5. Синтаксис языка 61

Синтаксис и семантика (61). α-цепочки (62). Синтаксически правильные α-цепочки (63).

§ 6. Абстрактная модель вычислительной машины 65

Язык и машина (65). Структура машины (66). Оперативный комплекс (67). К экономии операндов (69). Реализация программы (70). Внешний комплекс (71). Средства ввода и вывода информации (72).

§ 7. Повышение размерности операндов 75

Параметры размерности (75). Выполнение операций (76). Представления в оперативном комплексе (79). Примеры (80). Рекомендации (81).

§ 8. Л-операторы и подпрограммы 81

Два уровня языка ЛЯПАС (81). Задача о кратчайшем покрытии (82). Структура приближенного алгоритма (83). Подпрограммы (85). Составление подпрограмм (88). Компиляция (90). Согласование совокупностей операндов (92). Подпрограммы-многополюсники (95). Операторы как операнды (96). Составные элементы перечня (98). «Штриховка» внутренних операндов подпрограмм (98).

§ 9. Иерархическое программирование 99
Принципы иерархического программирования (99). Задача о размещении отдыхающих в лодках (100). Абстрактная формулировка задачи (101). Представление исходной информации (102). Первичное разложение задачи (103). Подпрограмма прикрапок (105). Нахождение максимальных совместимых подмножеств (106). Задача включения в матрицу максимальных векторов (110). Анализ подмножеств из A на совместимость (111). Подпрограмма перебор (112). Подпрограмма максопод (113). Программа решения задачи в целом (116). Обсуждение составленной программы (118).

§ 10. Оформление подпрограмм 120
Стандартизация формы представления подпрограмм (120). Внешнее описание подпрограммы (121). Л-программа (124). Прочие пункты описания (124). Пример: подпрограмма прикрапок (126). Пример: подпрограмма профакт (126). Оформление других подпрограмм (128). Две формы Л-оператора (131).

§ 11. Дополнительные операции и кодирование Л-программ 133
Разрядность вычислительных машин (133). Неограниченный сдвиг (134). Согласование форм представления операндов (136). Десятичная печать (136). Вывод информации на АЦПУ (137). Прочие операции (139). О структуре дополнительной памяти (140). Кодирование Л-программ (141).

Глава 2. Операции над булевыми и троичными матрицами 144

§ 1. Оптимизация покрытий булевых матриц 144
Задача о переводчиках (144). О поиске кратчайших покрытий (144). Вспомогательные подпрограммы (146). Приближенный алгоритм нахождения кратчайшего покрытия (148). О точном решении задачи (151). Правила поиска (152). Пример (153). Ускорение поиска (155). Схема точного алгоритма нахождения кратчайшего покрытия (157). Программа окрапок 4 (157).

§ 2. Поиск минимальных разбиений 162
Получение точного решения (162). Метод прямого размещения (163). Анализ текущей ситуации и поиск хороших шагов (165). Выбор сомнительного шага (166). Общая структура алгоритма прямого размещения (167). Пример (168). Подпрограммы поиска и реализации хороших шагов (171). Программа прямого размещения (172). Перебор в критических ситуациях (173). Пример нахождения точного решения (175).

§ 3. Задачи диагностики 176
Диагностическая матрица (176). Диагностические тесты (178). Нахождение минимального теста (180). Алгоритм нахождения кратчайшего столбцового покрытия (182). Вспомогательные подпрограммы (184). Программа окрапок 5 (186). Приближенный алгоритм (188). Пример (189). Вспомогательные подпрограммы (190). Программы бедиэкс (194). Оценки эффективности программ (195).

§ 4. Канонизация булевых матриц 197
Булево пространство (197). Эквивалентность булевых матриц (198). Алгоритм β-канонизации (199). Анализ на β-эквивалентность (203).

§ 5. Троичные матрицы 204
Вариант задачи о размещении отдыхающих по лодкам (204). Сведение задачи к предыдущей (206). Кодирование троичных векторов и матриц (207). Л-программы (208). Вариант задачи диагностики (212) Отношения между троичными векторами (213). Интервалы булева пространства (215). α-эквивалентность троичных матриц (216). Простейшие равносильные преобразования (217).

§ 6. Сжатие троичных матриц 219
Постановка задачи (219). Применение операции склеивания (219). Нахождение одной из кратчайших форм (221). Полу-

чение множества максимальных строк (223). Примеры (225). Детали программирования (228).

§ 7. О вырожденных троичных матрицах 229

Задача анализа матрицы на вырожденность (229). Алгоритм анализа (230). Пример (232). Л-программы (234). Устранение избыточности в троичной матрице (237). Л-программа (239).

§ 8. К максимальному сжатию троичных матриц 239

Простая матрица поглощений (239). Вариант задачи о максимальном сжатии (241). Поиск простых совокупностей (241). Метод построения простой матрицы поглощений (242). Пример (244). О переборе сочетаний строк (246). Нахождение ядра безызбыточных форм (247). Приведение критерия простоты к более простому виду (250). Примеры (251). Нахождение квазиядра кратчайшей формы (255).

§ 9. Согласование частичных двублочных разбиений . . . 256

Частичные двублочные разбиения (256). Отношение импликации (258). Матрицы разбиений (259). Нахождение кратчайшей имплицирующей формы (260). Пример (261). Поиск приближения к кратчайшей имплицирующей форме (263). Пример (264). Л-программа (266). Оценка эффективности программы (267).

Глава 3. Булевы функции, логические уравнения, графы . . 269

§ 1. Булевы функции 369

Другая интерпретация булева пространства (269). Булевы функции (270). Алгебраические формы булевых функций (271). Дизъюнктивные нормальные формы (273). Операции импликации и равенства (276). Импликанты булевых функций (277).

§ 2. Минимизация дизъюнктивных нормальных форм . . . 279

Обсуждение постановки задачи (279). Исходная форма (280). Получение безызбыточной д н ф (281). Нахождение ядра (287). Получение множества простых импликант (289). Нахождение циклического остатка (291). Построение простой матрицы Квайна (298). Получение кратчайшей д н ф (305).

§ 3. Минимизация слабо определенных булевых функций 306

Булевы функции, неполностью и слабо определенные (306). Задача нахождения кратчайшей д н ф (307). Нахождение множества $\sup M^*$ (308). Вспомогательные Л-программы (310). Л-программа получения матрицы Квайна (311). Дальнейший путь решения задачи (313). Л-программы (314). Программы нахождения кратчайших д н ф (316).

§ 4. Метод конкурирующих интервалов 317

Постановка задачи (317). Метод решения (318). Схема алгоритма (321). Пример (322). Уточнения алгоритма (325). Пример для уточненного алгоритма (325). Л-программа (329). Вспомогательные программы (331).

§ 5. Эксперименты над алгоритмами 334

Об эффективности алгоритмов (334). Исследование эффективности алгоритмов получения кратчайшего покрытия (336). Результаты исследования (337). Организация эксперимента над программой минимизации д н ф булевых функций (340). Экспериментальные данные (342). Интерпретация результатов (344). Эксперименты с программой к в а с о б у (345). Сравнительный анализ эффективности программ минимизации слабо определенных булевых функций (351). Рандомизация программы м и н с о б (354).

§ 6. Решение системы логических уравнений 358

Постановка задачи (358). Алгоритм последовательного раскрытия скобок (361). Л-программа (365). Лексикографический перебор комбинаций (366). Упрощение системы уравнений (367). Поиск одного из корней системы (371).

§ 7. Задачи теории графов 372
Определение связности графа (372). Анализ графа на двусвязность (374). Л-программа (376). Нахождение множества достижимых вершин графа (377). Кратчайшие цепи в графе (378). Нахождение максимальных полных подграфов (382). Л-программы (385). Сравнение алгоритмов (387).

Глава 4. **Синтез дискретных автоматов** 388
§ 1. Функциональные модели автоматов 388
Релейные устройства и дискретные автоматы (388). Комбинационный автомат (391). Элементарные преобразователи (392). Конъюнкции элементарных событий и связь между ними (393). Пример (394). Последовательностные автоматы (396). Синхронный автомат (398). Матричная модель синхронного автомата (399). Граф поведения автомата (401). Частичные автоматы (403). Асинхронный автомат (407).

§ 2. Минимизация числа состояний 408
Эквивалентность состояний (408). Пример (410). Граф условий эквивалентности (410). Минимизация числа состояний полного автомата (411). Реализация частичных автоматов (413). Совместимость состояний (414). Совмещение состояний (415). Минимизация частичного автомата (419). Графическая формулировка задачи (421). Нахождение точного решения (423). Предостережение (428).

§ 3. Структурные модели автоматов 430
Элементы и базисы (430). Сети (434). Комбинационные сети (436). Синхронные сети (438). Асинхронные сети (439).

§ 4. Кодирование внутренних состояний 442
Кодирование состояний синхронного автомата (442). Состязания между элементами памяти асинхронного автомата (443). Критерий отсутствия опасных состязаний (444). Кодирующая матрица и требования к ней (446). Матричная формулировка задачи (446). Получение и упрощение матрицы условий (447). Нахождение кодирующей матрицы (449). Рассмотрение K-множеств (450). О неоптимальности метода прямых переходов (451). Соседнее кодирование состояний (454). Графические представления (455). Поиск решения (456).

§ 5. Синтез комбинационных автоматов 461
Общие положения (461). Базис произвольных д и ф (462). Реализация одной булевой функции в базисе произвольных д и ф (464). О реализации системы булевых функций (466). Минимизация числа конъюнкторов (467). Выбор элементарных конъюнкций (469). Обсуждение результата (472). Инверсные конъюнкторы и инверсные дизъюнкторы (473). Синтез многоярусных сетей (475).

§ 6. Третий уровень языка ЛЯПАС 476
Большие программы и связанные с ними проблемы (476). Суперпрограммы (479). Принципы реализации суперпрограмм (481). Пример суперпрограммы (482). О локальной эффективности алгоритмов (486). Метапрограммы (487). О методике построения метапрограмм (488).

§ 7. О программе синтеза асинхронных автоматов . . . 489
Постановка задачи (489). Определение макроструктуры программы (490). Пример реализации программы (492). Интерпретация результатов (496).

Приложение. Список подпрограмм 498

Литература 502

Предметный указатель 505

ПРЕДИСЛОВИЕ

Предлагаемая вниманию читателя книга посвящена рассмотрению логических задач, возникающих при проектировании релейных устройств. Так принято называть технические устройства, структура и поведение которых хорошо описываются с помощью дискретных переменных, главным образом булевых, то есть принимающих значение 0 или 1.

Эти задачи не так уже сильно отличаются от логических задач, известных читателю из популярной литературы: о волке, козе и капусте, о трех мудрецах с тремя колпаками, о выяснении запутанного соответствия между именами и специальностями некоторых лиц и т. д. При решении таких задач приходится оперировать, как правило, не с числами, а с некоторыми другими объектами: довольно сложными структурами, последовательностями, отношениями, сочетаниями и т. п. При этом оказывается, что читателю неизвестны какие-либо систематические методы решения этих задач, в связи с чем они рассматриваются как головоломки, испытывающие сообразительность читателя, и решение их находится путем обширного перебора различных предположений и догадок, в ходе которого проверяется допустимость перебираемых вариантов.

Действительно, в настоящее время отсутствует теория, позволяющая охватить достаточно широкий круг логических задач и предложить более или менее общие и в то же время эффективные методы их решения.

Между тем, логические задачи приобретают все большее значение. Достаточно заметить, например, что вся современная математика является преимущественно символической, поскольку подавляющая часть рассматриваемых в ней преобразований сводится к операциям над символами, представляющими собой отнюдь не числа.

Выполнение этих преобразований и в особенности их поиск можно рассматривать как типичные логические задачи. Формализация решения этих задач, то есть построение алгоритмов поиска и доказательства новых теорем, введения новых полезных понятий и т. п., становится все более насущной.

Указанная проблема находится в поле зрения математической логики, в которой, однако, вычислительная сторона развита пока довольно слабо, хотя уже первые шаги свидетельствуют о быстром прогрессе в данном направлении. Хорошей к этому иллюстрацией может служить выраженное в 1957 году мнение, что, например, «для доказательства теоремы 2.45 из «Principia Mathematica» Уайтхеда и Рассела потребуется машинное время порядка нескольких тысяч лет» *). Спустя три года была опубликована статья Ван Хао, в которой сообщалось о составленной им программе, затрачивающей на поиск доказательства упомянутой теоремы всего около трех секунд **). Как замечает в связи с этим Ван Хао, природа и размеры трудностей на пути автоматизации математического творчества оказались «преувеличенными из-за недооценки комбинированных возможностей математической логики и вычислительных машин».

Можно указать еще на ряд областей науки и техники, в которых решаемые задачи зачастую оказываются логическими. Эти задачи возникают при решении производственных проблем, связанных с выбором оптимальных вариантов в некоторых сложных ситуациях, при автоматизации перевода с одного языка на другой, при диагностике неисправностей в некоторой аппаратуре и т. д. Однако главным источником постановок логических задач (и потребителем методов их решения) в на-

*) Newell A., Shaw J. C., Simon H. A., Empirical explorations of the logic theory machine: a case study in heuristics, Report P-951, Rand Corporation, 1957, March. Русский перевод: Ньюэлл А., Шоу Дж., Саймон Г., Эмпирические исследования машины «Логик-теоретик»; пример изучения эвристики, Сб. «Вычислительные машины и мышление», 113—144, Изд. «Мир», М., 1967.

**) Wang Hao, Toward mechanical mathematics IBM J. Res. Devel., vol. 4, № 1, 2—22, 1960. Русский перевод: Ван Хао, На пути к механической математике. Кибернетический сборник, № 5, 114—165, ИЛ, М., 1962.

стоящее время является, по-видимому, теория дискретных автоматов.

Эта теория оперирует с разновидностями абстрактной модели релейного устройства, называемой дискретным автоматом. Основной проблемой данной теории является задача синтеза, исходной информацией для которой служит описание требуемых функциональных свойств автомата, а также перечень типов элементов, играющих при синтезе роль «кирпичей». В результате синтеза получается описание структуры автомата, то есть перечень использованных элементов и связей между ними.

Задача синтеза является, вообще говоря, весьма сложной, однако в каждом конкретном случае она может быть разложена на некоторую совокупность других, более простых задач логического характера: эквивалентные преобразования булевых выражений с целью оптимизации представляемых этими выражениями структур автоматов, решение систем логических уравнений, некоторые задачи теории графов и теории кодирования, операции со строчками символов (например поиск, подстановка, расширение и т. п.).

Многие из этих задач исследованы относительно хорошо, из чего, однако, не следует, что они решаются просто. Как правило, эти задачи характеризуются необходимостью рассмотрения многих вариантов, перебор которых неизбежен и может быть лишь в какой-то степени сокращен путем совершенствования решающих алгоритмов. Решение таких задач вручную весьма трудоемко и в ряде случаев практически невозможно, в связи с чем возникает необходимость в использовании с этой целью электронных вычислительных машин (ЭВМ).

Применение ЭВМ для автоматизации трудоемких расчетов стало уже довольно обыденным делом. Однако при этом львиную долю расходов машинного времени составляют расчеты численного характера, в которых основными объектами производимых операций служат числа. Математика оказалась достаточно хорошо подготовленной именно к такому использованию машин, поскольку методы численного анализа (состоящие главным образом из алгоритмов приближенных вычислений и способов оценки погрешностей) были уже достаточно сильно развиты и широко применялись до появления

ЭВМ. Хуже обстояло дело с методами решения логических задач, поскольку это направление стало интенсивно развиваться лишь в последние годы.

Это обстоятельство нашло отражение и в развитии работ по автоматизации программирования, приведя к тому, что большинство из созданных к настоящему времени алгоритмических языков и систем автоматического программирования ориентировано на задачи численного характера.

К числу немногих исключений из этого правила относится алгоритмический язык ЛЯПАС, описываемый в настоящей книге. Разработка данного языка была начата в конце 1962 года специально для представления алгоритмов решения логических задач, характерных для процессов синтеза дискретных автоматов. Язык ЛЯПАС создавался совместно с соответствующей системой автоматического программирования, что позволило добиться высокой эффективности системы, выражающейся в простоте программирования, компактности представления алгоритмов в языке и быстроте их трансляции на машинные языки, в результате которой получаются качественные машинные программы (компактные и быстродействующие). Параллельно и в тесной связи с этими работами проводились исследования в области алгоритмизации процессов синтеза, что обеспечило хорошую «настройку» системы на соответствующий круг задач.

Результаты работ как по развитию языка ЛЯПАС, так и по его применению были отражены в серии публикаций, из которых прежде всего стоит упомянуть тематические сборники [7, 11], а также монографию автора [3]. Настоящая книга отличается от этих публикаций прежде всего большей систематичностью изложения, упорядоченностью содержащегося в ней материала с единой точки зрения и ориентацией на значительно более широкий круг читателей. В нее включено также много нового материала.

Накопленный при эксплуатации системы ЛЯПАС опыт позволил внести в систему ряд коррекций, повышающих ее эффективность. Таким образом появился последний вариант системы (состоящей из алгоритмического языка ЛЯПАС и программ трансляции на машин-

ные языки), описываемый в настоящей книге и получивший наименование «система ЛЯПАС-70».

Заметим, что описание языка ЛЯПАС не составляет основного содержания настоящей книги, а лишь подготавливает средства для изложения последующего материала, оправдывающего название книги, а именно алгоритмов синтеза дискретных автоматов.

Одним из руководящих принципов, определяющих как методику разработки этих алгоритмов, так и правила их описания, является принцип иерархичности, по-видимому, неизбежный при оперировании со сложными системами. Этот принцип выражается в том, что рассматриваемый сложный объект расчленяется на некоторое, относительно небольшое число взаимосвязанных блоков, причем каждый из них оказывается уже существенно «проще» объекта в целом. Блоки, остающиеся все же довольно сложными, подвергаются следующему шагу разложения и т. д. Этот принцип особенно ценен при синтезе объектов, позволяя разбить весь процесс синтеза на серию этапов, на каждом из которых решается сравнительно несложная задача, соответствующая одному шагу разложения. Применение принципа иерархичности вполне оправдывается «узостью» человеческого внимания, позволяющего, например, воспринимать формулы из нескольких десятков символов, но неспособного охватить как единое целое более сложные формулы, если только в них не будет замечено некоторой регулярности.

Следует, однако, заметить, что обоснование принципа иерархичности имеет более глубокий характер, не будучи связано исключительно с особенностями человеческого восприятия. Действительно, достаточно обратиться к структурам, созданным в процессе эволюции жизни, чтобы убедиться в универсальности этого принципа, которому подчиняется строение любого живого организма. По-видимому, главной причиной распространения принципа является представляемая им возможность эффективного использования отработанных ранее структур при синтезе новой, более сложной структуры, или, если можно так выразиться, использования уже имеющихся средств решения более простых задач при отыскании способа решения вновь возникшей задачи. Очевидно,

что главным при таком подходе является умение разложить новую задачу на старые, то есть свести в основном ее решение к уже известным методам решения старых задач.

Многократное применение операции разложения приводит к получению многоступенчатых иерархических структур большой сложности, однако каждый шаг разложения оказывается сравнительно несложным, а число конкурирующих при этом вариантов разложения относительно небольшим; эти варианты часто можно перебрать и выбрать из них наилучший, решая таким образом задачу оптимизации структуры в рамках данного принципа.

Как мы уже отметили, в настоящей книге принцип иерархичности нашел применение главным образом при рассмотрении алгоритмов решения различных задач. Этот же принцип выразился в многоступенчатой структуре языка ЛЯПАС. Он же определил в значительной степени и характер самой книги.

В первой главе рассматриваются довольно простые объекты — булевы векторы — и вводится ряд несложных операций над ними. Эти элементарные построения положены в основу описываемого в этой же главе алгоритмического языка ЛЯПАС. Этим самым подготавливается язык для описания уже несколько более сложных объектов, рассматриваемых во второй главе. Там вводятся булевы и троичные матрицы и операции над ними, которые представляются затем как разложения на более простые операции над булевыми векторами, оформляемые в виде программ в языке ЛЯПАС. Решаемые во второй главе задачи обладают массой полезных интерпретаций, и некоторые из них давно известны в теории автоматов. Абстрактная формулировка этих задач позволяет пролить дополнительный свет на их алгоритмическую сущность и найти новые пути к повышению эффективности методов их решения. Она же облегчает использование решающих их алгоритмов в качестве строительного материала при построении алгоритмов решения более сложных задач, рассматриваемых в следующей, третьей главе. Здесь объектами рассмотрения являются булевы функции, логические уравнения, графы — основной материал для математических аппаратов, ис-

пользуемых в теории автоматов. Решаемые в этой главе задачи оказываются в общем случае еще более сложными, однако довольно хорошо разлагаются на задачи, рассматриваемые в предыдущей главе. Иерархическая структура такого разложения соответствующим образом отражается в предлагаемых программах. Следующая ступень усложнения приводит нас к задачам теории дискретных автоматов, рассматриваемым в четвертой главе. Эти задачи являются далеко не тривиальными, однако читатель уже подготовлен к восприятию алгоритмов их решения.

Можно считать, что в каждой из этих глав вводится некоторый язык, ориентированный на представление объектов определенного характера и определенной степени сложности. Действительно, описываемые в предшествующей главе алгоритмы образуют удобную систему для разложения других алгоритмов, рассматриваемых в следующей главе и образующих, в свою очередь, язык более высокого уровня, ориентированный на решение более сложных задач.

Другой характерной чертой настоящей книги является детальность описываемых в ней алгоритмов, обусловленная именно последовательным применением принципа иерархичности: разложение алгоритмов доводится до уровня элементарных операций над булевыми векторами. Как правило, алгоритмы решения рассматриваемых в книге задач оформляются в виде подпрограмм на языке ЛЯПАС, проверенных на вычислительной машине. Этим самым в максимальной степени облегчается их использование как при решении тех задач, для которых они непосредственно предназначены, так и в качестве частей более сложных программ, решающих другие задачи.

Многие из предлагаемых алгоритмов были подвергнуты экспериментально-статистическим исследованиям на вычислительной машине, целью которых являлось получение количественных оценок эффективности алгоритмов и выяснение границ их разумного применения. Полученные при этом результаты позволяют сравнивать различные алгоритмы, решающие одинаковые задачи, и для каждой конкретной ситуации выбирать лучший для нее алгоритм.

Уже говорилось, что основным источником рассматриваемых в настоящей книге задач послужила теория дискретных автоматов. Задачи этой теории довольно сложны, поэтому они излагаются в своей более или менее непосредственной форме только в четвертой главе, где и вводятся в связи с этим основные понятия теории, в том минимальном объеме, который необходим для постановки и анализа рассматриваемых задач. До этого, в предыдущих главах, ведется лишь подготовка к этому рассмотрению и исследуются более «очищенные» абстрактные задачи, например задача нахождения кратчайших покрытий, задача минимизации булевой функции или некоторые задачи теории графов. Такая подготовка значительно облегчает рассмотрение более сложных задач синтеза дискретных автоматов, поскольку в значительной степени их решение сводится именно к решению задач указанных типов.

Стоит подчеркнуть, что рассматриваемые в книге абстрактные задачи имеют значение далеко не только в теории дискретных автоматов (в частности, поэтому автор старался до поры до времени не навязывать постановкам этих задач «автоматную» интерпретацию, оставляя читателю достаточный простор для воображения). В связи с этим предлагаемые в книге алгоритмы решения этих задач могут представить интерес не только для специалистов в области проектирования релейных устройств, но и для представителей других специальностей.

В книге широко использованы результаты исследований в области теории автоматов и в смежных областях, нашедшие отражение во многих опубликованных работах, выборочная библиография которых приведена в конце книги. Однако в связи со спецификой настоящей книги эти результаты подверглись существенному преломлению и форма их изложения подчинена принятой в книге терминологии. Например, при рассмотрении некоторых методов минимизации булевых функций использованы работы Квайна [21, 22], однако изложение ведется в рамках построенной в книге модели, основными объектами которой являются булевы векторы и матрицы и которая оказывается более удобной при создании «машинных» алгоритмов. Определенная часть книги пред-

ставляет собой доведение известных методов решения некоторых задач до строгой алгоритмической формы, при котором, как правило, неизбежен критический пересмотр метода и его переработка с учетом особенностей машинного решения задачи. Наконец, ряд результатов носит существенно оригинальный характер, и здесь, прежде всего, следует отметить предлагаемые алгоритмы минимизации слабо определенных булевых функций большого числа переменных и булевых функций, заданных непосредственно в дизъюнктивной нормальной форме. Стоит упомянуть также алгоритмы нахождения кратчайших покрытий булевой матрицы и алгоритм анализа троичной матрицы на вырожденность, который можно интерпретировать как алгоритм, выясняющий, равна ли тождественно единице некоторая булева функция, заданная в дизъюнктивной нормальной форме. Удалось добиться высокой эффективности этих алгоритмов, что особенно важно потому, что, как показала практика их использования при решении других задач, область их возможного применения весьма широка.

Таким образом, книга в целом может служить достаточно полным руководством по языку ЛЯПАС и его применению для программирования задач логического характера. Она знакомит читателя и с основной проблематикой теории дискретных автоматов, однако главное внимание при этом обращается на вычислительные трудности решения различных задач этой теории.

Учитывая характер книги, автор счел возможным обходиться, как правило, без системы прямых ссылок на использованную литературу. Вместо этого к каждой главе дается библиография, которая, разумеется, не претендует на полноту и содержит лишь ту литературу, которая наиболее тесно связана с содержанием книги.

Большую помощь автору при подготовке рукописи оказал коллектив проблемной лаборатории счетно-решающих устройств Сибирского физико-технического института при Томском государственном университете, где и проводились многолетние исследования по развитию алгоритмического языка ЛЯПАС и его применению для решения задач синтеза дискретных автоматов. При написании параграфа 3 «Задачи диагностики» главы 2 был использован материал, любезно предоставленный

А. А. Уткиным, оказавшим также существенную помощь в улучшении текста рукописи в целом. В параграф 7 «Задачи теории графов» главы 3 включены некоторые результаты, полученные Ю. В. Поттосиным, а в § 9 главы 2 описывается программа, разработанная автором совместно с А. Я. Янковской. Параграф 2 «Минимизация дизъюнктивных нормальных форм» главы 3 написан автором совместно с С. В. Быковой, выполнившей к тому же всю работу по составлению, отладке и оптимизации программ, описываемых в данном параграфе, а также по организации их экспериментального исследования, некоторые из результатов которого приводятся в параграфе 5 этой же главы.

Представляемый в книге материал неоднократно обсуждался как в стенах лаборатории, так и за их пределами, что позволило своевременно выявлять и устранять разнообразные недостатки как логического, так и стилистического характера. Особенно ценной в этом отношении оказалась помощь со стороны П. П. Пархоменко, рецензировавшего рукопись этой книги.

ГЛАВА 1

ЯЗЫК ЛЯПАС

§ 1. Булевы векторы и действия над ними

Булев вектор. Переменная, принимающая значение 0 или 1, называется логической, или *булевой переменной*, а вектор, компонентами которого служат такие переменные, носит название *булева вектора.* Число компонент определяет *длину* или *размерность вектора.* Условимся нумеровать компоненты вектора слева направо, начиная с нуля. Например, вектор u размерности 8 представляет собой следующую последовательность компонент:

$$u^0\, u^1\, u^2\, u^3\, u^4\, u^5\, u^6\, u^7,$$

и если

$$u = 10011101,$$

то это означает, что

$$u^0 = 1, \quad u^1 = 0, \ \ldots, \ u^7 = 1.$$

Рассмотрение булевых векторов представляет особый интерес потому, что именно с этими векторами непосредственно оперируют современные цифровые вычислительные машины и именно такие векторы используются для выражения различных величин, задающих исходные данные для решаемых на машинах задач и получаемый результат. Значения булевых векторов могут при этом интерпретироваться самым различным образом. Отметим пока две интерпретации, наиболее распространенные.

Теоретико-множественная интерпретация. Поставим совокупность компонент булева вектора во взаимно однозначное соответствие с некоторым конечным множеством, число элементов которого равно длине рассматриваемого вектора. Это означает, что каждой из компонент вектора будет отвечать ровно один

из элементов множества и обратно, каждому из элементов множества будет отвечать только одна из компонент вектора. Задавая конечное множество в виде перечня его элементов, заключенного в фигурные скобки, условимся перечислять элементы множества в том же порядке, в котором следуют друг за другом, слева направо, компоненты булева вектора, поставленного в соответствие с данным множеством.

Например, утверждая, что булев вектор

$$\boldsymbol{u} \equiv u^0 u^1 u^2 u^3 u^4 u^5 u^6 u^7$$

соответствует множеству

$$A \equiv \{a, b, c, d, e, f, g, h\},$$

мы будем подразумевать, что компонента u^0 ставится в соответствие элементу a, компонента u^1 — элементу b и т. д., наконец, компонента u^7 ставится в соответствие элементу h.

Продолжая эту интерпретацию, положим, что конкретным значениям булева вектора соответствуют определенные подмножества того множества, которое поставлено в соответствие полной совокупности компонент данного вектора. Условимся считать, что эти подмножества образуются из тех элементов заданного множества, которым соответствуют компоненты со значением 1, и будем говорить, что именно эти подмножества *представляются* конкретными значениями булева вектора. Возвращаясь к предыдущему примеру, заметим, что если вектор $\boldsymbol{u}$ принимает значение 1 0 0 1 1 1 0 1, то он представляет подмножество $\{a, d, e, f, h\}$; представляя одноэлементное множество $\{d\}$, он принимает значение 0 0 0 1 0 0 0 0, и т. д. Пустое подмножество, не содержащее ни одного элемента и обозначаемое символом $\varnothing$, представляется значением 0 0 0 0 0 0 0 0.

Введем более формальный способ выражения указанной связи, основанный на использовании матриц бинарных отношений.

Булева матрица бинарного отношения. Различные объекты могут находиться в некоторых отношениях между собой, среди которых особое внимание заслуживают так называемые *бинарные отношения*, свя-

зывающие объекты попарно. Примером бинарного отношения может служить отношение «больше», представляемое с помощью символа $>$. Например, в таком отношении находятся числа 5 и 2 ($5 > 2$), но не находятся числа 3 и 7, то есть выражение $3 > 7$ будет неверным.

Допустим, что нам требуется представить некоторое бинарное отношение, условно обозначаемое символом $\circ$, между элементами множества $A = \{a_0, a_1, \ldots, a_{n-1}\}$, с одной стороны, и элементами множества $B = \{b_0, b_1, \ldots, b_{m-1}\}$, с другой стороны. Если эти множества достаточно малы, то для требуемого представления удобно построить *булеву матрицу* (прямоугольную таблицу, элементы которой принимают значения 0 или 1) рассматриваемого *бинарного отношения*, которую мы обозначим через $C = \|A \circ B\|$ и определим следующим образом: элемент c_i^j этой матрицы, находящийся на пересечении i-й строки и j-го столбца, принимает значение 1, если имеет место отношение $a_i \circ b_j$, и принимает значение 0 в противном случае.

Например, отношение «больше» между элементами множества $A = \{1, 2, 3, 4\}$ и множества $B = \{1, 2, 3, 4, 5, 6\}$ представляется следующей булевой матрицей бинарного отношения:

$$\|A > B\| = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Другим примером может служить выражение отношения принадлежности элементов множества

$$B = \{a, b, c, d, e, f, g, h\}$$

элементам множества

$$A = \{\{a, d, e\},\ \{c, f, g\},\ \{b, c, f, h\},\ \{d\},\ \{a, e\}\},$$

которыми, как нетрудно видеть, служат некоторые подмножества из B (можно считать, что $A = \{a_0, a_1, a_2, a_3, a_4\}$, где, в свою очередь, $a_0 = \{a, d, e\}$, $a_1 = \{c, f, g\}$, $a_2 = \{b, c, f, h\}$, $a_3 = \{d\}$ и $a_4 = \{a, e\}$). Это отношение

достаточно наглядно представляется матрицей *)

$$\| A \ni B \| = \begin{array}{c} \begin{matrix} a & b & c & d & e & f & g & h \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{matrix}$$

Условимся обозначать через $\|\{A\} \circ B\|$ однострочную булеву матрицу, выражающую отношение $\circ$ между единственным элементом A множества $\{A\}$, с одной стороны, и элементами множества B, с другой стороны.

Например, если $B = \{a, b, c, d, e, f, g, h\}$, $A = \{a, e, f, h\}$, $D = \{d\}$ и $E = \varnothing$, то

$$\|\{A\} \ni B\| = 10001101,$$
$$\|\{D\} \ni B\| = 00010000,$$
$$\|\{E\} \ni B\| = 00000000.$$

Числовая интерпретация. При данной интерпретации булевых векторов считается, что их значения представляют натуральные числа. В этом случае мы будем пользоваться широко известной *позиционной двоичной системой счисления*, задающей следующее соответствие между множеством 2^n различных значений булева вектора u размерности n и множеством 2^n целых чисел, начиная с нуля и кончая числом $2^n - 1$.

Значение булева вектора	Представляемое число	Значение булева вектора	Представляемое число
00 ... 0000	0	00 ... 0110	6
00 ... 0001	1	00 ... 0111	7
00 ... 0010	2	. . . . .	. . .
00 ... 0011	3	11 ... 1100	2^n-4
00 ... 0100	4	11 ... 1101	2^n-3
00 ... 0101	5	11 ... 1110	2^n-2
		11 ... 1111	2^n-1

*) В дальнейшем специальные обозначения строк и столбцов рассматриваемых булевых матриц могут не применяться, если в этом не будет особой нужды.

Связь между значениями компонент u^i булева вектора $\boldsymbol{u}$ и представляемым числом k устанавливается следующей формулой:

$$k = \sum_{i=0}^{n-1} 2^{n-1-i} u^i.$$

Условимся символически представлять эту связь выражением $k = [\boldsymbol{u}]$, полагая, например, что если $\boldsymbol{u} = 00001001$, то $[\boldsymbol{u}] = 2^{8-1-4} + 2^{8-1-7} = 2^3 + 2^0 = 9$, если $[\boldsymbol{u}] = 3$, то $\boldsymbol{u} = 00000011$, и т. д.

Введем ряд операций над булевыми векторами размерности n, для удобства разбив эти операции на четыре типа.

Логические операции. К этому типу мы отнесем операции, выполнение которых распадается на независимую реализацию некоторой из функций алгебры логики (а именно, инверсии, дизъюнкции, конъюнкции или дизъюнкции с исключением) для каждой из n компонент в отдельности.

Операция *инверсия* вектора $\boldsymbol{u}$ заключается в получении вектора, значение каждой компоненты которого отличается от значения соответствующей компоненты вектора $\boldsymbol{u}$. Поскольку значением любой компоненты булева вектора может служить либо 0, либо 1, это означает, что на местах единичных компонент вектора $\boldsymbol{u}$ в получаемом векторе будут стоять нули, а на местах нулевых компонент — единицы.

Операция инверсии записывается с помощью символа $\neg$, называемого оператором инверсии и ставящегося после символа вектора, который подвергается данной операции.

Например, если

$$\boldsymbol{u} = 01110000$$

и $\boldsymbol{w} = \boldsymbol{u}\neg$ (то есть вектор $\boldsymbol{w}$ получается в результате инверсии вектора $\boldsymbol{u}$), то

$$\boldsymbol{w} = 10001111.$$

Операции инверсии булева вектора соответствует теоретико-множественная операция *дополнение*. Действительно, если вектору $\boldsymbol{u}$ поставлено в соответствие

множество $A = \{a, b, c, d, e, f, g, h\}$ и значением 01110000 вектора u представляется подмножество $B = \{b, c, d\}$ множества A, то значением 10001111 вектора w, получаемого в результате инверсии вектора u, представляется подмножество $C = \{a, e, f, g, h\}$, содержащее те и только те элементы множества A, которые отсутствуют в подмножестве B. Другими словами, значением получаемого вектора w представляется множество $C = A \setminus B$, являющееся дополнением множества B до множества A.

Операция *дизъюнкция* векторов u и v приводит к получению вектора, значение каждой компоненты которого определяется следующим образом: компонента принимает значение 1 в том и только в том случае, когда значением 1 обладает хотя бы одна из соответствующих компонент векторов u и v. Эта операция записывается через $u \vee v$, где $\vee$ — оператор дизъюнкции.

Например, если

$$u = 01011100,$$
$$v = 11000101$$

и $w = u \vee v$, то

$$w = 11011101.$$

Операции дизъюнкции булевых векторов соответствует теоретико-множественная операция *объединение*. Так, если в данном примере значением 01011100 вектора u представлено множество $B = \{b, d, e, f\}$ и значением 11000101 вектора v представлено множество $C = \{a, b, f, h\}$, то значением 11011101 вектора w представляется множество $D = \{a, b, d, e, f, h\}$, являющееся объединением множества B и C, то есть множество $D = B \cup C$, содержащее те и только те элементы, которые принадлежат по крайней мере одному из множеств B или C.

Операция *конъюнкция* векторов u и v заключается в получении вектора, компоненты которого принимают значение 1 в том и только в том случае, когда значением 1 обладают соответствующие компоненты обоих векторов u и v. Оператором конъюнкции служит символ $\wedge$.

Операции конъюнкции булевых векторов соответствует теоретико-множественная операция *пересечение*.

Например, если

$$u = 11001100,$$
$$v = 10101010,$$

$w = u \wedge v$ и если, как и в предыдущих примерах, совокупности всех компонент каждого из рассматриваемых векторов поставлено в соответствие множество $A = \{a, b, c, d, e, f, g, h\}$, то вектор w принимает значение

$$10001000,$$

представляющее множество $D = B \cap C = \{a, e\}$, являющееся пересечением множеств $B = \{a, b, e, f\}$ и $C = \{a, c, e, g\}$, представляемых значениями векторов u и v. Это множество содержит те и только те элементы, которые принадлежат как множеству B, так и множеству C.

Операция *дизъюнкция с исключением*, выполняемая над векторами u и v, приводит к получению вектора, в котором значение 1 принимают только те компоненты, соответствующие которым компоненты векторов u и v имеют значение 1 лишь в одном из этих векторов. Оператором дизъюнкции с исключением служит символ $\oplus$.

Например, если

$$u = 10001001,$$
$$v = 01100111$$

и $w = u \oplus v$, то

$$w = 11101110.$$

Рассмотренные операции могут выражаться друг через друга. Нетрудно проверить, что если $w = u \vee v$, то $w\urcorner = (u\urcorner) \wedge (v\urcorner)$, если $w = u \oplus v$, то $w = (u \vee v) \wedge \wedge ((u \wedge v)\urcorner)$ и т. п.

Модульные арифметические операции. По аналогии с известными арифметическими операциями сложения, вычитания, умножения и деления чисел определим соответствующие операции над булевыми векторами, используя их числовую интерпретацию. Поскольку векторы обладают размерностью n, число их различных значений равно 2^n. Поэтому указанные операции в данном случае определяются по модулю 2^n, и это означает, что результат каждой из этих операций может отличаться от результата соответствующей

арифметической операции на $2^n \cdot k$, где k — произвольное целое число, и должен принадлежать диапазону чисел от 0 до $2^n - 1$. Назовем вводимые операции модульными арифметическими операциями.

Операция *сложение* булевых векторов u и v приводит к получению вектора, представляющего остаток от деления на 2^n арифметической суммы чисел $[u]$ и $[v]$. Иными словами, если $w = u + v$, то $[w] = ([u] + [v]) \bmod 2^n$. В этих формулах символ + служит оператором арифметического сложения, если он стоит между символическими выражениями чисел; если же он окружен непосредственно символами булевых векторов, он играет роль оператора сложения булевых векторов.

Например, если

$$u = 00001001,$$

$$v = 00000011$$

и $w = u + v$, то

$$w = 00001100.$$

Действительно, согласно введенной ранее формуле

$$[u] = \sum_{i=0}^{n-1} 2^{n-1-i} u^i,$$

$[u] = 2^{8-1-4} + 2^{8-1-7} = 2^3 + 2^0 = 9$. Аналогично определяется значение $[v] = 3$. Следовательно, $[w] = (9 + 3) \bmod 2^8 = 12 = 2^3 + 2^2 = 2^{8-1-4} + 2^{8-1-5}$.

Если же

$$u = 11001010,$$

$$v = 01101101$$

и $w = u + v$, то

$$w = 00110111,$$

поскольку $[u] = 202$, $[v] = 109$, а $(202 + 109) \bmod 2^8 = 55 = [w]$.

Полезно иметь представление о механизме реализации этой операции, для чего целесообразно ввести в рассмотрение вспомогательный вектор t, называемый вектором переноса и обладающий той же размерностью n.

Вектор t определяется следующим образом: его произвольная i-я компонента принимает значение 1 в том

случае, когда не менее двух из соседних справа $(i+1)$-вых компонент векторов u, v и t обладают значением 1, и принимает значение 0 в противном случае (крайняя справа компонента вектора t всегда имеет значение 0). Компоненты вектора w непосредственно зависят от соответствующих компонент векторов u, v и t: w^i принимает значение 1 в том случае, когда число таких компонент из u^i, v^i и t^i, которые обладают значением 1, оказывается нечетным, и принимает значение 0, когда это число оказывается четным.

Дополнив рассмотренные выше примеры значением вектора переноса t, нетрудно проследить выполнение описанной операции:

$$u = 00001001,$$
$$v = 00000011,$$
$$t = 00000110,$$
$$w = 00001100$$

и

$$u = 11001010,$$
$$v = 01101101,$$
$$t = 10010000,$$
$$w = 00110111.$$

Аналогично определяется операция *вычитание* булева вектора v из булева вектора u. Результат этой операции представляет остаток от деления на 2^n арифметической разности $[u]-[v]$, увеличенной на 2^n: если $w = u - v$, то $[w] = ([u]-[v]+2^n) \bmod 2^n$.

Например, если

$$u = 00001001,$$
$$v = 00000011$$

и $w = u - v$, то

$$w = 00000110,$$

если же

$$u = 00000101,$$
$$v = 00001010$$

и $w = u - v$, то

$$w = 11111011.$$

Заметим, что эта операция реализуется не более сложным по сравнению со сложением способом, использующим известную процедуру «заема» в соседних слева разрядах.

Операция *перемножение* булевых векторов u и v определяется как такая операция, в результате которой получается вектор, представляющий остаток от деления на 2^n арифметического произведения $[u]$ на $[v]$: если $w = u \times v$, то $[w] = [u] \cdot [v] \bmod 2^n$.

Например, если $w = u \times v$,

$$u = 00000101,$$
$$v = 00001100,$$

то

$$w = 00111100,$$

если же $w = u \times v$,

$$u = 01000110,$$
$$v = 10001100,$$

то

$$w = 01001000.$$

Чтобы облегчить в дальнейшем выражение операции арифметического перемножения через операции над булевыми векторами, введем наряду с уже рассмотренной операцией операцию *перемножение с округлением*, поставив ей в соответствие оператор $\overline{\times}$ определяемый следующим образом: если $w = u \overline{\times} v$, то $[w] = ([u] \cdot [v] - [u \times v]) : 2^n$.

Например, если $w = u \times v$.

$$u = 01000110,$$
$$v = 10001100,$$

то

$$w = 00100110.$$

Наконец, определим правила деления вектора u на вектор v. Поскольку $[u]$ может не делиться на $[v]$ нацело, введем две операции, одна из которых обеспечивает получение целой части частного, другая — остатка от деления $[u]$ на $[v]$.

Результатом операции *деление*[1] вектора u на вектор v служит вектор, представляющий целую часть частного арифметического деления $[u]$ на $[v]$. Соответствующий оператор обозначается через $:^1$.

Результатом операции *деление*[2] вектора u на вектор v служит вектор, представляющий остаток от арифметического деления $[u]$ на $[v]$. Соответствующий оператор обозначается через $:^2$.

Например, если $w = u :^1 v$, $x = u :^2 v$,

$$u = 00001101,$$
$$v = 00000101,$$

то

$$w = 00000010,$$
$$x = 00000011.$$

Заметим, что обе операции деления u на v не определяются для случая, когда $[v] = 0$.

Предоставим читателю возможность убедиться в справедливости следующих выражений:

$$[u] \cdot [v] = [u \overline{\times} v] \cdot 2^n + [u \times v],$$
$$u = ((u :^1 v) \times v) + (u :^2 v).$$

Характеристические операции. К этому типу относятся следующие две операции, служащие для определения некоторых характеристик булевых векторов.

Операция *взвешивание* булева вектора u состоит в подсчете числа p единичных компонент вектора и выражается через $u\nabla$: если $w = u\nabla$, то $[w] = p$.

Например, если $w = u\nabla$ и

$$u = 01100101,$$

то

$$w = 00000100,$$

то есть $[w] = 4$.

Операция *нахождение номера левой единицы* вектора u определяется своим названием и обозначается через $u\vdash$: если $w = u\vdash$, то $[w] = i$, где i — номер самой левой из компонент вектора u, имеющих значение 1.

Например, если $w = u \vdash$ и

$$u = 00010110,$$

то

$$w = 00000011.$$

Операции сдвига. Относящиеся к этому типу операции осуществляют некоторые простейшие перестановки компонент булева вектора.

Операция *сдвиг влево* булева вектора u на величину, задаваемую значением вектора v, заключается в получении вектора, i-я компонента которого принимает значение $(i + [v])$-й компоненты вектора u, если $i + [v] < n$, и значение 0 — в противном случае. Записью этой операции служит выражение $u < v$.

Например, если $w = u < v$,

$$u = 01101010,$$
$$v = 00000100,$$

то

$$w = 10100000.$$

Аналогично определяется операция *сдвиг вправо*, которой соответствует оператор $>$. Если $w = u > v$, то i-я компонента вектора w принимает значение $(i - [v])$-й компоненты вектора u, если $i - [v] \geqslant 0$, и значение 0 — в противном случае.

Например, если $w = u > v$,

$$u = 01101010,$$
$$v = 00000101,$$

то

$$w = 00000011.$$

Операция *нормализация* вектора u заключается в смещении компонент вектора влево до тех пор, пока на место крайней слева компоненты не станет единица (исключение составляет случай, когда все компоненты вектора u имеют значение 0: в этом случае значение вектора u не меняется). Величина смещения при этом оказывается равной m компонентам, где m — номер самой левой из компонент вектора u, имеющих значение 1.

Нулевые значения m левых компонент вектора передаются m правым компонентам вектора, представляющего результат операции. Оператором нормализации служит символ $\vdash\!\leftarrow$.

Например, если $w = u \vdash\!\leftarrow$ и

$$u = 00010110,$$

то

$$w = 10110000,$$

а если

$$u = 10011000,$$

то w имеет то же значение

$$10011000,$$

что и u (поскольку левая компонента подвергающегося нормализации вектора u имеет значение 1). Если же $w = u \vdash\!\leftarrow$ и

$$u = 00000000,$$

то

$$w = 00000000.$$

§ 2. Операнды языка ЛЯПАС

Описанные выше операции над булевыми векторами используются в качестве элементов языка, играющего роль посредника между человеком и универсальной цифровой вычислительной машиной (УЦВМ) и созданного для представления алгоритмов синтеза дискретных автоматов, благодаря чему он и получил название ЛЯПАС (логический язык для представления алгоритмов синтеза). Следует, однако, заметить, что область его эффективного применения оказалась значительно шире, захватывая широкий круг логических задач, то есть таких задач, в которых основную роль играют не числа (использование последних в данном случае ограничивается областью натуральных чисел), а множества, отношения, функции алгебры логики, графы и другие величины, достаточно просто задаваемые посредством булевых векторов.

Машинное представление булевых векторов. При решении задач на УЦВМ булевы векторы,

в свою очередь, представляются состояниями ячеек запоминающих устройств и разнообразных регистров, каждый из которых состоит из некоторой совокупности элементов, обладающих двумя устойчивыми состояниями. Одно из этих состояний условно называют нулевым, другое — единичным, и, в зависимости от того, в каком из состояний находится элемент, говорят, что на нем «записан» либо 0, либо 1. Состояние регистра в целом определяется как комбинация состояний его элементов и, если регистр состоит из n двоичных элементов, может представлять значение булева вектора размерности n. Аналогично интерпретируется состояние ячейки запоминающего устройства, из чего следует, что вся информация, с которой оперирует машина, задается в форме некоторой совокупности булевых векторов. Размерность этих векторов, которая может быть различной для вычислительных машин разных типов, для современных УЦВМ не выходит, как правило, из диапазона трех — шести десятков.

Выбор размерности булевых векторов. Для начального этапа развития вычислительной техники был характерен индивидуальный выбор указанной размерности, производимый при проектировании каждого нового типа УЦВМ согласно критерию повышения эффективности конкретных вычислительных машин. Появившееся вследствие этого разнообразие размерностей булевых векторов, с которыми оперируют современные УЦВМ, превратилось в одно из основных препятствий на пути наметившейся в последнее время тенденции к созданию универсальных языков программирования, универсальных в том смысле, что они должны быть общими для машин различных типов.

Язык ЛЯПАС также является универсальным в данном смысле, причем эта универсальность обеспечивается главным образом путем рациональной стандартизации размерности булевых векторов, представляющих преобразуемую информацию. Эти векторы должны вписываться в ячейки УЦВМ практически любого типа, в то же время достаточно полно используя их (следует помнить, что скорость обработки информации быстро растет по мере увеличения размерности используемых векторов). По ряду соображений, связанных с особенностями

решения логических задач (например, с тем обстоятельством, что для представления функции алгебры логики от n переменных удобно использовать булев вектор размерности 2^n), размерность булевых векторов целесообразно выбирать таким образом, чтобы она равнялась целой степени двойки.

С учетом всех перечисленных условий и выбрана *стандартная размерность* булевых векторов, равная $2^5 = 32$. Множество различных значений, которые может принимать булев вектор стандартной размерности, условимся обозначать через U. Очевидно, что мощность этого множества (число его элементов) равна 2^{32}.

Выражение алгоритма решения какой-либо задачи в алгоритмическом языке, воспринимаемом вычислительной машиной, называется *программой* и представляет собой некоторую последовательность символов данного языка. Принято делить эти символы на три класса: *операнды* — символы преобразуемых величин, *операторы* — символы выполняемых над ними операций и *метки* — вспомогательные символы, используемые при построении программы. В свою очередь, операнды языка делятся на типы, с целью обеспечения удобств при представлении величин различного характера.

В язык ЛЯПАС введены следующие типы операндов.

Переменные. Так называются операнды, обозначаемые символами

$$a,\ b,\ c,\ d,\ e,\ f,\ g,\ h,\ i,\ j,\ k,\ l,\ m,\ n,\ p,$$
$$q,\ r,\ s,\ t,\ u,\ v,\ w,\ x,\ y,\ z,\ ж,\ л,\ ф,\ ш,\ э,\ ю,\ я.$$

Они принимают значения из U, то есть являются булевыми векторами стандартной размерности, и обычно интерпретируются в теоретико-множественном смысле.

Основные комплексы. Основные комплексы (или просто *комплексы*) обозначаются символами

$$A,\ B,\ C,\ D,\ E,\ F,\ G,\ H,\ I,\ J,\ K,\ L,\ M,\ N,\ P,\ Q,\ R,\ S,$$
$$T,\ U,\ V,\ W,\ X,\ Y,\ Z,\ Ж,\ Л,\ Ф,\ Ш,\ Э,\ Ю,\ Я$$

и представляют собой некоторые последовательности булевых векторов стандартной размерности, то есть являются цепочками элементов, каждый из которых принимает значение из U.

Мощность комплекса (число элементов в нем) условимся представлять с помощью символа σ, считая, например, что $\sigma(A)$ есть мощность комплекса A. Элементы комплекса будем нумеровать, полагая, что начальный элемент имеет нулевой номер. Комплекс можно задавать непосредственным перечислением его элементов, обозначая их соответствующими строчными буквами с нижним индексом — номером: например, выражение

$$A = \{a_0, a_1, a_2, a_3, a_4, a_5\}$$

означает, что комплекс A содержит всего 6 элементов, следующих друг за другом в указанном порядке и пронумерованных соответствующим образом.

Комплексы могут рассматриваться как булевы (состоящие из нулей и единиц) матрицы с 32 столбцами и числом строк, равным мощности комплекса.

Индексы. Индексы обозначаются через

$$a, b, c, d, e, f, g, h, i, j, k, l, m, n, p, q, r,$$
$$s, t, u, v, w, x, y, z, ж, л, ф, ш, э, ю, я$$

и принимают значения из U наравне с переменными, но, в отличие от последних, эти значения интерпретируются как числа. В связи с этим индексы могут быть использованы при обозначении отдельных элементов комплексов. Например, через a_c обозначается $[c]$-й элемент комплекса A. Очевидно, что для представления любого элемента этого комплекса достаточно лишь присвоить соответствующее значение индексу c.

Константы. Константы образуют четвертый (и последний) тип операндов. Они обладают фиксированными значениями и, в свою очередь, подразделяются на натуральные и стандартные.

Натуральные константы представляют целые числа в диапазоне от 0 до 127, и символами этих констант служат непосредственные обозначения представляемых чисел. Значения этих операндов выбираются из U, причем при интерпретации выбранных значений как чисел оказывается, что результат интерпретации всегда совпадает с символом данной натуральной константы.

Заметим, что для удобства последующего изложения при представлении констант языка, а также при нуме-

рации элементов комплексов и компонент булевых векторов принимается восьмеричная система счисления, в которой ряд целых неотрицательных чисел представляется в виде последовательности

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, ..., 76, 77, 100, 101, ...

Натуральные константы образуют множество, представленное следующей ниже таблицей, в левом столбце которой заданы символы натуральных констант, а в правом — их векторные значения:

0 — 0000 0000 0000 0000 0000 0000 0000 0000
1 — 0000 0000 0000 0000 0000 0000 0000 0001
2 — 0000 0000 0000 0000 0000 0000 0000 0010
. . . . . .
7 — 0000 0000 0000 0000 0000 0000 0000 0111
10 — 0000 0000 0000 0000 0000 0000 0000 1000
. . . . . .
176 — 0000 0000 0000 0000 0000 0000 0111 1110
177 — 0000 0000 0000 0000 0000 0000 0111 1111

Стандартные константы образуют следующее множество, представляемое аналогичным образом:

c_0 — 1000 0000 0000 0000 0000 0000 0000 0000
c_1 — 0100 0000 0000 0000 0000 0000 0000 0000
c_2 — 0010 0000 0000 0000 0000 0000 0000 0000
. . . . . .
c_{37} — 0000 0000 0000 0000 0000 0000 0000 0001
d_0 — 1010 1010 1010 1010 1010 1010 1010 1010
d_1 — 1100 1100 1100 1100 1100 1100 1100 1100
d_2 — 1111 0000 1111 0000 1111 0000 1111 0000
d_3 — 1111 1111 0000 0000 1111 1111 0000 0000
d_4 — 1111 1111 1111 1111 0000 0000 0000 0000
e_0 — 0101 0101 0101 0101 0101 0101 0101 0101
e_1 — 0011 0011 0011 0011 0011 0011 0011 0011
e_2 — 0000 1111 0000 1111 0000 1111 0000 1111
e_3 — 0000 0000 1111 1111 0000 0000 1111 1111
e_4 — 0000 0000 0000 0000 1111 1111 1111 1111
f_0 — 0000 0000 0000 0000 0000 0001 1000 0000
f_1 — 0000 0000 0000 0000 0000 0001 1100 0000
f_2 — 0000 0000 0000 0000 0000 0001 1110 0000
f_3 — 0000 0000 0000 0000 0000 0001 1111 0000

f_4 — 0000 0000 0000 0000 0000 0001 1111 1000
f_5 — 0000 0000 0000 0000 0000 0001 1111 1100
f_6 — 0000 0000 0000 0000 0000 0001 1111 1110
f_7 — 0000 0000 0000 0000 0000 0001 1111 1111
f_{10} — 1111 1111 1111 1111 1111 1111 1111 1111

Константы $c_0, c_1, \ldots, c_{37}$ замечательны тем, что каждая из них имеет лишь одну единичную компоненту и номер этой компоненты совпадает с номером константы в данной серии.

Специальные комплексы. Совокупность констант $c_0, c_1, \ldots, c_{37}$ образует специальный комплекс C, выбор отдельных элементов которого может производиться с помощью индексов, так же, как и в случае рассмотренных выше комплексов A, B, C, ..., $Я$. Аналогично определяются специальные комплексы D, E и F, представляющие соответствующие серии стандартных констант $d_0, d_1, \ldots, d_4$; $e_0, e_1, \ldots, e_4$; $f_0, f_1, \ldots, f_{10}$.

Мы рассмотрели пока 4 из общего числа 16 специальных комплексов

$$A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q,$$

основное отличие которых от основных комплексов A, B, ..., $Я$ заключается в том, что как число, так и смысл элементов каждого специального комплекса фиксированы. С другими специальными комплексами мы встретимся в следующих параграфах. Здесь упомянем еще специальный комплекс B, содержащий информацию о мощностях комплексов A, B, C, ..., $Я$, A, B, ..., Q. Этот комплекс состоит из 48 элементов $b_0, b_1, b_2, \ldots, b_{57}$ (вспомним, что нумерация элементов комплекса производится в восьмеричной системе счисления), причем

$$[b_0] = \sigma(A), \quad [b_1] = \sigma(B), \ldots, [b_{37}] = \sigma(Я),$$
$$[b_{40}] = \sigma(A), \ldots, [b_{57}] = \sigma(Q).$$

Обратим внимание на некоторые различия между двумя возможными способами выбора элементов комплексов. При одном из этих способов номер элемента указывается посредством соответствующей натуральной константы, и этот способ, естественно, применим лишь в том случае, когда данный номер не превышает 177. Номер выбираемого элемента при этом жестко фикси-

руется. Другой способ, основанный на использовании индексов, является более гибким в этом смысле: выше уже отмечалось, что, например, посредством выражения $\boldsymbol{a}_c$ можно представить любой из элементов комплекса $\boldsymbol{A}$, достаточно лишь подобрать соответствующее значение для индекса c. Кроме того, при этом способе практически отсутствует ограничение на величину номера представляемого элемента комплекса.

Символика типов операндов. В заключение данного параграфа построим общую классификацию рассмотренных операндов, введя специальные символы для обозначения представителей соответствующих классов.

Через $\pi_п$ обозначим операнд, относительно которого известно лишь, что он принадлежит множеству переменных $\{\boldsymbol{a}, \boldsymbol{b}, \ldots, \boldsymbol{я}\}$, то есть

$$\pi_п \in \{\boldsymbol{a}, \boldsymbol{b}, \ldots, \boldsymbol{я}\}.$$

Аналогично вводятся обозначения для элементов других классов:

$$\pi_и \in \{a, b, \ldots, я\},$$

$$\pi_ч \in \{0, 1, \ldots, 177\},$$

$$\pi_с \in \{c_0, c_1, \ldots, c_{37}, d_0, d_1, \ldots, d_4, e_0, e_1, \ldots, e_4, f_0, f_1, \ldots, f_{10}\},$$

$$\pi_к \in \{\boldsymbol{A}, \boldsymbol{B}, \ldots, \boldsymbol{Я}, A, B, \ldots, Q\},$$

$$\pi_к^- \in \{\boldsymbol{A}, \boldsymbol{B}, \ldots, \boldsymbol{Я}, A, B, G, H, \ldots, Q\},$$

$$\pi_о \in \{\boldsymbol{A}, \boldsymbol{B}, \ldots, \boldsymbol{Я}\},$$

$$\pi_{пи} \in \{\pi_п, \pi_и\},$$

$$\pi_{ич} \in \{\pi_и, \pi_ч\},$$

$$\pi_э \in \{\pi_п, \pi_и, \pi_ч, \pi_с\},$$

$$\pi^- \in \{\pi_п, \pi_и, \pi_{эк}^-\},$$

$$\pi \in \{\pi_э, \pi_{эк}\},$$

где через $\pi_{эк}$ обозначен произвольный элемент, принадлежащий какому-либо из комплексов $\boldsymbol{A}, \boldsymbol{B}, \ldots, \boldsymbol{Я}, A, B, \ldots, Q$, а через $\pi_{эк}^-$ — произвольный элемент, который может принадлежать любому комплексу, за исключением комплексов C, D, E и F. Обозначим также

через $\pi_{эск}$ произвольный элемент некоторого специального комплекса A, B, ..., Q, а через $\pi^{-}_{эск}$ — произвольный элемент любого из специальных комплексов, кроме комплексов C, D, E и F.

§ 3. Линейные программы

Л-программа. Представление алгоритма решения какой-либо задачи в языке ЛЯПАС называется *Л-программой.* Л-программа представляет собой цепочку символов языка ЛЯПАС, замыкаемую символом «.» (*меткой конца программы*) и отображающую некоторую последовательность операций, реализация которой приводит к получению решения соответствующей задачи. В простейшем случае, когда Л-программа является *линейной*, операции выполняются в том же порядке, в котором они представлены в Л-программе, то есть подряд, и каждая из них реализуется ровно один раз. В данном параграфе мы ограничимся рассмотрением именно таких Л-программ.

В дальнейшем, за исключением случаев, когда это может привести к каким-либо недоразумениям, будем называть Л-программы просто *программами.*

Вычислительные операции. В первом параграфе был рассмотрен ряд одноместных операторов, каждый из которых действует на один операнд (примеры одноместных операторов: $\neg$, ∇, $\vdash\!\leftarrow$ и т. д.) и ряд двуместных операторов, каждый из которых действует на два операнда ($\vee$, $\oplus$, $-$, $>$ и т. д.). Введем общие символы для их обозначения:

$$\rho_1 \in \{\neg, \nabla, \vdash\!\leftarrow, \ldots\},$$
$$\rho_2 \in \{\vee, \oplus, -, >, \ldots\}.$$

Выражения типа $\pi\rho_1$ (например, $a\nabla$, $d\vdash\!\leftarrow$) и $\pi\rho_2\pi$ (например, $b + 5$, $m \wedge n$, $c_a > 12$), содержащие информацию об операторе и операндах, на которые этот оператор действует, представляют *элементарные вычислительные операции.* В двуместных операциях фигурируют левые и правые операнды (в перечисленных примерах двуместных операций левыми оказываются операнды

b, m, c_a, правыми — 5, n, 12, в одноместных — лишь левые (a, d)).

Цепочка выражений, представляющих элементарные вычислительные операции, является линейной программой. Длину цепочки в целом удается сократить, учитывая близкую связность, типичную для большинства вычислительных процессов и выражающуюся в том, что результат предшествующей операции весьма часто задает значение операнда последующей операции.

Введем *собственную переменную* τ, которая не будет фигурировать в программах в явном виде, но всегда будет подразумеваться. Положим, что она принимает значения из U и всегда представляет результат реализации предшествующей операции, пробегая, таким образом, в процессе реализации программы некоторый ряд значений. В то же время τ может служить левым операндом последующей операции.

Далее, условимся выражать одноместные и двуместные операции в программе через ρ_1 и $\rho_2\pi$ (подразумеваются же $\tau\rho_1$ и $\tau\rho_2\pi$, соответственно). К числу допустимых операций отнесем также операцию присвоения собственной переменной τ значения операнда π, обозначаемую в программе просто через π.

Таким образом, выражение

$$a + b$$

будет рассматриваться как составное, представляющее две операции: операцию присвоения собственной переменной τ значения индекса a, обозначаемую через a, и операцию сложения полученного переменной τ значения со значением индекса b, обозначаемую через $+b$.

В первом параграфе были рассмотрены две операции деления булевых векторов, порождаемые операторами $:^1$ и $:^2$. Введя соответствующую элементарную операцию в язык ЛЯПАС, мы объединим операторы $:^1$ и $:^2$ в один оператор $:$, определив его следующим образом: при реализации операции *деление*, записываемой в виде $:\pi$, собственная переменная τ принимает значение, представляющее остаток от деления числа, заданного предшествующим значением τ, на $[\pi]$, а целая часть частного будет представлена значением, принимаемым индексом $я$.

Заметим, что индекс *я* принадлежит к операндам специального назначения, которые нельзя использовать при программировании наравне с другими операндами и к которым относятся все индексы и переменные, обозначаемые буквами русского алфавита *ж*, *л*, *ф*, *ш*, *э*, *ю*, *я*. Для чего именно они предназначены, увидим позднее.

Простейший пример линейной программы. Рассмотрим небольшую программу, реализация которой приводит к выбрасыванию из булева вектора, представленного значением переменной ***а***, всех единиц, кроме правой.

Допустим, что переменная ***а*** имеет следующее значение:

$$00110100$$

(в целях простоты в последующих примерах мы будем ограничиваться, как правило, рассмотрением 8-компонентных булевых векторов, вместо стандартных 32-компонентных), и запишем указанную программу:

$$\boldsymbol{a} \urcorner + 1 \wedge \boldsymbol{a}.$$

Программа состоит из четырех операций, символически представленных в левом столбце следующей ниже таблицы, называемой *таблицей реализации программы*. В правом столбце этой таблицы приводятся значения, принимаемые собственной переменной τ и представляющие результаты реализации данных операций:

$\boldsymbol{a}$	00110100
$\urcorner$	11001011
$+1$	11001100
$\wedge \boldsymbol{a}$	00000100

Результат решения задачи в целом также оказывается представленным переменной τ, ее последним значением.

Операции присвоения значений. Реализация рассмотренных до сих пор операций не может привести к изменению значений каких-либо операндов, кроме τ. В связи с этим, чтобы обеспечить возможность присвоения нового значения любому операнду (разу-

меется, кроме констант), дополним список операций следующими операциями.

Операция *присвоение*, представляемая символически как $\Rightarrow \pi^{-}$, заключается в присвоении операнду π^{-} текущего значения собственной переменной τ (напомним, что операндом типа π^{-} может служить переменная, индекс или элемент какого-либо комплекса, не являющийся константой).

Например, если $a = 00110100$, то результатом выполнения программы

$$a \urcorner + 1 \wedge a \Rightarrow b,$$

отличающейся от только что рассмотренной программы наличием операции $\Rightarrow b$, явится принятие переменной b значения

$$00000100.$$

Обобщим оператор $\Rightarrow$ на случай обмена информацией между группами операндов, рассмотрев выражение типа

$$(\xi_1\ \xi_2\ \ldots\ \xi_k) \Rightarrow \pi_0,$$

где $\xi_i \in \{\pi_п, \pi_и, \pi_ч, \pi_о, \pi_{эск}\}$, а $i \in \{1, 2, \ldots, k\}$. Положим, что это выражение представляет операцию расширения основного комплекса π_0 путем добавления новых элементов, принимающих значения операндов $\xi_1, \xi_2, \ldots, \xi_k$, с сохранением порядка их следования. При этом значение каждого из операндов типа $\pi_п$, $\pi_и$, $\pi_ч$ или $\pi_{эск}$ будет присвоено одному из новых элементов расширяемого комплекса, если же некоторый из операндов ξ_i оказывается комплексом, то естественно, что для его представления в расширяемый комплекс потребуется добавить столько элементов, сколько их имеется в комплексе ξ_i.

Например, если $\sigma(A) = 5$, то есть если комплекс A состоит из пяти элементов a_0, a_1, a_2, a_3 и a_4, то реализация операции

$$(a\ c\ p\ b_1) \Rightarrow A$$

выразится в том, что в комплекс A будут добавлены четыре новых элемента a_5, a_6, a_7 и a_{10}, причем a_5 примет значение переменной a, a_6 примет значение

индекса c и т. д. Если же $\sigma(A) = 5$ и $\sigma(B) = 3$, то при реализации операции

$$(b\ d\ e\ n\ B) \Rightarrow A$$

комплекс A будет расширен на семь элементов a_5, a_6, a_7, a_{10}, a_{11}, a_{12} и a_{13}, причем последний из них примет значение элемента b_2 комплекса B, предпоследний — значение элемента b_1, и т. д.

Заметим, что если некоторый из входящих в рассмотренное выражение операндов ξ_i оказывается комплексом, то этот комплекс не должен быть в то же время расширяемым комплексом, то есть будем считать, что выражения типа $(acA) \Rightarrow A$ смысла не имеют.

Положим, что выражение

$$\pi_0 \Rightarrow (\xi_1\ \xi_2\ \ldots\ \xi_k),$$

где $\xi_i \in \{\pi_п, \pi_и, \pi_о, \pi_{эск}^-\}$ и $i \in \{1, 2, \ldots, k\}$ представляет обратную операцию — «распаковку» комплекса π_0 с конца, производимую в таком порядке, что ξ_k (если это не комплекс) принимает значение последнего элемента комплекса π_0, ξ_{k-1} — предпоследнего, и т. д. Если же некоторый из операндов ξ_i оказывается комплексом, то считается, что мощность этого комплекса должна быть известна заранее. Элементы этого комплекса принимают значения соответствующих элементов «распаковываемого» комплекса. При этом элементы, «отдающие» свои значения, выбрасываются из «распаковываемого» комплекса π_0, мощность которого соответственно уменьшается. Очевидно, что эта операция определяется лишь для случая, когда мощность комплекса π_0 не меньше суммарной мощности операндов, перечисляемых в скобках.

Например, если $\sigma(A) = 7$ и $\sigma(C) = 3$, то при реализации операции

$$A \Rightarrow (n\ C\ q)$$

индекс q примет значение элемента a_6, элементы c_2, c_1 и c_0 комплекса C примут значения элементов a_5, a_4 и a_3, соответственно, и переменная n приобретет значение элемента a_2. После этого в комплексе A останутся лишь два элемента a_0 и a_1.

Нетрудно проверить, что рассмотренные две операции при определенных условиях обратимы. Например, после реализации выражения

$$(a\ c\ m\ b) \Rightarrow A\ \ A \Rightarrow (a\ c\ m\ b)$$

окажется, что операнды a, c, m, b и A примут свои первоначальные значения.

Введем также операцию *обмен*, представляемую с помощью оператора обмена $\Leftrightarrow$ и позволяющую программировать обмен значениями между операндами типа π^- (переменными, индексами и элементами комплексов, не являющимися константами).

Например, при реализации операции $a \Leftrightarrow c_i$ происходит обмен значениями между переменной a и $[i]$-м элементом комплекса C.

При реализации операции типа $\Rightarrow \pi^-$ собственная переменная сохраняет свое значение, совпадающее в конце операции со значением операнда π^-, при реализации операции $\xi_1 \Leftrightarrow \xi_2$ τ принимает новое значение операнда ξ_2, наконец, при реализации операций $(\xi_1, \xi_2, \ldots, \xi_k) \Rightarrow \pi_0$ или $\pi_0 \Rightarrow (\xi_1, \xi_2, \ldots, \xi_k)$ τ принимает нулевое значение.

Следующие четыре операции могут быть выражены через предыдущие и вводятся исключительно с целью сокращения наиболее типичных выражений.

Операция *положительное элементарное приращение*, символически представляемая как $\Delta\pi^-$, служит сокращением выражения $\pi^- + 1 \Rightarrow \pi^-$, заменяя, таким образом, цепочку из трех операций: π^-, $+1$ и $\Rightarrow \pi^-$. Пользуясь символом эквивалентности $\sim$, дадим следующее формальное определение этой операции:

$$\Delta\pi^- \sim \pi^- + 1 \Rightarrow \pi^-.$$

Аналогично вводится операция *отрицательное элементарное приращение*, представляемая с помощью оператора $\overline{\Delta}$:

$$\overline{\Delta}\pi^- \sim \pi^- - 1 \Rightarrow \pi^-.$$

Введем также операцию *присвоение нулевого значения*:

$$\circ\pi^- \sim 0 \Rightarrow \pi^-$$

и операцию *присвоение максимального значения*:

$$\overline{\bigcirc}\,\pi^- \sim f_{10} \Rightarrow \pi^-.$$

Например, если исходное значение индекса a равно

00010011,

то при выполнении операции $\triangle a$ индекс a примет значение

00010100,

при выполнении операции $\overline{\triangle} a$ — значение

00010010,

при выполнении операции $\bigcirc a$ — значение

00000000

и при выполнении операции $\overline{\bigcirc} a$ — значение

11111111.

Примеры арифметических программ. Описанная система элементарных операций позволяет довольно просто составлять программы вычислений арифметического характера, если только фигурирующие в них числа на всех этапах вычислений будут оставаться натуральными и не будут выходить из диапазона от 0 до $2^{32} - 1$. Считая, что данное условие выполняется, рассмотрим следующее арифметическое выражение:

$$f = ((a + b)\, c - a + d)\, c$$

и составим программу для вычисления значения f по заданным значениям переменных a, b, c и d:

$$a + b \times c - a + d \times c \Rightarrow f.$$

В этой программе для представления переменных арифметического выражения мы выбрали индексы языка ЛЯПАС, обозначаемые для удобства теми же буквами. Программа расчленяется на семь элементарных операций, при последовательной реализации которых собственная переменная τ принимает ряд значений, представляющих промежуточные результаты вычислений.

Процесс реализации программы отображен на следующей таблице, каждая строка которой соответствует

одному элементарному такту вычислений. В первом столбце таблицы показаны элементарные операции, реализуемые в соответствующие такты, во втором — те выражения, вычисление которых заканчивается в этих тактах, в третьем — значения, получаемые собственной переменной τ, в четвертом — их натуральная интерпретация. При этом предполагается, что переменным правой части вычисляемого арифметического выражения приданы следующие значения: $a = 1$, $b = 5$, $c = 3$, $d = 4$.

a	a	00000001	1
$+b$	$a+b$	00000110	6
$\times c$	$(a+b)c$	00010010	18
$-a$	$(a+b)c-a$	00010001	17
$+d$	$(a+b)c-a+d$	00010101	21
$\times c$	$((a+b)c-a+d)c$	00111111	63
$\Rightarrow f$	$f=((a+b)c-a+d)c$	00111111	63

В данном случае при представлении промежуточных результатов оказалось возможным ограничиться использованием лишь собственной переменной τ. В других ситуациях может возникнуть необходимость привлечения дополнительных операндов, специально для представления промежуточной информации. Например, при программировании вычислений по арифметическому выражению

$$f = ((c - a + d)b + ac)c$$

с этой целью можно использовать индексы e и g, придав программе следующий вид:

$$c - a + d \times b \Rightarrow e\, a \times c \Rightarrow g\, e + g \times c \Rightarrow f.$$

Более экономной по числу используемых операндов является, однако, следующая программа:

$$c - a + d \times b \Rightarrow e\ \ a \times c + e \times c \Rightarrow f.$$

Примеры логических программ. Обратимся к программам логического характера, иллюстрируя их таблицами реализаций с двумя столбцами: в первом из них перечисляются последовательно выполняемые

элементарные операции, во втором — значения, принимаемые собственной переменной τ.

Следующая программа, исходя из значений индексов a и b, строит булев вектор, j-я компонента которого принимает значение 1, если она удовлетворяет условию $[a] \leqslant j \leqslant [b]$ и принимает значение 0 в противном случае. Полученный вектор представляется значением переменной m:

$$c_a < 1 - c_b \Rightarrow m.$$

Пусть, например, $[a] = 2$ и $[b] = 6$. Тогда таблица реализации данной программы примет следующий вид:

$$\begin{array}{rl} c_a & 00100000 \\ <1 & 01000000 \\ -c_b & 00111110 \\ \Rightarrow m & 00111110 \end{array}$$

Заметим, что в основе рассмотренной программы лежит эффективное использование операции вычитания:

$$\begin{array}{r} 01000000 \\ -\ 00000010 \\ \hline 00111110 \end{array}$$

Рассмотрим следующую задачу. Заданы значения переменных a, b и c. Требуется подсчитать, сколько из этих переменных имеет значение 1 в 0-й компоненте, сколько — в 1-й, сколько — во 2-й и т. д., и выразить результат значениями переменных m и n так, чтобы пара j-х компонент этих переменных представляла в позиционном двоичном коде число единиц в j-х компонентах переменных a, b и c, причем правые (младшие) разряды полученных кодов содержались в переменной n.

Согласно рассмотренным ранее правилам представления чисел в позиционной двоичной системе, данная задача может быть решена достаточно просто: каждое из представляемых здесь чисел может принимать одно из четырех возможных значений — 0, 1, 2 или 3. Если это число является нечетным, младший разряд кода принимает значение 1, в противном случае — значение 0. Если это число не меньше чем 2, то значение 1 принимается старшим разрядом кода, принимающим значение 0 при представлении чисел 0 или 1.

Следуя этим правилам, построим программу решения поставленной задачи, причем так, чтобы реализуемые по этой программе вычисления производились параллельно по всем компонентам рассматриваемых переменных:

$$a \oplus b \oplus c \Rightarrow n$$

$$a \wedge b \Rightarrow m\, a \wedge c \vee m \Rightarrow m\, b \wedge c \vee m \Rightarrow m.$$

В первой строке определяются значения всех младших разрядов кодов, приписываемые компонентам переменной n, во второй строке вычисляются значения старших разрядов — при этом используется то обстоятельство, что старший разряд должен принять значение 1, если по крайней мере две из переменных a, b и c обладают единичными значениями соответствующих компонент.

Пусть переменные a, b и c обладают значениями

01001100,
11010010,
01010101,

соответственно. В этом случае таблица реализации программы примет следующий вид (для удобства мы укрупним некоторые из операций):

$a \oplus b$	10011110
$\oplus c \Rightarrow n$	11001011
$a \wedge b \Rightarrow m$	01000000
$a \wedge c$	01000100
$\vee m \Rightarrow m$	01000100
$b \wedge c$	01010000
$\vee m \Rightarrow m$	01010100

Полученные значения переменных m и n представляют в компактном виде результаты проведенных вычислений, то есть числа единиц в каждой из компонент переменных a, b и c:

01010100
11001011

Представленное число: 13021211.

Рассмотрим теперь программу преобразования представления натурального числа в двоичной позиционной системе в представление в *коде Грея*, находящее ряд технических применений. В этом представлении начинающейся с нуля последовательности натуральных чисел соответствует последовательность булевых векторов, открываемая вектором, все компоненты которого имеют значение 0. Каждый же последующий вектор получается из предшествующего по следующему правилу: изменяется значение самой правой из тех компонент, смена значения одной из которых не приводит к получению вектора, совпадающего с каким-либо из предыдущих векторов.

Таким образом, соседним числам ставятся в соответствие векторы, отличающиеся значением только одной компоненты, что и обусловило целесообразность использования данного способа кодирования при решении некоторых технических задач. Диапазон представимых чисел ограничивается числом компонент вектора так же, как и в случае позиционного двоичного кода, то есть различные значения n компонентного булева вектора представляют 2^n начальных членов последовательности натуральных чисел:

Значение булева вектора	Представляемое число
00 ... 0000	0
00 ... 0001	1
00 ... 0011	2
00 ... 0010	3
00 ... 0110	4
00 ... 0111	5
00 ... 0101	6
00 ... 0100	7
. . . . .	. . .
10 ... 0010	2^n-4
10 ... 0011	2^n-3
10 ... 0001	2^n-2
10 ... 0000	2^n-1

Существует следующее простое правило преобразования двоичного позиционного кода в код Грея: значение каждой компоненты результативного вектора определяется как сумма по модулю 2 (дизъюнкция с исключением) соответствующей компоненты исходного вектора и соседней слева компоненты исходного же вектора. Именно это правило положено в основу следующей небольшой программы, в которой переменная a представляет некоторое натуральное число (в рассматриваемом примере — число 91) в двоичном коде, а переменная b принимает в ходе реализации программы значение, являющееся представлением заданного числа в коде Грея:

$$a > 1 \oplus a \Rightarrow b.$$

Таблица реализации данной программы:

a	01011011
>1	00101101
$\oplus a$	01110110
$\Rightarrow b$	01110110

Несколько более сложной является приводимая ниже программа, которую можно использовать при переборе всех значений булева вектора размерности n с заданным числом m единичных компонент. В ней отражен следующий алгоритм перебора: последующее значение получается из предшествующего путем замены крайней справа комбинации соседних компонент типа 10 на 01 и сдвига всех единиц, расположенных правее, вплотную к этой комбинации. Этот алгоритм оказывается достаточным для перебора всех искомых значений, если в начальном значении все единицы будут расположены в крайних слева позициях; в последнем из перебираемых значений все единицы займут крайние справа позиции, и уже нельзя будет найти комбинацию типа 10.

Например, при

$$n = 5 \text{ и } m = 3$$

значения булева вектора перебираются в следующей последовательности:

1 1 1 0 0
1 1 0 1 0
1 1 0 0 1
1 0 1 1 0
1 0 1 0 1
1 0 0 1 1
0 1 1 1 0
0 1 1 0 1
0 1 0 1 1
0 0 1 1 1

(в таблице выделены жирным шрифтом существенные для данного алгоритма комбинации типа 10).

Программа имеет следующий вид:

$$a+1 \wedge a \Rightarrow b - 1 \oplus a \bigtriangledown - 2 \Rightarrow c$$

$$a+1 \oplus a < 1+1 < c \oplus b \Rightarrow a$$

Указанный булев вектор представляется в ней переменной a, для запоминания промежуточной информации использованы переменная b и индекс c. В таблице реализации приводятся параллельно два примера, отличающиеся исходным значением переменной a. Проводимые вычисления можно разбить на четыре этапа, разделенные в таблице пунктиром. На первом из них из исходного значения булева вектора удаляется группа единиц, расположенных в крайних справа позициях (то есть таких единиц, правее которых нет нулей), и результат фиксируется значением переменной b. На втором этапе определяется величина сдвига, которому впоследствии надо будет подвергнуть данную группу; на третьем этапе эта группа выделяется в явном виде, причем к ней добавляются слева еще две единицы, используемые затем для преобразования комбинации 10 в 01 (с помощью оператора $\oplus$). Это преобразование, совмещенное со сдвигом влево упомянутой группы единиц (если таковая существует), выполняется на последнем, четвертом этапе.

Таблица реализации:

a	01100111	01011100
$+1$	01101000	01011101
$\wedge a$	01100000	01011100
$\Rightarrow b$	01100000	01011100
-1	01011111	01011011
$\oplus a$	00111000	00000111
∇	00000011	00000011
-2	00000001	00000001
$\Rightarrow c$	00000001	00000001
a	01100111	01011100
$+1$	01101000	01011101
$\oplus a$	00001111	00000001
<1	00011110	00000010
$+1$	00011111	00000011
$<c$	00111110	00000110
$\oplus b$	01011110	01011010
$\Rightarrow a$	01011110	01011010

§ 4. Циклические программы

При реализации линейной программы, как мы видели, отдельные операции выполняются в том же порядке, в котором они входят в выражение программы, вследствие чего реализация сводится к однократному последовательному выполнению всех операций. Этот порядок, называемый *прямым*, может нарушаться лишь в программах, характеризующихся наличием в них операций перехода, с которыми мы сейчас познакомимся. Эти операции позволяют строить так называемые *циклические программы*, допускающие многократное выполнение отдельных участков программы, образующих *циклы*.

Предложения. Программа разбивается на *предложения*, начало каждого из которых отмечается посредством метки, состоящей из символа § и символа натуральной константы — номера предложения. Начальному предложению программы присваивается всегда номер 0, нумерация остальных предложений может быть довольно

произвольной, хотя предпочтительно нумеровать их подряд, используя константы 0, 1, 2, 3 и т. д. Последнее из предложений замыкается меткой конца программы — символом «•». В каждом из предложений представляется некоторая последовательность операций, и каждая из этих операций, кроме той, которая стоит в самом начале предложения, может выполняться только непосредственно после выполнения операции, предшествующей ей в выражении программы.

В особом положении находятся операции, соответствующие началам предложений, то есть операции, символы которых следуют сразу же за метками § $\pi_ч$.

Операции перехода. К данному типу операций относится операция *безусловный переход*, обозначаемая через $\rightarrow \pi_ч$. Выполнение этой операции состоит в том, что непосредственно вслед за реализацией предшествующей (символам $\rightarrow \pi_ч$) части программы начинается реализация предложения с номером $[\pi_ч]$, то есть операций, следующих за символами § $\pi_ч$. Будем говорить, что в этом случае совершается переход в предложение номер $[\pi_ч]$.

Например, реализация программы

§ 0 $1 \Rightarrow a\, 2 \Rightarrow b$

§ 1 $a \Leftrightarrow b + a \Rightarrow b \rightarrow 1$

заключается в однократном выполнении предложения 0 и бесконечном числе реализаций предложения 1, представляющего циклическую часть программы. Заметим, что получаемые при этом индексом a значения образуют последовательность 1, 2, 3, 5, 8, 13, 21, ... (в десятичной системе), известную под названием ряда Фибоначчи: каждый последующий член ряда равен сумме двух предшествующих.

Циклы, реализуемые конечное число раз, организуются с помощью описываемых ниже операций условного перехода.

Операция *условный переход по нулю* представляется символически как $\circ\!\!\rightarrow \pi_ч$ и реализуется как операция $\rightarrow \pi_ч$, если выполняется одно из следующих условий:

А) операции $\circ\!\!\rightarrow \pi_ч$ непосредственно предшествует операция $+\pi$ или $-\pi$, при реализации которых оказывается, что $[\tau \pm \pi] \neq [\tau] \pm [\pi]$, где τ и π —, соответ-

ственно, левый и правый операнды, подвергающиеся действию оператора + или — (при операторе «+» данное условие может быть записано как $[\tau] + [\pi] \geqslant 2^n$, при операторе «—» оно конкретизируется в форму $[\tau] < [\pi]$),

Б) операции $\circ\!\!\rightarrow \pi_q$ не предшествуют ни операция $+\pi$, ни операция $-\pi$, и текущим значением собственной переменной τ является 000...0.

В остальных случаях прямой порядок выполнения операций нарушаться не будет.

Таким образом, операция $\circ\!\!\rightarrow \pi_q$ соответствует точке ветвления программы, реализуя переход к выполнению одного из участков программы: один из них следует непосредственно за выражением данной операции в программе, другой представляет начало предложения номер $[\pi_q]$.

Например, если мы желаем получить начальный отрезок ряда Фибоначчи, ограниченный сверху числом 100 (в восьмеричной системе счисления это число записывается как 144), то можно следующим образом скорректировать рассмотренную выше программу:

$$\S\ 0 \quad 1 \Rightarrow a\, 2 \Rightarrow b$$

$$\S\ 1 \quad a \Leftrightarrow b + a \Rightarrow b\, a - 144 \circ\!\!\rightarrow 1.$$

Очевидно, что как только значение индекса a достигнет или превысит заданный порог, произойдет «выход из цикла» и реализация программы прекратится.

В некоторых случаях удобно использовать выражение типа

$$a - b \circ\!\!\rightarrow 1 \circ\!\!\rightarrow 2,$$

выполняемое следующим образом: если $[a] < [b]$, то совершается переход в предложение 1, если же $[a] = [b]$, то после реализации операции $a - b$ собственная переменная τ примет значение 00...0, и это явится причиной перехода в предложение 2 в результате реализации операции $\circ\!\!\rightarrow 2$ (нужно учесть, что эта операция не следует непосредственно за операцией $a - b$, в связи с чем результат ее выполнения определяется условием Б)). Наконец, во всех остальных ситуациях прямой порядок выполнения операций нарушаться не будет.

Выражением типа

§ 0 $\overline{\ominus} a$

§ 1 $\Delta a \oplus b_2 \circ\!\!\rightarrow 2\, c_a \Rightarrow a \rightarrow 1$

§ 2

удобно пользоваться при организации перебора всех элементов некоторого комплекса. В данном случае перебираются элементы комплекса C, значения которых последовательно присваиваются переменной a, причем операция $\overline{\ominus}\, a$ применяется для того, чтобы первое же значение, приобретаемое индексом a в предложении 1, оказалось нулевым и, таким образом, выборка началась бы с начального элемента комплекса C, то есть с элемента c_0. Каждое из последовательно принимаемых затем индексом a значений 1, 2, 3, ... сравнивается с помощью оператора $\oplus$ с мощностью комплекса C, представляемой операндом b_2. Как только оказывается, что сравниваемые величины совпадают, собственная переменная τ принимает значение 00...0, происходит переход в предложение 2 и перебор элементов комплекса C на этом прекращается.

Не более сложной является организация перебора элементов комплекса в обратном порядке, начиная с последнего. Соответствующей иллюстрацией может служить следующее выражение:

§ 0 $b_2 \Rightarrow a$

§ 1 $\overline{\Delta}\, a\, c_a \Rightarrow a\, a \circ\!\!\rightarrow 2 \rightarrow 1$

§ 2

Условия выхода из цикла проверяются в этой программе операцией $a \circ\!\!\rightarrow 2$, реализующей выход, как только окажется, что индекс a имеет значение 00...0.

Операция *условный переход по единице* $\mapsto \pi_{\text{ч}}$ по своему действию противоположна операции $\circ\!\!\rightarrow \pi_{\text{ч}}$: условия перехода к реализации предложения номер $[\pi_{\text{ч}}]$ для операции $\circ\!\!\rightarrow \pi_{\text{ч}}$ совпадают с условиями перехода к реализации последующего участка программы для операции $\mapsto \pi_{\text{ч}}$.

Программа вычисления значения полинома. Рассмотрим в качестве примера алгоритм вычисления значения полинома

$$P(x) = c_p x^p + \ldots + c_2 x^2 + c_1 x + c_0,$$

предполагая, что значения величин x, c_0, c_1, ..., c_p представляются натуральными числами, а значение полинома не превосходит $2^n - 1$.

Пусть значения перечисленных исходных величин задаются индексом x и элементами c_0, c_1, ..., c_p, составляющими комплекс C. Тогда искомая величина — значение полинома $P(x)$ — будет получена при реализации следующей программы и будет представлена значением индекса b:

§ 0 $b_2 \Rightarrow a \circ b$

§ 1 $\overline{\Delta}\, a\, b \times x + c_a \Rightarrow b\ a \longmapsto 1.$

Данная программа отражает известную схему Горнера

$$P(x) = (\ldots((c_p x + c_{p-1})\, x + c_{p-2})\, x + \ldots + c_1)\, x + c_0,$$

сводящую вычисление полинома к многократному повторению операции

$$b \times x + c_a \Rightarrow b,$$

представляющей основное содержание циклической части программы (предложение 1). В предложении 0 отражена подготовка к вхождению в цикл — придается нулевое значение индексу b, в котором будет «накапливаться» результат вычислений, и устанавливается такое значение индекса a, которое позволит начать выборку элементов из комплекса C с последнего (напомним, что значением элемента b_2 специального комплекса B служит мощность комплекса C, то есть, в данном случае, число $p + 1$ коэффициентов полинома). В предложение 1, кроме уже рассмотренной операции $b \times x + c_a \Rightarrow b$, входит операция $\overline{\Delta}\, a$, обеспечивающая перебор всех элементов из комплекса C, и представлено условие выхода из цикла, реализуемое операцией $a \longmapsto 1$: если $[a] \neq 0$, то предложение 1 реализуется вновь, в противном случае совершается выход из цикла и выполнение программы завершается (поскольку достигается символ «.»).

Следует заметить, что рассмотренная программа имеет смысл только для того случая, когда число коэффициентов c_i отлично от нуля. В противном случае

индекс a получит в предложении 0 значение 00...0, но это значение не дойдет до операции $a \longmapsto 1$, так как при первой же реализации операции $\overline{\triangle}\, a$ оно будет заменено на значение 11...1, и это приведет к последствиям, которые могут оказаться неожиданными для программиста: предложение 1 будет реализовано 2^{32} раз (то есть свыше 4 000 000 000 раз!), пока индекс a не примет вновь значение 00...0 и не совершится, наконец, выход из цикла. Все это время будет производиться формальная выборка элементов комплекса $\boldsymbol{C}$, не имеющих смысла в данном случае.

Разумеется, подобные ситуации могут возникать лишь вследствие некоторых ошибок, допущенных при составлении программ или их использовании. К сожалению, ошибки такого рода являются довольно типичными, почему мы и обращаем особое внимание на их последствия.

Программа упорядочения элементов комплекса. Рассмотрим задачу упорядочения элементов комплекса по их весам, полагая, что вес булева вектора равен числу его единичных компонент. Эта задача решается следующей ниже программой, перегруппировывающей элементы комплекса $\boldsymbol{A}$ так, что в результате они оказываются расположенными в порядке убывания их весов.

Программа является выражением в языке ЛЯПАС следующего алгоритма: сначала в комплексе $\boldsymbol{A}$ отыскивается элемент с максимальным весом и обменивается своим значением с элементом $\boldsymbol{a_0}$, затем элемент $\boldsymbol{a_0}$ из рассмотрения исключается и производится поиск элемента с максимальным весом среди остающихся элементов комплекса $\boldsymbol{A}$, причем найденный элемент обменивается значением с элементом $\boldsymbol{a_1}$. Таким образом, определяются последовательно новые значения элементов $\boldsymbol{a_0}, \boldsymbol{a_1}, \boldsymbol{a_2}, \ldots$ ..., пока, наконец, они не будут найдены для всех элементов комплекса $\boldsymbol{A}$.

$\S\ 0 \quad \circ b$

$\S\ 1 \quad b \Rightarrow a \Rightarrow c\, \boldsymbol{a_c} \bigtriangledown \Rightarrow d$

$\S\ 2 \quad \triangle\, a \oplus b_0 \circ\!\!\rightarrow 3\, \boldsymbol{a_a} \bigtriangledown \Rightarrow e\, d - e \longmapsto 2\, a \Rightarrow c\, e \Rightarrow d \rightarrow 2$

$\S\ 3 \quad \boldsymbol{a_b} \Leftrightarrow \boldsymbol{a_e}\, \triangle\, b + 1 \oplus b_0 \longmapsto 1.$

В программе имеются два вложенных друг в друга цикла. Во внешнем цикле, состоящем из предложений 1, 2, 3, последовательно, начиная с a_0, перебираются элементы, значения которых определяются, здесь же производится необходимая подготовка к вхождению во внутренний цикл. Во внутреннем цикле (§ 2) ищется максимальный по весу элемент в заданной части комплекса A, не содержащей начальных элементов с уже установленными значениями. При этом индексом c фиксируется номер максимального по весу среди уже просмотренных элементов анализируемого остатка комплекса A, индексом d — вес этого элемента. Значения этих индексов обновляются при встрече элемента с бо́льшим весом. Перебор элементов комплекса A во внутреннем цикле осуществляется с помощью индекса a; пробегаемые им значения играют роль номеров анализируемых элементов. Номера элементов, значения которых определяются путем обмена с найденными в остатке максимальными по весу элементами, представляются индексом b. Отметим удобство применения здесь оператора $\Leftrightarrow$: организация обмена путем использования выражения типа

$$a_b \Rightarrow a\, a_c \Rightarrow a_b\, a \Rightarrow a_c$$

выглядела бы значительно более громоздкой. При проверке условий выхода из циклов используется, как и в предыдущей программе, один из элементов специального комплекса B, и в данном случае — элемент b_{0}, представляющий мощность комплекса A.

Заметим, что эта программа может быть использована лишь в том случае, когда $[b_0] \geqslant 2$, так как в противном случае произойдет ложное «зацикливание» программы. В самом деле, условия выхода из внутреннего цикла проверяются при выполнении операции $\triangle a \,\dot{\oplus}\, b_0 \circ\!\!\rightarrow 3$, входящей в предложение 2. Перебор анализируемых здесь значений индекса a начинается со значения $[a] = 1$, поэтому если $[b_0] = 0$, то окажется, что момент выхода из цикла уже упущен. Аналогичное явление может иметь место и при проверке условий выхода из внешнего цикла, совершаемой операцией $\triangle b \,\dot{+}\, 1 \,\dot{\oplus}\, b_0 \mapsto 1$, замыкающей выражение программы — здесь перебор значений индекса b начинается с $[b] = 2$,

откуда и следует, что программа будет правильно работать лишь при условии $[b_0] \geqslant 2$.

Реализация программы. Пусть, например, комплекс A имеет следующее исходное значение, которое можно рассматривать как булеву матрицу, то есть прямоугольную таблицу, элементами которой могут служить только нули и единицы:

$$\begin{bmatrix} 10101101 \\ 00010000 \\ 11111011 \\ 00100110 \\ 10001000 \end{bmatrix}$$

В этом случае реализация данной программы может быть отображена следующей последовательностью номеров предложений, называемой *трассой* реализации программы и показывающей, в каком порядке выполняются операции, представленные различными предложениями программы:

0—1—2—2—2—2—2—3—1—2—2—2—2—3—1—
—2—2—2—3—1—2—2—3.

Предложение номер 3 реализуется 4 раза. Обмен значениями в первый раз производится между элементом a_0 и элементом a_2, максимальным по весу среди всех элементов, во второй раз — между элементом a_1 и элементом a_2, максимальным среди остающихся, затем между a_2 и a_3 и, наконец, между a_3 и a_4, после чего комплекс A принимает окончательное значение:

$$\begin{bmatrix} 11111011 \\ 10101101 \\ 00100110 \\ 10001000 \\ 00010000 \end{bmatrix}$$

Разумеется, рассмотренная программа не является единственной или лучшей среди программ, решающих поставленную задачу, и приводится исключительно в качестве иллюстрации.

Операция ухода с возвращением. В ряде случаев удобной для использования при программировании оказывается операция *уход с возвращением*, пред-

ставляемая с помощью пары операторов ⇉ (оператор ухода) и ! (оператор возвращения). Выполняется эта операция следующим образом: при встрече выражения $\rightrightarrows \pi_ч$ совершается переход к реализации предложения номер $[\pi_ч]$, если же в процессе последующей реализации программы будет встречен оператор !, то осуществляется возвращение в «точку ухода» и начинается реализация той части программы, которая следует непосредственно за выражением $\rightrightarrows \pi_ч$.

Поскольку в одной и той же программе может встретиться несколько операций рассмотренного типа, могут возникнуть трудности при нахождении «точки ухода» в каждом конкретном случае. Чтобы избежать их, уточним смысл рассматриваемой операции, введя понятие *глубины реализации* программы и положив, что началу реализации программы соответствует нулевая глубина и что глубина может меняться лишь при действии операторов ⇉ и !, причем делается это следующим образом.

Операция $\rightrightarrows \pi_ч \alpha$, где α — некоторое произвольное выражение в языке ЛЯПАС, означает, что глубина реализации программы должна быть увеличена на единицу, «координаты» выражения α должны быть поставлены в соответствие новому значению глубины и запомнены и должен быть совершен переход к реализации предложения номер $[\pi_ч]$. Оператор ! осуществляет переход к реализации того выражения, «координаты» которого запомнены на текущей глубине, вслед за чем глубина реализации уменьшается на единицу.

Операция перебора единиц. Следующая операция носит комбинированный характер, совмещая операцию изменения значений некоторых операндов с операцией условного перехода.

Операция *перебор единиц*, представляемая выражением $\pi^- \dot{X} \pi_ч \pi_и$, эквивалентна по своему действию следующей последовательности других элементарных операций, уже знакомых нам:

$$\pi^- \circ\!\!\rightarrow \pi_ч \pi^- \vdash \Rightarrow \xi\, C\xi \oplus \pi^- \Rightarrow \pi^- \xi \Rightarrow \pi_и,$$

где через ξ обозначен особый операнд, фиксирующий промежуточный результат и играющий роль индекса, а через $C\xi$ обозначен $[\xi]$-й элемент специального

комплекса C. При реализации этой операции в значении операнда π^- удаляется левая единица, а ее координата сообщается индексу $\pi_и$ и собственной переменной τ. Если же исходным значением операнда π^- служит 00...0, то τ также принимает значение 00...0, $\pi_и$ сохраняет свое прежнее значение, и начинается реализация предложения номер $[\pi_ч]$.

Программа транспонирования булевой матрицы. Иллюстрируя операцию перебора единиц, рассмотрим программу транспонирования булевой матрицы, представленной значением комплекса $\boldsymbol{A}$. Напомним, что при транспонировании матрицы каждый ее элемент обменивается своим значением с симметричным ему элементом, причем для элемента, расположенного в i-й строке и в j-м столбце, симметричным будет тот элемент, который находится в j-й строке и в i-м столбце.

Рассматриваемая программа осуществляет последовательный перебор строк матрицы $\boldsymbol{A}$ с присвоением их значений переменной $\boldsymbol{a}$ (§ 1) и передает далее эти значения одноименным столбцам формируемой матрицы $\boldsymbol{B}$ путем последовательного нахождения единичных компонент в текущем значении переменной $\boldsymbol{a}$ и присвоения значения 1 соответствующим элементам очередного столбца матрицы $\boldsymbol{B}$ (§ 2).

§ 0 $\overline{\circ}\, b$

§ 1 $\triangle b \oplus b_0 \circ\!\!\rightarrow 3\boldsymbol{a}_b \Rightarrow \boldsymbol{a}$

§ 2 $\boldsymbol{a}\,\dot{\mathrm{X}}\,1\,a\,c_b \vee \boldsymbol{b}_a \Rightarrow \boldsymbol{b}_a \rightarrow 2$

§ 3 .

Результат применения этой программы окажется представленным комплексом $\boldsymbol{B}$, причем предполагается, что вначале все элементы комплекса $\boldsymbol{B}$ имеют нулевые значения. Предполагается также, что число строк транспонируемой матрицы не превышает 32 и что число ее столбцов заранее представлено мощностью комплекса $\boldsymbol{B}$. Обратим внимание на употребление операции $\overline{\circ}\, b$, позволяющей начать производимый в предложении 1 перебор значений индекса b со значения 00...0, несмотря на то, что в начале этого предложения стоит операция $\triangle b$, а также рассматривать случай $[b_0] = 0$, чего нельзя было делать в предыдущих программах.

Представление случайных процессов. С целью обеспечения возможности моделирования случайных процессов, а также с целью представления алгоритмов с элементами случайного выбора в язык ЛЯПАС вводится *случайная переменная я*. Считается, что эта переменная служит генератором случайных значений булева вектора стандартной размерности 32, равномерно распределенных по всей области из 2^{32} возможных значений. Это означает, что при каждом употреблении операнда *я* в составе какой-либо операции в процессе реализации программы каждая из компонент этой переменной, независимо друг от друга, принимает с равной вероятностью одно из двух значений: 0 или 1.

Вводится также операция *случайный перебор единиц*, записываемая в форме $\pi^{-}\ddot{X}\pi_{ч}\pi_{и}$ и отличающаяся от рассмотренной ранее операции $\pi^{-}\dot{X}\pi_{ч}\pi_{и}$ лишь тем, что очередной выбор единицы в коде значения операнда π^{-} производится случайно, с равными вероятностями выбора каждой единицы.

Например, эта операция с успехом используется в следующей программе случайного распределения натуральных чисел 1, 2, ..., k в качестве значений k элементов комплекса $D([b_3] = k \leqslant 32)$.

§ 0 $b_3 - 1 \Rightarrow a\,0 - c_a \Rightarrow a \circ a$

§ 1 $\triangle\, a\, a\ddot{X} 2\, b\, a \Rightarrow d_b \to 1$

§ 2 .

§ 5. Синтаксис языка

Любая Л-программа представляет собой некоторую цепочку символов языка ЛЯПАС, однако далеко не всякая цепочка этих символов может рассматриваться как программа, а только такая, которая удовлетворяет ряду синтаксических и семантических правил данного языка.

Синтаксис и семантика. Синтаксис языка задается совокупностью правил, разбивающих множество всех цепочек на два класса: синтаксически правильные цепочки и синтаксически неправильные цепочки, то есть такие, в которых встречаются некоторые недопустимые сочетания символов. Обычно синтаксические правила достаточно просты и могут быть строго сформулированы,

благодаря чему проверку их выполнения в конкретных цепочках можно поручить вычислительной машине. Значительно более сложной является задача формулировки и проверки семантических правил, выделяющих из множества синтаксически правильных цепочек «осмысленные» цепочки, которые только и могут служить программами. Дело в том, что синтаксические правила задают лишь ограничения на сочетания самих символов языка, в то время как семантические правила должны учитывать смысл выражений в рассматриваемом языке, определяемый, в частности, значениями операндов, соответствующих встречающимся в цепочках символам. Значения же ряда операндов вычисляются лишь в процессе реализации программы.

Сформулируем *синтаксис* языка ЛЯПАС.

α-цепочки. Назовем *α-цепочкой* произвольную конечную последовательность групп символов α:

$$\alpha \in \{\pi,\ \rho_1,\ \rho_2\pi,\ \rho_3\pi_ч,\ \rho_4\pi^-,\ \Rightarrow\pi^-,\ \pi_o\Rightarrow(\xi_1^-\xi_2^-\ \ldots\ \xi_k^-),$$
$$(\xi_1\xi_2\ \ldots\ \xi_k)\Rightarrow\pi_o,\ \pi^-\dot{X}\pi_ч\pi_и,\ \pi^-\ddot{X}\pi_ч\pi_и,\ \pi^-\Leftrightarrow\pi^-,\ !,\ \S\ \pi_ч\},$$

где, в свою очередь,

$$\rho_1 \in \{\vdash,\ \leftarrowtail,\ \urcorner,\ \triangledown\},$$
$$\rho_2 \in \{+,\ -,\ \vee,\ \wedge,\ \oplus,\ \times,\ \overline{\times},\ >,\ <,\ :\},$$
$$\rho_3 \in \{\rightarrow,\ \circ\!\!\rightarrow,\ \mapsto,\ \twoheadrightarrow\},$$
$$\rho_4 \in \{\bigcirc,\ \overline{\bigcirc},\ \triangle,\ \overline{\triangle}\},$$
$$\pi_п \in \{a, b, \ldots, я\},$$
$$\pi_и \in \{a, b, \ldots, я\},$$
$$\pi_ч \in \{0,\ 1,\ \ldots,\ 177\},$$
$$\pi_c \in \{c_0,\ c_1,\ \ldots,\ c_{37},\ d_0,\ d_1,\ \ldots,\ d_4,\ e_0,\ e_1,\ \ldots,\ e_4,\ f_0,\ f_1,\ \ldots,\ f_{10}\},$$
$$\pi_o \in \{\boldsymbol{A},\ \boldsymbol{B},\ \ldots,\ \boldsymbol{Я}\},$$
$$\pi_к \in \{\boldsymbol{A},\ \boldsymbol{B},\ \ldots,\ \boldsymbol{Я},\ A,\ B,\ \ldots,\ Q\},$$
$$\pi_к^- \in \{\boldsymbol{A},\ \boldsymbol{B},\ \ldots,\ \boldsymbol{Я},\ A,\ B,\ G,\ H,\ \ldots,\ Q\},$$
$$\pi_{пи} \in \{\pi_п,\ \pi_и\},$$
$$\pi_{ич} \in \{\pi_и,\ \pi_ч\},$$

$$\pi_э \in \{\pi_п, \pi_и, \pi_ч, \pi_с\},$$
$$\pi^- \in \{\pi_п, \pi_и, \pi^-_{эк}\},$$
$$\pi \in \{\pi_э, \pi_{эк}\},$$
$$\xi_l \in \{\pi_э, \pi_о, \pi_{эск}\},$$
$$\xi_l^- \in \{\pi_п, \pi_и, \pi_о, \pi^-_{эск}\}$$

и где через $\pi_{эк}$ обозначен произвольный элемент комплекса $\pi_к$, через $\pi^-_{эк}$ — элемент комплекса $\pi^-_к$, через $\pi_{эск}$ — элемент специального комплекса и через $\pi^-_{эск}$ — элемент, который может принадлежать любому специальному комплексу, за исключением комплексов *C*, *D*, *E* и *F*.

Например, α-цепочкой является следующая последовательность символов:

$$a + b \vdash \Rightarrow m\, d\, a \wedge c_m \triangle a \Rightarrow a,$$

поскольку она разбивается на α-группы указанных ниже типов

$$\pi,\ \rho_2\pi,\ \rho_1,\ \Rightarrow\pi^-,\ \pi,\ \pi,\ \rho_2\pi,\ \rho_4\pi^-,\ \Rightarrow\pi^-;$$

выражения же

$$c \wedge b \Rightarrow c_1,\quad a + d\,\dot{X}\,5a,\quad m + - a \Rightarrow b$$

не являются α-цепочками, так как в них встречаются недопустимые сочетания

$$\Rightarrow c_1,\quad + d\,\dot{X},\quad + -.$$

Символы $\S\,\pi_ч$ разбивают α-цепочку на *предложения*, номерами которых служат значения символа $\pi_ч$ в стоящей в начале каждого предложения метке $\S\,\pi_ч$.

Синтаксически правильные α-цепочки. *Синтаксически правильной* назовем такую α-цепочку, которая удовлетворяет следующим условиям.

Во-первых, α-цепочка должна замыкаться символом «.», играющим двойную роль: с одной стороны, он служит признаком конца программы (реализация программы прекращается при встрече этого символа), с другой стороны, им всегда замыкается символическое выражение программы.

Во-вторых, номера предложений в данной α-цепочке не должны повторяться и должны образовывать такое множество N, чтобы для всех встречающихся в α-цепочке сочетаний $\rho_3 п_ч$, $\dot{Х} п_ч$ и $\ddot{Х} п_ч$ выполнялось условие $[п_ч] \in N$. Начальное предложение должно иметь номер 0, то есть α-цепочка должна начинаться символами $\S 0$.

Третье ограничение удобно сформулировать, воспользовавшись понятием *ориентированного графа*, как некоторой совокупности точек (*вершин* графа), соединенных друг с другом при помощи некоторой системы стрелок (*дуг* графа). Построим ориентированный граф, вершины которого будут соответствовать предложениям программы, а дуги — возможным переходам по операторам ρ_3, $\dot{Х}$ и $\ddot{Х}$ и переходам по непосредственному следованию предложений (в случае, если предшествующее предложение не оканчивается символом $\rightarrow п_ч$ или !).

Вообразим теперь некоторую точку, которая может путешествовать по графу, двигаясь по его дугам в направлении, указанном стрелкой, и которую можно поместить вначале в произвольную вершину графа. Любая совокупность дуг, по которым может последовательно пройти воображаемая точка, называется *путем* в графе, а такой путь, пройдя который, воображаемая точка возвратится в исходную вершину, называется *контуром*.

Отметим дуги, соответствующие переходам по оператору $\dashv\!\!\rightarrow$, и вершины, соответствующие предложениям, оканчивающимся символом !. В построенном таким образом графе не должно существовать контуров, содержащих отмеченные дуги. В графе, полученном из первого путем выбрасывания отмеченных дуг, не должно существовать путей, идущих от вершины, соответствующей начальному предложению (а эта вершина должна быть не отмеченной), к какой-либо из отмеченных вершин.

Например, изображенный на рис. 1.5.1 граф, в котором отмеченные вершины представлены светлыми кружками, а отмеченные дуги — пунктирными линиями, удовлетворяет сформулированным условиям. Граф же, показанный на рис. 1.5.2, данным условиям не удовлетворяет, поскольку в нем существует контур, проходящий через вершины *2*, *4* и *6* и содержащий отмеченную дугу *a*.

Кроме того, в этом графе существует путь, образованный неотмеченными дугами и ведущий из вершины *0* в отмеченную вершину *5*.

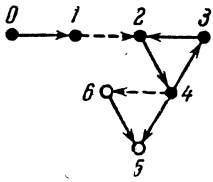


Рис. 1.5.1.

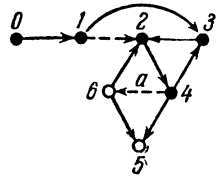


Рис. 1.5.2.

В заключение сформулируем следующее правило: *любая Л-программа представляется синтаксически правильной α-цепочкой.*

§ 6. Абстрактная модель вычислительной машины

Язык и машина. Записанные в языке ЛЯПАС программы могут быть реализованы практически на любой современной универсальной вычислительной машине. Каждая из этих машин является довольно сложным электронным устройством, однако, чтобы пользоваться ею, достаточно знать лишь основные характеристики машины. Применение языка ЛЯПАС сводит эту необходимую информацию к достаточно малому объему.

В данном параграфе будет рассмотрена упрощенная абстрактная модель вычислительной машины, реализующей Л-программы. Ознакомление с этой моделью полезно для лучшего усвоения языка ЛЯПАС. В самом деле, ЛЯПАС является языком императивного типа — выражения в этом языке (Л-программы) содержат информацию о том, что должна делать машина. Поэтому семантика языка — смысл выражений в этом языке — может быть выражена в терминах реакций машины. Рассматриваемая модель достаточно хорошо согласуется с основными принципами устройства и функционирования вычислительных машин, благодаря чему знакомство с нею окажется достаточным для подготовки к реализации программ на реальных вычислительных машинах.

Полное согласование данной модели с конкретными вычислительными машинами производится системой автоматического программирования, важными элементами которой являются трансляторы, переводящие программы, выраженные на языке ЛЯПАС, на собственные, машинные языки реальных машин. Сами трансляторы представляют собой программы, выраженные в собственном языке заданной машины, то есть в том языке, на который осуществляется перевод. Можно считать, что трансляторы имитируют рассматриваемую в данном параграфе абстрактную модель на реальных вычислительных машинах, то есть заставляют их вести себя подобно данной модели. Процесс этой имитации полностью автоматизирован, поэтому, пользуясь языком ЛЯПАС, мы избавлены от необходимости знакомства с особенностями машин, на которых реализуются составленные нами программы. Например, отпадает надобность в знании набора элементарных операций используемых вычислительных машин. Вычислительная машина как бы достраивается: к ее входу присоединяется транслятор, и входом полученной системы в целом становится входной язык транслятора, в данном случае — ЛЯПАС.

Структура машины. Переходим к рассмотрению модели, называя ее в дальнейшем просто *машиной.*

Двумя основными блоками машины являются память и *оперативное устройство,* реализующее программу, извлекаемую им из памяти. Кроме того, в состав машины входят устройство ввода информации и устройство вывода информации.

Память предназначена для фиксирования (запоминания) исходных данных к решаемой на машине задаче, промежуточной информации и результатов вычислений, а также для хранения реализуемой программы и различного рода вспомогательной информации. Она делится на две части — *оперативную память и дополнительную память.* Такое подразделение обусловлено исключительно техническими особенностями реальных вычислительных машин, память которых делится аналогичным образом, причем основной объем памяти приходится на дополнительную память, относительно дешевую, но медленно действующую, оперативная же память является

значительно более быстродействующей, но в то же время и более дорогой, вследствие чего ее объем разумно ограничивается.

Учитывая указанное различие характеристик двух типов памяти, программы следует составлять таким образом, чтобы их реализация происходила в основном в пределах оперативной памяти. Лишь в тех случаях, когда объем оперативной памяти оказывается недостаточным, следует прибегать к использованию дополнительной памяти, включая в составляемую программу операции обмена информацией между двумя типами памяти. Поскольку львиная доля времени, затрачиваемого при работе с дополнительной памятью, приходится на поиск требуемой порции информации, а транспортировка уже найденной порции производится относительно быстро, целесообразно по возможности уменьшить число обращений к дополнительной памяти при реализации программы за счет увеличения объема передаваемых порций информации.

Оперативная память представляет собой совокупность *ячеек*, пронумерованных числами 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, ..., $N-1$, представленными в восьмеричной системе счисления. Объем оперативной памяти N пока фиксировать не будем, считая величину N параметром, принимающим конкретное значение при согласовании данной модели с какой-либо реальной вычислительной машиной. Пока достаточно знать, что N имеет порядок нескольких тысяч.

Каждая из ячеек, в свою очередь, состоит из 32 двоичных *элементов*, пронумерованных числами 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, ..., 36, 37. Каждый из этих элементов может находиться в одном из двух возможных состояний, благодаря чему в одной ячейке памяти может быть представлено значение одного булева вектора стандартной размерности. Смена значения булева вектора сводится, как нетрудно видеть, к соответствующему изменению состояний двоичных элементов ячейки, в которой представляется значение данного вектора.

Оперативный комплекс. В § 2 говорилось об определенном типе операндов языка ЛЯПАС — о специальных комплексах. В частности, эти комплексы оказываются полезными при установлении связи между

операндами ЛЯПАСа и памятью машины. Делается это следующим образом.

Определим *оперативный комплекс* K как такой комплекс, значение i-го элемента которого представляется в i-й ячейке оперативной памяти машины. При таком определении задача размещения информации в оперативной памяти становится задачей установления определенного соответствия между комплексом K и другими операндами языка ЛЯПАС, представляющими перерабатываемую информацию.

Комплекс A примем за *комплекс начал* всех комплексов $\boldsymbol{A}$, $\boldsymbol{B}$, ..., $\boldsymbol{Я}$, A, B, ..., Q в оперативном комплексе K, то есть положим, что элемент a_0 будет представлять номер того элемента оперативного комплекса, который отождествляется с элементом $\boldsymbol{a}_0$, элемент a_1 ставится в аналогичное соответствие с элементом $\boldsymbol{b}_0$ и т. д. Установим принцип непрерывности в представлении комплексов, что означает следующее: если i-й элемент какого-либо комплекса отождествлен с j-м элементом оперативного комплекса, то его $i+1$-й элемент (если он существует) отождествляется с $j+1$-м элементом оперативного комплекса.

В § 2 уже упоминался специальный комплекс B, содержащий информацию о мощностях комплексов:

$$[b_0] = \sigma(\boldsymbol{A}), \quad [b_1] = \sigma(\boldsymbol{B}), \quad \ldots, \quad [b_{37}] = \sigma(\boldsymbol{Я}),$$
$$[b_{40}] = \sigma(A), \quad \ldots, \quad [b_{57}] = \sigma(Q).$$

Назовем этот комплекс *комплексом мощностей*.

Включая в программу операции, которые изменяют значения элементов комплексов A и B, можно программировать рост и сокращение разнообразных комплексов, а также их перемещение в пределах оперативного комплекса K. Однако это не разрешается делать для специальных комплексов A, B, ..., Q, так как, в отличие от основных комплексов $\boldsymbol{A}$, $\boldsymbol{B}$, ..., $\boldsymbol{Я}$, мощности и местоположение специальных комплексов фиксируются и программно изменяться не должны.

Для облегчения программирования операций с группами переменных и индексов введем *комплекс переменных* G, положив $g_0 = \boldsymbol{a}$, $g_1 = \boldsymbol{b}$, ..., $g_{37} = \boldsymbol{я}$, и *комплекс индексов* H, положив $h_0 = a$, $h_1 = b$, ... $h_{37} = я$.

К экономии операндов. У читателя может возникнуть законный вопрос: а что делать, если при составлении некоторой программы перечисленный набор операндов окажется недостаточным (то есть не хватит переменных, индексов или комплексов)?

Практика программирования показывает, что такие случаи довольно редки. Если все же соответствующая критическая ситуация возникает, то ее можно разрешить путем перехода от использования одиночных булевых векторов (переменных и индексов) к использованию булевых матриц (комплексов) и от комплексов — к совокупностям комплексов.

Выше уже рассматривалось обобщение операции присвоения, позволяющее «упаковывать» текущие значения отдельных операндов в некоторый один комплекс, где они могут храниться до поры до времени, пока эти значения не потребуются. Тогда производится «распаковка» данного комплекса: обратная передача значений его элементов отдельным операндам.

Пользуясь этой операцией в тех случаях, когда в число упомянутых отдельных операндов входят некоторые комплексы, следует запомнить, что при реализации, например, операции $\boldsymbol{A} \Rightarrow (\boldsymbol{B})$ число выбираемых с конца из комплекса $\boldsymbol{A}$ элементов будет равно мощности комплекса $\boldsymbol{B}$, заданной значением элемента b_1. Поэтому, если нужно запомнить на некоторое время значение комплекса $\boldsymbol{B}$, зная, что к моменту, когда это значение потребуется, мощность данного комплекса не изменится, можно пользоваться выражениями типа $(\boldsymbol{B}) \Rightarrow \boldsymbol{A}$ и $\boldsymbol{A} \Rightarrow (\boldsymbol{B})$. Если же такой гарантии нет, то разумно вместе со значением комплекса запоминать и его мощность (а в иных случаях — и местоположение, задаваемое значением соответствующего элемента специального комплекса A) и пользоваться выражениями типа $(\boldsymbol{B}b_1) \Rightarrow \boldsymbol{A}$ и $\boldsymbol{A} \Rightarrow (\boldsymbol{B}b_1)$.

В некоторых случаях обрабатываемую информацию целесообразно представлять в форме *секционированных* комплексов, как мы будем называть комплексы, разбивающиеся на *подкомплексы*, подобно тому, как оперативный комплекс K разбивается на комплексы $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, ... Разбиение секционированного комплекса на подкомплексы удобно задавать с помощью другого комплекса,

элементы которого отмечают, например, номера тех элементов секционированного комплекса, которые являются первыми в соответствующих секциях.

Например, если мощность секционированного комплекса A равна семи ($b_0 = 7$) и если разбиение этого комплекса задается описанным образом комплексом D, содержащим три элемента:

$$[d_0] = 0,\ [d_1] = 4,\ [d_2] = 5,$$

то это значит, что комплекс A разбивается на три подкомплекса, первый из которых образуется элементами a_0, a_1, a_2, a_3 комплекса A, второй — элементом a_4, третий — элементами a_5, a_6.

Разбиение секционированного комплекса можно задавать и по-иному, перечисляя, например, номера последних элементов или мощности подкомплексов. Особое преимущество секционированных комплексов перед обычными заключается в том, что число секций (подкомплексов) легко изменять программным путем, то есть уже в процессе реализации программы.

Реализация программы. Реализуемая машиной Л-программа задается в виде последовательности символов, закодированных определенным образом (подробное изложение способа кодирования будет дано ниже), и располагается в памяти машины, по возможности — в оперативной памяти. Реализация программы обеспечивается оперативным устройством, последовательно считывающим и выполняющим отдельные операции в том порядке, который соответствует вышеописанным правилам следования операций. Выполнение большинства операций сводится к определенному изменению значений некоторых операндов, то есть к изменению состояний элементов, образующих ячейки памяти, соответствующие данным операндам.

Кроме описанной, пронумерованной памяти, машина обладает еще одной ячейкой памяти стандартного размера (в 32 элемента), предназначенной для представления значения собственной переменной τ. Наличие такой ячейки приближает рассматриваемую модель к реальным машинам одноадресного типа, чем и объясняется близость принципов программирования для этих машин

к тем принципам, которые были положены в основу языка ЛЯПАС. Естественно, что данная ячейка является наиболее динамичным элементом памяти, поскольку ее состояние изменяется при реализации почти каждой операции.

Для данной модели все рассмотренные выше операции языка ЛЯПАС являются *элементарными*, то есть выполняемыми за один такт, без каких-либо промежуточных изменений состояний элементов памяти. Реализация каждой из элементарных операций производится исключительно оперативным устройством, структура которого приспособлена именно к заданному набору элементарных операций, без промежуточного обмена информацией с памятью машины. В крайнем случае, если реализуемый оперативным устройством алгоритм выполнения одной элементарной операции требует фиксации некоторой промежуточной информации, то необходимые для этого элементы памяти органически включаются в состав оперативного устройства и не считаются принадлежащими описанной выше памяти машины.

При использовании реальной вычислительной машины операции Л-программы перестают быть элементарными, в связи с чем реализации программы предшествует ее перевод на язык реальной машины, при котором каждая операция Л-программы заменяется некоторой цепочкой элементарных операций данной реальной машины. В этом случае может возникнуть потребность в дополнительной памяти, нужной для передачи информации между этими операциями. Такая потребность может быть удовлетворена путем отщепления некоторой части оперативной памяти используемой машины и ее специализации в указанном направлении. Перевод Л-программы на язык реальной машины осуществляется, как уже говорилось, специальной программой — транслятором, и является тем более простым, чем ближе набор элементарных операций реальной машины к набору операций языка ЛЯПАС.

Внешний комплекс. Кроме оперативного комплекса, соответствующего оперативной памяти машины, введем в рассмотрение *внешний комплекс*, соответствующий дополнительной памяти машины. Обмен информа-

цией между этими двумя комплексами производится с помощью следующих двух операций.

Операция *посылка во внешний комплекс* записывается как *зап* $\xi_1\xi_2\xi_3$, где ξ_1, ξ_2, $\xi_3 \in \{\pi\}$, и означает передачу значений $[\xi_2]$ рядом расположенных элементов оперативного комплекса, начинающихся с элемента номер $[\xi_1]$, соответственно $[\xi_2]$ рядом расположенным элементам внешнего комплекса, начинающимся с элемента номер $[\xi_3]$*).

Аналогично определяется операция *посылка из внешнего комплекса*, записываемая как *счит* $\xi_1\xi_2\xi_3$ и означающая передачу информации в обратном направлении: $[\xi_2]$ элементов внешнего комплекса, начиная с элемента номер $[\xi_3]$, передают свои значения соответствующей серии элементов оперативного комплекса, начинающейся с элемента номер $[\xi_1]$.

Средства ввода и вывода информации. Переходя к рассмотрению средств связи машины с «внешним миром», примем две формы представления информации вне машины: *печатную форму* и *перфоформу*. Первая из них будет использоваться при выводе информации из машины, вторая — как при выводе, так и при вводе.

Учитывая особенности выходных устройств современных вычислительных машин, примем, что значения булевых векторов при их представлении в печатной форме будут, как правило, переводиться в восьмеричную систему счисления (подразумевается численная интерпретация этих значений). Для такого представления достаточно 11 восьмеричных разрядов, каждый из которых соответствует трем двоичным компонентам булева вектора (за исключением самого левого разряда, на долю которого приходится лишь две двоичные компоненты, в связи с чем он может принимать значения только из множества {0, 1, 2, 3}).

Например, если некоторый булев вектор обладает значением

0100 1100 1010 0000 0001 1110 1111 0010,

*) В § 11 производится уточнение смысла значений операнда ξ_3, учитывающее особенности организации дополнительной памяти современных вычислительных машин.

то при переводе в восьмеричную систему его компоненты разбиваются на следующие группы:

01 001 100 101 000 000 001 111 011 110 010,

каждая из которых заменяется одной восьмеричной цифрой: 11450017362.

При представлении булева вектора в перфоформе ему в соответствие ставится группа из 32 позиций на некотором конкретном носителе информации — перфокарте или перфоленте. Каждая из этих позиций отвечает соответствующей компоненте булева вектора, и в том и только в том случае, когда эта компонента принимает значение 1, данная позиция перфорируется — в ней делается пробивка в виде небольшого отверстия. Представленная в такой форме информация легко может быть считана устройством ввода информации.

Подлежащая обработке на машине информация подготавливается в виде последовательности булевых векторов с фиксированными значениями, называемой *входной последовательностью*. В соответствии с планом ввода информации в машину эта последовательность заранее разбивается на порции и вводится (*устройством ввода информации*) в оперативную память в порядке реализации операции *ввод*, представляемой символически в виде оператора *ввод* и следующей за ним совокупности символов операндов, принимающих значения вводимых булевых векторов. Такими операндами могут быть переменные, индексы, комплексы и отдельные элементы специальных комплексов, если только номера этих элементов заданы натуральными константами (таким образом, налагается запрет на ввод отдельных элементов основных комплексов и на ввод отдельных элементов специальных комплексов в тех случаях, когда номера этих элементов задаются посредством индексов). Указанный перечень символов операндов должен замыкаться символом ;.

Ввод значений операндов производится последовательно, в порядке перечисления операндов, причем каждое из этих значений задается отдельной порцией. Разумеется, размеры этих порций должны удовлетворять определенным условиям — например, порция, задающая новое значение некоторого индекса, должна состоять

только из одного булева вектора, если же вводимая порция представляет значение, присваиваемое некоторому комплексу, то ее размер должен определяться мощностью данного комплекса.

Обратим внимание на необходимость выполнения еще одного существенного условия: при вводе значения некоторого комплекса его параметры (определяющие местоположение комплекса и его мощность) должны быть заданы текущими значениями соответствующих элементов специальных комплексов A и B. В частности, значения этих элементов могут быть введены при реализации той же операции ввода, по которой производится ввод самого комплекса; требуется лишь, чтобы в данном случае символ комплекса следовал в перечне после символов соответствующих этому комплексу элементов специальных комплексов A и B.

Например, при реализации операции *ввод* $\boldsymbol{a}\,c\,i\,a_2\,\boldsymbol{C}\,\boldsymbol{A}$; последовательно вводятся значения операндов $\boldsymbol{a}$, c, i, a_2, $\boldsymbol{C}$ и $\boldsymbol{A}$, причем к началу выполнения этой операции значениями элементов a_0, b_0 и b_2 уже должны быть заданы координаты начала комплекса $\boldsymbol{A}$ и его мощность, а также мощность комплекса $\boldsymbol{C}$. Значение же, представляющее координату начала комплекса $\boldsymbol{C}$, принимается элементом a_2 уже в процессе реализации данной операции, перед тем, как начинает вводиться значение самого комплекса $\boldsymbol{C}$.

На *устройстве вывода информации* реализуются следующие две операции.

Операция *печать*, записываемая в виде *печ*, означает выдачу на печать (в восьмеричной системе счисления) текущего значения собственной переменной τ. Если же оператор *печ* следует непосредственно за символом комплекса (например, в случае $\boldsymbol{A}$ *печ*), на печать выдаются последовательно значения всех элементов указанного комплекса. В начале выдаваемой последовательности приводится номер данного комплекса, а также значения соответствующих элементов комплексов A и B.

Операция *перфорация*, представляемая символически как *перф*, означает выдачу текущего значения собственной переменной τ на перфорацию, то есть в перфоформе. Эта операция обладает известными удобствами, например, в том случае, когда выдаваемая при реализа-

ции алгоритма информация предназначена для последующего ввода в машину. По аналогии с вышеописанной операцией печати, в том случае, когда символ *перф* следует непосредственно за символом комплекса, на перфорацию выдаются последовательно значения всех элементов этого комплекса.

§ 7. Повышение размерности операндов

Стандартная размерность булевых векторов, представляемых рассмотренными выше операндами ЛЯПАСа, при решении ряда логических задач оказывается явно недостаточной. Фигурирующие в этих задачах векторы большей размерности можно, конечно, представлять группами векторов стандартной размерности и составлять особые программы, оперирующие с этими группами, как с самостоятельными векторами. Однако составление таких программ вручную довольно трудоемко. В то же время желательно предусмотреть возможность непосредственного использования программ, подготовленных для оперирования с операндами стандартной размерности, и в том случае, когда размерность операндов превышает стандартную.

Параметры размерности. Поставленная задача решается путем расширения языка ЛЯПАС на основе введения специальных меток повышения размерности. Работа по соответствующему расширению Л-программ (программное объединение необходимого количества операндов стандартной размерности для представления каждого из операндов повышенной размерности) возлагается на систему автоматического программирования.

Введем четыре независимых *параметра размерности* ω_1, ω_2, ω_3 и ω_4, значениями которых, задаваемых индексами *ж*, *л*, *ф* и *ш*, соответственно, могут служить натуральные числа. Эти параметры будут определять размерности переменных и элементов комплексов, отмеченных спереди символами $'$, $''$, $'''$ и $''''$, соответственно. Будем всегда считать, что размерность элементов, принадлежащих одному и тому же комплексу, должна совпадать. Повышение размерности индексов не предусматривается, поскольку диапазон представляемых ими

чисел и так достаточно велик. Допустим возможность изменения значений индексов *ж*, *л*, *ф* и *ш* в разумных пределах при реализации программы.

Выполнение операций. Операции над операндами с повышенной размерностью определим как обобщение уже рассмотренных операций над операндами стандартной размерности, с заменой значения $n=32$ на $n = 32 \cdot \omega$, где $\omega \in \{\omega_1, \omega_2, \omega_3, \omega_4\}$ будем называть *коэффициентом размерности.*

В число операторов, на которые распространяется данное обобщение, включим одноместные операторы

$$\neg,\ \nabla,\ \dashv,\ \vdash,\ \Delta,\ \overline{\Delta},\ \circ,\ \bar{\circ}$$

и двуместные операторы

$$+,\ -,\ \vee,\ \wedge,\ \oplus,\ <,\ >,$$

и также операторы $\Rightarrow$ и $\Leftrightarrow$.

Условимся, что правым операндом операторов $<$ и $>$ может быть лишь операнд стандартной размерности ($n = 32$).

Положим также, что переменная повышенной размерности может служить левым операндом операторов $\dot{X}$ и $\ddot{X}$.

Стремясь не усложнять без особой в том нужды систему автоматического программирования, условимся, что действие операторов $\times$, $\overline{\times}$ и : на операнды повышенной размерности не распространяется.

Уточним правила выполнения операций над операндами повышенной размерности, используя следующую символику:

$$\rho_1 \in \{\vdash, \dashv, \neg, \nabla\},$$
$$\rho_4 \in \{\circ, \bar{\circ}, \Delta, \overline{\Delta}\},$$
$$\rho_2^- \in \{+, -, \vee, \wedge, \oplus\},$$
$$\pi^+ \in \{\pi_{\text{п}}, {}^{\nu}\pi_{\text{п}}, \pi_{\text{эок}}, {}^{\nu}\pi_{\text{эок}}\},$$

где

$${}^{\nu} \in \{', '', ''', ''''\},$$
$$\pi_{\text{п}} \in \{\boldsymbol{a}, \boldsymbol{b}, \ldots, \boldsymbol{я}\}$$

и где через $\pi_{\text{эок}}$ обозначен произвольный элемент основного комплекса.

При выполнении операции π^+ собственная переменная τ принимает размерность и значение операнда π^+. То же происходит и при реализации операции $\rho_4\pi^+$.

Одноместные операции типа ρ_1 в том случае, когда переменная τ обладает повышенной размерностью, выполняются совершенно аналогично случаю стандартной размерности. При этом размерность переменной τ сохраняется, за исключением случая реализации операции ∇, когда она принимает стандартное значение.

Реализация двуместных операций $\rho_2^-\pi^+$ также не вызывает затруднений, если размерность операнда π^+ совпадает с размерностью собственной переменной τ. Однако случай, когда эти размерности различны, заслуживает более подробного рассмотрения. Перед выполнением операции в этом случае размерности операндов выравниваются путем добавления справа к значению операнда меньшей размерности соответствующего количества нулей. В результате реализации операции размерность переменной τ принимает максимальное из значений размерностей данных двух операндов.

Несколько иначе выполняется операция $\rho_2^-\pi_{\text{ич}}$: операнд $\pi_{\text{ич}}$ (индекс или натуральная константа) взаимодействует с правыми компонентами собственной переменной τ, то есть дополняется нулями слева, если τ обладает повышенной размерностью.

Пусть, например, $[\text{ж}] = 3$. Тогда операнд $'\boldsymbol{a}$ будет обладать размерностью $32 \cdot 3 = 96$ и его можно представить в виде некоторой тройки $\alpha\beta\gamma$ булевых векторов стандартной размерности, обозначаемых здесь через α, β и γ, соответственно. При реализации операции $'\boldsymbol{a} \vee \boldsymbol{b}$ в этом случае с переменной $\boldsymbol{b}$ будет взаимодействовать (по правилам, задаваемым оператором $\vee$) лишь вектор α, и собственная переменная τ примет значение $\tilde{\alpha}\beta\gamma$, где через $\tilde{\alpha}$ обозначен результат этого взаимодействия.

Если же при аналогичных обстоятельствах выполняется операция $'a + b$, то индекс b будет непосредственно взаимодействовать лишь с вектором γ и τ примет значение $\tilde{\alpha}\tilde{\beta}\tilde{\gamma}$, в котором результат этого взаимодействия будет представлен вектором $\tilde{\gamma}$, а векторы $\tilde{\alpha}$ и $\tilde{\beta}$ будут отличаться от векторов α и β, соответственно,

лишь в случае возникновения межразрядных переносов в местах сочленения векторов. Следует, однако, учитывать, что операция $b + 'a$ будет выполняться иначе, поскольку в этом случае сначала значение индекса b будет передано собственной переменной τ, а уж затем произойдет выравнивание порядков, на этот раз — по стандартным правилам (значение переменной τ будет дополнено нулями справа). Поэтому при выполнении данной операции будет получена триада $\tilde{\alpha}\beta\gamma$, где вектор $\tilde{\alpha}$ — результат взаимодействия индекса b с вектором α.

Если правым операндом оператора ρ_2^- служит константа из комплексов C, D или E, то она должна экстраполироваться до аналогичной константы, размерность которой будет совпадать с размерностью собственной переменной τ. Например, если переменная τ обладает размерностью q и если значение индекса i удовлетворяет условию $[i] < q$, то c_i является вектором-константой с единственной единичной компонентой, а именно, $[i]$-й.

Таким образом, размерность собственной переменной τ определяется размерностью обрабатываемых при реализации Л-программы операндов. Однако, опять-таки с целью упрощения программирующей системы, положим, что через символ смены предложений § может переноситься лишь значение τ стандартной размерности.

При реализации операции присвоения $\Rightarrow \pi^-$ как собственная переменная τ, так и операнд π^- сохраняют свои размерности. При этом, в зависимости от того, размерность какого из этих операндов больше, происходит или отбрасывание правых компонент значения τ при его передаче операнду π^-, или присвоение нулевых значений правым компонентам операнда π^- (если размерность этого операнда превышает размерность переменной τ). Запрещается использование операции обмена в том случае, когда символом повышенной размерности отмечен один и только один из операндов оператора $\Leftrightarrow$, то есть в ситуациях типа $\boldsymbol{a} \Leftrightarrow \boldsymbol{'b}$, $\boldsymbol{''a} \Leftrightarrow \boldsymbol{c}$, и т. п.

Допустим возможность использования выражений типа $(\boldsymbol{a''\, b'\, c\, d}) \Rightarrow \boldsymbol{C}$, при реализации которых производится «упаковка» (или «распаковка») в один комплекс нескольких операндов различной размерности.

Условия перехода по операторам $\circ\!\!\rightarrow$, $\mapsto$, $\dot{X}$ и $\ddot{X}$ при действии над операндами повышенной размерности определяются аналогично случаю операндов стандартной размерности.

Наконец, условимся, что выражения типа *'а печ*, *"В перф* и т. п. представляют выдачу значений соответствующих операндов повышенной размерности на печать или перфорацию.

Представления в оперативном комплексе. При использовании механизма автоматического повышения размерности операндов следует учесть особенности представления этих операндов в оперативном комплексе K. Эти особенности заключаются в следующем.

Как переменные, так и элементы комплексов повышенной размерности представляются некоторыми группами соседних элементов комплекса K. Естественно, что при этом переменная размерности $32 \cdot k$ представляется группой из k элементов комплекса K, подобно тому, как представляется некоторый основной комплекс, состоящий из k элементов стандартной размерности. Местоположение такой группы определяется значением одного из элементов специального комплекса N: посредством значения элемента n_0 задается номер первого из элементов оперативного комплекса K, образующих группу, представляющую значение переменной ***а*** повышенной размерности, то есть отмеченной одним из символов $'$, $''$, $'''$ или $''''$, элемент n_1 характеризует аналогичным образом переменную ***b*** повышенной размерности, и т. д.

При реализации программы требуется, чтобы к моменту выполнения операции над некоторой переменной повышенной размерности значение соответствующего элемента комплекса N уже было задано. Это может быть сделано заранее, путем ввода исходных данных, или уже в процессе реализации программы, то есть программным путем. Программист должен побеспокоиться и о том, чтобы место, отводимое в комплексе K для представления переменной повышенной размерности, оказалось достаточным в течение всего процесса реализации программы.

На представление некоторого комплекса, состоящего из m элементов размерности $32 \cdot k$, уходит $m \cdot k$ элементов

оперативного комплекса. Местоположение представляемого комплекса в комплексе K устанавливается обычным образом, то есть значением одного из элементов комплекса начал A, соответствующий же элемент комплекса мощностей B принимает значение m.

Примеры. Для иллюстрации методики действия с операндами повышенной размерности воспользуемся рассмотренной в § 4 задачей транспонирования булевой матрицы, представленной значением комплекса $\boldsymbol{A}$. Положим, что число столбцов исходной матрицы равно 32, в связи с чем размерность элементов комплекса $\boldsymbol{A}$ может быть стандартной, число же строк матрицы превышает 32 и задается параметрически, будучи равно значению b_0.

Очевидно, что при транспонировании матрицы число строк и число столбцов поменяются своими значениями и для представления результата данной операции потребуется комплекс с элементами повышенной размерности. В приводимой ниже программе решения сформулированной задачи таким комплексом является комплекс $\boldsymbol{B}$. Размерность его элементов вычисляется при реализации программы: параметр ω_1 принимает значение, равное целому числу, ближайшему сверху к частному от деления b_0 на 32 (в программе численные величины представлены в восьмеричной системе счисления).

Пусть данный алгоритм несколько отличается от приведенного в § 4 — здесь используется метод «прощупывания» всех элементов исходной матрицы и их восстановления в новой системе координат.

§ 0 $40 \Rightarrow b_1\, b_0 + 37 : 40\ \textit{я} \Rightarrow \textit{ж}\ \bar{\circ}\ a$

§ 1 $\triangle\ \dot{a} \oplus 40 \circ\!\!\rightarrow 3\ \circ\ '\boldsymbol{b}_{\boldsymbol{a}}\ \bar{\circ}\ b$

§ 2 $\triangle\ b \oplus b_0 \circ\!\!\rightarrow 1\ \boldsymbol{a}_b \wedge \boldsymbol{c}_{\boldsymbol{a}} \circ\!\!\rightarrow 2\ '\boldsymbol{b}_{\boldsymbol{a}} \vee c_b \Rightarrow\ '\boldsymbol{b}_{\boldsymbol{a}} \rightarrow 2$

§ 3 .

Рассмотрим другой пример. Допустим, что требуется придать переменной $\boldsymbol{d}$, размерность которой, определяемая параметром ω_1 ($\omega_1 = 4$), равна 128, такое значение, чтобы левые 32 компоненты этой переменной совпадали по значению с соответствующими компонентами переменной $\boldsymbol{a}$ стандартной размерности, а следующие три группы в 32 компоненты каждая совпадали аналогич-

ным образом с компонентами переменных ***m***, ***p*** и ***q***. Поставленная задача может быть решена следующей программой:

§ 0 $(a\,m\,p\,q) \Rightarrow A\,A \Rightarrow ('d)$.

Рекомендации. При составлении программ следует учитывать, что повышение размерности операндов описанным методом может повлечь за собой резкое замедление скорости реализации программы. Особое внимание следует обратить на оформление «узких мест» в программе, то есть тех относительно небольших ее участков, на которые падает основная часть времени, затрачиваемого на выполнение программы. В то же время бóльшая по объему часть программы не содержит, как правило, «узких мест» и поэтому ее преобразование, соответствующее повышению размерности некоторых операндов и производимое системой автоматического программирования, может происходить относительно безболезненно.

§ 8. Л-операторы и подпрограммы

Два уровня языка ЛЯПАС. Совокупность рассмотренных выше средств программирования (за исключением описанной в § 7 методики повышения размерности операндов) образует основное содержание *первого уровня* языка ЛЯПАС. Этот уровень предназначается для представления микроструктуры алгоритмов и оказывается достаточно удобным при подготовке программ, размеры которых ограничены двумя-тремя сотнями символов. При составлении более сложных программ их обозримость ухудшается, и воспринимать данные программы как единое целое становится все труднее. Сказываются особенности восприятия человека, привыкшего оперировать непосредственно схемами ограниченной сложности.

Однако это не означает, что сложный объект принципиально «невоспринимаем». Можно попытаться найти некоторое разложение его на более простые и «понятные» объекты, то есть найти схему отношений между рассматриваемым сложным объектом и «понятными» объектами, образующими базу разложения. Если сложность

полученной схемы невелика, то такая схема воспринимается относительно легко, а посредством нее — и исходный сложный объект.

Эта весьма общая идея реализуется в алгоритмических языках путем введения средств выражения одних алгоритмов через другие, позволяющих создавать многоступенчатые системы алгоритмов, в которых алгоритмы низших ступеней могут использоваться в качестве элементов алгоритмов высших ступеней. Подобная иерархическая структура присуща и системам автоматического программирования, воспринимающим данные языки: такие системы обладают библиотеками *подпрограмм*, то есть программ, реализующих некоторые общеупотребительные вычислительные процессы и оформленных таким образом, что их можно использовать наряду с элементарными операциями при составлении новых программ и, в частности, при составлении новых подпрограмм.

К числу алгоритмических языков данного типа относится и язык ЛЯПАС, точнее — его *второй уровень*, в котором представлены средства выражения алгоритмов с иерархической, многоступенчатой структурой. Ознакомление с этими средствами начнем с рассмотрения следующего конкретного примера.

Задача о кратчайшем покрытии. Допустим, что задана некоторая булева матрица $\boldsymbol{U}$, то есть матрица, элементы которой могут принимать значения только из множества {0, 1}. Обозначим через $\boldsymbol{U}_p^q$ минор этой матрицы, образованный пересечением подмножества p ее строк $\boldsymbol{u}_i$ с подмножеством q ее столбцов $\boldsymbol{u}^j$. Требуется найти такое подмножество n строк данного минора, чтобы эти строки в совокупности «покрывали» все столбцы, принадлежащие множеству q. Это означает, что для каждого столбца $\boldsymbol{u}^j \in q$ должна найтись такая строка $\boldsymbol{u}_i \in n$, чтобы образуемый их пересечением элемент u_i^j матрицы $\boldsymbol{U}$ обладал значением 1. При этом желательно, чтобы множество n, называемое *покрытием*, содержало как можно меньше элементов — строк рассматриваемого минора.

Покрытие, содержащее минимально возможное число элементов, называется *кратчайшим*. Будем считать, что

поставленная задача решается точно при нахождении именно кратчайшего покрытия и решается приближенно, если полученное покрытие оказывается не кратчайшим. Если нет гарантий, что получаемые по некоторому алгоритму решения всегда будут точными, то такой алгоритм будем называть *приближенным*.

Например, пусть мы имеем дело со следующей булевой матрицей, строки и столбцы которой для удобства пронумерованы:

	1	2	3	4	5	6	7	8	9	10	
	0	1	1	0	0	0	1	0	0	1	1
	0	1	0	0	0	0	0	1	1	0	2
	0	1	1	0	0	1	0	1	0	1	3
	1	0	0	1	1	1	0	0	0	1	4
	1	1	0	0	1	0	1	1	0	0	5
	1	0	1	0	0	1	1	0	1	1	6
	0	1	0	1	0	0	1	1	0	0	7
	1	0	1	1	1	0	1	0	0	0	8

Допустим, что множество p состоит из строк 1, 2, 4, 6, 7, 8, а множество q — из столбцов 1, 2, 4, 5, 7, 8, 9. Тогда покрытием минора U_p^q является, к примеру, совокупность строк 1, 2, 4 или совокупность 2, 4, 6. Однако наилучшим, согласно постановке задачи, покрытием оказывается совокупность 2, 8, поскольку она содержит минимально возможное количество строк, то есть является кратчайшим покрытием.

Заметим, что поставленная задача является вариантом известной задачи о нахождении оптимальных покрытий таблицы Квайна и имеет массу полезных интерпретаций. С некоторыми из них мы познакомимся позднее.

Структура приближенного алгоритма. Рассмотрим здесь весьма простой приближенный алгоритм решения сформулированной задачи с тем, чтобы использовать его в качестве иллюстрации способа расчленения алгоритмов на некоторые части с самостоятельным значением.

Назовем минимальным столбцом минора такой столбец, который содержит минимальное число единиц, а максимальной строкой — строку, содержащую максимальное число единиц. Согласно условиям задачи, требуется

покрыть каждый столбец минора, то есть включить в решение по крайней мере одну из строк минора, содержащих единицу в данном столбце. Трудно предсказать, какую из этих строк лучше выбрать с тем, чтобы минимизировать общее число строк в решении. Однако можно утверждать, что вероятность наилучшего выбора будет расти с уменьшением числа единиц в рассматриваемом столбце, поскольку при этом уменьшается число возможных вариантов выбора — например, если в столбце имеется лишь одна единица, то выбор будет однозначным и, следовательно, наилучшим. В этом смысле предпочтительнее начинать поиск покрытия некоторого минора с рассмотрения его минимального столбца. С другой стороны, среди покрывающих этот столбец строк разумнее выбрать максимальную, так как она покроет наибольшее число столбцов минора, и включить в решение именно ее. После этого остается решить аналогичную задачу для непокрытой еще части исходного минора, которая может быть представлена некоторым новым минором, и найти таким образом следующий элемент решения, затем рассмотреть следующий остаток и т. д., пока, наконец, все столбцы исходного минора не окажутся покрытыми.

В основе описанного алгоритма лежат соображения, не гарантирующие оптимальность получаемого решения, а лишь повышающие (по сравнению с другими случайными выборами очередного шага) вероятность того, что это решение окажется оптимальным или по крайней мере будет достаточно близко к оптимальному. Поэтому алгоритм является приближенным. Опишем его несколько более формально.

Реализация алгоритма начинается с выбора минимального столбца e^1 минора U_p^q и нахождения максимальной строки f_1 в миноре $U_{p^*}^q$, где p^* — множество строк, имеющих единичные элементы на пересечении со столбцом e^1. После этого минор U_p^q заменяется на минор $U_{p_1}^{q_1}$, где p_1 получается из p удалением строки f_1, а q_1 получается из q выбрасыванием всех столбцов, имеющих единичные элементы на пересечении со строкой f_1. Затем ищется минимальный столбец e^2 в миноре $U_{p_1}^{q_1}$ и т. д., до тех пор, пока множество еще непокрытых

столбцов q_i на некотором i-м шаге не окажется пустым. Совокупность выбранных к этому моменту строк представит искомое решение:

$$n = \{f_1, f_2, \ldots, f_{i-1}\}.$$

Решая вышеприведенный пример, данный алгоритм находит сначала минимальный столбец представленного минора, а именно, столбец 5 (пусть в том случае, когда минимальных столбцов несколько, выбирается первый из них, и аналогичным же образом производится выбор среди конкурирующих строк), затем выбирается максимальная среди строк, имеющих единичный элемент в столбце 5, а именно, строка 8, которая становится первым элементом искомого покрытия. После этого из исходного минора удаляется как строка 8, так и покрываемые ею столбцы 1, 4, 5 и 7, в результате чего получается новый минор, образованный пересечением строк 1, 2, 4, 6 и 7 со столбцами 2, 8 и 9. Он обрабатывается таким же образом, что приводит к получению второго элемента покрытия — строки 2. После этого множество непокрытых столбцов минора оказывается пустым, а это и служит признаком того, что полученная совокупность строк является покрытием.

Подпрограммы. Очевидно, что основные части описанного алгоритма могут быть представлены в виде двух операторов, имеющих самостоятельный смысл. Один из них будет обеспечивать нахождение минимальных столбцов в задаваемых минорах, другой — нахождение максимальных строк.

Подобного рода операторы с самостоятельным значением будем называть *Л-операторами* (в отличие от операторов первого уровня языка, называемых *элементарными*, или *л-операторами*), а программы, в которых они встречаются — *внешними* программами (в отличие от *внутренних* программ, или *подпрограмм*, необходимых для реализации самих Л-операторов). Вводимым Л-операторам и реализующим их подпрограммам будут присваиваться индивидуальные наименования, довольно произвольные сокращения которых будут служить *именами* этих подпрограмм и операторов, используемыми во внешней программе. Перед именем Л-оператора в программе ставится метка *, за именем должен следовать

в установленном порядке *перечень его операндов*, замыкаемый символом-меткой //.

Например, рассмотренную операцию нахождения минимального столбца минора можно представить выражением

$$* \text{минсто } U\ p\ q\ d\ //,$$

где минсто — имя оператора *нахождения минимального столбца*, а символы, входящие в перечень, конкретизируют набор операндов, определяющих объекты действия данного операнда: комплекс U представляет матрицу U (то есть элементами комплекса служат строки матрицы), p и q — переменные, значения которых определяют некоторые подмножества p и q строк и столбцов матрицы U, выделяя их из множеств всех строк и столбцов этой матрицы (выделяемые элементы множеств отмечаются единичными значениями соответствующих им компонент указанных переменных), наконец, d — переменная, которая должна принять значение, представляющее искомый минимальный столбец минора U_p^q. Положим, что это значение должно выделить подмножество строк минора, имеющих единичные элементы на пересечении с найденным столбцом, иными словами — представить найденный столбец в транспонированном виде.

В том случае, когда размерность некоторой величины (булева вектора или булевой матрицы) оказывается меньшей, чем размерность представляющего ее операнда, возникает вопрос: какие именно из компонент операнда следует использовать для представления? Здесь полезно ввести стандартный способ представления, при котором всегда используются в необходимом количестве левые компоненты операнда, неиспользуемые же компоненты принимают значение 0.

За символ стандартного представления примем «: :» и будем пользоваться выражением «a : : m» вместо выражения «переменная a представляет стандартным образом булев вектор m», «C : : N» вместо «комплекс C представляет стандартным образом булеву матрицу N» и т. д.

Например, если a : : m и если $m = 01001101$, то переменная a имеет значение 01001101000. . .0.

Во избежание возможных недоразумений заметим, что смысл операнда Л-оператора фиксируется его местоположением в перечне. Например, выражение

$$* \text{минсто } \boldsymbol{A\,d\,l\,e} \,//,$$

где Л-оператор минсто определен выше, означает операцию нахождения минимального столбца минора булевой матрицы, представленной комплексом $\boldsymbol{A}$. Минор образуется пересечением совокупности строк, представленной значением переменной $\boldsymbol{d}$, с совокупностью столбцов, заданной значением переменной $\boldsymbol{l}$. Результат же выполнения данной операции будет представлен, соответствующим образом, значением переменной $\boldsymbol{e}$.

По аналогии определяется оператор макстро — *нахождения максимальной строки минора*, и выражение

$$* \text{макстро } \boldsymbol{U\,p\,q}\,d \,//$$

означает операцию применения данного Л-оператора к минору $\boldsymbol{U}_p^q$, представляемому так же, как и в рассмотренном примере. Условимся, что в данном случае значением индекса d будет представлен номер найденной максимальной строки минора $\boldsymbol{U}_p^q$.

Используя оба этих оператора, можно весьма компактно выразить описанный алгоритм получения покрытия минора булевой матрицы:

$$\S\ 0 \quad \circ \boldsymbol{n}$$

$$\S\ 1 \quad * \text{минсто } \boldsymbol{U\,p\,q\,d} \,// * \text{макстро } \boldsymbol{U\,d\,q}\,d \,// \boldsymbol{n} \vee c_d \Rightarrow \boldsymbol{n} \urcorner \wedge \boldsymbol{p} \Rightarrow \boldsymbol{p}\,\boldsymbol{u}_d \urcorner \wedge \boldsymbol{q} \Rightarrow \boldsymbol{q} \mapsto 1.$$

При реализации этой программы текущие значения переменных $\boldsymbol{p}$ и $\boldsymbol{q}$ будут всегда выделять из матрицы, представленной комплексом $\boldsymbol{U}$, некоторый непокрытый остаток, а последовательно находимые элементы решения будут накапливаться в текущем значении переменной $\boldsymbol{n}$ (в виде единичных значений, принимаемых соответствующими компонентами этой переменной). Результат выполнения программы — перечень строк минора, образующих искомое покрытие, будет представлен конечным значением переменной $\boldsymbol{n}$.

Составление подпрограмм. Пока что мы не рассматривали, каким конкретно образом реализуются, в свою очередь, Л-операторы минсто и макстро. Сделано это может быть по-разному, то есть одному и тому же Л-оператору может соответствовать много различных программ.

Например, операция *макстро $Udqd//$ может быть реализована следующей программой, последовательно просматривающей строки минора, подсчитывающей число единиц в них и запоминающей максимальную среди уже просмотренных:

§ 0 $\circ bd \Rightarrow a \vdash \Rightarrow d$

§ 1 $a\dot{X}2a u_a \wedge q \bigtriangledown \Rightarrow c\,b - c \longmapsto 1\,a \Rightarrow d\,c \Rightarrow b \to 1$

§ 2 .

Исходная информация для этой программы задается значениями операндов U, d и q, результат — операндом d. Такого рода операнды, представляющие информационный вход и выход программы, будем называть *внешними* (для данной программы) операндами. Именно они и входят в перечень операндов реализуемого программой Л-оператора. Остальные операнды, фигурирующие в программе (для рассматриваемой программы к таким операндам относится переменная a, а также индексы a, b и c), называются *внутренними* и предназначаются для представления вспомогательной информации и промежуточных результатов, получаемых в ходе выполнения программы. При окончании реализации программы значения этих операндов могут оказаться измененными по сравнению с их исходными значениями, но этот эффект можно рассматривать не как основной, а как побочный и несущественный.

Рассмотренная программа еще не является подпрограммой, поскольку она привязана к конкретным внешним операндам U, d, q и d. Для того чтобы превратить программу в подпрограмму, требуется произвести соответствующее абстрагирование, придав ей такой вид, чтобы стало возможным ее автоматическое включение во внешнюю программу, в которой в ее первоначальном виде содержится лишь символ соответствующего Л-оператора, реализуемого данной подпрограммой. При этом

считается, что заранее не известно, какие конкретно операнды войдут в перечень тех операндов, на которые действует Л-оператор во внешней программе.

Указанное абстрагирование достигается путем замены символов ее внешних операндов на символы

$$\alpha,\ \beta,\ \gamma,\ \delta,\ \varepsilon,\ \zeta,\ \eta,\ \vartheta,\ \varkappa,\ \lambda,\ \mu,\ \nu,\ \xi,\ \pi,\ \rho,\ \sigma,$$

в строгом соответствии с местом операндов в перечне, сопровождающем символ Л-оператора. Например, символом γ заменяется символ того внешнего операнда оформляемой подпрограммы, который занимает в соответствующем перечне третье место.

В рассматриваемом примере такая замена приведет к получению следующего выражения:

$$\S\ 0\quad \circ b\,\beta \Rightarrow \boldsymbol{a} \vdash \Rightarrow \delta$$
$$\S\ 1\quad \boldsymbol{a}\,\dot{X}\,2\,a\,\boldsymbol{a}_{\boldsymbol{a}} \wedge \gamma \nabla \Rightarrow c\,b - c \longmapsto 1\,a \Rightarrow \delta\,c \Rightarrow b \to 1$$
$$\S\ 2\quad .$$

В этом виде программа принимает более универсальный характер, так как на места ее внешних операндов, занимаемые сейчас символами α, β, γ и δ, могут быть поставлены символы любых конкретных операндов, если только эти операнды относятся к подходящему типу (символ α может быть заменен символом некоторого комплекса, символы β и γ — символами переменных, а символ δ — символом индекса). Поэтому она легко может быть «настроена» на внешнюю программу, в которой содержится перечень конкретных операндов, сопровождающий символ Л-оператора, реализуемого данной подпрограммой.

Подпрограмма, реализующая Л-оператор минсто, отыскивающий минимальный столбец заданного минора, может быть составлена на основе принципа последовательного подсчета числа единиц в каждом столбце минора. Поскольку соответствующей элементарной операции в ЛЯПАСе нет, в отличие от случая, когда «взвешиваются» строки и можно использовать оператор ∇, процесс подсчета числа единиц в столбце представляется сложнее, с помощью отдельного предложения (в данном

случае — предложения 2). Подпрограмме в целом можно придать следующий вид:

§ 0 $\gamma \Rightarrow b \bar{\circ} d$
§ 1 $b \dot{X} 4 b \beta \Rightarrow a \circ c \circ c$
§ 2 $a \dot{X} 3 a \alpha_a \wedge c_b \circ\!\!\rightarrow 2 \triangle c c_a \vee c \Rightarrow c \rightarrow 2$
§ 3 $c - d \mapsto 1 c \Rightarrow d c \Rightarrow \delta \rightarrow 1$
§ 4 .

Компиляция. Рассмотрим теперь простой способ использования данных подпрограмм в вышеописанной программе получения покрытия минора булевой матрицы. По этому способу каждый Л-оператор во внешней программе вместе с сопровождающим его перечнем операндов заменяется реализующей его подпрограммой, причем символы α, β, ... заменяются соответствующими им элементами перечня. Кроме того, производится перенумерация предложений в используемых подпрограммах таким образом, чтобы в получаемой в результате данной подстановки программе не встречались предложения с одинаковыми номерами. С этой целью, нумеруя вновь предложения некоторой подпрограммы в момент замены ею символа очередного Л-оператора, мы будем просто продолжать использование чисел из натурального ряда, начатое во внешней программе. При этом, наряду с заменой номеров предложений, стоящих в обрабатываемой подпрограмме непосредственно вслед за символами §, должна быть произведена соответствующая замена операндов операторов перехода $\rightarrow$, $\circ\!\!\rightarrow$, $\mapsto$, $\dashv\!\!\rightarrow$, $\dot{X}$ и $\ddot{X}$.

Попутно обратим внимание на одну существенную деталь в оформлении подпрограмм, заключающуюся в том, что нумерация предложений в них должна производиться начальным отрезком натурального ряда чисел, начинающимся с 0 (пара символов § 0 играет роль *метки начала подпрограммы*). Наряду с этим внутри подпрограммы не должно быть переходов к предложению 0, поэтому в случае необходимости (когда в подпрограмме имеется переход к ее «начальной» операции) это предложение должно быть оформлено как «пустое», то есть не содержащее других символов, кроме символов § 0.

При подстановке подпрограммы производится перенумерация всех ее предложений, кроме нулевого (имеющего номер 0), которое вносится во внешнюю программу с отбрасыванием символов § 0. Таким образом, нулевое предложение подпрограммы теряет самостоятельность и становится лишь некоторой частью одного из предложений программы. Очевидно, что такая операция, позволяющая сократить число предложений в синтезируемой программе, допустима именно потому, что внутри подпрограммы нет переходов к предложению 0. Кроме того, при подстановке подпрограммы всегда отбрасывается символ « . » ее окончания.

Следуя указанным правилам, совершим подстановку подпрограмм минсто и макстро в рассматриваемую программу, то есть в программу

$$\S\ 0\quad \circ \boldsymbol{n}$$
$$\S\ 1\quad * \text{минсто}\, \boldsymbol{U\,p\,q\,d}\ /\!/\ * \text{макстро}\, \boldsymbol{U\,d\,q\,d}\ /\!/$$
$$\boldsymbol{n} \vee c_d \Rightarrow \boldsymbol{n}\ \urcorner \wedge \boldsymbol{p} \Rightarrow \boldsymbol{p}\,\boldsymbol{u}_d\ \urcorner \wedge \boldsymbol{q} \Rightarrow \boldsymbol{q} \mapsto 1.$$

Результатом подстановки явится следующая программа:

$$\S\ 0\quad \circ \boldsymbol{n}$$
$$\S\ 1$$
$$\left.\begin{array}{ll} & \boldsymbol{q} \Rightarrow \boldsymbol{b}\,\bar{\circ}\,d \\ \S\ 2 & \boldsymbol{b}\,\dot{\mathrm{X}}\,5\,b\,\boldsymbol{p} \Rightarrow \boldsymbol{a} \circ c \circ \boldsymbol{c} \\ \S\ 3 & \boldsymbol{a}\,\dot{\mathrm{X}}\,4\,a\,\boldsymbol{u}_a \wedge c_b \circ\!\!\rightarrow 3\ \triangle\ c\,c_a \vee \boldsymbol{c} \Rightarrow \boldsymbol{c} \rightarrow 3 \\ \S\ 4 & c - d \mapsto 2\,c \Rightarrow d\,\boldsymbol{c} \Rightarrow \boldsymbol{d} \rightarrow 2 \\ \S\ 5 & \end{array}\right\} \text{минсто}$$
$$\left.\begin{array}{ll} & \circ b\,\boldsymbol{d} \Rightarrow \boldsymbol{a} \vdash \Rightarrow d \\ \S\ 6 & \boldsymbol{a}\,\dot{\mathrm{X}}\,7\,a\,\boldsymbol{u}_a \wedge \boldsymbol{q}\ \triangledown \Rightarrow cb - c \mapsto 6a \Rightarrow dc \Rightarrow b \rightarrow 6 \\ \S\ 7 & \end{array}\right\} \text{макстро}$$
$$\boldsymbol{n} \vee c_d \Rightarrow \boldsymbol{n}\ \urcorner \wedge \boldsymbol{p} \Rightarrow \boldsymbol{p}\,\boldsymbol{u}_d\ \urcorner \wedge \boldsymbol{q} \Rightarrow \boldsymbol{q} \mapsto 1.$$

Таким образом, составленная на втором уровне ЛЯПАСа программа перешла в программу первого уровня, в которой для удобства выделены те ее участки, которые были получены в результате включения подпрограмм минсто и макстро.

Описанный процесс преобразования программы второго уровня в программу первого уровня называется *компиляцией*. Этот процесс в рассматриваемом случае достаточно прост и строго формален, в силу чего его нетрудно реализовать программным путем, с помощью специальной программы, называемой *компилятором*. Наличие компилятора в составе какой-либо системы автоматического программирования является одним из основных условий ее высокой эффективности, облегчая использование информации, накапливаемой в виде системы подпрограмм, и значительно упрощая в связи с этим процесс составления сложных программ.

Согласование совокупностей операндов. При более внимательном рассмотрении процесса компиляции можно заметить, что совокупность операндов внешней программы должна быть определенным образом согласована с совокупностями внутренних операндов используемых подпрограмм. Существует опасность «пересечения» этих совокупностей, которая выражается в том, что значение какого-либо операнда внешней программы может быть потеряно при реализации подпрограммы, если в последней этот операнд также фигурирует. Достаточным условием предотвращения подобных ситуаций является следующее правило: во внешней программе не должны встречаться те операнды, которые входят в совокупности внутренних операндов подпрограмм, используемых в данной программе.

Сформулированное условие не является, однако, необходимым, в этом можно убедиться на рассмотренном примере. В данном случае набор $\boldsymbol{U}$, $\boldsymbol{p}$, $\boldsymbol{q}$, $\boldsymbol{n}$, $\boldsymbol{d}$, c операндов внешней программы пересекается с набором $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, a, b, c, d внутренних операндов подпрограммы минсто: оба набора содержат индекс c. Тем не менее это не приводит в данном случае к каким-либо неприятностям, поскольку во внешней программе индекс c предназначается лишь для запоминания одного из промежуточных результатов — номера находимой максимальной строки минора, и к началу реализации подпрограммы минсто эта информация оказывается уже использованной, в связи с чем индекс c освобождается.

Таким образом, учет особенностей конкретного алгоритма позволяет в ряде случаев обойти правило «непе-

ресечения операндов» и сократить в связи с этим общее число используемых операндов. Однако делать это следует с особой осторожностью, помня о последствиях возможных ошибок.

Чтобы облегчить программирование на втором уровне, введем *дополнительные наборы* переменных и индексов, считая, что к этим наборам всегда относятся переменные и индексы, являющиеся внутренними операндами подпрограмм, в то время как операнды программ, не являющихся подпрограммами, никогда не могут к ним относиться. Значения входящих в дополнительные наборы операндов представляются состояниями особых ячеек оперативной памяти, отличных от тех ячеек, которые предназначены для представления значений переменных и индексов основных наборов, рассматривавшихся до сих пор. Таким образом устраняется опасность пересечения переменных и индексов внешней программы (если только последняя не является, в свою очередь, подпрограммой) с внутренними операндами используемых в ней подпрограмм. Поэтому при составлении внешней программы можно концентрировать внимание лишь на внешних операндах тех подпрограмм, которые соответствуют используемым Л-операторам; не задаваясь вопросом о составе их внутренних операндов.

Условимся не применять, как правило, специальную символику для обозначения операндов дополнительных наборов, представляя их теми же символами, что и соответствующие операнды основных наборов. Лишь в тех случаях, когда это действительно необходимо, будем отмечать переменные и индексы дополнительных наборов, используя с этой целью символ «тильда», проставляемый над символом операнда: например, $\tilde{a}$ — символ переменной a из дополнительного набора, $\tilde{m}$ — символ индекса m из дополнительного набора, и т. д.

Достаточно сложные программы могут иметь многоступенчатую иерархическую структуру, составленную из Л-операторов, многие из которых могут быть выражены опять-таки через Л-операторы, выражающиеся, в свою очередь, через другие Л-операторы, и т. д. Порядок оформления подпрограмм таких Л-операторов имеет свои особенности.

Поскольку внутренние переменные и индексы этих подпрограмм выбираются из того же набора, что и переменные и индексы подчиненных подпрограмм, необходимо тщательное информационное согласование. В связи с этим установим следующее правило.

При оформлении каждой подпрограммы требуется экономно расходовать в качестве ее внутренних операндов переменные и индексы, используя их в алфавитном порядке и отмечая последние из них. При этом нужно учитывать, какие из переменных и индексов уже используются в подчиненных подпрограммах. Последние (по алфавиту) из используемых в оформляемой подпрограмме переменных и индексов упоминаются во внешнем описании подпрограммы, с правилами составления которого мы познакомимся позже. Например, если в описании некоторой подпрограммы упомянуты переменная ***e*** и индекс *d*, то это означает, что в качестве внутренних операндов данной подпрограммы в совокупности со всеми подчиненными ей подпрограммами используются переменные ***a***, ***b***, ***c***, ***d***, ***e*** и индексы *a*, *b*, *c*, *d* дополнительных наборов и что никакой из этих операндов не должен встречаться во внешней программе, использующей данную подпрограмму.

Особо следует оговорить правила использования комплексов. Комплексы также могут играть роль как внешних, так и внутренних операндов, однако дополнительный набор комплексов не введен (делать это было признано нецелесообразным по техническим причинам, связанным с построением системы автоматического программирования), и поэтому при программировании приходится обходиться практически одним основным набором, так как набор специальных комплексов слишком специфичен.

Положим, что использование комплексов как внутренних операндов подпрограмм следует производить в обратном алфавитном порядке, начиная с комплекса ***Z*** (напомним, что операнды, представляемые буквами русского алфавита, предназначаются для специального использования), в остальном придерживаясь тех же правил, которые установлены для переменных и индексов. Например, если для использования в оформляемой подпрограмме в качестве внутренних операндов требуется

выбрать два комплекса, а в подчиненных ей подпрограммах уже использованы комплексы ***Z***, ***Y*** и ***X*** в качестве внутренних операндов этих подпрограмм, то следует выбрать комплексы ***W*** и ***V*** и упомянуть в описании оформляемой подпрограммы комплекс ***V*** как последний из использованных.

Подпрограммы-многополюсники. Формальный характер производимой при компиляции подстановки элементов перечня на места символов α, β, γ, ... позволяет расширить понятие внешнего операнда подпрограммы. Рассмотрим некоторые из вытекающих отсюда возможностей.

Реализация каждой из разобранных выше подпрограмм может начинаться лишь с операции, представленной в самом начале подпрограммы, отмеченном символами § 0 — это так называемый *входной полюс* подпрограммы, и кончаться лишь при достижении заключительного символа «.», замыкающего подпрограмму — это ее *выходной полюс*. Подпрограммы такого типа могут быть названы поэтому *двуполюсными*.

Из синтаксических определений следует, что любая программа, не являющаяся подпрограммой, всегда является двуполюсной. Иначе обстоит дело с подпрограммами более общего типа, являющимися не чем иным, как «кусками» более сложных программ, вырезанными из них довольно произвольным образом. Кроме указанных двух полюсов, называемых *основными*, они могут обладать *дополнительными* полюсами, как входными, так и выходными, обеспечивающими многообразие связей подпрограммы с внешней программой.

В то время как основной выходной полюс подпрограммы привязан к ее концу (отмеченному символом «.»), дополнительные выходные полюсы могут задаваться в произвольных участках подпрограммы, будучи представлены там сочетаниями одного из символов перехода ($\rightarrow$, $\circ\!\!\rightarrow$, $\mapsto$, $\twoheadrightarrow$, $\dot{X}$, $\ddot{X}$) с символом внешнего операнда (α, β, ..., σ), имеющего в данном случае сугубо специальное значение.

Допустим теперь, что требуется замкнуть дополнительный полюс подпрограммы, представленный в ней символами $\circ\!\!\rightarrow\alpha$ на предложение 5 внешней программы. Очевидно, что для этого достаточно будет поставить

символ 5 на первую позицию (соответствующую операнду α) в перечне, сопровождающем символ соответствующего Л-оператора во внешней программе. При производимой компилятором подстановке подпрограммы выражение $\circ\!\!\rightarrow \alpha$ будет заменено $\circ\!\!\rightarrow 5$ и требуемое замыкание будет осуществлено.

Если же подпрограмма имеет дополнительные входные полюсы, то они должны обязательно соответствовать началам каких-либо предложений подпрограммы, и связь с ними задается во внешней программе с помощью тех же операторов перехода, причем за их символами следует символ Л-оператора и его имя, а также номер соответствующего предложения подпрограммы, представляемый натуральной константой. Например, выражение $\rightarrow *$ сосна 4, принадлежащее некоторой программе, означает операцию безусловного перехода к одному из дополнительных полюсов подпрограммы сосна, а именно — к предложению 4 этой подпрограммы.

В соответствии со сказанным можно говорить о дополнительных входных полюсах подпрограмм, указывая номера соответствующих предложений, выходные же дополнительные полюсы удобнее обозначить посредством символов α, β, ...

Использование подпрограмм-многополюсников позволяет, вообще говоря, разбивать сложные программы на подпрограммы весьма произвольным образом. Однако при этом следует всегда стремиться к тому, чтобы оформленные в виде подпрограмм части программы имели самостоятельный смысл, чтобы число внешних операндов этих подпрограмм было разумно ограничено, а сами подпрограммы были по возможности более универсальными, то есть применимыми в возможно большем числе различных ситуаций.

Операторы как операнды. Определенные возможности представляет и использование операторов в качестве внешних операндов подпрограмм. Например, такая возможность реализуется в приводимой ниже подпрограмме проскал, вычисляющей скалярное произведение двух комплексов по произвольному двуместному оператору, из числа элементарных операторов ЛЯПАСа. Скалярное произведение комплексов $\boldsymbol{A}$ и $\boldsymbol{B}$ по двуместному оператору ρ_2 определяется при этом как такой ком-

плекс C, для которого $\sigma(C)=\min(\sigma(A), \sigma(B))$ и при $i \in \{0, 1, \ldots, \sigma(C)-1\}$ $c_i = a_i \rho_2 b_i$.

Определим смысл внешних операндов подпрограммы. Пусть β и γ представляют «перемножаемые» комплексы, а δ — результативный комплекс. Это представление носит абстрактный характер, так как в самой подпрограмме нет никаких указаний на то, какие именно комплексы имеются в виду. Конкретизация наступает лишь при использовании подпрограммы, когда символы β, γ и δ будут заменены символами конкретных комплексов (например, символами A, B и C). Аналогичным образом используем операнд α для абстрактного представления двуместного оператора, по которому должно вычисляться скалярное произведение: при использовании подпрограммы символ α должен быть заменен символом некоторого конкретного двуместного оператора, который и должен указываться в той программе, где используется подпрограмма п р о с к а л.

Например, операция вычисления скалярного произведения комплексов A и B по оператору $\oplus$, с представлением результата вычислений значением комплекса C, может быть записана как

$$* \text{ прскал} \oplus A\,B\,C\ //$$

Подпрограмма п р о с к а л имеет следующий вид:

§ 0 $\bar{\circ} a\, t_\beta \Rightarrow b_\delta - b_\gamma \circ\!\!\rightarrow 1\, b_\gamma \Rightarrow b_\delta$
§ 1 $\triangle\, a \oplus b_\delta \circ\!\!\rightarrow 2\, \beta_a\, \alpha\, \gamma_a \Rightarrow \delta_a \rightarrow 1$
§ 2 .

Центральное место в этой подпрограмме принадлежит операции

$$\beta_a\, \alpha\, \gamma_a \Rightarrow \delta_a,$$

осуществляющей «перемножение» [a]-х элементов комплексов β и γ по оператору α и фиксирующей получаемый при этом результат значением [a]-го элемента комплекса δ.

Особо следует остановиться на смысле символов b_β, b_γ и b_δ. Эти символы обозначают элементы специального комплекса B, которые представляют мощности «абстрактных» комплексов β, γ и δ, соответственно. При компиляции они должны быть обработаны особым образом:

на места символов β, γ и δ должны быть подставлены не символы конкретных комплексов, а их номера в форме натуральных констант. Например, если комплексы β, γ и δ конкретизируются как ***A***, ***B*** и ***C***, то символы b_β, b_γ и b_δ превратятся после компиляции в b_0, b_1 и b_2, соответственно.

Точно таким же образом производится и обработка при компиляции символов типа a_α, a_β, ..., применяемых в подпрограммах для обозначения элементов специального комплекса A, представляющих «начала» «абстрактных» комплексов α, β, ..., соответственно.

Составные элементы перечня. Пока мы ограничивались рассмотрением таких элементов перечня, каждый из которых представляется одним символом. Такие элементы будем называть *простыми*, в отличие от *составных*, для представления которых требуется не менее двух символов. К составным элементам перечня относятся, в частности, символы элементов комплексов, состоящие, по сути, из двух символов: символа комплекса и символа индекса или натуральной константы. Чтобы составной элемент перечня не мог быть воспринят как совокупность нескольких простых элементов, условимся заключать его в круглые скобки, строя выражения типа

$$*\text{минсто}\ \boldsymbol{A}(c_5)\boldsymbol{d}(m_i)//',\quad *\text{макстро}\ \boldsymbol{B}\,\boldsymbol{d}(a_q)e//$$

и т. п. В процессе компиляции, когда составные элементы перечня подставляются на места символов α, β, ..., окаймляющие их скобки отбрасываются.

Введение составных элементов перечня открывает широкие возможности использования самых разнообразных выражений языка ЛЯПАС в качестве конкретных внешних операндов подпрограммы. Однако делать это можно только после тщательного знакомства с деталями процесса компиляции, в противном случае легко допустить ошибки, последствия которых трудно предугадать.

«Штриховка» внутренних операндов подпрограмм. Остановимся на вопросе «штриховки» внутренних операндов подпрограмм, как мы будем впредь называть процедуру установления для них требуемой размерности. Очевидно, что размерность внутренних операндов подпрограмм зависит от размерности внешних и

что «штриховка» последних должна сопровождаться соответствующей штриховкой внутренних операндов. Условимся представлять информацию, необходимую для выполнения такой штриховки, перечислением связей, существующих между внешними и внутренними операндами.

Способ этого перечисления должен стать ясным из рассмотрения следующего примера: выражение $(\alpha\, b\, e\, n)(\beta\, c)$ означает, что при любом использовании описываемой подпрограммы размерность внутренних операндов b, e и n должна совпадать с размерностью внешнего операнда α, а размерность внутреннего операнда c должна совпадать с размерностью внешнего операнда β. Следовательно, если окажется, что эти внешние операнды будут обладать повышенной размерностью, то нужно будет соответствующим образом заштриховать и указанные внутренние операнды.

Эта процедура выполняется автоматически в процессе компиляции, что позволяет при работе с операндами повышенной размерности пользоваться обычными подпрограммами, составленными для случая операндов стандартной размерности, если только они снабжены указанной выше информацией.

§ 9. Иерархическое программирование

Принципы иерархического программирования. Так называется программирование с использованием второго уровня языка, когда вначале определяется крупноблочная структура алгоритма, а затем уже составляются подпрограммы, соответствующие отдельным блокам алгоритма.

Первый из этих двух этапов является определяющим. Он неразрывно связан с анализом поставленной задачи и выбором метода ее решения, который делается с учетом вычислительных возможностей машин. На этом же этапе устанавливается форма представления перерабатываемой информации, достаточно удобная для использования как внутри машины, так и вне ее.

Второй уровень языка ЛЯПАС является языком открытого, растущего типа. Набор его Л-операторов может постоянно расширяться, получая пополнение при составлении новых программ, которые так разбиваются на

подпрограммы, чтобы последние носили по возможности более универсальный характер, с тем, чтобы область их потенциального применения была как можно шире. С другой стороны, при составлении новой программы целесообразно использовать некоторые из уже имеющихся подпрограмм, выделяя из алгоритма соответствующие блоки. Как правило, это удается сделать, и тем естественнее, чем больше запас накопленных подпрограмм. Очевидно, что при расширении этого запаса эффективность программирования возрастает.

Если составленная программа в целом не слишком специфична и может быть использована в дальнейшем при построении алгоритмов решения некоторых других задач, то она оформляется как подпрограмма.

К числу существенных этапов разработки программы относится отладка программы, в ходе которой отыскиваются и устраняются всевозможные ошибки, а также последующее исследование эффективности полученной программы, при котором выявляются «узкие» места в программе и намечаются пути ее совершенствования. Эти этапы выполняются с помощью машин и будут рассмотрены позже.

В настоящем параграфе мы остановимся на следующих этапах иерархического программирования: анализ постановки задачи, выбор формы представления перерабатываемой информации, разработка крупноблочной структуры алгоритма, составление соответствующих подпрограмм (или выбор из числа имеющихся), сочленение подпрограмм, оформление программы в целом, ее анализ. Рассмотрение этих этапов мы привяжем к конкретной логической задаче.

Задача о размещении отдыхающих в лодках. Эта задача может возникнуть перед организатором воскресной прогулки по реке, старающимся разместить отдыхающих в минимальном числе лодок, но так, чтобы ни в одной лодке не оказалось двух лиц, которые могли бы поссориться между собой и тем самым испортить прогулку. Считается, что организатору заранее известен полный список «несовместимых» пар. Положим также для простоты, что лодки достаточно велики и каких-либо ограничений на число помещаемых в одну лодку людей не существует.

Например, пусть число отдыхающих равно десяти. Обозначим их через $a_1, a_2, \ldots, a_{10}$. Допустим, что взаимные отношения отдыхающих характеризуются следующим списком «несовместимых» пар: (1,2), (2,4), (2,9), (2,10), (4,5), (4,6), (4,9), (5,6), (6,8), (6,10), (7,8), (7,9), (9,10). Здесь для простоты через (1,2) обозначена пара отдыхающих a_1 и a_2, которых нельзя посадить в одну лодку, аналогично показана пара a_2 и a_4 и т. д.

Совокупность перечисленных пар задает на множестве всех пар отдыхающих некоторую бинарную функцию, которую можно назвать *функцией несовместимости*. Эта функция получает обозримое выражение в виде графа (рис. 1.9.1), вершины которого (представленные цифрами) соответствуют отдыхающим, а ребра (отрезки прямых) отмечают пары отдыхающих, связанных отношением несовместимости. Граф данного типа называется *симметрическим*, поскольку он представляет симметрическое бинарное отношение: в рассматриваемом случае, например, если a_i несовместим с a_j, то и a_j несовместим с a_i.

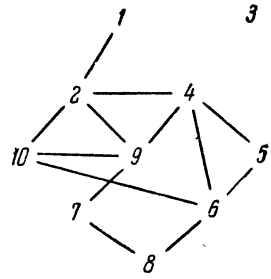


Рис. 1.9.1.

Поскольку построенный граф представляет функцию несовместимости, его можно назвать *графом несовместимости*.

Абстрактная формулировка задачи. Прежде всего, полезно «очистить» задачу от несущественных для решающего алгоритма деталей, приведя ее к некоторой абстрактной форме. При этом задача освобождается от конкретной интерпретации и ее сущность обнажается. Это, с одной стороны, упрощает процесс построения решающего алгоритма и, с другой стороны, расширяет значение получаемых результатов: построенный алгоритм может быть использован для решения, казалось бы, совершенно различных задач, формулировки которых по существу представляют собой лишь разные интерпретации одной и той же абстрактной задачи.

Задача о размещении отдыхающих в лодках может быть сформулирована следующим образом.

Задано некоторое множество A. На множестве пар элементов из A определена бинарная функция f, называемая функцией несовместимости и принимающая значение 1 на тех парах, которые называются несовместимыми. Требуется найти такое разбиение множества A на минимальное число подмножеств, при котором функция f будет обращаться в 0 на каждой паре элементов, принадлежащих одному и тому же подмножеству, то есть каждое подмножество будет содержать лишь совместимые пары.

Напомним, что *разбиением множества* A называется такое множество Q его подмножеств $A_1, A_2, \ldots, A_k$, которое покрывает множество A (то есть $A_1 \cup A_2 \cup \ldots \ldots \cup A_k = A$) и состоит из взаимно непересекающихся подмножеств (если $A_i \in Q$, $A_j \in Q$ и $A_i \neq A_j$, то $A_i \cap \cap A_j = \varnothing$). Разбиение, удовлетворяющее некоторым условиям и состоящее из минимального возможного числа подмножеств, назовем *минимальным*.

Подмножество A_i, не содержащее несовместимых пар элементов, может быть названо *совместимым подмножеством*, а сформулированная в абстрактной форме задача — задачей нахождения *минимального разбиения множества A на совместимые подмножества.*

Представляет интерес построение как «точных», так и «приближенных» алгоритмов решения этой задачи. Первые из них должны находить точное решение, соответствующее минимальному значению параметра k, вторые же должны давать достаточно хорошие приближения к точному решению, не гарантируя, однако, их оптимальность. В этом параграфе будет рассматриваться приближенный алгоритм.

Представление исходной информации. Поскольку операндами языка ЛЯПАС служат булевы векторы и матрицы, именно эти величины и стоит использовать для представления исходной информации к решаемой задаче.

Представим функцию несовместимости f булевой матрицей $\boldsymbol{F} = \|A * A\|$ бинарного отношения несовместимости $*$, полагая, что элемент f_i^j этой матрицы принимает значение 1 в том и только в том случае, когда выполняется отношение $a_i * a_j$, то есть когда элементы a_i и a_j множества A оказываются несовместимыми.

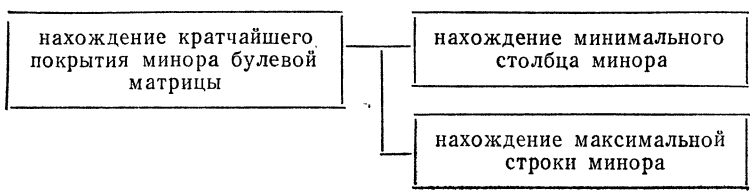
Для рассматриваемого нами примера матрица F принимает следующее значение:

$$
\begin{array}{c}
\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\
\left[\begin{array}{cccccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}\right]
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array}
\end{array}
$$

Матрица F оказывается не чем иным, как *матрицей смежности* графа несовместимости (рис. 1.9.1): единичными элементами матрицы отмечены *смежные* пары вершин графа, то есть соединенные некоторым ребром.

Первичное разложение задачи. Построение алгоритма решения некоторой задачи начинается (если только задача не слишком проста) с ее разложения на некоторую совокупность более простых задач, к решению которых можно свести в основном решение исходной задачи и которые образуют *базу разложения*. Это разложение отражает основную идею метода решения задачи, и его нахождение представляет собой наиболее «творческий» и труднее всего поддающийся формализации этап разработки алгоритма.

Например, рассмотренная выше задача о нахождении кратчайшего покрытия минора булевой матрицы была разложена на совокупность двух задач: задачу нахождения минимального столбца минора и задачу нахождения максимальной строки минора. Это разложение можно представить следующей схемой:



Данное разложение не является полным, то есть исходная задача не полностью сводится к задачам, образующим базу разложения. Однако оно достаточно эффективно, позволяя составить довольно компактную программу решения рассматриваемой задачи путем использования Л-операторов минсто и макстро, соответствующих элементам базы разложения.

Может оказаться, что некоторые из задач, входящих в базу разложения, будут, в свою очередь, довольно сложными и потребуют собственного разложения, а возникающие при этом новые задачи также могут быть разложены, и т. д. Совокупность порожденных таким образом задач образует иерархическую структуру. Аналогичной структурой будет обладать и система подпрограмм, решающих эти задачи, в связи с чем процесс разработки программы решения исходной задачи в целом может быть назван *иерархическим программированием.*

В рассмотренном случае (нахождение кратчайшего покрытия минора булевой матрицы) возникшие при разложении задачи оказались довольно простыми и не потребовали собственного разложения. Решающие эти задачи программы минсто и макстро достаточно просто выражаются в элементарных операциях ЛЯПАСа.

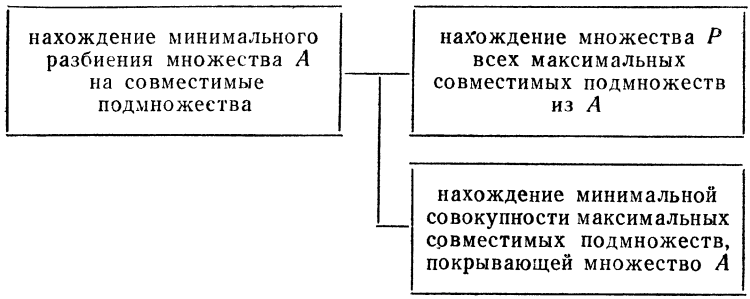
Порождаемая в этом случае иерархия задач оказалась простейшей, совпадая с *первичным разложением*, то есть непосредственным разложением исходной задачи, без последующего разложения задач, входящих в базу непосредственного разложения.

Возвратимся, однако, к задаче нахождения минимального разбиения множества A на совместимые подмножества.

Более или менее очевидно, что, решая эту задачу, достаточно ограничиться рассмотрением множества P всех *максимальных совместимых подмножеств* из A, то есть таких совместимых подмножеств A_i, для каждого из которых не существует отличного от него совместимого подмножества A_j из A, такого, что $A_i \subset A_j$. Из множества P следует выбрать некоторую совокупность, покрывающую множество A и состоящую из минимального числа совместимых подмножеств. К нахождению этой совокупности в основном и сводится решение всей за-

дачи: чтобы получить затем искомое разбиение, достаточно будет обеспечить взаимное непересечение выбранных совместимых подмножеств, удалив из них некоторые элементы.

Данное рассуждение представляет собой основную идею предлагаемого метода решения, которую уже нетрудно было бы оформить в более строгом виде. Эта идея отражается следующим первичным разложением решаемой задачи:



Подпрограмма прикрапок. Разумеется, сделанный шаг еще далеко не привел нас к детальному алгоритму решения поставленной задачи. Тем не менее становится ясно, за решение каких именно более простых задач следует взяться. Как найти множество P, мы рассмотрим позднее. Пока же займемся второй из задач, образующих базу найденного первичного разложения.

Нетрудно сообразить, что при решении этой задачи может пригодиться построенная в § 8 программа нахождения кратчайшего покрытия минора булевой матрицы. Конечно, можно составить специальную программу решения аналогичной задачи, которая будет более эффективной в рассматриваемой ситуации. Однако полезно познакомиться с методикой использования уже имеющихся программ. Во многих случаях такой путь более целесообразен.

Прежде всего, упомянутую программу следует оформить как подпрограмму с тем, чтобы ее можно было применить в программе решения рассматриваемой в этом параграфе задачи.

Программа имеет следующий вид:

§ 0 ○ $\boldsymbol{n}$
§ 1 * минсто $\boldsymbol{U\,p\,q\,d}$ // * макстро $\boldsymbol{U\,d\,q\,d}$ //
$\boldsymbol{n} \vee c_d \Rightarrow \boldsymbol{n}\ \urcorner \wedge \boldsymbol{p} \Rightarrow \boldsymbol{p}\,\boldsymbol{u}_d\ \urcorner \wedge \boldsymbol{q} \Rightarrow \boldsymbol{q} \mapsto 1.$

Исходная информация для нее задается комплексом $\boldsymbol{U}$, представляющим рассматриваемую булеву матрицу, а также переменными $\boldsymbol{p}$ и $\boldsymbol{q}$, выделяющими соответственно строки и столбцы в матрице. Совокупность строк матрицы, представляющая собой искомое покрытие, задается получаемым в результате значением переменной $\boldsymbol{n}$.

Эти операнды, имеющие в данной программе конкретный характер, при оформлении подпрограммы должны быть заменены абстрактными операндами, играющими роль внешних операндов подпрограммы. Заменив $\boldsymbol{U}$, $\boldsymbol{p}$, $\boldsymbol{q}$ и $\boldsymbol{n}$ на α, β, γ и δ, соответственно, получим выражение подпрограммы:

§ 0 ○ δ
§ 1 * минсто $\alpha\,\beta\,\gamma\,\boldsymbol{d}$ // * макстро $\alpha\,\boldsymbol{d}\,\gamma\,d$ //
$\delta \vee c_d \Rightarrow \delta\ \urcorner \wedge \beta \Rightarrow \beta\,\alpha_d\ \urcorner \wedge \gamma \Rightarrow \gamma \mapsto 1.$

Присвоим этой подпрограмме наименование прикрапок (приближенный алгоритм нахождения кратчайшего покрытия).

Нахождение максимальных совместимых подмножеств. Решить эту задачу можно, перебирая подмножества из A, находя среди них совместимые подмножества и выделяя среди последних максимальные совместимые подмножества. При этом не обязательно перебирать все подмножества из A: если, например, некоторое подмножество несовместимо, то и любое его расширение, получаемое добавлением к нему каких-либо из других элементов множества A, также будет несовместимо, следовательно, расширения несовместимых подмножеств можно не рассматривать. С другой стороны, если уже найдено некоторое максимальное совместимое подмножество, то незачем перебирать его подмножества: заранее известно, что все они совместимы, но не максимальны.

Алгоритм, полностью устраняющий указанную избыточность перебора, может оказаться довольно сложным. Рассмотрим поэтому метод частичного устранения этой избыточности.

Отображая перебор подмножеств из A перебором представляющих их булевых векторов, начнем его с вектора

1 2 3 4 5 6 7 8 9 10

[0 0 0 0 0 0 0 0 0 1],

содержащего лишь одну единицу, в правом разряде (число пронумерованных для удобства компонент вектора равно числу элементов множества A). Считая, что представляемое этим вектором подмножество, содержащее лишь один элемент a_{10}, безусловно совместимо, попытаемся его расширить, добавив единицу в следующий (справа налево) разряд:

1 2 3 4 5 6 7 8 9 10

[0 0 0 0 0 0 0 0 1 1].

Однако представляемое теперь подмножество $\{a_9, a_{10}\}$ оказывается несовместимым, так как $a_9 * a_{10}$ (см. рис. 1.9.1). Поэтому добавленную единицу следует выбросить и испытать следующее расширение $\{a_8, a_{10}\}$:

1 2 3 4 5 6 7 8 9 10

[0 0 0 0 0 0 0 1 0 1]

(крайняя слева единица как бы передвигается на один разряд влево). Новое подмножество совместимо, поэтому добавленная единица в векторе сохраняется, и производится следующее расширение:

1 2 3 4 5 6 7 8 9 10

[0 0 0 0 0 0 1 1 0 1].

Оно оказывается неприемлемым, поскольку несовместима пара a_7 и a_8, поэтому левая единица двигается влево:

0 0 0 0 0 1 0 1 0 1.

Продолжая процесс перебора аналогичным образом, мы получим через некоторое время вектор

1 0 1 0 1 0 0 1 0 1,

после чего продолжение процесса по аналогии становится невозможным. Назовем этот момент реализации перебора *особой точкой* и рассмотрим представляемое полученным здесь вектором подмножество $\{a_1, a_3, a_5, a_8, a_{10}\}$. Учитывая характер процесса его построения, нетрудно убедиться в том, что оно совместимо и максимально. Запомним этот результат.

В особой точке (после получения единицы в левом разряде) выбросим все единицы, левее которых нет нулей (здесь такой единицей оказывается первая слева), а затем сместим на один разряд влево левую из оставшихся единиц (допустим, что до смещения она находилась в позиции i). Этот шаг, называемый *особым*, достаточно прост и соответствует выбрасыванию пока из рассмотрения подмножеств, содержащих элемент a_i. Именно в этом шаге кроется причина неполноты устранения избыточности перебора, как мы увидим ниже, продолжая решение нашей задачи.

Итак, получим вектор

0100100101.

Подмножество из A, соответствующее этому вектору, оказывается несовместимым. Поэтому, продолжая уже знакомый процесс, сдвинем левую единицу влево:

1000100101

и вновь попадем в особую точку.

Представленное полученным здесь вектором подмножество оказывается совместимым, но максимально ли оно? Причиной его немаксимальности может быть лишь некоторый из сделанных ранее особых шагов. Но это означает, что в этом случае среди полученных ранее максимальных совместимых подмножеств должно найтись такое, которое является расширением данного подмножества. Поэтому проверку максимальности получаемых в особых точках совместимых подмножеств можно свести к сравнению полученного здесь вектора с векторами, образующими некоторую совокупность, в которой накапливаются результаты предшествующего перебора и в которой представлены все найденные к текущему моменту времени максимальные совместимые подмножества.

В данном случае достаточно сравнить вектор

1000100101

с полученным ранее вектором

1010100101,

чтобы убедиться в немаксимальности совместимого подмножества, представляемого первым из них.

Выполненный к данному моменту времени перебор можно проиллюстрировать следующей таблицей, в которой символом «+» отмечены совместимые из перебираемых подмножеств, символом «—» — несовместимые:

0000000001 +
0000000011 —
0000000101 +
0000001101 —
0000010101 —
0000100101 +
0001100101 —
0010100101 +
0110100101 —
1010100101 + максимальное
0100100101 —
1000100101 + немаксимальное

Перебор кончается при получении вектора, у которого все единицы окажутся слева, и к этому моменту будут найдены все максимальные совместимые подмножества из A. Совокупность представляющих их булевых векторов образует в рассматриваемом примере следующую булеву матрицу:

$$\| P \ni A \| = \begin{bmatrix} 1010100101 \\ 1011000101 \\ 1010101001 \\ 1011001001 \\ 1010100110 \\ 1010010010 \\ 0110100100 \\ 0110011000 \\ 1010011000 \\ 0110101000 \end{bmatrix}$$

Характер изложенного метода поиска максимальных совместимых подмножеств позволяет разложить данную задачу на следующие три: перебор подмножеств из A по описанным правилам, проверку перебираемых подмножеств на совместимость и запоминание тех из них, которые оказываются максимальными среди уже найденных совместимых подмножеств.

Начнем с последней из них:

Задача включения в матрицу максимальных векторов. Эту задачу можно сформулировать следующим образом. Заданы булев вектор m и булева матрица C, размерность строк которой совпадает с размерностью вектора m. Требуется определить, имеется ли в матрице C такая строка, которая совпадала бы с вектором m или была бы «больше» его, то есть содержала бы 1 во всех тех разрядах, в которых содержится 1 в векторе m. Если такой строки в матрице C не окажется, то в эту матрицу требуется добавить новую строку, равную вектору m.

Включаемые в матрицу C новые строки можно назвать *максимальными*, а решаемую задачу — задачей включения в матрицу максимальных векторов. Задача выглядит довольно простой и можно, не производя дальнейшего разложения, приступить к составлению подпрограммы ее решения, присвоив ей наименование включмак.

Подпрограмма включмак будет предназначаться для многократного использования в составе более сложных программ, строящих некоторую последовательность булевых векторов, из которой нужно выбрать и запомнить максимальные. Получаемая совокупность может оказаться довольно большой, и следует опасаться нехватки оперативной памяти, где будет фиксироваться эта совокупность при реализации программы. В связи с этим следует предусмотреть контроль за ростом данной совокупности, который можно осуществить методом «сторожа» и «аварийного выхода».

Суть метода заключается в установлении заранее допустимого размера результирующего комплекса, то есть комплекса, представляющего матрицу C, и в слежении за текущим значением мощности этого комплекса (соответствующая группа операций называется «сторожем»),

Как только мощность комплекса превосходит установленный предел, реализуется дополнительный «аварийный» выход из подпрограммы.

Определим внешние операнды подпрограммы: α — дополнительный «аварийный» выходной полюс подпрограммы, задаваемый натуральной константой, β — комплекс, представляющий матрицу $\boldsymbol{C}$, γ — переменная, представляющая булев вектор $\boldsymbol{m}$, δ — индекс, значение которого задает максимально допустимую мощность комплекса β.

Подпрограмма достаточно проста, поэтому приведем ее без дальнейших пояснений.

§ 0 $\bar{\bigcirc} a$

§ 1 $\triangle a \oplus b_\beta \circ\!\!\rightarrow 2\beta_a \urcorner \wedge \gamma \longmapsto 1 \rightarrow 3$

§ 2 $\gamma \Rightarrow \beta_a \triangle b_\beta \oplus \delta \circ\!\!\rightarrow \alpha$

§ 3 •

Анализ подмножеств из A на совместимость. Необходимым и достаточным условием совместимости некоторого подмножества A_i из A является совместимость каждой из содержащихся в подмножестве A_i пар элементов множества A.

Обозначив через B_i подмножество из A, образованное такими его элементами, каждый из которых несовместим по крайней мере с одним из элементов подмножества A_i, нетрудно сформулировать следующий критерий совместимости подмножества A_i: подмножества A_i и B_i не должны пересекаться.

Проверку этого критерия возложим на следующую подпрограмму, присвоив ей наименование анасовм:

§ 0 $\gamma \Rightarrow \boldsymbol{a}$

§ 1 $\boldsymbol{a} \dot{\mathrm{X}} \alpha a \beta_a \wedge \gamma \circ\!\!\rightarrow 1.$

В данной подпрограмме комплексом β представляется булева матрица $\boldsymbol{F}$, задающая функцию несовместимости, определенную на множестве пар элементов из A, переменной γ представляется анализируемое подмножество A_i (то есть значением этой переменной служит булев вектор $\| \{A_i\} \ni A \|$, принимающий значение 1 в тех компонентах, которые соответствуют элементам множества A, принадлежащим в то же время

подмножеству A_i), наконец, внешний операнд α играет роль дополнительного выходного полюса подпрограммы, реализуемого в том случае, когда рассматриваемое подмножество оказывается совместимым (в противном случае совершается выход через основной полюс).

Подпрограмма перебор. Описанный выше метод перебора подмножеств может быть реализован следующей подпрограммой, рассмотрение которой представляет особый интерес в связи с тем, что, во-первых, у нее полностью отсутствуют внутренние операнды и, во-вторых, она имеет три выходных и три входных полюса. Эту подпрограмму, следовательно, можно изобразить в виде многополюсника (рис. 1.9.2), обозначая дополнительные входные полюсы номерами соответствующих предложений в подпрограмме (в данном случае — символами *2* и *3*), а дополнительные выходные полюсы — символами α и β. Основные полюсы оставим необозначенными.

Рис. 1.9.2.

Эта подпрограмма, названная перебор, обладает четырьмя внешними операндами. Операнды α и β, как мы только что увидели, имеют специальное назначение, представляя дополнительные выходные полюсы подпрограммы. Значениями переменной γ служат булевы векторы $\| \{A_i\} \ni A \|$, представляющие перебираемые подмножества A_i множества A. Индекс δ задает номер того элемента множества A, который добавляется к варьируемому подмножеству при последней попытке его расширения.

В начале перебора следует обратиться к основному входному полюсу подпрограммы. При этом переменная γ должна иметь значение $00\ldots0$, а индекс δ должен задавать мощность множества A ($[\delta] = \sigma(A)$). Выход из подпрограммы через основной выходной полюс будет служить признаком конца перебора.

При реализации дополнительного выхода α значением переменной γ будет представлено подмножество, которое требуется проанализировать на совместимость. Если при такой проверке будет установлено, что данное

подмножество совместимо, то следует обратиться к основному входному полюсу, в противном случае — к дополнительному полюсу *3*. Если же реализуется дополнительный выход β, то в этом случае представляемое переменной γ подмножество нужно проверить на максимальность, сравнив его с ранее полученными максимальными совместимыми подмножествами. Эта проверка (с запоминанием данного значения переменной γ, если оно будет соответствовать максимальному совместимому подмножеству) будет производиться подпрограммой в к л ю м а к, при выходе из которой следует обращаться к дополнительному полюсу *2* подпрограммы п е р е б о р.

Таково внешнее описание подпрограммы. Внутренняя же ее структура полностью определяется описанным выше алгоритмом перебора, имея следующий вид:

§ 0

§ 1 $\delta \circ\!\!\rightarrow \beta \overline{\triangle}\, \delta c_\delta \vee \gamma \Rightarrow \gamma \rightarrow \alpha$

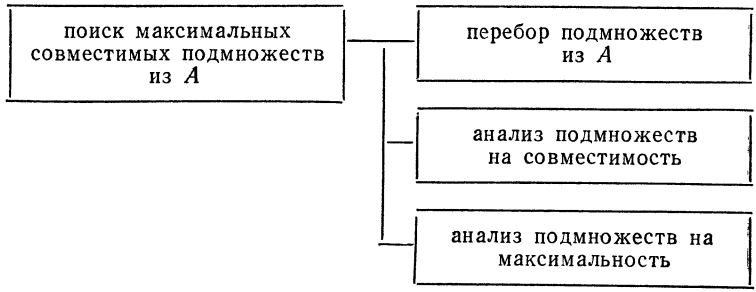
§ 2 $\gamma \urcorner \vdash \Rightarrow \delta c_\delta - 1 \wedge \gamma \Rightarrow \gamma \circ\!\!\rightarrow 4 \vdash \Rightarrow \delta$

§ 3 $c_\delta \oplus \gamma \Rightarrow \gamma \rightarrow 1$

§ 4 .

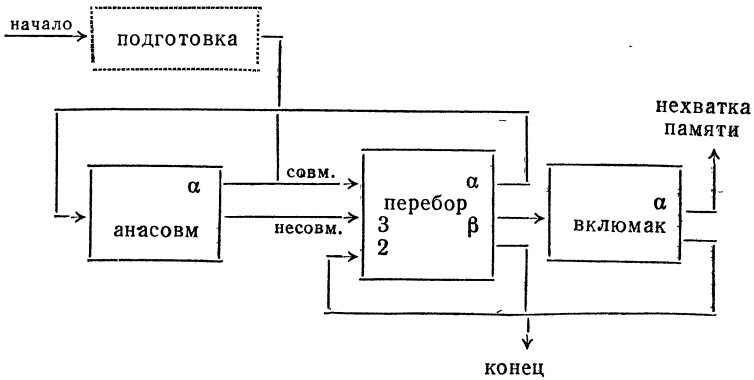
При обращении к основному входному полюсу подпрограммы (§ 0) к текущему значению переменной γ добавляется слева единица к крайней левой из уже имеющихся единиц (в самом начале перебора ставится единица в $(k-1)$-й разряд, где $k = \sigma(A)$) и совершается выход через полюс α. Если же при этом оказывается, что в предшествующем значении переменной γ 0-я компонента уже имела значение 1 (в этом случае индекс δ будет иметь значение 0), совершается выход через полюс β. При обращении к полюсу *3* производится то же и, кроме того, отбрасывается левая из уже имеющихся единиц. При обращении к полюсу *2* из значения переменной γ удаляются все единицы, левее которых нет нулей. Если после этого в значении γ не останется единиц, то совершается выход через основной полюс (что соответствует окончанию перебора), в противном случае над полученным вектором производятся те же операции, что и при обращении к полюсу *3*.

Подпрограмма максопод. Таким образом, задача нахождения всех максимальных совместимых подмножеств из A разлагается по следующей схеме:



Нами составлены подпрограммы решения задач, образующих базу разложения. Осталось разработать подпрограмму для решения разлагаемой задачи. Назовем ее максопод.

Центральным блоком этой подпрограммы является подпрограмма перебор, связывающая остальные блоки в единое целое. Анализ ее описания приводит нас к следующей схеме взаимодействия блоков, задающей структуру подпрограммы максопод:



Функции блока «подготовка» довольно просты: он должен подготовить исходные значения некоторых операндов к началу перебора подмножеств из A. Поэтому этот блок не оформляется в виде отдельной подпрограм-

мы, а будет представлен начальным предложением подпрограммы максопод.

Определим внешние операнды разрабатываемой подпрограммы. Пусть комплекс β представляет булеву матрицу $\boldsymbol{F} = \|A * A\|$, задающую функцию несовместимости на множестве всех пар элементов множества A, а комплекс γ предназначается для представления получаемого результата — множества P всех максимальных совместимых подмножеств из A в виде булевой матрицы $\|P \ni A\|$. На операнд α возложим функцию дополнительного выходного полюса подпрограммы, реализуемого в случае нехватки памяти, отводимой для представления результата. Положим, что объем этой памяти должен определяться заранее, путем присвоения максимально допустимого значения мощности комплекса γ элементу b_γ.

Теперь уже нетрудно найти формальное выражение данной подпрограммы:

§ 0 $\circ \boldsymbol{c}\, b_\gamma \Rightarrow b \circ b_\gamma\, b_\beta \Rightarrow c \rightarrow 4$

§ 1 * анасовм 4 β **c** // → * перебор 3

§ 2 * включмак 3 γ **c** *b* // → * перебор 2

§ 3 → α

§ 4 * перебор 1 2 **c** *c* //.

В качестве внутренних операндов здесь взяты переменная **c** и индексы *b* и *c*, поскольку операнды ***a***, ***b*** и *a* использованы в подчиненных подпрограммах.

Следует помнить, что если какой-либо внешний операнд подпрограммы играет роль дополнительного выходного полюса, то он не должен встречаться в перечнях операндов Л-операторов, входящих в выражение рассматриваемой подпрограммы. Например, было бы неправильно поместить символ α дополнительного выходного полюса подпрограммы максопод на первом месте перечня Л-оператора включмак, не вводя, таким образом, специального предложения, состоящего исключительно из операции → α.

Введение этого правила обусловлено особенностями компиляции, при которой производится перенумерация предложений подпрограммы при их «вписывании» в синтезируемую на первом уровне программу. Его аннулирование могло бы стать источником допускаемых

при компиляции ошибок или сильно усложнило бы процесс компиляции.

Программа решения задачи в целом. Возвратимся к рассмотрению первоначальной задачи нахождения минимального разбиения множества A на совместимые подмножества и составим программу решения этой задачи, оформив ее как подпрограмму, называемую расопод.

Положим, что внешний операнд α этой подпрограммы так же, как и в подпрограммах максопод и включмак, играет роль дополнительного выходного полюса, реализуемого при нехватке памяти. Комплекс β пусть представляет исходную информацию к решаемой задаче, заданную матрицей $F = \|A * A\|$, а комплекс γ предназначается для представления получаемого результата, оформляемого в виде булевой матрицы $\|Q \ni A\|$, где Q — искомое разбиение множества A, то есть такое множество его совместимых и взаимно непересекающихся подмножеств, которое покрывает множество A. Поскольку этот результат будет получен лишь в конце реализации программы, до этого комплекс γ может быть использован для фиксации промежуточных результатов, а именно, для представления матрицы $\|P \ni A\|$. Максимально допустимая мощность комплекса задается заранее значением элемента b_γ.

Кроме Л-операторов максопод и прикрапок, в выражение рассматриваемой подпрограммы должны входить операции согласования этих Л-операторов. Согласование производится путем вычисления значений переменных e и f, выделяющих минор в булевой матрице $\|P \ni A\|$, представленной комплексом γ после реализации подпрограммы максопод. В данном случае минор будет совпадать с матрицей.

Покрытие матрицы $\|P \ni A\|$, находимое подпрограммой прикрапок, представляется значением переменной g, выделяющей из данной матрицы некоторую совокупность строк. Далее организуется последовательный просмотр этих строк, в ходе которого из очередной строки удаляется единица в каждом столбце, содержащем единицу на пересечении с какой-либо из уже просмотренных строк. Эта операция отражает устранение взаимного пересечения входящих в найденное покрытие

подмножеств из A и приводит к получению искомого разбиения Q.

```
§ 0  * максопод 2 β γ //
       b_γ − 1 ⇒ a 0 − c_a ⇒ e
       b_β − 1 ⇒ a 0 − c_a ⇒ f
       * прикрапок γ e fg // ○ b ○ a
§ 1  g X̊ 3 a a ⌝ ∧ γ_a ⇒ γ_b ∨ a ⇒ a △ b → 1
§ 2  → α
§ 3  b ⇒ b_γ.
```

Полная совокупность подпрограмм, использованных при решении рассматриваемой задачи, образует следующую иерархическую структуру:

Вернемся, однако, к рассмотрению исходной задачи о размещении отдыхающих по лодкам. Выше уже было показано, как в ходе решения этой задачи будет получена булева матрица

$$\|P \ni A\| = \begin{array}{c} \begin{matrix} 1&2&3&4&5&6&7&8&9&10 \end{matrix} \\ \left[\begin{array}{cccccccccc} 1&0&1&0&1&0&0&1&0&1 \\ 1&0&1&1&0&0&0&1&0&1 \\ 1&0&1&0&1&0&1&0&0&1 \\ 1&0&1&1&0&0&1&0&0&1 \\ 1&0&1&0&1&0&0&1&1&0 \\ 1&0&1&0&0&1&0&0&1&0 \\ 0&1&1&0&1&0&0&1&0&0 \\ 0&1&1&0&0&0&1&1&0&0 \\ 1&0&1&0&0&1&1&0&0&0 \\ 0&1&1&0&1&0&1&0&0&0 \end{array}\right] \end{array} \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \\ j \end{matrix}$$

представляющая множество P всех максимальных совместимых подмножеств из A. Этот промежуточный результат вычисляется подпрограммой максопод

и служит исходной информацией для подпрограммы прикрапок.

Подпрограмма прикрапок начинает свою работу с выбора столбца 4 и строки b, содержащей единицу в данном столбце. Эта строка принимается за первый элемент покрытия. Последующий процесс чередования находимых минимальных столбцов непокрытого остатка матрицы с выбираемыми максимальными строками, содержащими единицу в найденном столбце, представляется последовательностью $9\,e\,2\,h$. Совокупность выбранных строк образует матрицу

$$\begin{matrix} & 1\,2\,3\,4\,5\,6\,7\,8\,9\,10 \\ \left[\begin{matrix} 1\,0\,1\,1\,0\,0\,0\,1\,0\,1 \\ 1\,0\,1\,0\,1\,0\,0\,1\,1\,0 \\ 0\,1\,1\,0\,0\,1\,1\,0\,0\,0 \end{matrix}\right] & \begin{matrix} b \\ e \\ h \end{matrix} \end{matrix}$$

Наконец, устранение взаимного пересечения представляемых строками этой матрицы подмножеств из A (эта операция выполняется предложением 1 подпрограммы расопод) приводит к окончательному результату — к матрице

$$\| Q \ni A \| = \begin{matrix} 1\,2\,3\,4\,5\,6\,7\,8\,9\,10 \\ \left[\begin{matrix} 1\,0\,1\,1\,0\,0\,0\,1\,0\,1 \\ 0\,0\,0\,0\,1\,0\,0\,0\,1\,0 \\ 0\,1\,0\,0\,0\,1\,1\,0\,0\,0 \end{matrix}\right] \end{matrix}$$

Таким образом, множество A разбивается на подмножества $\{a_1, a_3, a_4, a_8, a_{10}\}$, $\{a_5, a_9\}$ и $\{a_2, a_6, a_7\}$, каждое из которых состоит лишь из взаимно совместимых элементов.

Обсуждение составленной программы. Редко бывает, чтобы составленная однажды программа решения некоторой задачи не могла быть впоследствии существенно улучшена. Совершенствование программы должно начинаться с ее анализа, особенно тщательного при рассмотрении «узких мест», и выявления различных недостатков в программе. Полученная при этом информация используется затем для нахождения разумных путей, приводящих к более или менее широкой «перекройке» программы, которая может начинаться с первичного разложения задачи и кончаться программированием заново простейших вычислительных процессов.

Переделка может коснуться некоторых из используемых подпрограмм, другие же могут остаться нетронутыми.

Своими недостатками обладает и подпрограмма расопод. Перечислим наиболее существенные из этих недостатков.

Во-первых, эта подпрограмма не гарантирует точного решения, то есть не гарантирует получение действительно *минимального* разбиения множества A на совместимые подмножества (хотя в рассмотренном примере решение оказалось точным). Причиной этого является использование подпрограммы прикрапок, находящей лишь некоторые приближения к кратчайшим покрытиям булевой матрицы $\|P \ni A\|$. Возможно, что в связи с этим не совсем оправдано и применение здесь подпрограммы максопод, которая находит все максимальные совместимые подмножества из A: получаемая при этом полная информация, характеризующая исходную постановку задачи в удобной для последующего использования форме, используется потом далеко не полностью.

Можно, конечно, заменить подпрограмму прикрапок другой подпрограммой, гарантирующей получение кратчайшего покрытия булевой матрицы $\|P \ni A\|$ (такая подпрограмма будет описана в дальнейшем). Возможно, однако, что такая замена приведет к значительному увеличению времени, затрачиваемого на решение задачи, что не всегда приемлемо.

Другой из существенных недостатков обсуждаемой программы также тесно связан с использованием в ней подпрограммы прикрапок. Дело в том, что в своей непосредственной форме эта подпрограмма может оперировать лишь с такими булевыми матрицами, у которых как число строк, так и число столбцов не превышает 32. Это накладывает существенные ограничения на сложность исходной постановки задачи, поскольку число столбцов в указанных матрицах совпадает в данном случае с мощностью множества A, а число строк — с мощностью получаемого при реализации подпрограммы максопод множества P.

Снять эти ограничения можно, использовав описанный в § 7 механизм повышения размерности операндов и пересмотрев подпрограмму под соответствующим уг-

лом зрения. Следует, однако, учитывать возможность резкого снижения в связи с этим быстродействия программы. Если затрагиваемая подпрограмма относится к числу «узких мест» в программе, то лучше искать другие пути расширения диапазона ее действия.

В данном случае мы столкнулись лишь с одним из типов ограничений, вытекающим из способа кодирования перерабатываемой информации. В более сложных случаях «потолок» действия программы, выражаемый в предельных значениях некоторых параметров, характеризующих сложность решаемой задачи, можно оценить лишь путем постановки эксперимента на вычислительной машине.

§ 10. Оформление подпрограмм

Стандартизация формы представления подпрограмм. Как это было показано в предыдущем параграфе, сущность программирования на втором уровне языка ЛЯПАС заключается в разложении алгоритма на некоторые части с самостоятельным значением и в организации их взаимодействия. Совокупность связей между вводимыми при этом Л-операторами может иметь сложную многоступенчатую структуру, благодаря чему данное программирование и названо иерархическим. Основные преимущества этого типа программирования заключаются в том, что, во-первых, задача составления некоторой сложной программы заменяется задачей составления серии более простых программ и что, во-вторых, при этом можно использовать некоторые из ранее отработанных алгоритмов, оформленных в виде подпрограмм в языке ЛЯПАС и образующих в совокупности так называемую *эльтеку*. При этом достаточно ознакомиться лишь с внешними характеристиками используемых подпрограмм и совершенно необязательно вникать в детали их внутренней структуры.

Естественно, что эффективность иерархического программирования повышается с ростом эльтеки, в которую включаются все новые и новые программы, оформленные должным образом. Естественно также, что пользование эльтекой может быть существенно облегчено при стандартизации формы представления подпрограмм.

Существуют две основные формы представления содержимого эльтеки: *эльтека описаний*, содержащая полные описания всех подпрограмм в таком виде, который годится для публикаций, и *машинная эльтека*, представленная в форме, непосредственно воспринимаемой вычислительной машиной, снабженной системой автоматического программирования. Здесь мы ограничимся рассмотрением первой из этих форм.

В основе предлагаемой ниже формы описания подпрограммы лежит идея разложения описания на две части, одна из которых (постановка задачи и внешнее описание решающей ее подпрограммы) содержит лишь минимальную информацию, необходимую для организации правильного использования подпрограммы, другая же содержит информацию о методе решения задачи и о внутренней структуре решающего алгоритма.

Описание в целом состоит из следующих пунктов.

Полное название подпрограммы и ее сокращенное наименование.

1. Постановка решаемой задачи.
2. Внешнее описание подпрограммы.
3. Неформальное описание алгоритма.
4. Л-программа.
5. Описание контрольного примера.

Уточним содержание отдельных пунктов описания, начав со второго, поскольку суть первого пункта, по-видимому, достаточно ясна.

Внешнее описание подпрограммы. Заполняя пункт 2, мы должны стать в позицию возможного пользователя данной подпрограммы и рассматривать эту подпрограмму, как «черный ящик», описывая лишь вход и выход этого ящика.

Прежде всего, должен быть выяснен смысл всех внешних операндов подпрограммы в той терминологии, которая была использована при постановке задачи. Должна быть дана интерпретация значений, принимаемых этими операндами как перед, так и непосредственно после реализации подпрограммы. Поскольку этими значениями служат булевы векторы и булевы матрицы, с успехом могут быть использованы те два способа интерпретации, которые были рассмотрены в § 1. Дополним их следующими правилами.

А. Тип внешнего операнда конкретизируется следующими нижними индексами: п — переменная, и — индекс, к — комплекс, ч — натуральная константа (число), о — оператор, в — некоторое сложное выражение в языке ЛЯПАС (использование операнда такого типа производится лишь в исключительных случаях). Как правило, величины, представляемые переменными и индексами, могут представляться также элементами комплексов, что будет в дальнейшем подразумеваться.

Б. Фиксация момента времени, к которому относится рассматриваемое значение операнда, производится с помощью верхних индексов — и +. Через α^- представляется значение операнда α в момент времени, непосредственно предшествующий процессу реализации рассматриваемой подпрограммы, через α^+ представляется значение этого же операнда в момент окончания процесса реализации подпрограммы, а отсутствие верхнего индекса будет говорить о том, что представляется значение внешнего операнда, не изменяющееся при реализации данной подпрограммы.

В. Напомним, что, оперируя с булевыми векторами, размерность которых меньше чем 32, мы условились представлять их левыми компонентами операндов стандартной размерности, считая, что неиспользуемые компоненты данных операндов принимают значение 0. При описании смысла операндов в этом случае используется символ ::, например, выражение $\alpha_п :: 1011$ означает, что переменная α принимает значение 10110...0, а выражение $\beta_к :: \boldsymbol{D}$, где $\boldsymbol{D}$ — символ некоторой булевой матрицы, означает, что левыми компонентами элементов комплекса β служат компоненты соответствующих строк матрицы $\boldsymbol{D}$, взятые в аналогичном порядке. Заметим, что символ :: может быть использован, в частности, и при совпадении числа компонент операнда и представляемого вектора, в то время как символ = может использоваться только в этом последнем случае.

Мощности некоторых комплексов, являющихся внешними операндами, и их «координаты» в оперативной памяти должны быть определены уже к моменту начала реализации подпрограммы, для других же комплексов это необязательно. Поэтому следует перечислить те из элементов специальных комплексов A и B, значения

которых должны быть заданы в качестве исходных данных.

Следует перечислить также группы таких внешних операндов, размерности которых должны совпадать. Разумеется, это относится лишь к тем операндам, размерность которых может повышаться.

Может оказаться, что некоторые из операндов, несущих исходную информацию для решаемой задачи, могут быть использованы также для представления результатов вычислений. Наличие такой возможности представляет интерес с точки зрения экономии числа используемых операндов и объема оперативной памяти, особенно в том случае, когда дело касается комплексов. В связи с этим условимся перечислять в пункте 2 группы совместимых в данном смысле операндов, считая, например, что выражение

совмещение: $\alpha\,\delta$, $\beta\,\varepsilon$

означает, что для представления величин, задаваемых в данной подпрограмме операндами α и δ, достаточно использовать лишь один операнд такого же типа и что таким же образом могут быть совмещены операнды β и ε.

Далее идет перечень всех подпрограмм, непосредственно подчиненных данной подпрограмме.

В этом же пункте должны быть указаны последние из переменных, индексов и комплексов, использованных в качестве внутренних операндов рассматриваемой подпрограммы. Например, перечень ***d***, *b*, ***X*** будет в данном случае означать, что в качестве внутренних операндов подпрограммы использованы переменные ***a***, ***b***, ***c*** и ***d***, индексы *a* и *b* и комплексы ***Z***, ***Y*** и ***X*** (напомним, что порядок использования комплексов как внутренних операндов подпрограмм определяется перебором букв с конца латинского алфавита); перечень же —, *a*, — означает, что в подпрограмме использован лишь индекс *a*, а переменные и комплексы в качестве внутренних операндов данной подпрограммы не используются. Следует учитывать, что совокупность внутренних операндов подпрограммы содержит и все внутренние операнды подчиненных подпрограмм.

Вся остальная информация, знание которой необходимо для правильного использования подпрограммы, офор-

мляется в виде примечаний. В частности, здесь могут быть сообщены ограничения на некоторые параметры исходных данных, определяющие «потолок» алгоритма.

Л-программа. Сама подпрограмма, в символах ЛЯПАСа, приводится в пункте 4. Она состоит из *тела*, то есть основного текста, порядок составления которого рассматривался нами до сих пор, и открывается *паспортом*, содержащим информацию, необходимую для работы системы автоматического программирования.

В начале паспорта стоит символ *, сопровождаемый номером, который присваивается данной подпрограмме и записывается в виде пары трехразрядных восьмеричных чисел (первое из которых может выбираться в диапазоне от 001 до 177, а второе — от 001 до 776). Чтобы не путать эти числа с натуральными константами, условимся выделять их жирным шрифтом (или подчеркивать — в рукописном тексте) и называть *непосредственными константами*. Различным подпрограммам должны присваиваться различные номера.

Далее в паспорте может следовать (но не обязательно) информация, необходимая для осуществления штриховки внутренних операндов и задаваемая в форме, описанной в параграфе 8.

Например, паспорт

*** 001 053** $(\beta \boldsymbol{a})\,(\gamma \boldsymbol{cd})$

принадлежит подпрограмме, имеющей номер 001 053. При штриховке внешнего операнда β компилятор должен обеспечить аналогичную штриховку переменной $\boldsymbol{a}$, при штриховке γ — штриховку переменных $\boldsymbol{c}$ и $\boldsymbol{d}$. Например, если на третьем месте в перечне операндов реализуемого Л-оператора стоит $''\boldsymbol{m}$, то всюду в обрабатываемой компилятором подпрограмме перед символами $\boldsymbol{c}$ и $\boldsymbol{d}$ будет поставлен символ $''$.

Прочие пункты описания. В пункте 3 описывается суть алгоритма, реализуемого данной подпрограммой. Изложение здесь может вестись «вольным стилем», но так, чтобы оно было согласовано с введенными при постановке задачи понятиями и символикой. Здесь же может проясняться и смысл, придаваемый некоторым из внутренних операндов, используемых для представления промежуточной информации.

Пункт 5 отводится для описания примера, иллюстрирующего работу алгоритма и дополняющего тем самым описание. С другой стороны, данный пример является контрольным: он должен обязательно быть решен на машине с помощью описываемой подпрограммы, свидетельствуя, таким образом, об отсутствии ошибок в последней.

Приведем примеры оформленных по описанной схеме подпрограмм, как правило, уже знакомых нам по предыдущим параграфам.

Пример: подпрограмма прикрапок.

Получение кратчайшего покрытия (приближенный алгоритм) — прикрапок.

1. На булевой матрице A задан минор, образованный пересечением подмножества B из множества P строк матрицы с подмножеством C из множества S ее столбцов. Требуется найти одно из покрытий данного минора, то есть такую совокупность Q его строк, которая содержит по крайней мере одну единицу в каждом из столбцов минора. При этом желательно, чтобы найденное покрытие содержало по возможности меньшее число строк.

2. Внешние операнды:

$$\alpha_к :: A,$$
$$\beta_п^- :: \| \{B\} \Rightarrow P \|,$$
$$\gamma_п^- :: \| \{C\} \Rightarrow S \|,$$
$$\delta_п^+ :: \| \{Q\} \Rightarrow P \|.$$

Задаются: a_α, b_α.

Размерность: $\alpha\gamma$, $\beta\delta$.

Подпрограммы: минсто, макстро.

Внутренние операнды: d, d, —.

Примечание: предполагается, что покрытие существует.

3. Находится минимальный (по числу единиц) столбец минора, затем из строк минора, содержащих 1 в найденном столбце, выбирается максимальная и удаляется из минора вместе с покрываемыми ею столбцами. Эта совокупность операций повторяется до тех пор, пока множество столбцов минора не станет пустым, после чего совокупность Q выбранных описанным образом строк выдается в качестве решения.

4.

```
        * 001 010 (βd)
  ¬§ 0   ○δ
    § 1   * минсто α β γ d // * макстро α d γ d //
          δ ∨ c_d ⇒ δ ¬ ∧ β ⇒ β α_d ¬ ∧ γ ⇒ γ ⟼ 1.
```

5. Пусть

$$\begin{array}{c} \quad 1\,2\,3\,4\,5\,6\,7\,8\,9\,10 \\ A = \left|\begin{array}{c} 1\,0\,1\,1\,0\,0\,0\,1\,0\,1 \\ 0\,1\,0\,1\,0\,1\,0\,0\,1\,0 \\ 1\,1\,0\,0\,0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,0\,1\,0\,0\,0\,1 \\ 0\,1\,0\,0\,1\,0\,1\,1\,0\,0 \\ 1\,0\,1\,0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,1\,0\,1\,0 \\ 0\,1\,0\,1\,1\,0\,0\,1\,0\,1 \end{array}\right| \begin{array}{l} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array} \end{array}$$

$$\|\{B\} \ni P\| = 1\,1\,0\,1\,0\,1\,1\,1\,1,$$

$$\|\{C\} \ni S\| = 1\,1\,0\,1\,1\,0\,1\,0\,1\,1.$$

Тогда описанный алгоритм выбирает следующую последовательность чередующихся столбцов и строк:

$$1a\ 7f\ 9b,$$

находя таким образом покрытие $\{a, b, f\}$ и представляя его значением переменной δ:

$$\delta_{\text{п}}^{+} :: 1\,1\,0\,0\,0\,1\,0\,0\,0.$$

Заметим, что при наличии нескольких минимальных по числу единиц столбцов Л-оператор минсто выбирает левый из них, в то время как Л-оператор макстро всегда останавливает выбор на верхней из максимальных строк.

Пример: подпрограмма профакт. Рассмотрение следующей подпрограммы представляет интерес в связи с особым способом представления исходной информации, при котором совокупность натуральных чисел задается значением одной переменной.

Подсчет произведения факториалов — профакт

1. Подсчитывается произведение факториалов $N = \prod_{i=1}^{m} (n_i!)$ для заданных натуральных чисел $n_1, n_2, \ldots, n_m$.

2. Внешние операнды:

α_n — единичные значения компонент этой переменной служат правыми границами участков, мощности которых равны числам $n_1, n_2, \ldots, n_m$, например если

1 2 3 1 2 1 2 3 4 5 1 2 3 4

α::[0 0 1 0 1 0 0 0 0 1 0 0 0 1]

(сверху показана нумерация компонент, своя в каждом из участков), то это означает, что $n_1 = 3$, $n_2 = 2$, $n_3 = 5$ и $n_4 = 4$.

$[\beta_{\text{и}}^{+}] = N$.

Внутренние операнды: —, b, —.

Примечание: значения исходных чисел $n_1, n_2, \ldots, n_m$ ограничиваются в совокупности стандартной размерностью переменной α: $\sum_{i=1}^{m} n_i \leqslant 32$.

3. Значение переменной α просматривается поразрядно справа налево, начиная с крайней правой единицы, и реализуется перемножение последовательности натуральных чисел

$$2 \cdot 3 \cdot \ldots \cdot n_m \cdot 2 \cdot 3 \cdot \ldots \cdot n_{m-1} \cdot \ldots \cdot 2 \cdot 3 \cdot \ldots \cdot n_1.$$

4.

* **001 105**

§ 0 $\alpha \urcorner + 1 \wedge \alpha \vdash \Rightarrow a\,1 \Rightarrow \beta$

§ 1 $1 \Rightarrow b$

§ 2 $a \circ\!\!\rightarrow 3\ \overline{\triangle}\ a\alpha \wedge c_a \mapsto 1 \triangle b \times \beta \Rightarrow \beta \rightarrow 2$

§ 3 .

5. Пусть α::0 0 1 0 1 0 0 0 0 1 0 0 0 1. В этом случае будет реализовано перемножение следующей последовательности натуральных чисел:

$$2 \cdot 3 \cdot 4 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 2 \cdot 2 \cdot 3$$

и результат данной операции окажется представленным значением индекса β:

$$[\beta^{+}] = 34560.$$

Оформление других подпрограмм. Представим в описанной форме некоторые из других знакомых нам подпрограмм, ограничиваясь заполнением пунктов 1, 2 и 4, поскольку эти подпрограммы были уже достаточно подробно рассмотрены. В тех случаях, когда подпрограммы весьма просты, допускается объединение первых пунктов.

Нахождение минимального столбца минора — минсто

1. На булевой матрице $\boldsymbol{A}$ задан минор, образуемый пересечением подмножества B из множества P строк матрицы и подмножества C из множества S ее столбцов. Требуется найти минимальный по числу единиц столбец этого минора.

2. Внешние операнды:

$\alpha_{к} :: \boldsymbol{A}$,

$\beta_{п} :: \| \{B\} \ni P \|$,

$\gamma_{п} :: \| \{C\} \ni S \|$,

$\delta_{п}^{+}$ — найденный и представленный в транспонированном виде минимальный столбец минора; если таких столбцов несколько, то выбирается левый из них.

Задается: a_{α}.

Размерность: αγ, βδ.

Внутренние операнды: $\boldsymbol{c}$, $\boldsymbol{d}$, —

4.

$$* \; 001 \; 002 \; (\beta \, \boldsymbol{a} \, \boldsymbol{c}) (\gamma \, \boldsymbol{b})$$

§ 0 $\gamma \Rightarrow \boldsymbol{b} \, \bar{\circ} \, d$

§ 1 $\boldsymbol{b} \dot{\mathrm{X}} \, 4 \, b \, \beta \Rightarrow \boldsymbol{a} \circ c \circ \boldsymbol{c}$

§ 2 $\boldsymbol{a} \dot{\mathrm{X}} \, 3 \, a \, \alpha_{a} \wedge c_{b} \circ\!\!\rightarrow 2 \triangle c \, c_{a} \vee \boldsymbol{c} \Rightarrow \boldsymbol{c} \rightarrow 2$

§ 3 $c - d \mapsto 1 \, c \Rightarrow d \, \boldsymbol{c} \Rightarrow \delta \rightarrow 1$

§ 4 .

Нахождение максимальной строки минора — максстро

1. На булевой матрице A задан минор, образуемый пересечением подмножества B из множества P строк матрицы и подмножества C из множества S ее столбцов. Требуется найти максимальную по числу единиц строку минора.

2. Внешние операнды:

$\alpha_к :: A$,

$\beta_п :: \| \{B\} \ni P \|$,

$\gamma_п :: \| \{C\} \ni S \|$,

$[\delta_и^+] = i$, где i — номер найденной строки в матрице A; если максимальных строк несколько, выбирается первая из них.

Задается: a_α.
Размерность: $\alpha\gamma$.
Внутренние операнды: a, c, —.

4.

* 001 003 (βa)

§ 0 $\circ b\, \beta \Rightarrow a \vdash \Rightarrow \delta$

§ 1 $a \dot{X} 2\, a\, a_a \wedge \gamma \bigtriangledown \Rightarrow c\, b - c \mapsto 1\, a \Rightarrow \delta\, c \Rightarrow b \to 1$

§ 2 .

Включение в комплекс максимального элемента — включмак

1—2. Подпрограмма включмак включает значение переменной γ в комплекс β в том и только в том случае, когда комплекс β не содержит элементов, покрывающих значение переменной γ (будем говорить, что булев вектор a покрывает булев вектор b, если все компоненты вектора a, соответствующие единичным компонентам вектора b, также имеют значение 1). При включении нового элемента мощность комплекса β возрастает на единицу. Максимально допустимая мощность этого комплекса задается значением индекса δ. При достижении этого значения совершается выход через дополнительный полюс α.

Задаются: a_β, b_β.
Размерность: $\beta\gamma$.
Внутренние операнды: —, a, —.

4.

* **001 400**

§ 0 $\bar{\circ} a$

§ 1 $\triangle a \oplus b_\beta \circ\!\!\rightarrow 2\beta_a \urcorner \wedge \gamma \mapsto 1 \rightarrow 3$

§ 2 $\gamma \Rightarrow \beta_a \triangle b_\beta \oplus \delta \circ\!\!\rightarrow \alpha$

§ 3 .

Перебор элементов выпуклого множества — перебор

1—2. Оператор перебор осуществляет по определенным правилам перебор значений переменной γ, который можно трактовать как перебор подмножеств некоторого множества, поставленного во взаимно однозначное соответствие совокупности левых k компонент переменной γ, где k задается начальным значением индекса δ. Этот оператор оказывается полезным при поиске верхней границы некоторых подмножеств, выпуклых по отношению включения, когда этот поиск производится методом анализа отдельных элементов заданного множества на наличие у них некоторого свойства, выделяющего упомянутое выпуклое подмножество. Здесь *выпуклым* называется такое подмножество A_i множества A, которое обладает следующим свойством: если $a_j \subseteq a_k$ и $a_k \in A_i$, то $a_j \in A_i$. Описание порядка перебора опустим (оно было дано ранее).

4.

* **001 403**

§ 0

§ 1 $\delta \circ\!\!\rightarrow \beta \overline{\triangle} \delta c_\delta \vee \gamma \Rightarrow \gamma \rightarrow \alpha$

§ 2 $\gamma \urcorner \vdash \Rightarrow \delta c_\delta - 1 \wedge \gamma \Rightarrow \gamma \circ\!\!\rightarrow 4 \vdash \Rightarrow \delta$

§ 3 $c_\delta \oplus \gamma \Rightarrow \gamma \rightarrow 1$

§ 4 .

Получение скалярного произведения двух комплексов — проскал

Определим скалярное произведение комплексов $\boldsymbol{A}$ и $\boldsymbol{B}$ по двуместному оператору ρ_2 как комплекс $\boldsymbol{C}$, для которого $\sigma(\boldsymbol{C}) = \min(\sigma(\boldsymbol{A}), \sigma(\boldsymbol{B}))$ и при $i \in \{0, 1, \ldots, \sigma(\boldsymbol{C}) - 1\}$ $c_i = a_i \rho_2 b_i$.

2. Внешние операнды:

β и γ — исходные комплексы,

δ — результативный комплекс,

α — двуместный оператор.

Задаются: a_β, a_γ, a_δ, b_β, b_γ.
Размерность: βγδ.
Совмещение: βδ, γδ.
Внутренние операнды: —, a, —.
4.

* **001 051**

§ 0 $\overline{\circ}\, a\, b_\beta - b_\gamma \circ\!\!\rightarrow 1\, b_\gamma \Rightarrow b_\delta \rightarrow 2$

§ 1 $b_\beta \Rightarrow b_\delta$

§ 2 $\triangle\, a \oplus b_\delta \circ\!\!\rightarrow 3\, \beta_a\, \alpha\, \gamma_a \Rightarrow \delta_a \rightarrow 2$

§ 3 .

Две формы Л-оператора. На примере приведенной в параграфе 9 подпрограммы максопод мы встретились с двумя формами упоминания об использовании во внешней программе Л-операторов. Одну из них, соответствующую обращению к основному входному полюсу подпрограммы и включающую перечисление ее внешних операндов, назовем *прямой формой*. Другую, соответствующую обращению к какому-либо из дополнительных полюсов подпрограммы и содержащую только наименование этой подпрограммы и номер предложения, к которому производится обращение (этим номером не может быть 0), назовем *косвенной формой*.

Суть компиляции, сопровождающей просмотр внешней программы слева направо, заключается в том, что всякий раз при встрече прямой формы в синтезируемую программу первого уровня вставляется подпрограмма соответствующего Л-оператора, предварительно «настроенная» на заданное в этой форме перечисление операндов. При встрече же косвенной формы действия компилятора ограничиваются программированием запланированного во внешней программе перехода к одному из дополнительных полюсов подпрограммы, набор операндов которой при этом не меняется.

Это означает, что программа, в которой имеется косвенная форма упоминания о каком-то Л-операторе, но отсутствует соответствующая прямая форма, не имеет смысла, о чем следует помнить при программировании.

Если во внешней программе употребляется несколько прямых форм одного и того же Л-оператора, то синтезированная программа первого уровня будет иметь такое же число включенных подпрограмм, реализующих этот Л-оператор и отличающихся друг от друга только тем, что они могут быть настроены на различные наборы внешних операндов.

Поэтому, если объем подпрограммы достаточно велик, то, в целях сокращения объема Л-программы первого уровня, рекомендуется ограничиваться однократным употреблением прямой формы упоминания о соответствующем Л-операторе, вынося «настройку» на его операнды во внешнюю программу и оформляя дополнительное предложение, в начале которого ставится прямая форма.

Например, программа вида

§ 0 . . . . . * альфа ***a b c*** // . . .

§ 1 . . . * альфа ***b e c*** // . . .

§ 2 . . . . . * альфа ***d a c*** //.

где а л ь ф а — некоторый Л-оператор, может быть преобразована к следующему эквивалентному виду:

§ 0 . . . . . ***a*** ⇒ ***m b*** ⇒ ***n*** ↠ 3 . . .

§ 1 . . . ***b*** ⇒ ***m e*** ⇒ ***n*** ↠ 3 . . .

§ 2 . . . . . ***d*** ⇒ ***m a*** ⇒ ***n*** ↠ 3 → 4

§ 3 * альфа ***m n c*** //!

§ 4 .

Этим же приёмом следует пользоваться и в том случае, когда Л-оператор обладает несколькими входными полюсами и в то же время обращение к его основному полюсу производится в более чем одной точке внешней программы.

Действительно, соответствующая подпрограмма (назовем ее б э т а) может быть «вписана» в каждую из этих точек, породив таким образом несколько экземпляров подпрограммы. Но дело в том, что выражение типа «→ * бэта 2», символизирующее переход к дополнитель-

ному полюсу 2 подпрограммы бэта, не говорит, какой именно экземпляр подпрограммы имеется в виду. Поэтому компилятор замкнет все переходы такого типа (то есть на дополнительные полюсы) лишь на первое вхождение подпрограммы. Но это может идти вразрез с планами составителя программы.

§ 11. Дополнительные операции и кодирование Л-программ

В данном параграфе будут рассмотрены дополнительные операции языка ЛЯПАС, вводимые для повышения эффективности использования современных вычислительных машин. Некоторые из этих операций позволяют полнее учесть особенности устройств ввода и вывода информации, другие предназначены для использования преимущественно в системе автоматического программирования.

Разрядность вычислительных машин. Известно, что число двоичных разрядов в ячейках современных УЦВМ, как правило, превышает 32. Например, в широко распространенных машинах типа М-20, М-220, БЭСМ-3М оно равно 45. Таким же числом параллельных двоичных каналов обладают электромеханические устройства ввода и вывода информации, приданные данным машинам. Наконец, столько же разрядов содержится в регистре, на котором фиксируются значения собственной переменной τ. Однако рассмотренные до сих пор операции языка ЛЯПАС позволяют использовать лишь 32 разряда, выбираемые определенным образом (например, в случае машин упомянутых типов они получаются путем отбрасывания 9 левых и 4 правых из 45 разрядов). В то же время целесообразно предусмотреть возможность использования всей разрядной сетки машины, что позволяет, во-первых, повысить скорость ввода и вывода информации и, во-вторых, повысить компактность представления информации в памяти машины.

Введем в комплекс F дополнительную константу f_{11} специального назначения: положим, что, вне зависимости от разрядности используемой машины, разрядность этой константы совпадает с разрядностью машины, причем каждая из ее компонент имеет значение 1 (в отличие

от константы f_{10}, которая определена как константа стандартной размерности, содержащая 32 единичные компоненты. При представлении этой константы в ячейке большей размерности она дополняется нулями).

Остановимся на некоторых особенностях получения последовательности значений случайной переменной *я*. Эти значения получаются специальной программой, встроенной в систему автоматического программирования и реализующей итеративный процесс перехода между соседними значениями булева вектора, представляемого в ячейке, соответствующей переменной *я* дополнительного набора. При переходе от предшествующего значения этого вектора к последующему используются все 45 компонент (для машин типа М-20) вектора, однако соответствующие значения случайной переменной *я*, принадлежащей основному набору, задаются лишь тридцатью двумя из них, выбираемыми описанным выше способом.

Это следует учитывать в тех случаях, когда по каким-либо соображениям требуется повторить реализацию некоторого участка псевдослучайного процесса. Для фиксирования начала участка в таких случаях необходимо запомнить соответствующее значение указанного 45-разрядного вектора. Фиксация же значения 32-разрядной случайной переменной *я* оказывается недостаточной для достижения поставленной цели.

Неограниченный сдвиг. Введем операцию $\leqq\pi$ *неограниченный сдвиг*, интерпретируемую следующим образом: все содержимое регистра, в котором хранится значение собственной переменной τ, подвергается сдвигу на $q=[\pi] \bmod 2^7 - 2^6$ разрядов, причем положительному значению q соответствует сдвиг влево, отрицательному — вправо, а освобождающиеся при сдвиге разряды заполняются нулями.

С помощью этой операции можно программировать обмен информацией между 32 основными разрядами и остальными разрядами регистров и ячеек памяти. Например, если число двоичных разрядов в используемой машине равно 45, если на регистре, предназначенном для фиксирования значений собственной переменной τ, «записан» 45-разрядный булев вектор

101 111 000 000 000 000 000 000 000 000 000 000 000 011 110

и если индекс c обладает значением

00 000 000 000 000 000 000 000 001 000 100,

то в результате реализации операции $\Leftarrow c$ представленный на регистре булев вектор сдвинется влево на четыре разряда:

110 000 000 000 000 000 000 000 000 000 000 000 111 100 000.

Отбросив 9 левых и 4 правых разряда, можно выяснить, какое же значение собственной переменной τ будет представлено в данном случае состоянием регистра. Оно оказывается равным

00 000 000 000 000 000 000 000 000 011 110.

Если же значение индекса c при тех же прочих условиях равно

00 000 000 000 000 000 000 000 000 110 100,

то собственная переменная τ примет значение

00 010 111 100 000 000 000 000 000 000 000.

При реализации операций присвоения информация передается по всем 45 разрядам. Это значит, что, например, если при рассмотренных условиях реализуется операция $\Leftarrow c \Rightarrow a$, то получаемый 45-разрядный булев вектор

110 000 000 000 000 000 000 000 000 000 000 000 111 100 000

или

000 000 000 000 101 111 000 000 000 000 000 000 000 000 000

заносится в ту ячейку памяти, которая предназначена для представления значений переменной a, принимающей в данном случае значение

00 000 000 000 000 000 000 000 000 011 110

или

00 010 111 100 000 000 000 000 000 000 000.

Согласование форм представления операндов. Заметим, что рассмотренная операция используется в системе автоматического программирования ЛЯПАС-70, принятой на вооружение для машин типа М-20, М-220, БЭСМ-3М и БЭСМ-4. Точнее, она используется для согласования внутренней и внешней форм представления значений операндов ЛЯПАСа. Выше уже говорилось, что при реализации выражений языка ЛЯПАС на машине значения операндов представляются теми 32 разрядами, которые получаются при удалении 9 левых и 4 правых разрядов. Именно эта форма и называется *внутренней*. Однако она не совсем удобна при представлении информации вне машины, например, при использовании печатающего и перфорирующего устройств. Поэтому вне машины операнды ЛЯПАСа представляются во *внешней форме*, при которой используются правые 32 из имеющихся разрядов. При вводе информации в машину, в порядке реализации оператора *ввод*, 45-разрядные булевы векторы сдвигаются влево на 4 разряда, при выводе (с помощью операций *печ*, *перф*, $\pi_к$ *печ* и $\pi_к$ *перф*) — вправо, также на 4 разряда.

Однако в некоторых специальных случаях целесообразно вводить и выводить информацию без какого-либо сдвига булевых векторов. В связи с этим предложим следующие операции, отличающиеся от аналогичных операций, рассмотренных ранее, только отсутствием указанного сдвига:

операция *ввод без сдвига*, порождаемая оператором *ввод'*

операция *печ'*, называемая *печать без сдвига*,

операция *перф'*, называемая *перфорация без сдвига*.

Десятичная печать. При реализации рассмотренных выше операций печати и перфорации информация выводится на печать в восьмеричной форме, то есть каждая тройка двоичных разрядов заменяется одной восьмеричной цифрой. Однако иногда более удобной оказывается операция $печ_{10}$, называемая *десятичная печать* и использующая существующие печатающие устройства, приданные вычислительным машинам рассматриваемого типа. Эта операция выполняется аналогично операции *печ'*, то есть без сдвига, отличаясь от нее способом разбивки совокупности двоичных разрядов.

При этой разбивке 45 разрядов булева вектора группируются следующим образом:

0 0 0 00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000.

Три левых разряда при печати заменяются знаками + и — (0 заменяется на +, 1 заменяется на —), следующая пара двоичных разрядов представляется одной из цифр 0, 1, 2 или 3 (согласно правилу 00—0, 01—1, 10—2 и 11—3), наконец, следующие четверки (тетрады) разрядов заменяются одной десятичной цифрой каждая, согласно следующей таблице:

0000 — 0, 0101 — 5,
0001 — 1, 0110 — 6,
0010 — 2, 0111 — 7,
0011 — 3, 1000 — 8,
0100 — 4, 1001 — 9.

Печатающие устройства рассматриваемых машин построены так, что остающимся шести вариантам тетрад не соответствуют какие-либо десятичные цифры и при десятичной печати на местах таких тетрад появляются пробелы.

Например, булевы векторы

0 0 0 00 0000 1000 0011 0000 0110 1001 0001 0100 0000 0111,
1 0 1 01 0000 0110 1111 1111 0011 0000 1000 1111 1111 0010,
0 0 0 00 1111 0001 1000 0010 0001 0101 0000 1111 0001 0110

будут выданы в следующей форме:

+ + + 00 830691407,
— + — 10 6 308 2,
+ + + 0 182150 16

(интервал между двумя левыми и остальными цифрами обязан своим происхождением особенностям конструкции печатающего устройства).

Вывод информации на АЦПУ. Информация может выводиться из машины не только в цифровой форме, но и с использованием букв и других символов. Широкое распространение получило алфавитно-цифровое печатающее устройство (АЦПУ), алфавит которого содержит 78 символов, приводимых в следующей таблице.

Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код
0	000	$_{10}$	020	А	040	Р	060	D	077
1	001	↑	021	Б	041	С	061	F	100
2	002	(	022	В	042	Т	062	G	101
3	003	)	023	Г	043	У	063	I	102
4	004	×	024	Д	044	Ф	064	J	103
5	005	=	025	Е	045	Х	065	L	104
6	006	;	026	Ж	046	Ц	066	N	105
7	007	[	027	З	047	Ч	067	Q	106
8	010	]	030	И	050	Ш	070	R	107
9	011	*	031	Й	051	Щ	071	S	110
+	012	'	032	К	052	Ы	072	U	111
—	013	'	033	Л	053	Ь	073	V	112
/	014	≠	034	М	054	Э	074	W	113
,	015	<	035	Н	055	Ю	075	Z	114
.	016	>	036	О	056	Я	076	⁻	115
	017	:	037	П	057				

Вывод информации на АЦПУ осуществляется посредством операции π_0 *ацпу*, которая называется *вывод на АЦПУ* и реализует «распечатку» в 128-разрядные строки комплекса π_0, в каждом элементе которого (45-разрядном булевом векторе) содержится пять 9-разрядных двоичных кодов, представляющих перечисленные в таблице символы или относящихся к следующим управляющим кодам:

144 — признак адреса; после этого кода показывается адрес (от 000 до 177_8) занесения в строку следующего символа,

150 — признак транспорта, стимулирующий сдвиг бумажной ленты, на которую производится распечатка; следующий код (от 000 до 177_8) указывает количество интервалов (уменьшенное на единицу) прогона бумаги,

160 — признак запрета транспорта, позволяющий блокировать работу транспортного механизма, автоматически срабатывающего при заполнении очередной строки.

Для более подробного ознакомления с особенностями работы АЦПУ отошлем читателя к книге [10].

Прочие операции. Исключительно для упрощения контроля правильности работы машины вводится операция *контрольное суммирование*, обозначаемая через *сумма* $\xi_1\xi_2$, где ξ_1, $\xi_2 \in \{\pi_{\text{ич}}\}$. При реализации этой операции значение 45-разрядного булева вектора, находящееся на регистре, посредством специальной машинной операции циклического сложения (детали этой операции для нас несущественны) суммируется со значениями также 45-разрядных булевых векторов, расположенных в $[\xi_2]$ соседних ячейках ОЗУ, начиная с ячейки номер $[\xi_1]$. Значение полученной суммы остается на регистре. Эта операция может быть использована для организации двойного просчета какого-либо участка программы, а также для контроля правильности реализации операций обмена с внешней памятью. При этом следует учитывать, что после реализации операций *зап* $\xi_1\xi_2\xi_3$ и *счит* $\xi_1\xi_2\xi_3$ на регистре фиксируется полученная аналогичным образом 45-разрядная контрольная сумма переписанной совокупности векторов.

Предусматривая возможность использования программ, составленных непосредственно в машинном языке, введем операцию *переход к машинному языку*, представляемую символически как *маш* $\pi_{\text{ич}}$. При выполнении этой операции осуществляется переход к реализации некоторой программы на машинном языке, первая из команд которой представляется в ячейке ОЗУ, имеющей номер $[\pi_{\text{ич}}]$. В то же время переменная $\tilde{\omega}$ из дополнительного набора принимает значение номера той ячейки ОЗУ, с которой начинается участок машинной программы, соответствующей выражению, стоящему непосредственно вслед за символами *маш* $\pi_{\text{ич}}$. Этот номер может быть использован для организации возвращения к указанному участку программы.

Наконец, рассмотрим операцию *локализация*, обозначаемую через *локал* $\pi_{\text{и}}$ и позволяющую организовывать взаимодействие различных машинных программ — продуктов многократного применения транслятора — при решении некоторых сложных задач. Эта операция в принципе отлична от остальных операций первого уровня: она реализуется однократно, а именно транслятором, при трансляции Л-программы первого уровня на машинный язык. В результате реализации операции *локал* $\pi_{\text{и}}$

значением операнда $\pi_и$ становится номер ячейки ОЗУ, содержащей команду, с которой начинается машинное представление следующего за *локал* $\pi_и$ выражения.

О структуре дополнительной памяти. Вычислительная машина может обладать несколькими магнитными барабанами и лентами, на которых реализуется дополнительная память машины. Сквозная нумерация ячеек этой памяти, как правило, отсутствует. Чтобы обратиться к некоторой определенной ячейке дополнительной памяти, необходимо указать тип запоминающего устройства (барабан или лента) и его номер, а также указать номер ячейки в выбранном конкретном устройстве. В ленточных запоминающих устройствах предусматривается возможность оперирования не с отдельными ячейками, а лишь с такими их группами, которые расположены в начале зон, на которые заранее разбивается лента.

Лишь две операции первого уровня ЛЯПАСа имеют дело с дополнительной памятью, а именно, операции *зап* $\xi_1\xi_2\xi_3$ и *счит* $\xi_1\xi_2\xi_3$, в которых значением операнда ξ_3 задается номер некоторого элемента внешнего комплекса, то есть номер некоторой ячейки дополнительной памяти. Введем следующий способ представления этого номера значением булева вектора $\xi = (\xi^0, \xi^1, \ldots, \xi^{31})$: компонента ξ^0 конкретизирует тип устройства (0 — признак ленты, 1 — признак барабана), компоненты $(\xi^8, \xi^9, \ldots, \xi^{13})$ задают номер устройства (ленты или барабана), а компоненты $\xi^{14}, \xi^{15}, \ldots, \xi^{31}$ представляют номер ячейки (на барабане) или номер зоны (на ленте).

В случае ленты предполагается, что значение ξ_3 указывает на начальную ячейку заданной зоны *).

Например, значение

$$\xi_3 = 00\,000\,000\,000\,011\,000\,000\,000\,000\,001\,111$$

указывает на начальную ячейку 17-й зоны магнитной ленты № 3, а значение

$$\xi_3 = 10\,000\,000\,000\,010\,000\,000\,001\,110\,000\,001$$

указывает на ячейку № 1601 барабана № 2.

*) Это делается не всегда. Например, при использовании вычислительных машин Минск-2 и Минск-22 совокупность значений компонент $\xi^{14}, \xi^{15}, \ldots, \xi^{31}$ интерпретируется как номер ячейки на ленте.

Кодирование Л-программ. Символы языка ЛЯПАС представляются в машине булевыми векторами размерности 9, что позволяет ставить им в соответствие трехразрядные восьмеричные числа, называемые *кодами* данных символов. Соответствие задается нижеприводимой таблицей кодирования, на которую настроены существующие в настоящее время системы автоматического программирования с входным языком ЛЯПАС.

При кодировании с помощью этой таблицы некоторого символа ЛЯПАСа нужно сложить номер строки и столбца, в которых находится символ. Например, символ ¬ находится на пересечении строки номер 040 со столбцом номер 14, следовательно, кодом этого символа является 054.

Напомним, что элемент комплекса обозначается двумя символами, первый из которых показывает, какому комплексу принадлежит данный элемент, а второй задает порядковый номер элемента в комплексе. Например, обозначаемый через a_5 элемент комплекса A, имеющий порядковый номер 5, представляется парой кодов 100 405, а пара кодов 152 302 представляет тот элемент k_c оперативного комплекса K, номер которого задается значением индекса c.

Особым образом кодируются непосредственные константы. Они представляются кодами, совпадающими с этими константами: например, непосредственные константы **15**, **57**, **463** представляются кодами 015, 057, 463, соответственно. Нетрудно заметить порождаемую таким способом неоднозначность представления. Например, код 015 может представлять как непосредственную константу **15**, так и оператор $\Leftrightarrow$. Однако это не опасно, так как непосредственные константы опознаются по месту, которое они могут занимать в Л-программах.

Обратим внимание на правило кодирования Л-операторов, вытекающее из возможности их двоякого символического представления. С одним из них — именем, перед которым стоит символ * — мы уже знакомы. Другая символическая форма Л-оператора получается путем замены имени номером, задаваемым двумя непосредственными константами. Например, Л-оператор минсто представляется таким образом тройкой символов *001 002. Переход к кодированной форме,

Таблица кодирования символов ЛЯПАСа

	Коды	00	01	02	03	04	05	06	07	10	11	12	13	14	15	16	17
Операторы	000	/	§	(	)	!	*маш*	·	*сумма*	△	$\overline{\triangle}$	○	$\overline{\bigcirc}$	⇒	⇔	*зап*	*счит*
	020	′	″	‴	‴′									*	//		
	040	;	*ацпу*	Ẋ	Ẍ	→	⇉	⟜	↦	▽	*печ*$_{10}$	←	⊢	⌝		*печ*	*перф*
	060	⇐	<	>	*ввод*	*ввод′*	:	×	$\overline{\times}$	+	−	∧	∨	⊕	*локал*	*печ′*	*перф′*
Комплексы	100	***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***P***	***Q***
	120	***R***	***S***	***Г***	***U***	***V***	***W***	***X***	***Y***	***Z***	***Ж***	***Л***	***Ф***	***Ш***	***Э***	***Ю***	***Я***
	140	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*P*	*Q*
Внешн. операнды	160	α	β	γ	δ	ε	ζ	η	ϑ	ϰ	λ	μ	ν	ξ	π	ρ	σ
Переменные	200	Дополнительный набор переменных															
	220																
	240	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***	***p***	***q***
	260	***r***	***s***	***t***	***u***	***v***	***w***	***x***	***y***	***z***	***ж***	***л***	***ф***	***ш***	***э***	***ю***	***я***
Индексы	300	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*p*	*q*
	320	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*	*ж*	*л*	*ф*	*ш*	*э*	*ю*	*я*
	340	Дополнительный набор индексов															
	360																
Натуральные константы	400	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
	420	20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37
	440	40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57
	460	60	61	62	63	64	65	66	67	70	71	72	73	74	75	76	77
	500	100	101	102	103	104	105	106	107	110	111	112	113	114	115	116	117
	520	120	121	122	123	124	125	126	127	130	131	132	133	134	135	136	137
	540	140	141	142	143	144	145	146	147	150	151	152	153	154	155	156	157
	560	160	161	162	163	164	165	166	167	170	171	172	173	174	175	176	177

осуществляемый согласно таблице кодирования, достаточно прост и в данном случае приводит к кодам 034 001 002.

В некоторых ситуациях оказывается полезным использование пустого кода, символом которого является/.

При работе на 45-разрядных машинах коды Л-программ удобно разбивать на группы по пяти кодов и отводить для представления каждой группы одну ячейку памяти. Для облегчения обзора полезно при этом начинать каждое предложение программы с новой группы.

В заключение приведем пример закодированной описанным образом подпрограммы в к л ю м а к.

Цифровой код		Символическое представление
034 001 400 000 000		$*$ **001 400**
001 400 013 300 000	§ 0	$\overline{\circ}\, a$
001 401 010 300 074	§ 1	$\triangle\, a \oplus b_\beta \circ\!\!\rightarrow 2$
141 161 046 402 161		
300 054 072 162 047		$\beta_a \urcorner \wedge \gamma \longmapsto 1 \rightarrow 3$
401 044 403 000 000		
001 402 162 014 161	§ 2	$\gamma \Rightarrow \beta_a$
300 010 141 161 074		$\Delta b_\beta \oplus \delta \circ\!\!\rightarrow \alpha$
163 046 160 000 000		
001 403 006 000 000	§ 3	.

ОПЕРАЦИИ НАД БУЛЕВЫМИ И ТРОИЧНЫМИ МАТРИЦАМИ

§ 1. Оптимизация покрытий булевых матриц

Задача о переводчиках. Рассмотрим следующую задачу, не имеющую, казалось бы, никакого отношения к проблеме синтеза дискретных автоматов.

Допустим, что из некоторого числа переводчиков, каждый из которых владеет несколькими определенными языками, требуется скомплектовать минимальную по числу членов группу такую, чтобы она смогла обеспечить перевод с любого из интересующих нас языков.

Обозначим множество переводчиков, из которого можно производить выбор, через $A \equiv \{a_1, a_2, \ldots, a_m\}$, а множество интересующих нас языков — через $B \equiv \{b_1, b_2, \ldots, b_n\}$. Введем булеву матрицу $\|A \succ B\|$ отношения переводчиков к языкам, полагая, что $a_i \succ b_j$ в том и только в том случае, когда переводчик a_i знает язык b_j.

Например, если $A = \{a, b, c, d, e, f, g, h, i\}$, $B = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ и

$$\|A \succ B\| = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \left[\begin{array}{cccccccccc|c} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & b \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & c \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & d \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & e \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & f \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & g \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & h \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & i \end{array}\right] \end{array}$$

то это означает, что переводчик a знает языки 1, 4 и 8, переводчик b — языки 2, 6 и 7 и т. д.

О поиске кратчайших покрытий. Нетрудно видеть, что поставленная задача сводится к уже знакомой нам по первой главе (§ 8) задаче нахождения кратчайшего покрытия булевой матрицы $\|A \succ B\|$, то есть

нахождения такой минимальной совокупности строк матрицы, которая содержала бы не менее одной единицы в каждом столбце матрицы.

Оказывается, что целесообразность поиска кратчайших покрытий возникает при минимизации дизъюнктивных нормальных форм булевых функций, при синтезе логических схем некоторых типов, при решении систем логических уравнений (а эта задача, в свою очередь, имеет массу полезных интерпретаций), при поиске простейших диагностических тестов и во многих других задачах, эффективность методов решения которых оказывается, таким образом, существенно зависящей от совершенства используемых алгоритмов поиска кратчайших покрытий.

Важно отметить, что при решении практически интересных задач, подобных перечисленным, редко требуется получать все кратчайшие покрытия. Гораздо чаще оказывается достаточным нахождение лишь одного из них, неважно какого именно. Более того, довольно часто можно согласиться на получение не точного решения данной задачи, а некоторого приближения, если только оно не будет слишком уж плохим.

Выше (глава 1, § 8) был рассмотрен алгоритм получения одного из приближений к кратчайшему покрытию произвольного минора заданной булевой матрицы. Реализующая этот алгоритм программа была названа прикрапок. Она весьма проста и в то же время в ряде ситуаций позволяет получать достаточно удовлетворительные решения. Однако в других ситуациях более эффективной может оказаться описываемая ниже программа прикрапок1. Эту программу можно рекомендовать использовать в тех случаях, когда требуется найти покрытие минора, образованного произвольной совокупностью столбцов заданной булевой матрицы и всеми ее строками. В отличие от программы прикрапок, увеличение числа строк матрицы свыше 32 не влечет за собой повышения размерности каких-либо из операндов программы прикрапок1, которое, в свою очередь, могло бы вызвать резкое замедление скорости ее реализации. Данное свойство рассматриваемой программы обусловлено заменой подпрограмм минсто и максстро на соответствующим образом построенные их

модификации минстол и макстрок. Кроме того, приняты некоторые меры, позволяющие улучшить качество получаемых решений, а именно, введена процедура упрощения миноров, представляющих в процессе решения непокрытые остатки исходной матрицы. Эта процедура реализуется подпрограммой сокрамат.

Вспомогательные подпрограммы. Эти подпрограммы достаточно просты, в связи с чем в их описании опускается пункт 3.

Нахождение минимального столбца булевой матрицы — минстол

1. Требуется найти минимальный по числу единиц столбец булевой матрицы $\boldsymbol{A}$ из числа столбцов, входящих в подмножество P множества V всех столбцов матрицы. Если же таких минимальных столбцов будет несколько, требуется выбрать левый из них.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{A}$,

$\beta_п :: \| \{P\} \Rightarrow V \|$,

$[\gamma_и^+]$ — номер найденного столбца.

Задаются: a_α, b_α.

Размерность: $\alpha\beta$.

Внутренние операнды: $\boldsymbol{a}$, d, —.

4.

```
      * 001 126 (αa)
§ 0   β⇒a⊽d
§ 1   a X 4 b ○ a ○ c
§ 2   α_a ∧ c_b o-> 3 △ c
§ 3 . △ a⊕b_α |-> 2 c − d |-> 1 b ⇒ γ c ⇒ d → 1
§ 4   .
```

5. Пример:

$$\alpha :: \begin{bmatrix} 10110010 \\ 11100101 \\ 10011010 \\ 00011010 \\ 10100100 \\ 11010101 \end{bmatrix}$$

$\beta ::$ [10110100]

$[\gamma^+] = 2$

Нахождение максимальной среди отмеченных строк булевой матрицы — максстрок

1. Среди строк булевой матрицы A, имеющих единицу в i-м столбце, требуется найти максимальную по числу единиц в столбцах, образующих подмножество P множества V всех столбцов матрицы A. Если таких максимальных строк окажется несколько, выбрать первую из них.

2. Внешние операнды:

$\alpha_к :: A$,
$[\beta_и] = i$,
$\gamma_п :: \| \{P\} \ni V \|$,
$[\delta_и^+]$ — номер найденной строки.

Задаются: a_α, b_α.
Размерность: $\alpha\gamma$.
Внутренние операнды: a, c, —.

4.

* 001 127 (αa)
§ 0 $c_\beta \Rightarrow a \circ \delta \circ c \circ a$
§ 1 $\alpha_a \wedge a \circ\!\!\rightarrow 2\alpha_a \wedge \gamma \bigtriangledown \Rightarrow b$
$c - b \longmapsto 2a \Rightarrow \delta b \Rightarrow c$
§ 2 $\bigtriangleup a \oplus b_\alpha \longmapsto 1$.

5. Пример:

$$\alpha :: \begin{bmatrix} 1\,0\,1\,1\,0\,0\,0\,1\,1\,1\,0\,1 \\ 0\,1\,1\,0\,1\,0\,1\,1\,1\,0\,1\,0 \\ 0\,1\,0\,1\,1\,1\,0\,0\,0\,0\,1\,0 \\ 1\,1\,0\,0\,0\,0\,1\,0\,1\,1\,1\,0 \end{bmatrix}$$

$$\gamma :: [1\,1\,0\,0\,0\,1\,1\,1\,1\,1\,0\,0],$$

$$[\beta] = 1, \quad [\delta^+] = 3.$$

Сокращение булевой матрицы — сокрамат

1. Требуется построить булеву матрицу B, включая в нее такие строки булевой матрицы A, которые не поглощаются другими строками этой же матрицы в столбцах, принадлежащих подмножеству P множества V всех столбцов матрицы A. Однако, если в матрице A оказывается несколько строк, равных по значению

в выделенной совокупности столбцов, то выбирается лишь первая из них.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{A}$,

$\beta_п :: \| \{P\} \ni V \|$,

$\gamma_к^+ :: \boldsymbol{B}$.

Задаются: a_α, a_γ, b_α.
Размерность: $\alpha\beta\gamma$.
Совмещение: $\alpha\gamma$.
Внутренние операнды: $\boldsymbol{a}$, c, —.

4.

$* \ \mathbf{001\ 125}\ (\alpha\ \boldsymbol{a})$

§ 0 $\circ c \bar{\circ} b$

§ 1 $\triangle b \oplus b_\alpha \circ\!\!\rightarrow 4\, \alpha_b \wedge \beta \Rightarrow \boldsymbol{a} \bar{\circ} a$

§ 2 $\triangle a \oplus b_\alpha \circ\!\!\rightarrow 3\, \alpha_a \urcorner \wedge \boldsymbol{a} \mapsto 2$

$\alpha_a \wedge \beta \oplus \boldsymbol{a} \mapsto 1\, a - b \circ\!\!\rightarrow 1 \rightarrow 2$

§ 3 $\alpha_b \Rightarrow \gamma_c \triangle c \rightarrow 1$

§ 4 $c \Rightarrow b_\gamma$.

5. Пример:

$$\alpha :: \begin{bmatrix} 11000000 \\ 10100100 \\ 01111011 \\ 10110101 \\ 01100110 \end{bmatrix} \qquad \beta :: [\,11000110\,] \qquad \gamma^+ :: \begin{bmatrix} 11000000 \\ 10100100 \\ 01100110 \end{bmatrix}$$

Приближенный алгоритм нахождения кратчайшего покрытия. Этот алгоритм реализуется следующей подпрограммой.

Нахождение приближения к кратчайшему покрытию — прикрапок1

1. Ставится задача нахождения приближения к кратчайшему покрытию булевой матрицы $\boldsymbol{A}$ в столбцах, входящих в подмножество P множества V всех столбцов матрицы. Совокупность строк матрицы $\boldsymbol{A}$, представляющая полученное решение, составит матрицу $\boldsymbol{B}$.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{A}$,

$\beta_п :: \| \{P\} \ni V \|$,

$\gamma_к^+ :: \boldsymbol{B}$.

Задаются: a_α, a_γ, b_α.

Размерность: $\alpha\beta\gamma$.

Подпрограммы: сокрамат, минстол, макстрок.

Внутренние операнды: $\boldsymbol{c}$, g, —.

3. Алгоритм, реализуемый программой прикрапок1, весьма прост: сначала находится минимальный по числу единиц столбец из множества P, затем из числа тех строк матрицы $\boldsymbol{A}$, которые содержат единицу в данном столбце, выбирается максимальная по числу единиц в столбцах множества P, и представляющий ее элемент комплекса α переносится в комплекс γ. После этого исходная матрица упрощается путем выбрасывания строк, поглощаемых в еще не покрытых столбцах какими-либо другими строками, остающимися в матрице. Описанная процедура повторяется до тех пор, пока не будут покрыты все столбцы, входящие в множество P.

4.

$* \ \mathbf{001\ 407}\ (\alpha\ \boldsymbol{c})$

§ 0 $\beta \Rightarrow \boldsymbol{c} \circ g \to 2$

§ 1 сокрамат $\alpha\, \boldsymbol{c}\, \alpha$ //

§ 2 минстол $\alpha\, \boldsymbol{c}\, f$ // макстрок $\alpha\, f\, \boldsymbol{c}\, e$ //

$a_e \Rightarrow \gamma_g \triangle g\, a_e \urcorner \wedge \boldsymbol{c} \Rightarrow \boldsymbol{c} \mapsto 1\, g \Rightarrow b_\gamma$.

5. Допустим, что

$$\boldsymbol{A} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \left[\begin{array}{cccccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{matrix}$$

и

$$\beta = 1111111111100\ldots0.$$

Совокупность столбцов матрицы, которые требуется покрыть, представляется переменной c, принимающей сначала значение

$$1111111111100\ldots0.$$

Левым из минимальных по весу оказывается столбец 6, а максимальной по весу строкой, из числа строк, содержащих единицу в столбце 6, оказывается строка f. Поэтому строка f выбирается в качестве первого элемента решения, покрывая столбцы 1, 3, 6 и 9. Анализируя непокрытый остаток матрицы

$$\begin{array}{c} 2\,4\,5\,7\,8\,10 \\ \left[\begin{array}{c|c} 0\,1\,0\,0\,1\,0 & a \\ 1\,0\,0\,1\,0\,0 & b \\ 1\,0\,1\,0\,1\,0 & c \\ 0\,1\,0\,0\,1\,1 & d \\ 0\,1\,0\,1\,0\,0 & e \\ 0\,0\,0\,0\,0\,0 & f \\ 1\,0\,1\,1\,0\,0 & g \\ 0\,0\,1\,0\,1\,0 & h \\ 0\,0\,1\,1\,0\,1 & i \end{array}\right] \end{array}$$

(элементы уже покрытых столбцов для удобства не показаны), выбросим поглощаемые строки. В результате получим следующую матрицу:

$$\begin{array}{c} 2\,4\,5\,7\,8\,10 \\ \left[\begin{array}{c|c} 1\,0\,1\,0\,1\,0 & c \\ 0\,1\,0\,0\,1\,1 & d \\ 0\,1\,0\,1\,0\,0 & e \\ 1\,0\,1\,1\,0\,0 & g \\ 0\,0\,1\,1\,0\,1 & i \end{array}\right] \end{array}$$

Применив к этой матрице описанную процедуру, выберем столбец 2 и строку c, играющую в данном случае роль второго элемента конструируемого решения.

Аналогичным образом находятся остальные элементы решения: ими оказываются строки d и e. Таким образом,

полученный ответ представляется матрицей

$$\begin{bmatrix} 1\,0\,1\,0\,0\,1\,0\,0\,1\,0 \\ 0\,1\,1\,0\,1\,0\,0\,1\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,0\,0\,1\,0\,0\,0 \end{bmatrix} \begin{matrix} f \\ c \\ d \\ e \end{matrix}$$

содержащей по крайней мере одну единицу в каждом из столбцов, то есть покрывающей все столбцы. Заметим, однако, что найденное покрытие не является кратчайшим. Действительно, существует покрытие, содержащее только три строки: d, f и g. Оно представляется матрицей

$$\begin{bmatrix} 0\,0\,1\,1\,0\,0\,0\,1\,0\,1 \\ 1\,0\,1\,0\,0\,1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1\,0\,1\,0\,0\,0 \end{bmatrix} \begin{matrix} d \\ f \\ g \end{matrix}$$

О точном решении задачи. Точное решение задачи о покрытии булевой матрицы, то есть нахождение кратчайшего покрытия (покрытия, содержащего минимальное число элементов), является значительно более трудоемким. Тем не менее в некоторых ситуациях возникает необходимость в получении именно точного решения, например в тех случаях, когда алгоритм получения точных решений может служить эталоном, позволяющим оценивать качество решений, даваемых приближенными алгоритмами.

Решить поставленную задачу «в принципе» нетрудно. Например, можно воспользоваться методом полного перебора всевозможных совокупностей строк матрицы, их анализа и выявления среди них тех, которые являются покрытиями, наконец, выбора минимальных из полученных таким образом покрытий. Этот метод прост, но, к сожалению, настолько трудоемок, что практическое его применение, как правило, невозможно, если только число строк в матрице не слишком мало (например, не менее чем 30).

Более совершенные алгоритмы позволяют существенно сократить указанный перебор (заметим, что полное его устранение оказывается невозможным) и снизить

в связи с этим затраты времени на получение точного решения. Рассмотрим один из таких алгоритмов.

Введем понятие *дерева поиска* как схемы реализации алгоритма, содержащей информацию о том, из каких элементарных шагов состоит процесс решения некоторой конкретной задачи, в какой последовательности они совершаются и к какому результату они приводят. При этом понятие элементарного шага и его результата для задач различных типов может конкретизироваться по-разному.

В рассматриваемом алгоритме будут фигурировать элементарные шаги двух типов: выбор столбца булевой матрицы и выбор строки. Выбор строки сопряжен с ее добавлением в некоторое переменное подмножество строк, среди значений которого ищется кратчайшее покрытие, с анализом получаемого на данном шаге значения этого подмножества и с принятием решения о том, следует ли продолжать наращивать его мощность. Выбор столбца определяет *точку ветвления* вычислительного процесса, задавая подмножество строк, среди которых делается выбор на следующем шаге.

Правила поиска. Сформулируем следующие правила, положенные в основу алгоритма поиска кратчайшего покрытия.

1. *Сокращение матрицы.* Так мы назовем правило, говорящее о том, что при поиске одного из кратчайших покрытий булевой матрицы из нее можно предварительно удалить все поглощаемые строки (напомним, что строка матрицы называется поглощаемой, если в этой же матрице существует другая строка, не равная данной и содержащая единицы во всех тех столбцах, в которых имеются единицы в данной строке) и привести подобные, то есть оставить по одной среди равных строк.

2. *Итерация.* Это правило заключается в том, что если кратчайшее покрытие булевой матрицы ищется среди покрытий, содержащих некоторую заранее определенную совокупность строк, то эту задачу можно свести к задаче получения кратчайшего покрытия такой булевой матрицы, которая получается из исходной путем выбрасывания столбцов, покрываемых указанной совокупностью строк.

3. *Разложение по минимальному столбцу.* Поиск покрытий достаточно вести среди таких совокупностей, которые содержат по крайней мере одну из строк, имеющих единицу в некотором фиксированном столбце матрицы. Очевидно, что число таких совокупностей тем меньше, чем меньше единиц содержится в выбранном столбце. В связи с этим разумно останавливать выбор на минимальном (по числу единиц) столбце матрицы. Для определенности положим, что если в матрице содержится несколько минимальных столбцов, то имеется в виду левый из них.

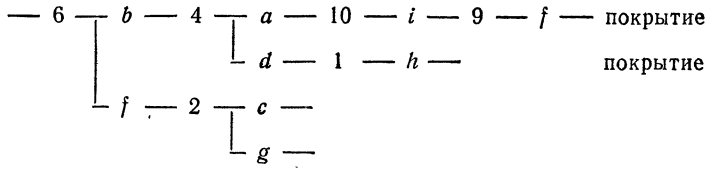
4. *Ограничение мощности покрытия.* Если в процессе решения найдено некоторое покрытие, то дальнейший поиск кратчайшего покрытия можно вести, ограничившись рассмотрением таких подмножеств строк, мощность которых меньше мощности найденного покрытия.

Заметим, что правила 1 и 4 основаны на использовании существенных ограничений в постановке задачи — требуется найти кратчайшее покрытие и только одно (неважно, какое именно). Правила 2 и 3 обладают более общим характером и могут, например, быть использованы при поиске безызбыточных покрытий, то есть таких покрытий, из которых нельзя удалить ни один элемент, с тем чтобы полученный остаток не перестал быть покрытием. Доказательства справедливости данных правил достаточно просты, в связи с чем их поиск предоставляется читателю.

Пример. Сформулированные правила позволяют значительно ограничить количество анализируемых подмножеств строк, в чем можно убедиться, вернувшись к рассмотрению уже знакомой нам булевой матрицы:

$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \left[\begin{array}{cccccccccc|c} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & b \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & c \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & d \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & e \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & f \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & g \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & h \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & i \end{array}\right] \end{array}$$

В данном случае процесс решения будет представлен следующим деревом поиска:



Рассмотрим это дерево.

На первом шаге вычислений выбирается столбец 6 — минимальный столбец исходной матрицы. Он содержит две единицы, в соответствии с чем в переменное подмножество строк (вначале оно пусто), среди которых производится поиск кратчайшего покрытия, нужно включить по крайней мере одну из строк b или f. Сначала рассматривается вариант включения верхней из них, строки b. После выбрасывания поглощаемых ею столбцов исходной матрицы и сокращения остатка возникает задача поиска кратчайшего покрытия следующей матрицы:

$$A' = \begin{matrix} \\ a \\ c \\ d \\ f \\ h \\ i \end{matrix} \begin{matrix} \begin{matrix} 1 & 3 & 4 & 5 & 8 & 9 & 10 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{matrix}$$

В этой матрице минимальным оказывается столбец 4, определяющий следующую точку ветвления процесса вычислений — выбор одной из двух строк, a или d. Выбор верхней из них приводит к матрице

$$\begin{matrix} \\ f \\ h \\ i \end{matrix} \begin{matrix} \begin{matrix} 3 & 5 & 9 & 10 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{matrix}$$

Нахождение следующего минимального столбца, столбца 10, приводит к однозначному выбору строки, а именно, строки i. После этого матрица сокращается до одного элемента, и построение покрытия завершается присоединением к текущему подмножеству строк строки f.

Итак, полученное покрытие состоит из четырех строк: b, a, i и f. Затем следует вернуться к последней из точек ветвления, с тем чтобы рассмотреть другой вариант достраивания текущего подмножества строк. В данном случае эта точка связана с выбором столбца 4, а другой вариант — с выбором строки d и рассмотрением совокупности строк b и d. Непокрытый этой совокупностью остаток исходной матрицы $\boldsymbol{A}$ может быть получен из матрицы $\boldsymbol{A}'$ путем выбрасывания столбцов, покрываемых строкой d, и последующего сокращения остатка. Оказывается, что он будет содержать только одну строку:

$$\begin{matrix} 1\,5\,9 \\ [1\,1\,1]\,h \end{matrix}$$

Следующий шаг тривиален: добавляя в множество $\{b, d\}$ строку h, мы получаем покрытие $\{b, d, h\}$, оказывающееся лучше предыдущего, так как оно содержит только три элемента. Дальнейший перебор производится уже среди двухэлементных множеств, не содержащих строки b, и отражается нижней ветвью дерева поиска. Он ограничивается рассмотрением только двух подмножеств, $\{f, c\}$ и $\{f, g\}$, получаемых аналогичным путем. Поскольку ни одно из них не является покрытием, процесс вычислений прекращается. Тем самым устанавливается, что найденное покрытие $\{b, d, h\}$ является кратчайшим.

Ускорение поиска. Сложность дерева поиска отражает в какой-то мере количество времени, затрачиваемого на поиск решения, и, в свою очередь, сильно зависит от числа ярусов в дереве, равного максимальной мощности перебираемых подмножеств строк матрицы. В связи с этим одним из эффективных путей повышения скорости поиска кратчайшего покрытия является уменьшение числа ярусов в дереве поиска. Введем следующее дополнительное правило.

5. *Уменьшение числа ярусов.* При поиске покрытий мощностью не более k мощность перебираемых подмножеств строк матрицы можно ограничить величиной k — — 2, дополнив перебор поиском двухэлементных покрытий непокрытых остатков матрицы.

Поиск двухэлементных покрытий осуществляется относительно легко с помощью следующей программы.

Нахождение двустрочного покрытия булевой матрицы — двупокр2

1. В заданной булевой матрице A требуется найти пару строк, являющуюся покрытием подмножества P множества V всех столбцов этой матрицы, или установить отсутствие такой пары.

2. Внешние операнды:

$\alpha_к :: A$,

$\beta_п :: \| \{P\} \ni V \|$,

$\gamma_к^+ ::$ матрица, образованная найденной парой строк,

$\tau^+ = 0$, если и только если двустрочное покрытие отсутствует.

Задаются: a_α, a_γ, b_α.
Размерность: $\alpha\beta\gamma$.
Внутренние операнды: a, c, —.

4.

* **001 124** (αa)

§ 0 $\beta \vdash \Rightarrow a \supset b$

§ 1 $\triangle\, b \oplus b_\alpha \circ\!\!\rightarrow 3\,\alpha_b \wedge c_a \circ\!\!\rightarrow 1\alpha_b \urcorner \wedge \beta \Rightarrow a \supset c$

§ 2 $\triangle\, c \oplus b_\alpha \circ\!\!\rightarrow 1\,\alpha_c \urcorner \wedge a \mapsto 2\,\alpha_b \Rightarrow \gamma_0$

$\alpha_c \Rightarrow \gamma_1\, 2 \Rightarrow b_\gamma$

§ 3 .

5. Пример:

$$\alpha :: \begin{bmatrix} 0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,1\,1 \\ 0\,0\,1\,1\,0\,1\,0\,0\,1\,1\,0\,0 \\ 1\,0\,1\,1\,0\,0\,1\,0\,1\,0\,1\,1 \\ 0\,1\,1\,0\,1\,0\,0\,1\,0\,1\,1\,0 \\ 1\,1\,0\,1\,0\,0\,1\,1\,0\,1\,1\,0 \end{bmatrix}$$

$$\beta :: [\,0\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,0\,]$$

$$\gamma^+ :: \begin{bmatrix} 0\,0\,1\,1\,0\,1\,0\,0\,1\,1\,0\,0 \\ 0\,1\,1\,0\,1\,0\,0\,1\,0\,1\,1\,0 \end{bmatrix}$$

Время вычислений можно уменьшить еще более, запоминая те промежуточные булевы матрицы, которые соответствуют точкам ветвления в дереве поиска. При этом в момент рассмотрения некоторого подмножества строк $\{a_1, a_2, \ldots, a_k\}$, образованного путем последовательного включения строк $a_1, a_2, \ldots, a_k$ (начиная с пу-

стого множества), достаточно хранить в памяти $k-1$ матрицу, не считая исходной. Первая из этих дополнительных матриц была получена при рассмотрении подмножества $\{a_1\}$ путем удаления из исходной матрицы столбцов, покрываемых строкой a_1, и сокращения непокрытого остатка; вторая аналогичным образом получается из первой путем удаления столбцов, покрываемых строкой a_2, и т. д. Кроме этих матриц, полезно запоминать вспомогательную промежуточную информацию: номера выбираемых столбцов и строк и т. п.

Схема точного алгоритма нахождения кратчайшего покрытия. Эта схема строится на основе упорядоченного использования перечисленных выше правил и дана на следующей странице.

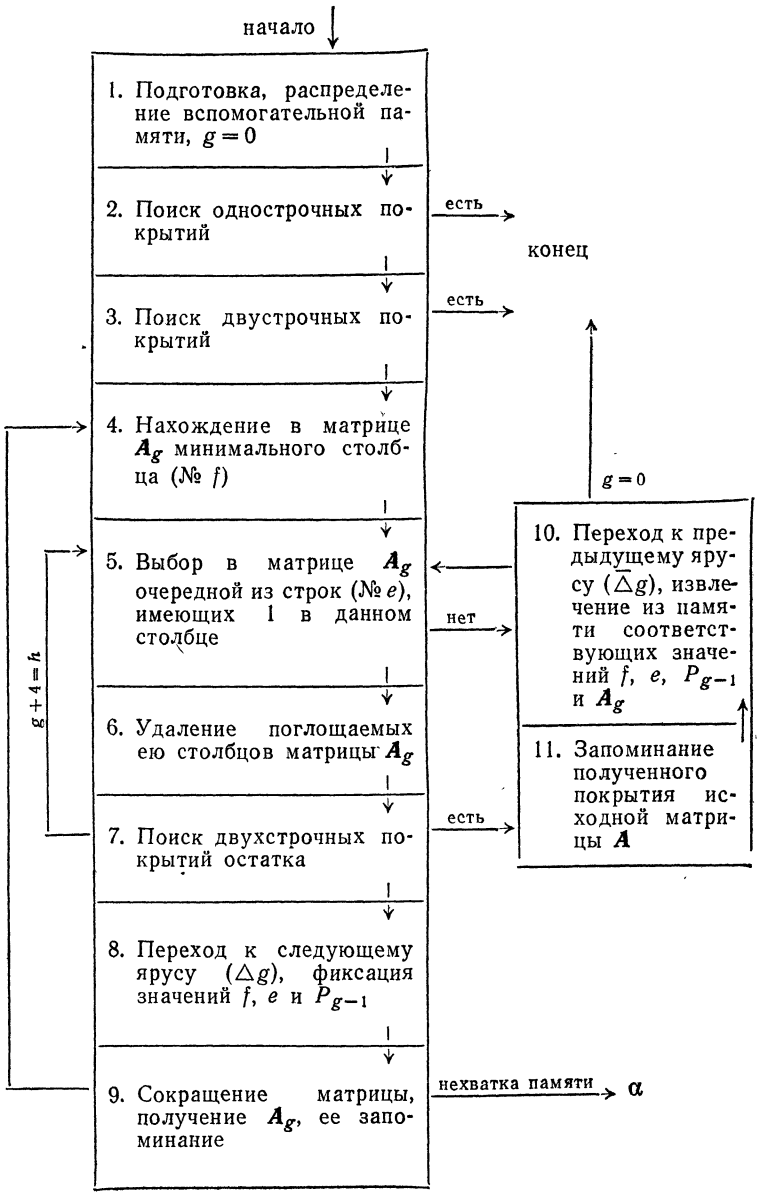
Схема отражает правила построения и обхода дерева поиска. Через g в ней обозначен номер проходимого яруса дерева, совпадающий с мощностью рассматриваемого при этом подмножества тех строк матрицы A, которые к текущему моменту времени уже включены в строящееся покрытие, через P_g обозначена совокупность непокрываемых при этом столбцов матрицы A и через A_g — остаток матрицы A, в котором ищутся недостающие элементы покрытия.

Обратим внимание на переход от блока 7 к блоку 5, реализуемый в том случае, когда оказывается, что двустрочное покрытие остатка матрицы A (которое ищется блоком 7) не существует и в то же время выполняется условие $g+4=h$. Через h здесь обозначена мощность уже найденного покрытия, кратчайшего среди рассмотренных. Смысл этого перехода легко понять, вспомнив, что после нахождения покрытия мощности h не стóит рассматривать совокупности, состоящие более чем из $h-1$ строк, а именно такие совокупности и рассматривались бы в данном случае при переходе к блоку 8.

Программа окрапок4. Рассмотренная схема положена в основу следующей подпрограммы.

Нахождение одного кратчайшего покрытия — окрапок4

1. Решается задача нахождения одного из кратчайших покрытий булевой матрицы A в столбцах, образующих подмножество P множества V всех столбцов матрицы A.


начало
1. Подготовка, распределение вспомогательной памяти, $g = 0$
2. Поиск однострочных покрытий
есть
конец
3. Поиск двустрочных покрытий
есть
4. Нахождение в матрице A_g минимального столбца (№ f)
$g = 0$
10. Переход к предыдущему ярусу ($\triangle g$), извлечение из памяти соответствующих значений f, e, P_{g-1} и A_g
5. Выбор в матрице A_g очередной из строк (№ e), имеющих 1 в данном столбце
нет
11. Запоминание полученного покрытия исходной матрицы A
$g + 4 = h$
6. Удаление поглощаемых ею столбцов матрицы A_g
7. Поиск двухстрочных покрытий остатка
есть
8. Переход к следующему ярусу ($\triangle g$), фиксация значений f, e и P_{g-1}
9. Сокращение матрицы, получение A_g, ее запоминание
нехватка памяти
α

2. Внешние операнды:

$\alpha_{ч}$ — аварийный выход, соответствующий нехватке памяти,

$\beta_{к}$ :: $\boldsymbol{A}$, продолжение комплекса β служит для запоминания промежуточной информации,

$[\gamma]_{и}$ — максимально допустимая мощность комплекса β,

$\delta_{п}$ :: $\| \{P\} \equiv V \|$,

$\varepsilon_{к}^{+}$ :: матрица, образованная из строк — элементов найденного кратчайшего покрытия.

Задаются: a_β, a_ε, b_β.
Размерность: βε.
Подпрограммы: двупокр2, минстол, сокрамат.
Внутренние операнды: $\boldsymbol{d}$, h, $\boldsymbol{Y}$.
Примечание: если покрытие отсутствует, величина b_1 принимает значение 0.

4.

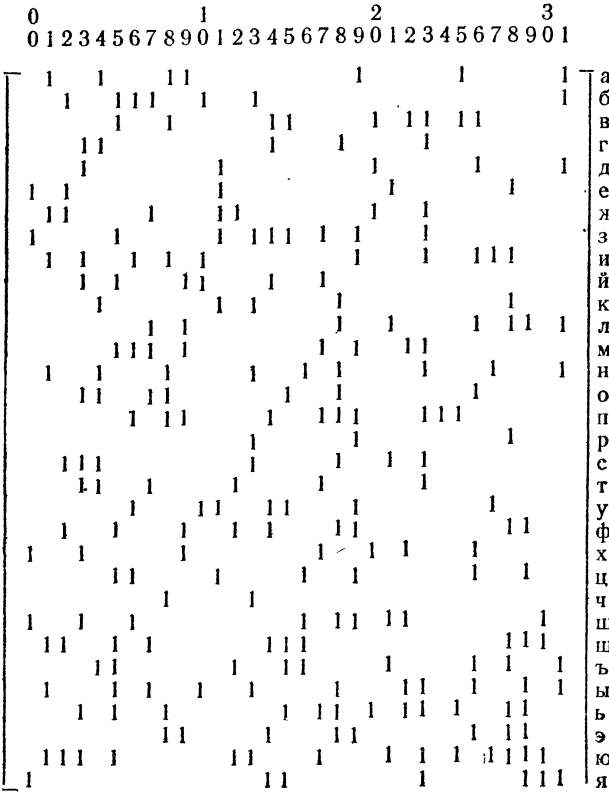
		* **001 130** ($\beta\, c\, d$)	Номера блоков схемы
§	0	$a_\beta \Rightarrow a_{30}\, b_\beta \Rightarrow b_{30} \circ g \,\overline{\circ}\, h$ $\delta \Rightarrow \boldsymbol{d} \Rightarrow \boldsymbol{c} \circ a$	1
§	1	$\beta_a \urcorner \wedge \boldsymbol{d} \circ\!\!\rightarrow 10\ \triangle a \oplus b_\beta \longmapsto 1$ $\twoheadrightarrow 2 \circ\!\!\rightarrow 3\ 2 \Rightarrow h \rightarrow 11$	2
§	2	* двупокр2 $\boldsymbol{Z}\, \boldsymbol{d}\, \varepsilon$ // !	3
§	3	* минстол $\boldsymbol{Z}\, c\, f$ // $\overline{\circ}\, e$	4
§	4	$\triangle\, e \oplus b_{30} \circ\!\!\rightarrow 5\ \boldsymbol{z}_e \wedge c_f \circ\!\!\rightarrow 4$	5
		$\boldsymbol{z}_e \urcorner \wedge \boldsymbol{c} \Rightarrow \boldsymbol{d} \twoheadrightarrow 2 \longmapsto 6$ $g + 4 \oplus h \circ\!\!\rightarrow 4$	6,7
		$\triangle\, g\, a_{30} \Rightarrow a_{27} \Rightarrow d \quad b_{30} \Rightarrow b_{27}$ $(c\, f\, e\, d) \Rightarrow \boldsymbol{Z}\, \boldsymbol{d} \Rightarrow \boldsymbol{c}$ $a_{30} + b_{30} \Rightarrow a_{30} + b_{30} - a_\beta - \gamma \longmapsto \alpha$	8
		* сокрамат $\boldsymbol{Y}\, \boldsymbol{c}\, \boldsymbol{Z}$ // $\rightarrow 3$	9

§ 5 $g \circ\!\!\rightarrow 11\,\overline{\triangle}\, g\, a_{30} - 1 \Rightarrow b$] 10
$a_{30} - k_b \Rightarrow b_{30}\, k_b \Rightarrow a_{30}$
$Z \Rightarrow (c\, f\, e\, d) \rightarrow 4$]

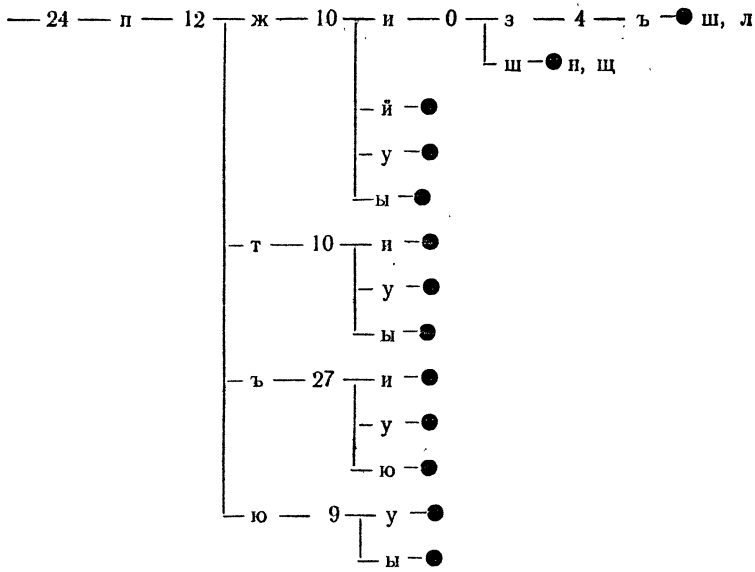
§ 6 $a_{30} \Rightarrow a_{27}\, 2 \Rightarrow h$] 11
§ 7 $y_e \Rightarrow \varepsilon_h \triangle h\, a_{27} \oplus a_\beta \circ\!\!\rightarrow 5$
$a_{27} - 1 \Rightarrow b - 1 \Rightarrow c$
$k_b \Rightarrow a_{27}\, k_c \Rightarrow e \rightarrow 7$]

§ 10 $\beta_a \Rightarrow \varepsilon_0\, 1 \Rightarrow h$
§ 11 $h \urcorner\!\!\mapsto 12 \circ h$
§ 12 $h \Rightarrow b_\varepsilon$.

5. Например, анализируя булеву матрицу



представленную значением комплекса β (подразумевается, что пустые места в данной матрице занимаются нулями), программа окрапок4 реализует обход следующего дерева поиска:



Ребра этого дерева соответствуют выбираемым строкам матрицы, вершины, обозначенные цифрами, — минимальным столбцам, находимым подпрограммой минстол непосредственно после сокращения матрицы, производимого подпрограммой сокрамат, а конечные вершины, показанные кружками, соответствуют моментам времени, в которые применяется лишь подпрограмма двупокр2, осуществляющая поиск двустрочных покрытий остатка. Первым находится покрытие мощностью 7, состоящее из строк п, ж, и, з, ъ, ш, л, затем — покрытие из строк п, ж, и, ш, н, щ, в дальнейшем ищутся покрытия мощностью не более 5, но ввиду отсутствия таковых за решение принимается указанное шестиэлементное покрытие, представляемое значением комплекса ε:

$$\varepsilon^{+}::\left|\begin{array}{l}
0\,1\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1 \\
0\,1\,1\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0 \\
1\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0 \\
0\,1\,0\,1\,0\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,1\,1\,1\,0\,0\,0 \\
0\,1\,1\,0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,0\,0\,0\,1\,0\,0\,1\,1\,1\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0
\end{array}\right|\begin{array}{l}\text{н}\\ \text{щ}\\ \text{ш}\\ \text{и}\\ \text{ж}\\ \text{п}\end{array}$$

При этом выбранные строки подвергаются некоторой перестановке.

§ 2. Поиск минимальных разбиений

Получение точного решения. Вспомним рассмотренную в главе 1 задачу о размещении отдыхающих по лодкам, которой было присвоено затем более абстрактное наименование задачи о нахождении минимального разбиения некоторого множества на совместимые подмножества. Вернувшись к этой задаче, нетрудно будет составить программу нахождения ее точного решения, поскольку теперь в нашем распоряжении имеется подпрограмма окрапок4, гарантирующая получение кратчайших покрытий булевых матриц, число строк в которых может быть довольно большим.

Программа может быть составлена по аналогии с программой расопод и оказывается при этом весьма простой. Поэтому ограничимся приведением ее в уже оформленном виде.

Нахождение минимального разбиения на совместимые подмножества — расопод1

1. На множестве пар элементов некоторого множества A задано отношение несовместимости (будем обозначать выражением $a_i * a_j$ факт несовместимости элементов a_i и a_j из множества A). Требуется найти такое разбиение Q множества A на минимальное число подмножеств, при котором каждое подмножество будет содержать лишь совместимые пары элементов.

2. Внешние операнды:

$\alpha_к ::\| A * A \|$,

$\beta_к ::\| Q \ni A \|$, $\sigma(Q) \leqslant 32$,

$\gamma_к$ — комплекс памяти,

δ — дополнительный выходной полюс, реализуемый при нехватке памяти.

Задаются: a_α, a_β, a_γ, b_α, b_γ.
Внутренние операнды: ***e***, *i*, ***Y***.

4.

* **001 211**

§ 0 $b_\gamma \Rightarrow i$ * максопод $2\,\alpha\,\gamma$ //
$b_\alpha - 1 \Rightarrow a\,0 - c_a \Rightarrow \boldsymbol{e}$
* окрапок4 $2\,\gamma\,i\,\boldsymbol{e}\,\beta$ // $\overline{\ominus}\,a \circ \boldsymbol{a}$

§ 1 $\triangle\, a \oplus b_\beta \circ\!\!\rightarrow 3\,\boldsymbol{a}\ \urcorner \wedge \beta_a \Rightarrow \beta_a \vee \boldsymbol{a} \Rightarrow \boldsymbol{a} \rightarrow 1$

§ 2 $\rightarrow \delta$

§ 3 .

Метод прямого размещения. Можно предложить совершенно иной метод решения рассматриваемой задачи, суть которого легко объяснить, воспользовавшись начальной интерпретацией задачи.

Представим себе, что отдыхающие, общим числом n, собрались на берегу, а мы должны произвести посадку в лодки при условии, что весь процесс посадки должен быть разбит на n шагов, на каждом из которых отыскивается место в какой-либо лодке для одного из отдыхающих. Этот процесс можно рассматривать как последовательность из n ситуаций, в каждой из которых решается одна и та же задача: какого конкретно отдыхающего обеспечить в данный момент местом в лодке и в какой именно? В результате решения такой задачи каждый раз из предшествующей ситуации возникает последующая, в которой на берегу остается уже на одного человека меньше (что можно рассматривать как изменение текущей ситуации). Описанный процесс в целом назовем процессом *прямого размещения* (отдыхающих по лодкам).

Решая поставленную задачу, мы не знаем заранее, сколько же потребуется лодок (обозначим это число через k) для размещения всех отдыхающих. Очевидно только, что разным способам размещения могут соответствовать различные значения k и что некоторые из этих способов должны быть оптимальными, то есть приводить к минимально возможному значению k ($k = k_{\min}$).

Примем величину $k_{\min}$ за оценку качества исходной ситуации (когда все отдыхающие находятся на берегу),

и будем аналогичным образом оценивать и все остальные ситуации, возникающие в процессе прямого размещения, считая при этом, что оптимальное для этих ситуаций решение следует искать лишь при условии, что лиц, уже размещенных в лодках, пересаживать нельзя.

Несмотря на то, что мы не можем непосредственно подсчитывать значение оценки качества ситуации (для этого потребовалось бы найти и реализовать оптимальный способ размещения), мы можем сравнивать по этой оценке различные ситуации и в ряде случаев делать вполне определенные утверждения.

Например, рассматривая некоторый шаг процесса прямого размещения, иногда можно утверждать, что оценка последующей ситуации равна оценке предшествующей ситуации, то есть качество текущей ситуации при реализации рассматриваемого шага не меняется. Назовем такой шаг *хорошим*. В других же случаях заведомо известно, что реализация рассматриваемого шага приведет к увеличению оценки текущей ситуации — назовем такой шаг *плохим*. Наконец, при некоторых обстоятельствах остается неизвестным, приведет ли реализация рассматриваемого шага к возрастанию оценки текущей ситуации или эта оценка останется прежней (очевидно, что уменьшиться она не может). Назовем такой шаг *сомнительным*.

Нетрудно показать, что если при реализации метода прямого размещения все шаги оказываются хорошими, то полученное решение должно быть оптимальным, то есть число занятых лодок будет минимальным. Однако может оказаться, что в некоторых из возникающих ситуаций нельзя будет найти хороших шагов (будем называть такие ситуации *критическими*) и придется реализовать какой-либо из сомнительных шагов. Заметим в связи с этим, что если в текущей ситуации нельзя найти хороших шагов, то обязательно должны быть сомнительные шаги, так как из самого определения оценки качества ситуации следует, что множество всех возможных в текущей ситуации шагов не может состоять исключительно из плохих шагов. Естественно, что при реализации сомнительного шага теряются гарантии на оптимальность получаемого затем решения, хотя, конечно, не теряется возможность такой оптимальности.

Отметим относительность понятия «сомнительный шаг», обусловленную отсутствием эффективных средств установления более определенной характеристики рассматриваемого шага. В конечном счете каждый из сомнительных шагов оказывается или плохим, или хорошим, то есть его реализация или приводит к ухудшению качества получаемого решения, или позволяет нам избежать этого.

Анализ текущей ситуации и поиск хороших шагов. Текущая ситуация характеризуется уже реализованным распределением части отдыхающих по лодкам и перечнем пока оставшихся на берегу и образующих некоторое множество R. При решении задачи распределения этой оставшейся части удобно описывать ситуацию двумя матрицами: матрицей $\|R * R\|$, задающей отношение несовместимости для тех отдыхающих, которых еще надо разместить, и матрицей $\|B * R\|$, задающей отношение несовместимости между упомянутыми отдыхающими и частично заполненными лодками, образующими множество $B(b_i * r_j$, если отдыхающего r_j нельзя посадить в лодку $b_i)$.

Из сказанного выше следует, что должно быть оказано предпочтение хорошим шагам, так как их реализация не приводит к ухудшению качества текущей ситуации: если, исходя из предшествующей ситуации, можно решить задачу размещения так, чтобы обойтись, например, k лодками, то эта возможность сохраняется и для последующей ситуации. Поэтому, анализируя очередную ситуацию, представленную текущими значениями матриц $\|B * R\|$ и $\|B * R\|$, целесообразно прежде всего искать хорошие шаги.

Предложим следующие два правила нахождения хорошего шага.

1. Если в матрице $\|B * R\|$ существует столбец без нулей, то это означает, что соответствующий этому столбцу отдыхающий не может быть посажен ни в одну из лодок, принадлежащих множеству B. Следовательно, необходимо начать заполнение новой лодки, посадив в нее данного отдыхающего и включив ее в множество B.

2. Если в матрице $\|R * R\|$ существует некоторая i-я строка, для которой можно найти поглощающую ее j-ю строку в матрице $\|B * R\|$ с i-й компонентой, имеющей

значение 0, то это означает, что отдыхающий r_i может быть помещен в лодку b_j, и это не приведет к изменению отношения несовместимости между данной лодкой и остающимися на берегу отдыхающими. Отсюда следует, что данный шаг будет заведомо хорошим.

Поясним второе правило на примере. Допустим, что очередная ситуация описывается следующей парой матриц:

$$\|R*R\| = \begin{matrix} 1\,2\,3\,4\,5 \\ \left[\begin{matrix} 0\,1\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,0 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 2 \\ 3, \\ 4 \\ 5 \end{matrix} \quad \|B*R\| = \begin{matrix} 1\,2\,3\,4\,5 \\ \left[\begin{matrix} 1\,0\,1\,1\,0 \\ 0\,0\,0\,0\,1 \end{matrix}\right] \end{matrix} \begin{matrix} a \\ b \end{matrix}.$$

В этой ситуации искомое соотношение выполняется между лодкой a и отдыхающими r_2 и r_5. Поэтому, не теряя возможности получения оптимального решения в целом, можно посадить отдыхающих r_2 и r_5 в лодку a и перейти к рассмотрению последующей ситуации, характеризующейся парой матриц:

$$\|R*R\| = \begin{matrix} 1\,3\,4 \\ \left[\begin{matrix} 0\,1\,0 \\ 1\,0\,1 \\ 0\,1\,0 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 3, \\ 4 \end{matrix} \quad \|B*R\| = \begin{matrix} 1\,3\,4 \\ \left[\begin{matrix} 1\,1\,1 \\ 0\,0\,0 \end{matrix}\right] \end{matrix} \begin{matrix} a \\ b \end{matrix}.$$

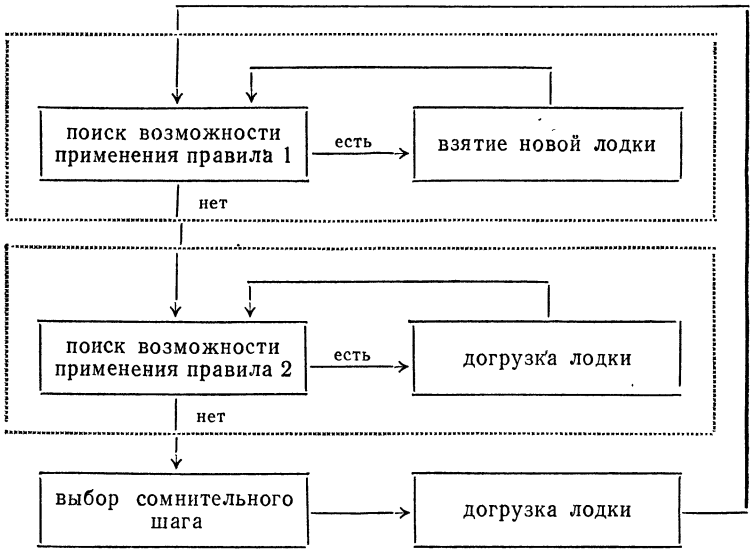
Выбор сомнительного шага. Если в рассматриваемой ситуации хороший шаг не находится, остается выбрать какой-либо из сомнительных шагов. При этом приходится мириться с возможностью ухудшения качества текущей ситуации. Можно лишь попытаться уменьшить вероятность такого ухудшения на основе некоторых общих соображений.

Предложим следующее правило выбора сомнительного шага, которым рекомендуется пользоваться лишь в тех ситуациях, в которых сформулированные выше правила нахождения хорошего шага оказываются неприменимыми.

3. В матрице $\|R*R\|$ нужно найти максимальную по числу единиц строку. Соответствующего этой строке отдыхающего следует поместить в одну из лодок, принадлежащих множеству B и совместимых с данным отдыхающим.

Конечно, данное правило не является наилучшим из всех, которые можно было бы предложить. В его защиту можно сказать следующее: во-первых, его применение приводит к ускоренному устранению единиц из матрицы $\|R * R\|$, что, в свою очередь, повышает вероятность применения в дальнейшем правила 2 нахождения хорошего шага, во-вторых, это правило весьма просто реализуется.

Общая структура алгоритма прямого размещения. Совокупность рассмотренных правил выбора хороших и сомнительных шагов естественным образом объединяется в следующей схеме алгоритма решения задачи в целом:



Отмеченные пунктиром участки схемы соответствуют задачам, для решения которых ниже будут предложены специальные подпрограммы. Остальная же часть схемы будет представлена текстом основной программы, в которую будут также входить непоказанные в схеме операции определения конца работы алгоритма. Очевидно, задачу можно считать решенной до конца, если множество R оставшихся на берегу отдыхающих станет пустым.

В порядке уточнения алгоритма положим, что если в некоторой ситуации среди оставшихся на берегу

отдыхающих имеется несколько претендентов на применение правила 1, то из них следует выбирать наименее сговорчивого, то есть такого, которому соответствует строка в матрице $\|R * R\|$, содержащая как можно более единиц. Таким путем будет повышаться вероятность возможности применения правила 1 на последующих шагах. Если же и после этого дополнительного отбора число претендентов останется большим одного, условимся выбирать первого из них (при таком условии будет удобно использовать в программе подпрограмму макстро).

Второе из сделанных уточнений является малосущественным с принципиальной точки зрения и вводится исключительно для устранения возможной неоднозначности толкования алгоритма в деталях. Это замечание относится также к следующим уточнениям, касающимся порядка выбора сомнительного шага и порядка применения правила 2.

При подготовке сомнительного шага условимся останавливать выбор также на первой среди максимальных строк матрицы $\|R * R\|$, и на первой среди лодок, принадлежащих множеству B и совместимых с выбранным отдыхающим. Наконец, при поиске возможности применения правила 2 условимся просматривать подряд все лодки из множества B, начиная с начала, и для каждой из них аналогичным образом перебирать всех отдыхающих из множества R, останавливая выбор на первой из встреченных пар, допускающих применение правила 2.

Пример. Вернемся к рассмотрению описанного в главе 1 примера, для которого исходная ситуация (все отдыхающие на берегу, то есть $R = A$) представляется одной матрицей:

$$\|R * R\| = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \left\|\begin{matrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{matrix}\right\| \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{matrix}$$

поскольку множество B вначале пусто (можно считать, что число строк матрицы $\|B * R\|$ равно при этом нулю). В этой ситуации правило 1 может быть применено к любому отдыхающему. Сделанное уточнение приводит нас к выбору отдыхающего r_2, как первого из наименее сговорчивых. Посадив его в лодку, которую мы обозначим символом a, скорректируем при этом матрицу $\|R * R\|$, удалив из нее второй столбец и вторую строку, а также введем соответствующую строку в матрицу $\|B * R\|$, которая становится теперь однострочной. Результатом последующих применений правила 1 будет занятие еще двух лодок, в одну из которых будет усажен отдыхающий r_4, в другую — отдыхающий r_9. Возникает ситуация, характеризующаяся матрицами

$$\|R * R\| = \begin{array}{c} \begin{matrix} 1 & 3 & 5 & 6 & 7 & 8 & 10 \end{matrix} \\ \left[\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} 1 \\ 3 \\ 5 \\ 6 \\ 7 \\ 8 \\ 10 \end{matrix}, \quad \|B * R\| = \begin{array}{c} \begin{matrix} 1 & 3 & 5 & 6 & 7 & 8 & 10 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} a \\ b \\ c \end{matrix}.$$

В полученной ситуации правило 1 оказывается уже неприменимым, но зато может быть использовано правило 2, согласно которому отдыхающий r_3 усаживается в лодку a, а отдыхающие r_1 и r_{10} следуют в лодку b. После этого возникает ситуация, равноценная с исходной (так как все реализованные до сих пор шаги были хорошими), но в которой дальнейшее применение правил 1 и 2 оказывается уже невозможным. Данная ситуация представляется матрицами

$$\|R * R\| = \begin{array}{c} \begin{matrix} 5 & 6 & 7 & 8 \end{matrix} \\ \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} 5 \\ 6 \\ 7 \\ 8 \end{matrix}, \quad \|B * R\| = \begin{array}{c} \begin{matrix} 5 & 6 & 7 & 8 \end{matrix} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} a \\ b \\ c \end{matrix}$$

и является критической.

Здесь приходится сделать сомнительный шаг. Согласно правилу 3, выбирается отдыхающий r_6 и усаживается

в лодку a, после чего появляется ситуация, в которой

$$\|R * R\| = \begin{matrix} \\ 0\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \end{matrix}\!\!\begin{matrix} 5\,7\,8 \\ \left[\begin{matrix} \\ \\ \end{matrix}\right. \\ \left.\begin{matrix} \\ \\ \end{matrix}\right] \end{matrix}\!\!\begin{matrix} 5 \\ 7, \\ 8 \end{matrix} \qquad \|B * R\| = \begin{matrix} \\ 1\,0\,1 \\ 1\,0\,0 \\ 0\,1\,0 \end{matrix}\!\!\begin{matrix} 5\,7\,8 \\ \left[\begin{matrix} \\ \\ \end{matrix}\right. \\ \left.\begin{matrix} \\ \\ \end{matrix}\right] \end{matrix}\!\!\begin{matrix} a \\ b \\ c \end{matrix}.$$

Больше критических ситуаций не встречается, и остаток отдыхающих размещается по лодкам, согласно правилу 2: r_7 усаживается в лодку a, r_8 — в b, наконец, r_5 — в c. После этого множество R становится пустым и задача считается решенной. Полученное решение (разбиение Q множества A) представляется матрицей

$$\|Q \ni A\| = \begin{matrix} \\ 0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,0 \\ 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,1 \\ 0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0 \end{matrix}\!\!\begin{matrix} 1\,2\,3\,4\,5\,6\,7\,8\,9\,10 \\ \left[\begin{matrix} \\ \\ \end{matrix}\right. \\ \left.\begin{matrix} \\ \\ \end{matrix}\right] \end{matrix}.$$

Описанный процесс прямого размещения для данного примера можно проиллюстрировать таблицей 2.2.1 реализации алгоритма, в которой знаком «+» отмечены хорошие шаги, а знаком «—» — сомнительные.

Таблица 2.2.1

Номер шага	Кто усаживается	В какую лодку	Тип шага	Кол-во занятых лодок
1	r_2	a	+	1
2	r_4	b	+	2
3	r_9	c	+	3
4	r_3	a	+	3
5	r_1	b	+	3
6	r_{10}	b	+	3
7	r_6	a	—	3
8	r_7	a	+	3
9	r_8	b	+	3
10	r_5	c	+	3

Более компактным является следующее символическое представление реализации процесса размещения:

$2a$—$4b$—$9c$—$3a$—$1b$—$10b$—$6a$—$7a$—$8b$—$5c$

Итак, лишь один из сделанных шагов оказался сомнительным. Привел ли он к потере качества решения? Можно утверждать, что в данном случае этого не произошло. Действительно, уже после реализации первых трех шагов, которые все были хорошими, число занятых лодок достигло трех. Это же число сохранилось и к концу реализации алгоритма, из чего следует, что полученное решение оптимально, то есть число использованных лодок является минимально необходимым.

Подпрограммы поиска и реализации хороших шагов. При представлении текущих значений матриц $\|R * R\|$ и $\|B * R\|$, размеры которых изменяются на протяжении всего процесса вычислений, удобно использовать рассмотренную ранее (глава 1, § 8) методику выделения миноров из более стабильных матриц. В данном случае различные значения матрицы $\|R * R\|$ можно рассматривать как различным образом выделяемые миноры неизменяемой матрицы $\|A * A\|$, а вместо матрицы $\|B * R\|$ удобнее оперировать непосредственно с матрицей $\|B * A\|$ и уже из этой матрицы выделять столбцовый минор $\|B * R\|$.

Эта методика применена в описываемых ниже подпрограммах.

Поиск и реализация хороших шагов по правилу 1 — хоро

1. Данная подпрограмма производит проверку возможности применения правила 1 с уточнениями и реализует вытекающие из этого правила хорошие шаги до тех пор, пока это возможно.

2. Внешние операнды:

$\alpha_к :: \| A * A \|$,

$\beta_п :: \| \{R\} \ni A \|$,

$\gamma_к :: \| B * A \|$,

$\delta_к :: \| Q \ni A \|$.

Задаются: a_α, a_γ, a_δ, b_α, b_γ, b_δ.
Внутренние операнды: $\boldsymbol{b}$, e, —.

4.

```
        * 001 206
§ 0  b_γ ⇒ e → 2
```

§ 1 * макстро $\alpha\, b\, \beta\, d$ //

$c_d \Rightarrow \delta_e\, a_d \Rightarrow \gamma_e \triangle e\, c_d \oplus \beta \Rightarrow \beta$

§ 2 $\overline{\circ}\, a\, \beta \Rightarrow b$

§ 3 $\triangle\, a \oplus e \circ\!\!\rightarrow 1$

$\gamma_a \wedge b \Rightarrow b \longmapsto 3$

$e \Rightarrow b_\beta \Rightarrow b_\delta$.

Поиск и реализация хороших шагов по правилу 2 — хорд

1. Данная подпрограмма производит проверку возможности применения правила 2 с уточнениями и реализует вытекающие из этого правила хорошие шаги до тех пор, пока это возможно.

2. Внешние операнды:

$\alpha_к :: \| A * A \|$,

$\beta_п :: \| \{R\} \Rightarrow A \|$,

$\gamma_к :: \| B * A \|$,

$\delta_к :: \| Q \Rightarrow A \|$.

Задаются a_α, a_γ, a_δ, b_α, b_γ, b_δ.
Внутренние операнды: b, b, —.

4.

* **001 207**

§ 0 $\overline{\circ}\, a$

§ 1 $\triangle\, a \oplus b_\gamma \circ\!\!\rightarrow 3$

$\gamma_a \neg \wedge \beta \Rightarrow a \Rightarrow b$

§ 2 $a\, \dot{X}\, 1\, b\, \alpha_b \wedge b \longmapsto 2$

$c_b \vee \delta_a \Rightarrow \delta_a\, c_b \oplus \beta \Rightarrow \beta\, \overline{\circ}\, a \rightarrow 1$

§ 3 .

Программа прямого размещения. Рассмотренный алгоритм решения задачи размещения отдыхающих по лодкам реализуется следующей программой, оформленной как подпрограмма.

Нахождение разбиения на совместимые подмножества методом прямого размещения — прямор

1. На множестве пар элементов некоторого множества A задано отношение несовместимости. Требуется найти такое разбиение Q множества A по возможности на минимальное число подмножеств, при котором каждое подмножество будет содержать лишь совместимые пары элементов.

2. Внешние операнды:

$\alpha_к :: \| A * A \|$,

$\beta_к^+ :: \| Q \ni A \|$,

$\gamma_к$ — комплекс памяти, $\sigma(\gamma) \leqslant \sigma(\alpha)$.

Задаются: a_α, a_β, a_γ, b_α.

Внутренние операнды: c, e, —.

4.

* **001 210**

§ 0 $b_\alpha - 1 \Rightarrow a\,0 - c_a \Rightarrow c \circ b_\beta \circ b_\gamma$

§ 1 * хоро $\alpha\, c\, \gamma\, \beta \,//\, c \circ\!\!\rightarrow 3$

* хорд $\alpha\, c\, \gamma\, \beta \,//\, c \circ\!\!\rightarrow 3$

* макстро $\alpha\, c\, c\, d \,//\, \overline{\circ}\, a$

§ 2 $\Delta\, a\, \gamma_a \wedge c_d \longmapsto 2$

$c_d \vee \beta_a \Rightarrow \beta_a\, \alpha_d \vee \gamma_a \Rightarrow \gamma_a$

$c_d \oplus c \Rightarrow c \longmapsto 1$

§ 3 .

Перебор в критических ситуациях. Предложенный выше точный алгоритм расопод 1 обладает существенным недостатком: получаемое в качестве промежуточного результата множество P максимальных совместимых подмножеств из A может оказаться довольно большим, что повлечет определенные затруднения при представлении матрицы $\|P \ni A\|$ в памяти машины, а также при реализации подпрограммы окрапок4.

Например, пусть множество A составлено из 30 элементов, которые можно разбить на тройки таким образом, что любые два элемента, принадлежащие различным тройкам, будут совместимы, а любые два элемента, взятые из одной и той же тройки, будут несовместимы. В этом случае элементами множества P будут служить такие подмножества из A, каждое из которых состоит из

10 элементов, выбранных из различных троек. Очевидно, что число таких различных подмножеств равно $3^{10} = 59049$.

В связи с этим рассмотрим иной метод получения точного решения нашей задачи, являющийся развитием метода прямого размещения и свободный от указанных недостатков.

Раньше уже говорилось, что источником возможного ухудшения качества решения является реализация сомнительных шагов, которые приходится делать в критических ситуациях, где неприменимы правила 1 и 2 нахождения хороших шагов. Между тем, в таких ситуациях всегда можно найти некоторую совокупность шагов, каждый из которых в отдельности является сомнительным, но среди которых должен находиться по крайней мере один хороший шаг. Назовем такую совокупность *полной*. Какой именно из шагов полной совокупности окажется хорошим, заранее не известно, но если мы переберем их по очереди и рассмотрим соответствующие им продолжения процесса вычислений, то мы найдем, в конце концов, оптимальное решение.

Производя подобный перебор в каждой из возникающих критических ситуаций, мы реализуем процесс, отображаемый деревом поиска, знакомым нам по предыдущему параграфу. Там же были рассмотрены некоторые приемы сокращения дерева поиска и ускорения таким путем процесса вычислений. Двумя из них мы воспользуемся и сейчас: во-первых, среди различных полных совокупностей, соответствующих некоторой анализируемой критической ситуации, будем выбирать по возможности такие, которые состоят из минимального числа сомнительных шагов, во-вторых, если в процессе поиска уже получено некоторое решение, то последующий перебор будем проводить лишь в тех границах, где может быть найдено лучшее решение.

Минимизация полной совокупности достигается при нахождении максимального по числу единиц столбца матрицы $\|B * R\|$. Этому столбцу соответствует отдыхающий, усадить которого труднее всех. Полная совокупность образуется вариантами посадки данного отдыхающего в совместимые с ним и уже частично загруженные лодки и вариантом взятия для него новой лодки.

Пример нахождения точного решения. Допустим, что

$$\|A*A\|=\begin{array}{c}\begin{matrix}1&2&3&4&5&6&7&8&9&10\end{matrix}\\ \left[\begin{array}{cccccccccc}0&1&0&0&0&1&0&1&0&1\\1&0&0&1&1&0&0&0&0&0\\0&0&0&0&0&0&0&1&0&0\\0&1&0&0&1&0&0&0&1&1\\0&1&0&1&0&0&1&0&1&0\\1&0&0&0&0&0&0&0&1&0\\0&0&1&0&1&0&0&0&0&1\\1&0&0&0&0&0&0&0&0&0\\0&0&0&1&1&1&0&0&0&1\\1&0&0&1&0&0&1&0&1&0\end{array}\right]\begin{matrix}1\\2\\3\\4\\5\\6\\7\\8\\9\\10\end{matrix}\end{array}$$

Следуя правилу 1, реализуем шаг $1a$ (усаживаем отдыхающего r_1 в лодку a), а также шаг $10b$, затем, согласно правилу 2, реализуются шаги $3b$, $6b$ и $8b$, после чего возникнет следующая критическая ситуация:

$$\|R*R\|=\begin{array}{c}\begin{matrix}2&4&5&7&9\end{matrix}\\ \left[\begin{array}{ccccc}0&1&1&0&0\\1&0&1&0&1\\1&1&0&1&1\\0&0&1&0&0\\0&1&1&0&0\end{array}\right]\begin{matrix}2\\4\\5\\7\\9\end{matrix}\end{array}\qquad \|B*R\|=\begin{array}{c}\begin{matrix}2&4&5&7&9\end{matrix}\\ \left[\begin{array}{ccccc}1&0&0&0&0\\0&1&0&1&1\end{array}\right]\begin{matrix}a\\b\end{matrix}\end{array}$$

Выберем левый среди максимальных столбцов матрицы $\|B*R\|$, соответствующий отдыхающему r_2. Этого отдыхающего можно усадить или в лодку b, или взять для него новую лодку c. Рассмотрим пока первый из этих вариантов: реализуем шаг $2b$, после чего возникнет критическая ситуация:

$$\|R*R\|=\begin{array}{c}\begin{matrix}4&5&7&9\end{matrix}\\ \left[\begin{array}{cccc}0&1&0&1\\1&0&1&1\\0&1&0&0\\1&1&0&0\end{array}\right]\begin{matrix}4\\5\\7\\9\end{matrix}\end{array}\qquad \|B*R\|=\begin{array}{c}\begin{matrix}4&5&7&9\end{matrix}\\ \left[\begin{array}{cccc}0&0&0&0\\1&1&1&1\end{array}\right]\begin{matrix}a\\b\end{matrix}\end{array}$$

Здесь процесс размещения ветвится аналогичным образом на отдыхающем r_4, которого можно усадить или в лодку a, или взять для него лодку c. Рассмотрение первого из этих вариантов и реализация шага $4a$ приводят к последующей серии хороших шагов $5c$, $9d$ и $7a$,

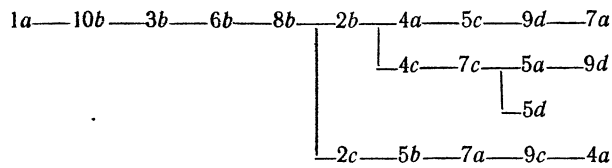
завершающих процесс размещения. При этом оказываются занятыми всего четыре лодки.

Вернемся к последней точке ветвления процесса вычислений и вместо шага $4a$ попробуем реализовать шаг $4c$. После этого находится хороший шаг $7c$, затем процесс вновь ветвится: возникает проблема выбора между шагами $5a$ и $5d$. Как первый из них, влекущий за собой хороший шаг $9d$, так и шаг $5d$ не приводят к решению лучшему, чем найденное ранее. Поэтому исследование данной ветви можно прекратить и, закончив таким образом перебор вариантов размещения отдыхающего r_4, возвратиться к рассмотрению предыдущей точки ветвления. Реализовав теперь шаг $2c$, мы, не сталкиваясь более с критическими ситуациями, последовательно находим и реализуем шаги $5b$, $7a$, $9c$ и $4a$, получая в результате размещение в трех лодках:

$$\| Q \Rightarrow A \| = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Поскольку перебор во всех критических ситуациях закончен, полученное размещение следует считать наилучшим.

Описанный процесс отображается следующим деревом поиска:



§ 3. Задачи диагностики

Диагностическая матрица. Рассмотрим два множества $A = \{a_1, a_2, \ldots, a_n\}$ и $B = \{b_1, b_2, \ldots, b_m\}$ некоторых явлений (событий), связанных отношением причинности: будем говорить, что явление a_i *влечет* (*имплицирует*) явление b_j ($a_i \to b_j$), если из факта существования явления a_i неизбежно следует существование явления

b_j, и явление a_i *отрицает* явление b_j $(a_i \to \bar{b}_j)$, если из факта существования явления a_i неизбежно следует отсутствие явления b_j. Можно считать, что множество A является множеством *причин*, а B — множеством *следствий*. Можно также рассматривать A как множество некоторых *явлений*, а B — как множество сопутствующих им *признаков* (в данном параграфе мы будем пользоваться именно такой терминологией).

Заслуживают внимания и более конкретные интерпретации. Например, A может быть множеством болезней, а B — множеством симптомов. Множество A может представлять собой совокупность всех возможных повреждений в некотором техническом устройстве, а через B может быть обозначено множество внешних признаков, по которым мы будем судить об исправности устройства.

Положим, что элементы множества B играют здесь роль некоторых величин, которые мы можем наблюдать, а элементы множества A — роль событий, о которых мы должны догадываться. Положим также, что вся информация, на основе которой мы можем строить свои предположения, или, как говорят, *ставить диагноз*, содержится в булевой матрице C, называемой *диагностической матрицей*. В этом параграфе мы ограничимся рассмотрением случая полной связи между множествами явлений A и B, когда для каждой пары явлений $a_i \in A$ и $b_j \in B$ имеет место или отношение $a_i \to b_j$ (в этом случае элемент c_i^j диагностической матрицы C принимает значение 1), или отношение $a_i \to \bar{b}_j$ (в этом случае $c_i^j = 0$).

Например, первая строка матрицы

$$C = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix}$$

показывает, что явление a влечет явления 1, 4, 5, 7 и 8 (или, если угодно, характеризуется признаками 1, 4, 5, 7 и 8) и отрицает явления 2, 3 и 6 (то есть характеризуется отсутствием признаков 2, 3 и 6).

Диагностические тесты. Будем говорить, что признак b_k *различает* явления a_i и a_j, если выполняется одно из следующих условий:

а) $a_i \to b_k$ и $a_j \to \bar{b}_k$,

б) $a_i \to \bar{b}_k$ и $a_j \to b_k$.

Будем говорить, что некоторая совокупность признаков $B_p \subseteq B$ *различает* подмножество явлений $A_q \subseteq A$, если для каждой пары a_i, $a_j \in A_q$ найдется различающий их признак $b_k \in B_p$. Совокупность $B_p \subseteq B$, различающую множество явлений A, назовем *безусловным диагностическим тестом* для A.

Аналогичные формулировки можно давать и при рассмотрении булевых матриц, интерпретируемых как диагностические матрицы. Например, будем говорить, что некоторый столбец булевой матрицы различает две ее строки, если одна из них содержит в данном столбце единицу, а другая — нуль. По аналогии определяется смысл утверждений «некоторое подмножество столбцов матрицы различает заданное подмножество ее строк» и «некоторая совокупность столбцов матрицы представляет безусловный диагностический тест».

Очевидна справедливость следующего утверждения: для того чтобы некоторая совокупность столбцов диагностической матрицы являлась безусловным диагностическим тестом, необходимо и достаточно, чтобы образованный данными столбцами минор не содержал одинаковых строк.

Любое подмножество B_p из B порождает некоторое разбиение A/B_p множества A, а именно, такое множество, элементами которого служат взаимно непересекающиеся подмножества из A, образованные из явлений, неразличимых между собой по признакам из B_p. Этим подмножествам будут соответствовать совокупности одинаковых строк минора диагностической матрицы C, образованного из столбцов, соответствующих элементам подмножества B_p. Очевидно, что подмножество B_p будет безусловным диагностическим тестом, если порождаемое им разбиение A/B_p будет полным, то есть если это разбиение будет образовано из подмножеств, каждому из которых будет принадлежать лишь один элемент из A.

Рассмотрим задачу опознавания некоторого заданного явления (элемента множества A) по совокупности

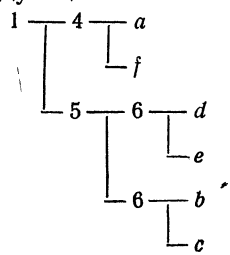
признаков (элементов множества B), считая, что матрица C нам известна. Процесс опознавания состоит в проверке значений некоторых признаков и в последующем умозаключении. При этом под проверкой значений признаков мы будем понимать процедуру проверки наличия или отсутствия признаков, ставя в соответствие наличию признака значение 1, а его отсутствию — значение 0.

Очевидно, что проверка значений признаков, образующих безусловный диагностический тест, дает информацию, достаточную для решения рассматриваемой задачи. Например, решая задачу опознавания для приведенной выше матрицы C, достаточно ограничиться рассмотрением первых шести признаков, соответствующих первым шести столбцам этой матрицы. Действительно, образованный этими столбцами минор

$$\begin{array}{c} 1\,2\,3\,4\,5\,6 \\ \left|\begin{array}{l} 1\,0\,0\,1\,1\,0 \\ 0\,0\,1\,1\,0\,1 \\ 0\,1\,0\,1\,0\,0 \\ 0\,0\,0\,1\,1\,1 \\ 0\,0\,0\,1\,1\,0 \\ 1\,0\,1\,0\,1\,0 \end{array}\right| \begin{array}{l} a \\ b \\ c \\ d \\ e \\ f \end{array} \end{array}$$

не содержит одинаковых строк — выбранное подмножество столбцов различает все множество строк диагностической матрицы и является, следовательно, безусловным диагностическим тестом.

Процесс опознавания может иметь динамический характер, когда выбор очередного проверяемого признака ставится в зависимость от результатов проверки значений предыдущих (при которой устанавливается наличие или отсутствие этих признаков). Например, для той же матрицы C можно предложить алгоритм опознавания, представляемый следующей схемой:



В этой схеме верхние ветви, исходящие из точек ветвления, соответствуют наличию проверяемого признака, нижние — отсутствию.

Пусть, например, имеет место явление *е*. На первом шаге алгоритма устанавливается отсутствие признака 1, в связи с чем последующая реализация алгоритма соответствует рассмотрению нижней ветви схемы. На втором шаге устанавливается наличие признака 5, и поэтому далее рассматривается верхняя ветвь, исходящая из текущей точки ветвления. Наконец, на третьем шаге устанавливается отсутствие признака 6 и тем самым явление полностью опознается.

Описанный алгоритм относится к *условным диагностическим тестам*, изучение которых порождает много интересных задач. Мы ограничимся, однако, рассмотрением безусловных диагностических тестов, называя их далее просто *тестами*.

Организация самой возможности проверки значения некоторого признака уже может быть связана с определенными затратами. Считая, что эти затраты одинаковы для каждого из признаков, разумно поставить задачу минимизации числа признаков, образующих тест. Назовем ее задачей нахождения *минимального теста*.

В данном параграфе будет рассмотрено несколько алгоритмов решения этой задачи, являющейся довольно типичным, хотя и далеко не единственным представителем проблематики нового научного направления, называемого *диагностикой*.

Нахождение минимального теста. Алгоритм нахождения точного решения поставленной задачи достаточно прост: сначала следует получить *матрицу различий*, каждая строка которой соответствует некоторой паре строк диагностической матрицы и показывает, какими компонентами эти строки отличаются, а затем найти для полученной матрицы кратчайшее покрытие, только не строчное, что рассматривалось раньше, а *столбцовое*, то есть состоящее из некоторых столбцов, покрывающих в совокупности все строки данной матрицы. Множество признаков, соответствующих элементам найденного покрытия, будет искомым решением.

Матрица различий легко получается путем перебора всех пар строк диагностической матрицы и вычисления,

с помощью оператора $\oplus$, соответствующих строк матрицы различий. Для рассматриваемого примера она примет следующий вид (справа показаны порождающие пары строк диагностической матрицы):

$$\begin{array}{c} 12345678 \\ \left[\begin{array}{c} 10101110 \\ 11001011 \\ 10000110 \\ 10000000 \\ 00110000 \\ 01100101 \\ 00101000 \\ 00101110 \\ 10011110 \\ 01001101 \\ 01001011 \\ 11111011 \\ 00000110 \\ 10110110 \\ 10110000 \end{array}\right. \begin{array}{l} ab \\ ac \\ ad \\ ae \\ af \\ bc \\ bd \\ be \\ bf \\ cd \\ ce \\ cf \\ de \\ df \\ ef \end{array} \end{array}$$

Прежде чем искать столбцовое покрытие, данную матрицу можно существенно упростить, выбросив из нее все поглощающие строки и поглощаемые столбцы. Напомним, что булев вектор a поглощает булев вектор b, если все компоненты вектора a, соответствующие единичным компонентам вектора b, также имеют значение 1. Заметим также, что поглощающая строка может выбрасываться из матрицы только в том случае, когда поглощаемая ею строка остается в матрице. Аналогичное замечание будет справедливо и в отношении выбрасываемых столбцов.

В результате выбрасывания поглощающих строк получаем матрицу

$$\begin{array}{c} 12345678 \\ \left[\begin{array}{c} 10000000 \\ 00110000 \\ 00101000 \\ 01001101 \\ 01001011 \\ 00000110 \end{array}\right] \end{array}$$

Последующее удаление поглощаемых столбцов приводит нас к матрице

$$\begin{array}{c} 1\,3\,5\,6\,7 \\ \left[\begin{array}{c} 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,0 \\ 0\,0\,1\,1\,0 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{array}\right] \end{array}$$

В матрице вновь появляется поглощающая строка, которую, следовательно, можно выбросить:

$$\begin{array}{c} 1\,3\,5\,6\,7 \\ \left[\begin{array}{c} 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,1\,0 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{array}\right] \end{array}$$

Затем без труда находится одно из кратчайших покрытий этой матрицы, которое будет кратчайшим покрытием и для исходной матрицы различий. Таким покрытием является, например, совокупность столбцов с номерами 1, 3, 5 и 6. Соответствующая совокупность признаков является минимальной для заданной диагностической матрицы.

Таким образом, основной операцией в рассмотренном методе получения минимального теста служит нахождение кратчайшего столбцового покрытия. Рассмотрим Л-программу решения этой задачи *).

Алгоритм нахождения кратчайшего столбцового покрытия. В основу описываемого ниже алгоритма положена уже знакомая нам идея обхода дерева поиска. Алгоритм обладает, однако, рядом отличительных особенностей.

Разложение по минимальному столбцу заменяется разложением по минимальной строке (коль скоро теперь ищутся столбцовые покрытия). В алгоритме отсутствуют операции устранения поглощающих строк и поглощаемых столбцов, обычно применяемые для упрощения об-

*) Приводимые в этом параграфе Л-программы принадлежат А. А. Уткину, так же как и результаты их экспериментального исследования.

рабатываемой матрицы (как это было показано на только что рассмотренном примере). Применение этих операций было сочтено невыгодным, так как они являются довольно трудоемкими и в то же время, как показали эксперименты на машине, в большинстве случаев не приводят к существенному сокращению размеров матрицы. Впрочем, затронутый вопрос является дискуссионным, и можно предположить, что при определенных условиях (например, в ситуациях, соответствующих вершинам дерева поиска, ближе расположенным к корню дерева) использование указанных операций окажется вполне целесообразным.

Л-программа, реализующая данный алгоритм, оформлена в виде подпрограммы, названной окрапок5 и позволяющей обрабатывать матрицы с числом столбцов до 32 и с числом строк, определяемым емкостью оперативной памяти машины. Существенной технической особенностью программы окрапок5 является использование приема, названного *сепарацией строк* и используемого для экономного запоминания непокрытой части матрицы: при включении нового столбца в строящееся столбцовое покрытие строки матрицы переставляются так, что все непокрытые строки оказываются в конце матрицы. Таким образом, для запоминания непокрытых остатков матрицы не требуется привлечения дополнительных комплексов, кроме комплекса, задающего исходную матрицу C, причем мощность этого комплекса фиксирована, будучи равной числу строк в матрице C.

Как и в программе окрапок4, мощность перебираемых подмножеств (в данном случае — столбцов) ограничивается величиной $k-2$, если ищутся покрытия мощности k, и этот перебор дополняется поиском двухэлементных покрытий непокрытых остатков матрицы.

Таким образом, в алгоритме выделяются следующие три основные операции: нахождение минимальной строки среди непокрытого остатка матрицы (попутно ищется одностолбцовое покрытие этого остатка), выделение непокрытого остатка строк (сепарация) и поиск двустолбцового покрытия остатка.

Переходим к рассмотрению соответствующих подпрограмм.

Вспомогательные подпрограммы.

Нахождение минимальной строки строчного минора булевой матрицы — министро

1. В булевой матрице $\boldsymbol{A}$ задан строчный минор, состоящий из соседних строк, начиная со строки номер p и кончая строкой номер q. Требуется найти минимальную (по числу единиц) строку $\boldsymbol{a}_i$ минора. Если минимальных строк будет несколько, выбрать первую из них.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{A}$,

$[\beta_и] = p$,

$[\gamma_и] = q + 1$,

$\delta_п^+ :: \boldsymbol{a}_i$.

Задается: a_α.

Внутренние операнды: —, c, —.

4.

```
        * 001 173
§ 0   β − 1 ⇒ a ⊂ b
§ 1   △ α − γ ⟼ 2 α_a ▽ ⇒ c − b ⟼ 1
      c ⇒ b α_a ⇒ δ → 1
§ 2   .
```

Разделение (*сепарация*) *строк минора булевой матрицы по заданному подмножеству столбцов* — сепара

1. В булевой матрице $\boldsymbol{A}$ задан строчный минор, состоящий из соседних строк, начиная со строки номер p и кончая строкой номер q. Из множества S всех столбцов матрицы $\boldsymbol{A}$ выделено некоторое подмножество S_i. Требуется произвести такую перестановку строк минора, чтобы все его строки, не содержащие единиц ни в одном из выделенных столбцов, оказались в конце минора, и выделить затем минор, состоящий из таких строк.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{A}$,

$[\beta_и] = p$,

$[\gamma_и] = q + 1$,

$\delta_п :: \| \{S_i\} \ni S \|$,

$e_п^+$ — номер (отсчитываемый в результативном значении матрицы $\boldsymbol{A}$) первой строки выделенного минора.

Задается: a_α.
Внутренние операнды: —, —, —.

4.

* 001 174

§ 0 $\beta \Rightarrow \varepsilon - 1 \Rightarrow \beta$

§ 1 $\triangle \beta - \gamma \mapsto 2\alpha_\beta \wedge \delta \circ\!\!\rightarrow 1\alpha_\beta \Leftrightarrow \alpha_\varepsilon \triangle \varepsilon \rightarrow 1$

§ 2 .

Нахождение одностолбцовых покрытий минора булевой матрицы — о д н о с т о п о к

1. В булевой матрице A, совокупность столбцов которой образует множество S, задан строчный минор, состоящий из соседних строк, начиная со строки номер p и кончая строкой номер q. Требуется найти множество S_i столбцов, каждый из которых содержит единицы во всех строках минора, то есть найти множество одностолбцовых покрытий минора.

2. Внешние операнды:

$\alpha_к :: A$,

$[\beta_и] = p$,

$[\gamma_и] = q + 1$,

$\delta_п^+ :: \| \{S_i\} \Rightarrow S \|$.

Задается: a_α.
Внутренние операнды: —, a, —.
Примечание: предполагается, что число строк в миноре не равно нулю.

4.

* 001 172

§ 0 $\beta - 1 \Rightarrow a \bar{\circ} \delta$

§ 1 $\triangle a - \gamma \mapsto 2\alpha_a \wedge \delta \Rightarrow \delta \mapsto 1$

§ 2 .

Нахождение двустолбцового покрытия минора булевой матрицы — д в у с т о п о к

1. В булевой матрице A, совокупность столбцов которой образует множество S, задан строчный минор, состоящий из соседних строк, начиная со строки номер p

и кончая строкой номер q. Требуется найти покрытие S_2 ($S_2 \subseteq S$) этого минора, состоящее из двух столбцов, или убедиться, что такого покрытия нет. Задается также некоторая строка a_i минора — очевидно, что искомое двустолбцовое покрытие, если оно существует, должно содержать по крайней мере один из столбцов, имеющих единицу в строке a_i.

2. Внешние операнды:

$\alpha_к :: A$,

$[\beta_и] = p$,

$[\gamma_и] = q + 1$,

$\delta_п^- :: a_i$,

$\varepsilon_п^+ :: \| \{S_2\} \ni S \|$, если покрытие S_2 существует,

0 — в противном случае.

Задается: a_α.

Внутренние операнды: —, b, —.

4.

* 001 175

§ 0 $\circ \varepsilon$

§ 1 $\delta \dot{X} 4 a \beta - 1 \Rightarrow b \bar{\circ} \varepsilon$

§ 2 $\triangle b - \gamma \longmapsto 3 \alpha_b \wedge c_a \longmapsto 2$

$\alpha_b \wedge \varepsilon \Rightarrow \varepsilon \circ\!\!\rightarrow 1 \rightarrow 2$

§ 3 $\varepsilon \vdash \Rightarrow b\, c_a \vee c_b \Rightarrow \varepsilon$

§ 4 .

Программа окрапок5. Л-программа решения рассматриваемой задачи в целом выглядит следующим образом.

Нахождение кратчайшего столбцового покрытия булевой матрицы — окрапок5

1. Решается задача нахождения одного из кратчайших столбцовых покрытий булевой матрицы A.

2. Внешние операнды:

$\alpha_к :: A$,

$\beta_к$ — комплекс памяти, $\sigma(\beta) = 32$,

$\gamma_к$ — комплекс памяти, $\sigma(\gamma) = 32$,

$\delta_{\text{п}}^{+}::\|\{S_i\}\Rightarrow S\|$, где S_i — подмножество из множества S всех столбцов матрицы $\boldsymbol{A}$, являющееся кратчайшим столбцовым покрытием.

Задаются: a_α, a_β, a_γ, b_α.

Подпрограммы: одностопок, министро, сепара, двустопок.

Внутренние операнды: $\boldsymbol{b}$, h, —.

Примечание: считается, что столбцовое покрытие существует.

3. Основные идеи алгоритма уже были описаны. Поясним лишь смысл некоторых из внутренних операндов.

Значение индекса e отмечает текущую глубину, то есть номер яруса дерева поиска, из которого извлекается рассматриваемая в текущий момент времени вершина. Индекс d задает предельную глубину поиска (определяемую мощностью уже найденного покрытия). Индекс f используется для запоминания числа строк в матрице $\boldsymbol{A}$. Смысл комплексов β и γ: значением i-го элемента комплекса β служит номер первой из строк матрицы $\boldsymbol{A}$, входящих в непокрытый на i-й глубине остаток матрицы, единичные компоненты i-го элемента комплекса γ отмечают нерассмотренные еще столбцы, содержащие единицы в минимальной строке, найденной на i-й глубине.

4.

* **001 176**

§ 0 $\overline{\circ}\, d \circ e\, b_\alpha \Rightarrow f \circ \beta_0\, c_0 \Rightarrow \gamma_0$

§ 1 $e - d \mapsto 5\, \beta_e \Rightarrow h \triangle e\, 2 \Rightarrow g$

* одностопок $\alpha\, h\, f\, \boldsymbol{a}\, //\, \circ\, \boldsymbol{b}\, \boldsymbol{a} \Rightarrow \gamma_e \mapsto 2$

* министро $\alpha\, h\, f\, \boldsymbol{b}\, // \to 4$

§ 2 $\boldsymbol{b} \Rightarrow \delta\, e + 1 \Rightarrow h - 2 - g \Rightarrow e \Rightarrow d \circ b$

§ 3 $\triangle\, b - h \mapsto 5\, \gamma_b \vdash \Rightarrow a\, c_a \vee \delta \Rightarrow \delta \to 3$

§ 4 $\boldsymbol{b} \Rightarrow \gamma_e \vdash \Rightarrow a\, c_a \Rightarrow \boldsymbol{a}\, e - 1 \Rightarrow a\, \beta_a \Rightarrow a$

* сепара $\alpha\, a\, f\, \boldsymbol{a}\, b\, //\, b \Rightarrow \beta_e \to 1$

§ 5 $e \wedge c_0 \mapsto 6\, \beta_e \Rightarrow h\, \alpha_h \Rightarrow \boldsymbol{a}$

* двустопок $\alpha\, h\, f\, \boldsymbol{a}\, \boldsymbol{b}\, //\, \circ\, g\, \boldsymbol{b} \mapsto 2$

$\gamma_e \vdash \Rightarrow a\, c_a \oplus \gamma_e \Rightarrow \boldsymbol{b} \mapsto 4\, \overline{\triangle}\, e \to 5$

§ 6 .

Приближенный алгоритм. Рассмотрим алгоритм построения приближенно-минимального теста, несколько напоминающий метод прямого размещения отдыхающих по лодкам, изложенный в предыдущем параграфе.

Как и в упомянутом методе, реализация алгоритма распадается на серию шагов, связывающих следующие друг за другом ситуации. Очередная, i-я ситуация будет в данном случае характеризоваться некоторой построенной на предыдущих шагах совокупностью B_i, выбранной из множества признаков B и состоящей ровно из i элементов. Если разбиение A/B_i множества A, порождаемое данной совокупностью, является полным, то это означает, что совокупность B_i является тестом и, следовательно, ее можно принять за искомое решение. В противном случае данную совокупность следует расширить, добавив к ней на i-м шаге (приводящем от i-й ситуации к $i+1$-й) один признак из множества $B \setminus B_i$, «лучший» в каком-то смысле.

Очевидно, что лучшим в полном смысле этого слова был бы такой признак, выбор которого привел бы впоследствии к построению строго минимального теста. Если бы на каждом шаге выбирался именно такой признак, алгоритм стал бы точным. Однако нахождение такого признака представляет собой задачу того же порядка трудности, что и сама задача нахождения строго минимального теста. В связи с этим в предлагаемом алгоритме выбирается признак, лишь условно называемый «лучшим».

Критерии выбора очередного признака в i-й ситуации могут быть различными. Например, «лучшим» может быть назван признак, который различает максимальное число пар элементов множества A, неразличимых совокупностью B_i. При этом он может выбираться

а) из множества $B \setminus B_i$,

б) из множества признаков, различающих некоторую произвольно взятую пару элементов множества A, неразличаемых в i-й ситуации (чем меньше таких признаков, тем легче их перебор; в то же время очевидно, что в любой тест должен входить по крайней мере один из таких признаков),

в) из множества признаков, различающих пару наиболее «близких» элементов из множества A, неразличимых в i-й ситуации («расстояние» здесь измеряется числом признаков, различающих данную пару).

Ниже будут рассмотрены Л-программы, соответствующие перечисленным вариантам и названные бедиэкс1, бедиэкс2 и бедиэкс3, соответственно (как сокращение от «безусловный диагностический эксперимент»).

Интересно сравнить их с методом поиска приближения к кратчайшему столбцовому покрытию матрицы различий, получаемому из диагностической матрицы.

Программа бедиэкс1 соответствует такому варианту указанного поиска, при котором каждый шаг алгоритма полностью определяется выбором максимального столбца в непокрытом остатке матрицы различий. Программа бедиэкс3, кроме того, при поиске пары наиболее «близких» строк в диагностической матрице выполняет работу, эквивалентную поиску минимальной строки в матрице различий. Программа бедиэкс2 отличается от нее тем, что вместо «ближайших» строк рассматривается произвольная пара строк диагностической матрицы, неразличимых в текущей ситуации, что соответствует выбору произвольной строки в непокрытом остатке матрицы различий.

Как показали выполненные на машине эксперименты, именно программа бедиэкс2 оказалась наилучшей с практической точки зрения.

Пример. Следуя алгоритму бедиэкс2, найдем тест для уже знакомой нам диагностической матрицы

$$C = \begin{array}{c} \begin{matrix} 1\,2\,3\,4\,5\,6\,7\,8 \end{matrix} \\ \left[\begin{array}{cc} 1\,0\,0\,1\,1\,0\,1\,1 & a \\ 0\,0\,1\,1\,0\,1\,0\,1 & b \\ 0\,1\,0\,1\,0\,0\,0\,0 & c \\ 0\,0\,0\,1\,1\,1\,0\,1 & d \\ 0\,0\,0\,1\,1\,0\,1\,1 & e \\ 1\,0\,1\,0\,1\,0\,1\,1 & f \end{array}\right] \end{array}.$$

Первый шаг. Выбираем в матрице C первую пару строк a, b и находим «лучший» из различающих ее столбцов. Им оказывается столбец 7, различающий девять пар строк (это число определяется как произведение

числа единиц на число нулей в данном столбце). Матрица C распадается на два минора:

$$\begin{array}{c} 1\,2\,3\,4\,5\,6\,7\,8 \\ \left[\begin{array}{c} 1\,0\,0\,1\,1\,0\,1\,1 \\ 0\,0\,0\,1\,1\,0\,1\,1 \\ 1\,0\,1\,0\,1\,0\,1\,1 \end{array}\right] \begin{array}{c} a \\ e \\ f \end{array} \\ \left[\begin{array}{c} 0\,0\,1\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,1\,1\,1\,0\,1 \end{array}\right] \begin{array}{c} b \\ c \\ d \end{array} \end{array}$$

Второй шаг. Выбираем первую пару строк в первом из этих миноров и находим единственный различающий эту пару столбец 1. Этот столбец дает следующее разложение матрицы C:

$$\begin{array}{c} 1\,2\,3\,4\,5\,6\,7\,8 \\ \left[\begin{array}{c} 1\,0\,0\,1\,1\,0\,1\,1 \\ 1\,0\,1\,0\,1\,0\,1\,1 \end{array}\right] \begin{array}{c} a \\ f \end{array} \\ [0\,0\,0\,1\,1\,0\,1\,1]\ e \\ \left[\begin{array}{c} 0\,0\,1\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,1\,1\,1\,0\,1 \end{array}\right] \begin{array}{c} b \\ c \\ d \end{array} \end{array}$$

Третий шаг. Выбираем пару строк a, f и находим «лучший» среди различающих ее столбцов (им оказывается столбец 3, который различает пару в первом миноре и две пары в последнем). Получаем разбиение:

$$\begin{array}{c} 1\,2\,3\,4\,5\,6\,7\,8 \\ [1\,0\,0\,1\,1\,0\,1\,1]\ a \\ [1\,0\,1\,0\,1\,0\,1\,1]\ f \\ [0\,0\,0\,1\,1\,0\,1\,1]\ e \\ [0\,0\,1\,1\,0\,1\,0\,1]\ b \\ \left[\begin{array}{c} 0\,1\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,1\,1\,1\,0\,1 \end{array}\right] \begin{array}{c} c \\ d \end{array} \end{array}$$

Четвертый шаг. Наконец, выбираем последнюю пару c, d и первый из различающих ее столбцов 2, после чего разбиение матрицы становится полным. Следовательно, совокупность столбцов {7, 1, 3, 2} является тестом.

Вспомогательные подпрограммы. Порождаемое совокупностью B_i разбиение A/B_i множества A

условимся представлять соответствующим разбиением диагностической матрицы $\boldsymbol{C}$ на строчные миноры, составленные из рядом расположенных строк матрицы $\boldsymbol{C}$ (будем называть подобные матрицы *секционированными*). На каждом шаге, когда в совокупность B_i добавляется некоторый новый признак, каждый из указанных миноров разлагается, в общем случае, на два новых строчных минора, первый из которых содержит в столбце, соответствующем новому признаку, только единицы, второй — только нули. Этот процесс сопровождается такой перестановкой строк внутри разлагаемых миноров, чтобы новые строчные миноры также состояли из рядом расположенных строк.

Заметим, что продолжая решение задачи, нет смысла рассматривать миноры, целиком состоящие из одинаковых строк (в частности, содержащие всего одну строку), так как очевидно, что не существует признаков, различающих строки таких миноров. Предусмотрим в связи с этим процедуру выбрасывания таких миноров из преобразуемой матрицы.

В описываемых ниже подпрограммах преобразуемая от шага к шагу секционированная диагностическая матрица $\boldsymbol{C}$ представляется тремя комплексами: комплексом α представляется собственно матрица $\boldsymbol{C}$, а комплексы β и γ задают текущее разбиение ее на строчные миноры. Значения j-х элементов этих комплексов задают, соответственно, номер первой строки (начало) и увеличенный на единицу номер последней строки (конец) j-го минора. Через S обозначается множество всех столбцов матрицы $\boldsymbol{C}$.

Удаление строчных миноров, состоящих из одинаковых строк — у д а м о с

1. Из секционированной матрицы $\boldsymbol{C}$ удаляются строчные миноры, состоящие из одинаковых (неразличимых) строк.

2. Внешние операнды:

$\alpha_к$, $\beta_к$, $\gamma_к$ — комплексы, задающие секционированную матрицу $\boldsymbol{C}$.

Задаются: a_α, a_β, a_γ, b_β.

Внутренние операнды: $\boldsymbol{a}$, c, $-$.

4.

* **001 034**

§ 0 $\overline{\circ} b\, b_\beta \Rightarrow c$

§ 1 $\triangle b - c \longmapsto 4\,\beta_b \Rightarrow a\,\alpha_a \Rightarrow a$

§ 2 $\triangle a - \gamma_b \longmapsto 3\,\alpha_a \oplus a \circ\!\!\rightarrow 2 \rightarrow 1$

§ 3 $\overline{\triangle}\, c\,\beta_c \Rightarrow \beta_b\,\gamma_c \Rightarrow \gamma_b\,\overline{\triangle}\, b \rightarrow 1$

§ 4 $c \Rightarrow b_\beta \Rightarrow b_\gamma$.

Вычисление оценки столбца секционированной матрицы — о ц е с т о

1. Вычисляется оценка $\triangle M$ заданного столбца s_i секционированной булевой матрицы $\boldsymbol{C}$ по следующей формуле:

$$\triangle M = \sum_{j \in J} N_j^0 N_j^1,$$

где N_j^0 и N_j^1 — число нулей и, соответственно, число единиц в данном столбце j-го строчного минора, J — множество номеров строчных миноров, а $\triangle M$ — приращение числа M различаемых пар строк, достигаемое включением данного столбца в строящийся тест.

2. Внешние операнды:

$\alpha_к$, $\beta_к$, $\gamma_к$ — комплексы, задающие секционированную матрицу $\boldsymbol{C}$,

$\delta_п :: \| \{\{s_i\}\} \ni S \|$,

$[\varepsilon_и^+] = \triangle M$.

Задаются: a_α, a_β, a_γ, b_β.

Внутренние операнды: —, c, —.

4.

* **001 036**

§ 0 $\overline{\circ} b \circ \varepsilon$

§ 1 $\triangle b - b_\beta \longmapsto 4\,\beta_b - 1 \Rightarrow a \circ c$

§ 2 $\triangle a - \gamma_b \longmapsto 3\,\alpha_a \wedge \delta \circ\!\!\rightarrow 2\,\triangle c \rightarrow 2$

§ 3 $\gamma_b - \beta_b - c \times c + \varepsilon \Rightarrow \varepsilon \rightarrow 1$

§ 4 .

Разбиение строчных миноров секционированной матрицы — разбимат

1. Каждый из строчных миноров секционированной булевой матрицы C разлагается, в общем случае, на два новых минора, в первый из которых входят строки, содержащие в столбце s_i единицу, во второй — строки, содержащие в этом столбце нуль. При этом производится такая перестановка строк внутри разлагаемого минора, чтобы новые миноры также состояли из рядом расположенных строк матрицы C.

2. Внешние операнды:

$\alpha_к$, $\beta_к$, $\gamma_к$ — комплексы, задающие секционированную матрицу C,

$\delta_п :: \| \{\{s_i\}\} \ni S \|$.

Задаются: a_α, a_β, a_γ, b_β.

Подпрограмма: сепара.

Внутренние операнды: —, d, —.

4.

* **001 032**

§ 0 $\bar{\circ} b\, b_\beta \Rightarrow c$

§ 1 $\triangle\, b - b_\beta \longmapsto 2\beta_b \Rightarrow a$

* сепара $\alpha\, a\, (\gamma_b)\, \delta\, d$ //

$\gamma_b \Rightarrow \gamma_c\, d \Rightarrow \gamma_b \Rightarrow \beta_c \triangle\, c \to 1$

§ 2 $c \Rightarrow b_\gamma \Rightarrow b_\beta$.

Нахождение пары ближних строк — минрасхэм

1. Находится пара неодинаковых строк, принадлежащих одному строчному минору секционированной булевой матрицы C и отличающихся своими значениями в минимальном числе столбцов, образующих подмножество S_i из S (числом таких столбцов измеряется расстояние между строками, согласно Хэммингу, почему данная подпрограмма и названа минрасхэм).

2. Внешние операнды:

$\alpha_к$, $\beta_к$, $\gamma_к$ — комплексы, задающие секционированную матрицу C,

$\delta_п^+ :: \| \{S_i\} \ni S \|$.

Задаются: a_α, a_β, a_γ, b_β.
Внутренние операнды: ***a***, *e*, —.

4.

```
        * 001 354
§ 0   ⊖ b ⊖ d
§ 1   △ b − b_β ⟼ 4 β_b − 1 ⇒ c
§ 2   △ c ⇒ a − γ_b ⟼ 1
§ 3   △ a − γ_b ⟼ 2 α_c ⊕ α_a ⇒ a ⟶ 3
      ▽ ⇒ e − d ⟼ 3 e ⇒ d a ⇒ δ → 3
§ 4   .
```

Программы бедиэкс. Описанные разновидности приближенного алгоритма нахождения минимального безусловного теста реализуются следующей серией программ.

Нахождение минимального теста — бедиэкс1, бедиэкс2, бедиэкс3

1. Для заданной булевой матрицы ***C*** находится минимальная (приближенно) совокупность столбцов S_i, различающая все ее неодинаковые строки.

2. Внешние операнды:

$\sigma_к :: \boldsymbol{C}$,

$\beta_к$ — комплекс памяти, $\sigma(\beta) \leqslant \sigma(\alpha)$,

$\gamma_к$ — комплекс памяти, $\sigma(\gamma) \leqslant \sigma(\alpha)$,

$\delta_п^+ :: \| \{S_i\} \ni S \|$, где S — множество всех столбцов матрицы ***C***.

Задаются: a_α, a_β, a_γ, b_α.
Подпрограммы:

в бедиэкс1: удамос, оцесто, разбимат;
в бедиэкс2: удамос, оцесто, разбимат;
в бедиэкс3: удамос, минрасхэм, оцесто, разбимат.

Внутренние операнды:

в бедиэкс1: ***e***, *e*, —;
в бедиэкс2: ***d***, *e*, —;
в бедиэкс3: ***d***, *e*, —.

4.

бедиэкс1:

* **001 201**

§ 0 $\circ\delta 1 \Rightarrow b_\beta \Rightarrow b_\gamma \circ \beta_0 b_\alpha \Rightarrow \gamma_0 \bar{\circ} \boldsymbol{e}$

§ 1 * удамос $\alpha\beta\gamma$ // $b_\beta \circ\!\!\rightarrow 5\ \boldsymbol{e} \Rightarrow \boldsymbol{d} \circ e$

§ 2 $\boldsymbol{d}\,\dot{\mathrm{X}}\,4\,a\,c_a \Rightarrow \boldsymbol{b}$ * оцесто $\alpha\beta\gamma\,\boldsymbol{b}\,d$ // $d \circ\!\!\rightarrow 3\ e - d \mapsto 2\ d \Rightarrow e\,\boldsymbol{b} \Rightarrow \boldsymbol{c} \rightarrow 2$

§ 3 $\boldsymbol{b} \oplus \boldsymbol{e} \Rightarrow \boldsymbol{e} \rightarrow 2$

§ 4 * разбимат $\alpha\beta\gamma\,\boldsymbol{c}$ // $\boldsymbol{c} \vee \delta \Rightarrow \delta \rightarrow 1$

§ 5 .

бедиэкс2:

* **001 202**

§ 0 $\circ\delta 1 \Rightarrow b_\beta \Rightarrow b_\gamma \circ \beta_0 b_\alpha \Rightarrow \gamma_0$

§ 1 * удамос $\alpha\beta\gamma$ // $b_\beta \circ\!\!\rightarrow 5$ $\beta_0 \Rightarrow a\,\alpha_a \Rightarrow \boldsymbol{d}$

§ 2 $\triangle\, a\,\alpha_a \oplus \boldsymbol{d} \circ\!\!\rightarrow 2 \Rightarrow \boldsymbol{d} \circ e$

§ 3 $\boldsymbol{d}\,\dot{\mathrm{X}}\,4\,a\,c_a \Rightarrow \boldsymbol{b}$ * оцесто $\alpha\beta\gamma\,\boldsymbol{b}\,d$ // $e - d \mapsto 3\ d \Rightarrow e\,\boldsymbol{b} \Rightarrow \boldsymbol{c} \rightarrow 3$

§ 4 * разбимат $\alpha\beta\gamma\,\boldsymbol{c}$ // $\boldsymbol{c} \vee \delta \Rightarrow \delta \rightarrow 1$

§ 5 .

бедиэкс3:

* **001 353**

§ 0 $\bar{\circ}\delta 1 \Rightarrow b_\beta \Rightarrow b_\gamma \circ \beta_0 b_\alpha \Rightarrow \gamma_0$

§ 1 * удамос $\alpha\beta\gamma$ // $b_\beta \circ\!\!\rightarrow 4$ * минрасхэм $\alpha\beta\gamma\,\boldsymbol{d}$ // $\circ e$

§ 2 $\boldsymbol{d}\,\dot{\mathrm{X}}\,3\,a\,c_a \Rightarrow \boldsymbol{b}$ * оцесто $\alpha\beta\gamma\,\boldsymbol{b}\,d$ // $e - d \mapsto 2\ d \Rightarrow e\,\boldsymbol{b} \Rightarrow \boldsymbol{c} \rightarrow 2$

§ 3 * разбимат $\alpha\beta\gamma\,\boldsymbol{c}$ // $\boldsymbol{c} \vee \delta \Rightarrow \delta \rightarrow 1$

§ 4 .

Оценки эффективности программ. Интересно сравнить эффективность рассмотренных алгоритмов нахождения теста: одного точного, основанного на применении подпрограммы окрапок5, и трех приближенных — бедиэкс1, бедиэкс2, бедиэкс3.

Для получения соответствующих оценок были поставлены эксперименты (на машине М-20). На вход исследуемых программ подавались серии из l псевдослучайных булевых матриц, генерируемых на базе использования последовательности значений случайной переменной $я$ и играющих роль диагностических матриц. Фиксировались лишь следующие параметры матриц: m — число строк, n — число столбцов и p — средняя плотность единиц (или вероятность того, что произвольно выбранный элемент матрицы имеет значение 1). Для каждой серии эффективность программ оценивалась двумя величинами: s — длиной найденного теста (в числе признаков) и t — временем его построения (в секундах), усредненными по числу l индивидуальных экспериментов в серии.

Результаты испытаний представлены в таблице 2.3.1.

Таблица 2.3.1

($n = 32$, $p = 1/4$)

Программа	8	16	32	64	128	256	512	m
бедиэкс1	3,89	5,76	8,02	11,0	13,6	17,0	21,0	
бедиэкс2	3,88	5,83	8,25	11,3	14,3	17,0	20,4	s
бедиэкс3	3,87	5,69	8,12	10,9	14,1	16,6	20,0	
окрапок5	3,84	5,2	8,0					
бедиэкс1	0,8	1,9	4,6	10,3	23,5	50	107	
бедиэкс2	0,4	0,8	1,8	4,1	7,0	20	54	t
бедиэкс3	0,4	1,5	5,2	20,7	90	350	1300	
окрапок5	1,1	22	720					
бедиэкс1	100	100	50	20	5	5	2	
бедиэкс2	200	200	200	50	10	10	5	l
бедиэкс3	100	100	50	50	10	5	1	
окрапок5	50	5	2					

Прокомментируем полученные результаты.

Границы применимости приближенных алгоритмов оказались, как и следовало ожидать, значительно шире, чем у точного алгоритма, исследованного лишь при $m \leqslant$

$\leqslant 32$. Испытания показывают, что качество приближенных решений достаточно высокое, в то же время быстродействие приближенных алгоритмов существенно превышает быстродействие точного. Особый интерес представляет самый быстродействующий из рассмотренных алгоритмов — б е д и э к с 2, который, например, при $m = 32$, $n = 32$ и $p = 1/4$ затрачивает на получение решения в 400 раз меньше времени, чем точный алгоритм, получаемый же им тест лишь в одном случае из четырех отличается от оптимального (превышая его на один элемент).

Следует отметить, что рассмотренные приближенные алгоритмы не гарантируют безызбыточности решения, то есть не исключена возможность того, что некоторый из входящих в полученный тест признаков окажется лишним. Однако вероятность такого события при тех размерах матрицы $\boldsymbol{C}$, на которые ориентируются данные алгоритмы, весьма мала. Кроме того, устранение избыточности в уже полученном решении представляет собой относительно несложную задачу, на которой мы, однако, не будем останавливаться.

§ 4. Канонизация булевых матриц

Б у л е в о п р о с т р а н с т в о. Множество всех различных n-компонентных булевых векторов назовем *булевым пространством* (размерности n) и обозначим его через M.

Пространство M может рассматриваться так же, как множество всех подмножеств некоторого множества $X = \{x_1, x_2, \ldots, x_n\}$, которое мы назовем *опорным*. Действительно, легко установить следующую взаимно однозначную связь между n-компонентными булевыми векторами $\boldsymbol{x}_i = (x_i^1, x_i^2, \ldots, x_i^n)$ и подмножествами X_i из X:

$$\boldsymbol{x}_i = \| \{X_i\} \ni X \|.$$

Очевидно, что это векторное представление подмножеств из X зависит от порядка, в котором располагаются элементы множества X. Например, если $X = \{a, b, c, d, e, f\}$, то подмножество $X_i = \{b, c, e\}$ будет представлено вектором 011010, если же элементы множества X будут перечислены в несколько ином порядке,

а именно, если $X=\{b, a, c, e, d, f\}$, то X_i будет представляться вектором 101100. От порядка расположения элементов в выбранном подмножестве данное представление не зависит. Например, любое из множеств $\{b, c, f\}$, $\{c, b, f\}$, $\{f, b, c\}$ и $\{f, c, b\}$, рассматриваемое как подмножество из множества $\{a, b, c, d, e, f\}$, представляется одним и тем же вектором 011001.

Заметим, что рассмотренная упорядоченность элементов в множестве X носит в данном случае чисто условный характер и введена исключительно с целью устранения неоднозначности векторного представления подмножеств из X.

Поскольку каждый n-компонентный булев вектор представляет собой некоторый элемент пространства M, естественно считать, что булева матрица $\boldsymbol{A}$ (с n столбцами) представляет некоторое подмножество M_i из пространства M:

$$\boldsymbol{A}=\| M_i \ni X \|.$$

Условимся при этом, что если в матрице $\boldsymbol{A}$ имеется несколько равных строк, то они представляют лишь один элемент пространства M.

Это представление неоднозначно: одно и то же подмножество M_i может быть представлено различными и не равными между собой матрицами.

Эквивалентность булевых матриц. Будем говорить, что матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ *α-эквивалентны*, если каждая строка любой из этих матриц равна некоторой строке другой из них.

Например, α-эквивалентны следующие матрицы:

$$\boldsymbol{A}=\begin{bmatrix}0\,1\,1\,0\,0\,1\,0\\1\,1\,0\,1\,0\,0\,1\\0\,1\,1\,0\,0\,1\,0\\1\,0\,0\,1\,0\,0\,0\end{bmatrix}\begin{matrix}1\\2\\3\\4\end{matrix}\qquad \boldsymbol{B}=\begin{bmatrix}1\,0\,0\,1\,0\,0\,0\\0\,1\,1\,0\,0\,1\,0\\1\,1\,0\,1\,0\,0\,1\\1\,0\,0\,1\,0\,0\,0\\0\,1\,1\,0\,0\,1\,0\end{bmatrix}\begin{matrix}1\\2\\3\\4\\5\end{matrix}$$

поскольку, с одной стороны, $\boldsymbol{a}_1=\boldsymbol{b}_2$, $\boldsymbol{a}_2=\boldsymbol{b}_3$, $\boldsymbol{a}_3=\boldsymbol{b}_2$, $\boldsymbol{a}_4=\boldsymbol{b}_1$ и, с другой стороны, $\boldsymbol{b}_1=\boldsymbol{a}_4$, $\boldsymbol{b}_2=\boldsymbol{a}_1$, $\boldsymbol{b}_3=\boldsymbol{a}_2$, $\boldsymbol{b}_4=\boldsymbol{a}_4$, $\boldsymbol{b}_5=\boldsymbol{a}_1$.

Очевидно, что α-эквивалентность булевых матриц является необходимым и достаточным условием равенства представляемых ими подмножеств. α-эквивалентность

матриц легко может быть установлена при их приведении к *α-канонической форме*, то есть такой форме, в которой α-эквивалентные матрицы становятся равными.

Процедура α-канонизации заключается в «приведении подобных», после которого в матрице остаются лишь не равные между собой строки, и в последующем упорядочении строк таким образом, чтобы представляемые ими числа (в числовой интерпретации булевых векторов) были расположены в возрастающем порядке.

В рассмотренном примере как матрица A, так и матрица B обладают следующей α-канонической формой:

$$\begin{bmatrix} 0110010 \\ 1001000 \\ 1101001 \end{bmatrix}$$

Будем говорить, что матрицы A и B *β-эквивалентны*, если существует такая перестановка столбцов одной из этих матриц, после которой данные матрицы становятся α-эквивалентными.

Нетрудно получить любое количество булевых матриц, β-эквивалентных исходной матрице. Для этого достаточно воспользоваться произвольными перестановками строк и столбцов, причем в множество строк исходной матрицы могут быть включены новые строки, равные каким-либо из уже имеющихся. Значительно труднее определить, являются ли две заданные матрицы β-эквивалентными.

В дальнейшем большее внимание будет уделено рассмотрению α-эквивалентности, в связи с чем условимся называть α-эквивалентные булевы матрицы *эквивалентными*.

Алгоритм β-канонизации. Трудности анализа булевых матриц на β-эквивалентность связаны с отсутствием достаточно простого алгоритма канонизации матриц, приводящего их к такой форме, в которой β-эквивалентные матрицы были бы равны.

Предлагаемый ниже алгоритм β-канонизации не дает таких гарантий. Производимая им канонизация может оказаться частичной, то есть может случиться, что две β-эквивалентные булевы матрицы после обработки их данным алгоритмом не будут равны между собой. Тем

не менее предлагаемый алгоритм достаточно эффективен: в большинстве случаев канонизация является полной (то есть приводит к форме, в которой β-эквивалентные матрицы равны), если же она оказывается частичной, то задача анализа матриц на β-эквивалентность, как правило, существенно облегчается.

Прежде всего, в исходной матрице следует «привести подобные», после чего все строки матрицы становятся взаимно не равными.

Дальнейшее изложение алгоритма будем сопровождать рассмотрением конкретного примера. Допустим, что задана матрица:

$$\begin{array}{c} 1\,2\,3\,4\,5\,6\,7\,8 \\ \left[\begin{array}{l} 1\,0\,0\,0\,1\,0\,1\,0 \\ 1\,0\,1\,0\,0\,1\,1\,1 \\ 0\,1\,0\,0\,1\,0\,0\,0 \\ 1\,1\,1\,0\,0\,0\,0\,1 \\ 0\,0\,1\,0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0\,0\,1\,1 \\ 1\,1\,0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,0\,1\,0\,0\,1 \\ 0\,1\,0\,1\,1\,0\,1\,0 \end{array}\right] \end{array} \begin{array}{l} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array}$$

«Взвесим» строки этой матрицы, то есть подсчитаем число единиц в каждой из них, а затем произведем перестановку строк, расположив их в монотонно неубывающем (по полученному весу) порядке. Разобьем полученную матрицу на секции, в каждую из которых будут входить строки с одинаковым весом. Назовем эти секции строчными.

Полученный результат примет следующий вид:

$$\begin{array}{l} 1\,2\,3\,4\,5\,6\,7\,8 \\ 0\,1\,0\,0\,1\,0\,0\,0\,c \\ \underline{0\,0\,1\,0\,0\,1\,0\,0\,e} \\ 1\,0\,0\,0\,1\,0\,1\,0\,a \\ \underline{1\,1\,0\,0\,0\,0\,1\,0\,g} \\ 1\,1\,1\,0\,0\,0\,0\,1\,d \\ 1\,0\,0\,1\,0\,0\,1\,1\,f \\ 1\,0\,1\,0\,1\,0\,0\,1\,h \\ \underline{0\,1\,0\,1\,1\,0\,1\,0\,i} \\ 1\,0\,1\,0\,0\,1\,1\,1\,b \end{array}$$

Взвесим теперь ту часть каждого столбца, которая принадлежит первой строчной секции, и используем полученный результат для секционирования матрицы по столбцам. В данном случае образуются две столбцовые секции: к одной из них будут относиться все такие столбцы матрицы, которые не содержат единиц в первой строчной секции, к другой — содержащие там одну единицу:

1478	2356	
0000	1010	c
0000	0101	e
1010	0010	a
1010	1000	g
1001	1100	d
1111	0000	f
1001	0110	h
0110	1010	i
1011	0101	b

На следующем шаге находятся веса тех частей каждого столбца, которые принадлежат второй строчной секции. Эти веса определяют очередное упорядочение столбцов внутри секций и разложение каждой из них на несколько новых секций. Получаем:

48	17	36	25	
00	00	00	11	c
00	00	11	00	e
00	11	00	01	a
00	11	00	10	g
01	10	10	10	d
11	11	00	00	f
01	10	10	01	h
10	01	00	11	i
01	11	11	00	b

Если произвести аналогичное разложение по остальным строчным секциям, то можно получить следующий

результат:

4	8	7	1	6	3	25	
0	0	0	0	0	0	11	c
0	0	0	0	1	1	00	e
0	0	1	1	0	0	01	a
0	0	1	1	0	0	10	g
0	1	0	1	0	1	10	d
1	1	1	1	0	0	00	f
0	1	0	1	0	1	01	h
1	0	1	0	0	0	11	i
0	1	1	1	1	1	00	b

После этого вновь перейдем к упорядочению строк (уже внутри имеющихся секций) и дроблению строчных секций, используя для этого веса тех участков строк, которые будут соответствовать рассматриваемым по порядку столбцовым секциям. Этот процесс совершенно аналогичен только что рассмотренной процедуре упорядочения столбцов. В результате получим:

4	8	7	1	6	3	25	
0	0	0	0	0	0	11	c
0	0	0	0	1	1	00	e
0	0	1	1	0	0	01	a
0	0	1	1	0	0	10	g
0	1	0	1	0	1	10	d
0	1	0	1	0	1	01	h
1	0	1	0	0	0	11	i
1	1	1	1	0	0	00	f
0	1	1	1	1	1	00	b

Многие из секций полученной матрицы содержат только по одному элементу (строке или столбцу); назовем такие секции *элементарными*. В данном случае не все секции являются элементарными, но дальнейшее их дробление описанным методом становится невозможным — будем говорить, что в этом случае рассматриваемая секционированная матрица задана в *циклической форме*.

Таким образом, выполненная в рассматриваемом примере канонизация носит лишь частичный характер, так как осталась столбцовая секция, составленная из столбцов 2 и 5, а также строчные секции a, g и d, h.

Анализ на β-эквивалентность. Таким образом, анализ булевых матриц на β-эквивалентность лишь частично сводится к их β-канонизации. Чем же следует ее дополнить?

Можно предложить следующий способ решения этой задачи.

Итак, пусть заданы булевы матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ и требуется выяснить, являются ли они β-эквивалентными. Допустим, что после «приведения подобных» и реализации описанной выше процедуры β-канонизации множество строк R^a матрицы $\boldsymbol{A}$ окажется разбитым на секции (подмножества из R^a) $R^a_1, R^a_2, \ldots, R^a_{n_a}$, а множество столбцов S^a — на секции $S^a_1, S^a_2, \ldots, S^a_{m_a}$. Аналогичным образом матрица $\boldsymbol{B}$ будет разбита на строчные секции $R^b_1, R^b_2, \ldots, R^b_{n_b}$ и столбцовые секции $S^b_1, S^b_2, \ldots, S^b_{m_b}$.

Необходимым условием β-эквивалентности рассматриваемых матриц является выполнение равенств $n_a = n_b$ и $m_a = m_b$, а также системы равенств $\sigma(R^a_1) = \sigma(R^b_1)$, $\sigma(R^a_2) = \sigma(R^b_2), \ldots, \sigma(R^a_{n_a}) = \sigma(R^b_{n_b})$, $\sigma(S^a_1) = \sigma(S^b_1), \sigma(S^a_2) = \sigma(S^b_2), \ldots, \sigma(S^a_{m_a}) = \sigma(S^b_{m_b})$. При невыполнении хотя бы одного из этих равенств можно считать установленным, что матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ не являются β-эквивалентными.

Если полученное разбиение является полным, то есть каждая из секций содержит лишь одну строку (или столбец), то β-эквивалентные матрицы должны стать равными. Если же разбиение оказывается неполным и в то же время выполняется указанное выше необходимое условие β-эквивалентности, то анализ следует продолжить. Сделать это можно следующим образом.

Среди неэлементарных секций матрицы $\boldsymbol{A}$ выберем минимальную (по числу элементов), если же минимальных секций окажется несколько, возьмем первую из них, считая при этом, что строчные секции следуют друг за другом сверху вниз, а столбцовые — слева направо и что строчные секции предшествуют столбцовым.

Допустим, что мощность выбранной секции равна k. Это значит, что существует $k!$ различных способов упорядочения элементов внутри секции. Однако мы не будем рассматривать их все и будем пока интересоваться

лишь способом выбора первого элемента данной секции. Сделаем этот выбор произвольно и разобьем рассматриваемую секцию на две: первую секцию образуем из одного первого элемента, вторую — из остальных элементов разбиваемой секции. Это разбиение может породить возможность продолжения процедуры β-канонизации, которую мы и реализуем по мере возможности, пока форма матрицы не станет опять циклической. Затем мы снова ищем минимальную среди неэлементарных секций матрицы и выделяем из нее описанным образом новую элементарную секцию, после чего снова продолжаем процедуру β-канонизации. Эти два процесса чередуются до тех пор, пока не будет достигнуто полное разбиение матрицы. Полученную при этом форму матрицы $\boldsymbol{A}$ назовем *условно β-канонической*.

Теперь обратимся к матрице $\boldsymbol{B}$. Можно утверждать, что существуют такие варианты выбора первого элемента минимальной секции в каждой из последовательно получаемых циклических форм матрицы $\boldsymbol{B}$, которые преобразуют эту матрицу к форме, совпадающей с условно β-канонической формой матрицы $\boldsymbol{A}$. Отсюда следует, что задача анализа матриц на β-эквивалентность будет полностью решена при реализации соответствующего перебора, отображаемого деревом поиска, вершины которого соответствуют образующимся при реализации алгоритма циклическим матрицам, а исходящие из них ребра — различным вариантам выбора первого элемента минимальной секции.

Экспериментально установлено, что этот перебор не слишком велик, то есть задача решается довольно быстро.

§ 5. Троичные матрицы

Вариант задачи о размещении отдыхающих по лодкам. Возвращаясь к задаче, рассмотренной в параграфах 1.9 и 2.2, допустим, что взаимные отношения отдыхающих носят более дифференцированный характер и каждый из отдыхающих указывает организатору прогулки как на тех лиц, с кем он хотел бы попасть в одну лодку, так и на тех, в чьей компании ему не хотелось бы очутиться.

Совокупность данных отношений выразим квадратной матрицей P, элементы которой принимают значения из множества $\{0, 1, -\}$: $p_i^j = 1$, если i-й отдыхающий желает попасть в одну лодку с j-м отдыхающим, $p_i^j = 0$ в противном случае и $p_i^j = -$, если i-му отдыхающему безразлично, окажется ли он с j-м отдыхающим в одной лодке или в различных.

Заметим, что значения 0 и 1 элементов матрицы имеют здесь иной смысл, чем ранее (раньше через 1 выражалось отношение несовместимости, а через 0 — отношение безразличия). Очевидно, что матрица P может оказаться несимметричной. Новая постановка задачи является естественным обобщением предыдущей.

Очевидно также, что в новой постановке задача не всегда имеет полное решение, удовлетворяющее всех отдыхающих. Нетрудно сформулировать необходимое и достаточное условие существования такого решения.

Рассмотрим следующую процедуру разбиения множества A всех отдыхающих на классы: перебирая подряд все элементы матрицы P, будем относить к одному классу такие элементы a_i и a_j, для которых $p_i^j = 1$. Если в результате образуется некоторый класс, содержащий, в частности, элементы a_k и a_l, для которых $p_k^l = 0$, то это значит, что поставленная задача решения не имеет, в противном случае решение существует.

Например, пусть матрица P обладает следующим значением (из очевидных соображений мы полагаем, что $p_i^j = 1$, если $i = j$):

	1	2	3	4	5	6	7	8	9	10	
⎡	1	–	–	–	–	–	–	–	–	1	⎤ 1
⎢	–	1	–	–	–	–	1	0	–	–	⎥ 2
⎢	–	–	1	–	–	–	–	–	–	1	⎥ 3
⎢	–	0	–	1	1	–	–	–	–	–	⎥ 4
⎢	–	–	–	–	1	0	–	–	1	–	⎥ 5
⎢	1	–	–	–	–	1	–	–	–	1	⎥ 6
⎢	–	–	–	–	–	–	1	–	0	–	⎥ 7
⎢	0	–	–	–	–	–	0	1	–	–	⎥ 8
⎢	–	–	–	1	1	–	–	–	1	0	⎥ 9
⎣	–	–	–	–	–	1	–	–	0	1	⎦ 10

Выполнив предписанную процедуру разбиения множества A, мы получим четыре класса: $A_1=\{1, 3, 6, 10\}$, $A_2=\{2, 7\}$, $A_3=\{4, 5, 9\}$ и $A_4=\{8\}$. Каких-либо «противоречий» внутри этих классов не существует, что нетрудно увидеть, произведя соответствующую перестановку строк и столбцов матрицы:

1	3	6	10	2	7	4	5	9	8	
1	–	–	1	–	–	–	–	–	–	1
–	1	–	1	–	–	–	–	–	–	3
1	–	1	1	–	–	–	–	–	–	6
–	–	1	1	–	–	–	–	0	–	10
–	–	–	–	1	1	–	–	–	0	2
–	–	–	–	–	1	–	–	0	–	7
–	–	–	–	0	–	1	1	–	–	4
–	–	0	–	–	–	–	1	1	–	5
–	–	–	0	–	–	1	1	1	–	9
0	–	–	–	–	0	–	–	–	1	8

Сведение задачи к предыдущей. Допустим, что решение задачи существует. В этом случае задача легко сводится к уже рассмотренной ранее, для чего достаточно обратиться к отношению несовместимости между полученными классами: классы A_i и A_j считаются взаимно несовместимыми, если существуют такие $a_k \in A_i$ и $a_l \in A_j$, для которых $p_k^l=0$ или $p_l^k=0$.

Переходя к старому способу кодирования, то есть выражая отношение несовместимости через 1 и отношение безразличия — через 0 (учитывая способ построения классов, можно обойтись лишь этими двумя отношениями), получим следующую матрицу несовместимости между классами:

$$\begin{matrix} & 1 & 2 & 3 & 4 \\ \left[\begin{matrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{matrix}\right] & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \end{matrix}$$

Далее можно применить один из уже известных нам алгоритмов (расопод, прямор и др.). В данном случае решение находится тривиальным образом, путем объединения класса A_1 с классом A_2, а класса A_3 — с классом A_4, в результате чего оказывается достаточным ограничиться двумя лодками: в одну из них усажи-

ваются отдыхающие 1, 2, 3, 6, 7, 10, в другую — отдыхающие 4, 5, 8 и 9.

Кодирование троичных векторов и матриц. Поскольку непосредственными операндами языка ЛЯПАС служат булевы векторы и матрицы, именно эти величины следует использовать и для представления рассматриваемых троичных матриц. Сделать это можно по-разному, и в различных ситуациях более удобными оказываются различные способы представления.

Наиболее простой является поэлементная форма кодирования, при которой каждый элемент представляемого троичного вектора (или матрицы) заменяется упорядоченной парой элементов множества {0, 1}. Нетрудно подсчитать, что в этом случае существует всего $4 \cdot 3 \cdot 2 = 24$ различных способа кодирования: значение 0 может быть представлено одной из комбинаций 00, 01, 10 или 11, значение 1 может быть представлено одной из трех остающихся пар, наконец, значение «—» можно представить одной из двух оставшихся неиспользованными пар.

Будем говорить, что троичный вектор (или троичная матрица) задан в (10, 01, 00)-коде, если значение 0 компоненты этой величины представляется парой 10, значение 1 — парой 01, а значение «—» — парой 00. Аналогично определяются (01, 11, 00)-код, (00, 11, 01)-код и т. д.

Например, матрица

$$\begin{bmatrix} 0 & - & - & 1 & - & 0 & 0 \\ 1 & 1 & - & 0 & - & - & - \\ 0 & 0 & 1 & - & 0 & 0 & 1 \end{bmatrix}$$

представляется в (10, 01, 00)-коде парой булевых матриц, первая из которых составлена из первых элементов кодирующих пар, вторая — из вторых:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Наличие некоторой избыточности в рассматриваемых простейших формах кодирования позволяет пользоваться не совсем определенными кодами. Например, в дальнейшем нами будет использован (01, 11, —0)-код, в котором символ «—» означает не третье значение кодирующей величины, а лишь то, что на место этого символа может

быть произвольно подставлен либо 0, либо 1. Так, приведенную выше троичную матрицу в этом коде допустимо представить следующей парой булевых матриц:

$$\begin{bmatrix} 0101000 \\ 1110001 \\ 0011001 \end{bmatrix} \quad \begin{bmatrix} 1001011 \\ 1101000 \\ 1110111 \end{bmatrix}$$

Л-программы. Оперируя с матрицей $\boldsymbol{P}$, заданной в (10, 01, 00)-коде, нетрудно найти требуемое разбиение множества A на классы, для чего достаточно использовать информацию, заключенную во второй из кодирующих матриц. Затем, по первой кодирующей матрице, строится матрица несовместимости для этих классов и проверяется, существует ли решение: решение не существует, если какой-либо из диагональных элементов построенной матрицы равен единице. Для поиска решения при его существовании используется, например, оператор прямор.

Построим следующие вспомогательные подпрограммы.

Получение классов — поклас

1. Задана квадратная булева матрица $\boldsymbol{C} = \| A \bigcirc A \|$, представляющая некоторое бинарное отношение $\bigcirc$ между элементами множества A. Требуется найти разбиение Q множества A, считая, что элементы a_i и a_j должны принадлежать одному классу, если $c_i^j = 1$, и максимизировать при этом число классов.

2. Внешние операнды:

$\alpha_к^- :: \boldsymbol{C} = \| A \bigcirc A \|$,

$\alpha_к^+ :: \| Q \Rightarrow A \|$.

Задаются: a_α, b_α.

Внутренние операнды: —, c, —.

4.

* **001 370**

§ 0 $\overline{\bigcirc}\, a\, b_\alpha \Rightarrow c$

§ 1 $\triangle\, a \oplus c \circ\!\!\rightarrow 3\, a \Rightarrow b$

§ 2 $\triangle\, b \oplus c \circ\!\!\rightarrow 1\, \alpha_a \wedge \alpha_b \circ\!\!\rightarrow 2$

$\alpha_a \vee \alpha_b \Rightarrow \alpha_a \overline{\triangle}\, c\, \alpha_c \Rightarrow \alpha_b \overline{\triangle}\, b \rightarrow 2$

§ 3 $c \Rightarrow b_\alpha$.

Симметрирование бинарного отношения — симбинот

1. Квадратной булевой матрицей A задано некоторое бинарное отношение. Требуется симметрировать его, считая, что если $a_i^j = 1$, то и элемент a_j^i также должен принять значение 1.

2. Внешние операнды:

$\alpha_к^-$ :: матрица A, задающая исходное бинарное отношение,

$\alpha_к^+$ :: матрица A^*, задающая симметрированное отношение.

Задаются: a_α, b_α.

Внутренние операнды: a, b, —.

4.

```
      * 001 371
§ 0   Ō a
§ 1   Δ a ⊕ b_a o-> 3 α_a ⇒ a
§ 2   a X 1 b a ⊕ b o-> 2 c_a ∨ α_b ⇒ α_b -> 2
§ 3   .
```

Получение матрицы несовместимости классов — манесок

1. Задано разбиение Q множества A и симметричное отношение $*$ несовместимости на множестве пар элементов множества A. Требуется получить отношение несовместимости на множестве пар заданных классов, считая, что классы A_i и A_j взаимно несовместимы, если существуют такие $a_k \in A_i$ и $a_l \in A_j$, что $a_k * a_l$.

2. Внешние операнды:

$\alpha_к$ :: $\| A * A \|$,

$\beta_к$ :: $\| Q \ni A \|$,

$\gamma_к^+$ :: $\| Q * Q \|$.

Задаются: a_α, a_β, a_γ, b_β.

Внутренние операнды: b, b, —.

4.

* 001 372

§ 0 $\overline{\circ}\, a$

§ 1 $\triangle\, a \oplus b_\beta \circ\!\!\rightarrow 5$

$\beta_a \Rightarrow \boldsymbol{a} \circ \boldsymbol{b} \circ \gamma_a$

§ 2 $\boldsymbol{a}\dot{\vee} 3\, b\, a_b \vee \boldsymbol{b} \Rightarrow \boldsymbol{b}$

§ 3 $\overline{\circ}\, b$

§ 4 $\triangle\, b \oplus b_\beta \circ\!\!\rightarrow 1$

$b_\beta \wedge \boldsymbol{b} \circ\!\!\rightarrow 4\, c_b \vee \gamma_a \Rightarrow \gamma_a \rightarrow 4$

§ 5 $b_\beta \Rightarrow b_\gamma$.

Объединение в подмножествах — о в п о д

1. Задано множество A, множество B некоторых из его подмножеств и множество C некоторых из подмножеств множества B. Для каждого элемента множества C требуется произвести объединение всех таких подмножеств из A, которые принадлежат данному элементу, то есть являются, в свою очередь, его элементами, и получить таким образом некоторое множество D подмножеств из A.

Например, если

$$A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\};$$
$$B = \{b_1, b_2, b_3, b_4\},$$

где $b_1 = \{a_1, a_4, a_5\}$, $b_2 = \{a_3\}$, $b_3 = \{a_6, a_7\}$, $b_4 = \{a_1, a_2, a_7\}$;

$$C = \{c_1, c_2, c_3\},$$

где $c_1 = \{b_1, b_2\}$, $c_2 = \{b_1, b_4\}$, $c_3 = \{b_3\}$, то

$$D = \{d_1, d_2, d_3\},$$

где $d_1 = \{a_1, a_3, a_4, a_5\}$, $d_2 = \{a_1, a_2, a_4, a_5, a_7\}$, $d_3 = \{a_6, a_7\}$.

2. Внешние операнды:

$\alpha_к :: \| B \ni A \|$,

$\beta_к :: \| C \ni B \|$,

$\gamma_к^+ :: \| D \ni A \|$.

Задаются: a_α, a_β, a_γ, b_β.

Совмещение: $\beta\gamma$.

Внутренние операнды: $\boldsymbol{a}$, b, —.

4.

* 001 373

§ 0 $\bar{\circ} a$

§ 1 $\triangle a \oplus b_\beta \circ\!\!\rightarrow 3$

$\beta_a \Rightarrow a \circ \gamma_a$

§ 2 $a \dot{\times} 1\, b\, \alpha_b \vee \gamma_a \Rightarrow \gamma_a \rightarrow 2$

§ 3 $\beta_\beta \Rightarrow \beta_\gamma$.

Задача в целом решается следующей программой.

Нахождение разбиения, удовлетворяющего дифференцированному отношению совместимости — радифс

1. На множестве пар элементов некоторого множества A троичной матрицей $\boldsymbol{P}$ задано дифференцированное отношение совместимости. Требуется найти, по возможности минимальное, разбиение Q множества A, удовлетворяющее данному отношению: элементы пар, отмеченных значением 0, должны попасть в различные классы разбиения, элементы пар, отмеченных значением 1, — в одни и те же классы, элементы пар, отмеченных значением «—», допускают произвольное размещение.

2. Внешние операнды:

$\alpha\beta_к^-$ :: матрица $\boldsymbol{P}$ в (10, 01, 00)-коде

(то есть первая из булевых матриц, представляющих в (10, 01, 00)-коде троичную матрицу $\boldsymbol{P}$, задается исходным значением комплекса α, а вторая — исходным значением комплекса β),

$\gamma_к^+ :: \| Q \Rightarrow A \|$,

$\delta_к$ — комплекс памяти, $\sigma(\delta) \leqslant \sigma(\alpha)$,

$\varepsilon_ч$ — дополнительный выход, соответствующий отсутствию решения.

Задаются: a_α, a_β, a_γ, a_δ, b_α, b_β.

Подпрограммы: поклас, симбинот, манесок, прямор, овпод.

Внутренние операнды: c, d, —.

3. (10, 01, 00)-код матрицы $\boldsymbol{P}$ можно рассматривать как пару булевых матриц, первая из которых задает несимметричное бинарное отношение несовместимости на

множестве пар элементов из A, а вторая — отношение «неразрывности». Описываемый алгоритм последовательно обрабатывает содержащуюся в этих матрицах информацию (при этом первая матрица симметрируется), после чего представляющие эти матрицы комплексы освобождаются для принятия промежуточных результатов вычислений.

4.

* 001 374

§ 0 * поклас β // * симбинот α // * манесок $\alpha\beta\gamma$ // $\overline{\circ}\, a$

§ 1 $\triangle\, a \oplus b_\gamma \circ\!\!\rightarrow 2$

$\gamma_a \wedge c_a \longmapsto \varepsilon \rightarrow 1$

§ 2 * прямор $\gamma\alpha\delta$ // * овпод $\beta\alpha\gamma$ //.

Вариант задачи диагностики. В некоторых ситуациях диагностическая матрица C, введенная в параграфе 2.3, также может оказаться троичной, то есть некоторые элементы этой матрицы могут иметь значение «—».

Пусть, например, таким значением обладает элемент c_i^j. Можно указать на две интерпретации этого значения. Одна из них соответствует тому случаю, когда неизвестно, обладает ли явление a_i признаком b_j, при другой интерпретации известно, что явление a_i и признак b_j просто не связаны: в рассматриваемом явлении данный признак может как присутствовать, так и отсутствовать. Очевидно, что в этих случаях признак b_j не может быть использован для опознавания явления a_i.

Получаемый при этом вариант задачи нахождения минимального диагностического теста не требует существенной переделки описанных ранее алгоритмов. Например, в точном алгоритме изменение затрагивает лишь первый его этап — получение матрицы различий.

Допустим, что диагностическая матрица имеет следующий вид:

$$\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \left[\begin{array}{ccccccc} 0 & - & 1 & 1 & - & 0 & 1 \\ - & 1 & 1 & 0 & - & 1 & 1 \\ 1 & 1 & - & 0 & 0 & 0 & 0 \\ 0 & 0 & - & - & 0 & 1 & 0 \\ 1 & - & - & 1 & - & - & - \end{array}\right] \end{array}$$

Перебирая, как и раньше, пары ее строк, получим матрицу различий

$$\begin{array}{c} 1\,2\,3\,4\,5\,6\,7 \\ \left[\begin{array}{c} 0\,0\,0\,1\,0\,1\,0 \\ 1\,0\,0\,1\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,1 \\ 1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,1 \\ 0\,1\,0\,0\,0\,0\,1 \\ 0\,0\,0\,1\,0\,0\,0 \\ 1\,1\,0\,0\,0\,1\,0 \\ 0\,0\,0\,1\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0 \end{array}\right] \end{array}.$$

последующее упрощение которой приводит к матрице

$$\begin{array}{c} 1\,4\,7 \\ \left[\begin{array}{c} 0\,0\,1 \\ 1\,0\,0 \\ 0\,1\,0 \end{array}\right] \end{array}$$

из которой непосредственно получается искомое решение — множество признаков {1, 4, 7}.

Коррекция приближенных алгоритмов **бедиэкс** также представляется достаточно простой: по сути, дело сводится к небольшому изменению в способе подсчета числа пар строк, различаемых выбранным на очередном шаге столбцом диагностической матрицы, или к иному порядку определения расстояния между строками (например, расстояние между строками 0 1 — — 0 1 — и 1 1 1 — 1 1 0 оценивается равным 2).

Отношения между троичными векторами. Ограничиваясь рассмотрением векторов одинаковой размерности и полагая, что булевы векторы представляют собой частный случай троичных векторов, введем следующую систему бинарных отношений.

Будем считать, что троичные векторы $\boldsymbol{u}$ и $\boldsymbol{v}$ ортогональны по i-й компоненте ($\boldsymbol{u}\,\mathrm{ort}_i\,\boldsymbol{v}$)*), если и только

*) Символ ort представляет собой сокращение от orthogonal — английского эквивалента для *ортогональный*. Аналогично, вводимые далее в этом же параграфе символы ins, adj, nei, abs и eqv образованы соответственно из intersecting — *пересекающийся*, adjacent — *смежный*, neighbour — *соседний*, absorbing — *поглощающий* и equivalent — *эквивалентный*.

если i-я компонента принимает значение 0 в одном из этих векторов и значение 1 — в другом (здесь и ниже мы полагаем, что $i \in \{0, 1, 2, \ldots, n-1\}$, где n — размерность рассматриваемых векторов). Будем говорить, что векторы u и v ортогональны (u ort v), если они ортогональны по крайней мере по одной из компонент. Если же векторы u и v не ортогональны, то будем считать, что они находятся в отношении *пересечения* (u ins v).

Например,

$$0\,1\,-\,-\,1\,1\,0\,1 \ \mathrm{ort}_0\ 1\,1\,0\,-\,1\,-\,0\,0,$$
$$1\,-\,1\,0\,0\,0\,-\,0 \ \mathrm{ins}\ 1\,0\,1\,0\,0\,0\,1\,0,$$
$$0\,-\,1\,1\,-\,0\,-\,1 \ \mathrm{ins}\ -\,1\,1\,1\,1\,-\,0\,1,$$
$$1\,1\,0\,1\,0\,0\,0\,1 \ \mathrm{ort}\ 1\,-\,-\,0\,0\,0\,-\,1.$$

Если и только если векторы u и v ортогональны и притом только по одной из компонент, будем говорить, что они находятся в отношении *смежности* (u adj v). Желая уточнить это отношение, будем в некоторых случаях пользоваться выражением «векторы u и v *смежны по i-й компоненте* (u adj_i v)». Если векторы u и v смежны по некоторой i-й компоненте и, кроме того, равны по значениям всех остальных компонент, то они находятся в отношении *соседства* (u nei v), или *соседства по i-й компоненте* (u nei_i v).

Например,

$$0\,-\,1\,1\,0\,0\,-\,1 \ \mathrm{adj}\ -\,0\,0\,1\,-\,-\,1\,1,$$
$$1\,1\,0\,1\,1\,0\,0\,0 \ \mathrm{adj}_3\ 1\,-\,0\,0\,1\,-\,0\,0,$$
$$1\,1\,-\,0\,1\,1\,-\,0 \ \mathrm{nei}_0\ 0\,1\,-\,0\,1\,1\,-\,0,$$
$$1\,0\,0\,0\,1\,0\,0\,1 \ \mathrm{nei}\ 1\,0\,0\,1\,1\,0\,0\,1.$$

Будем говорить, что вектор u *поглощает* вектор v (u abs v), если и только если все компоненты вектора u, значения которых отличны от «—», совпадают с соответствующими компонентами вектора v.

Например,

$$0\,1\,-\,1\,-\,-\,0\,0 \ \mathrm{abs}\ 0\,1\,1\,1\,-\,1\,0\,0,$$
$$1\,-\,1\,1\,-\,1\,1\,0 \ \mathrm{abs}\ 1\,-\,1\,1\,-\,1\,1\,0,$$
$$0\,0\,-\,1\,-\,1\,1\,- \ \mathrm{abs}\ 0\,0\,0\,1\,0\,1\,1\,1.$$

Нетрудно убедиться в том, что отношения ортогональности, пересечения, смежности и соседства являются *симметричными*, то есть в том, что, например, из $u\,\mathrm{ort}\,v$ всегда следует $v\,\mathrm{ort}\,u$, из $u\,\mathrm{adj}_j\,v$ следует $v\,\mathrm{adj}_j\,u$ и т. п. Отношение поглощения является *транзитивным*, то есть из u abs v и v abs w всегда следует u abs w.

Интервалы булева пространства. Очевидно, что некоторый троичный вектор u поглощает все такие булевы векторы, которые образуются из него подстановкой на места символов «—» всевозможных комбинаций из нулей и единиц. Обозначим множество этих векторов через $I(u)$. Их число, как нетрудно подсчитать, равно 2^m, где m — число компонент вектора u, обладающих значением «—».

Положим, следуя распространенному обычаю, что $1>0$ (единица *больше* нуля) и что двоичные величины a и b находятся в отношении $a \geqslant b$ (a *не меньше* b), если известно, что $a \in \{0, 1\}$ и $b=0$ или $a=1$ и $b \in \{0, 1\}$. Введем следующие отношения между булевыми векторами с n компонентами:

$u=v$ (u *равен* v), если $u^i=v^i$ для любого $i \in \{0, 1, \ldots, n-1\}$,

$u \geqslant v$ (u *не меньше* v), если $u^i \geqslant v^i$ для любого $i \in \{0, 1, \ldots, n-1\}$,

$v > v$ (u *больше* v), если $u^i \geqslant v^i$ для любого $i \in \{0, 1, \ldots, n-1\}$

и если $u^j > v^j$ для некоторого $j \in \{0, 1, \ldots, n-1\}$.

Аналогично определяются отношения $u \leqslant v$ (u *не больше* v) и $u < v$ (u *меньше* v).

Обозначим через $u(0)$ булев вектор, получаемый из троичного вектора u заменой всех символов «—» на 0, через $u(1)$ — вектор, получаемый из u заменой всех символов «—» на 1. Все другие булевы векторы, поглощаемые вектором u (обозначим через $u(i)$ произвольный из таких векторов), находятся между векторами $u(0)$ и $u(1)$, то есть удовлетворяют условию $u(0) < u(i) < u(1)$. С другой стороны, любой булев вектор $u(i)$, удовлетворяющий условию $u(0) \leqslant u(i) \leqslant u(1)$, поглощается вектором u.

В силу вышесказанного множество $I(u)$ всех булевых векторов, поглощаемых некоторым троичным вектором u, называется *интервалом булева пространства* M, соответствующим вектору u.

Легко подсчитать число различных интервалов булева пространства, обладающего размерностью n, — оно оказывается равным 3^n.

Используя понятие интервала, дадим следующую интерпретацию отношения ортогональности между троичными векторами: если векторы u и v находятся в отношении $u \operatorname{ort} v$, то соответствующие им интервалы $I(u)$ и $I(v)$ булева пространства M находятся в отношении непересечения $I(u) \cap I(v) = \varnothing$, и обратно, если $I(u) \cap I(v) = \varnothing$, то $u \operatorname{ort} v$. В более краткой символической записи это утверждение выглядит следующим образом:

$$u \operatorname{ort} v \leftrightarrow I(u) \cap I(v) = \varnothing.$$

Аналогичную интерпрерацию получают отношения пересечения и поглощения:

$$u \operatorname{ins} v \leftrightarrow I(u) \cap I(v) \neq \varnothing,$$

$$u \operatorname{abs} v \leftrightarrow I(u) \supseteq I(v).$$

α-эквивалентность троичных матриц. Будем считать, что троичная матрица U и булева матрица W *α-эквивалентны*, или *эквивалентны* ($U \operatorname{eqv} W$), если, с одной стороны, каждая из строк матрицы W поглощается по меньшей мере одной из строк матрицы U и если, с другой стороны, ни один из булевых векторов, не совпадающих с какой-либо из строк матрицы W, не поглощается ни одной из строк матрицы U.

Будем также говорить, что троичные матрицы U и V *α-эквивалентны*, или *эквивалентны* ($U \operatorname{eqv} V$), если существует булева матрица, эквивалентная каждой из матриц U и V.

Например.

$$\begin{bmatrix} 0\,1\,1\,0\,1\,1\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \end{bmatrix} \operatorname{eqv} [0\,1\,1\,-1\,1\,0\,0],$$

$$[0\,1\,1\,-0\,1\,-1] \operatorname{eqv} \begin{bmatrix} 0\,1\,1\,0\,0\,1\,0\,1 \\ 0\,1\,1\,0\,0\,1\,1\,1 \\ 0\,1\,1\,1\,0\,1\,0\,1 \\ 0\,1\,1\,1\,0\,1\,1\,1 \end{bmatrix},$$

$$\begin{bmatrix} 0\,1\,1 - 0\,0\,1\,0 \\ -\,1\,1\;1\;0\,0\,1\,0 \end{bmatrix} \text{eqv} \begin{bmatrix} 0\,1\,1\,0\,0\,0\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1\,0 \\ 1\,1\,1\,1\,0\,0\,1\,0 \end{bmatrix},$$

$$\begin{bmatrix} 0\,1\,0\,1 - 1\,0\,1 \\ 0\,0\,0\,1\,1\,1 - 1 \\ 0 - 0\,1\;1\;1\,0\,1 \end{bmatrix} \text{eqv} \begin{bmatrix} 0\,1\,0\,1 - 1\,0\,1 \\ 0\,0\,0\,1\;1\;1 - 1 \end{bmatrix}.$$

Очевидно, что введенное отношение α-эквивалентности является *рефлексивным*, *симметричным* и *транзитивным*, то есть любая матрица α-эквивалентна самой себе, из $\boldsymbol{U}$ eqv $\boldsymbol{V}$ следует $\boldsymbol{V}$ eqv $\boldsymbol{U}$, а из $\boldsymbol{U}$ eqv $\boldsymbol{V}$ и $\boldsymbol{V}$ eqv $\boldsymbol{W}$ следует $\boldsymbol{U}$ eqv $\boldsymbol{W}$.

Обозначив через $M(\boldsymbol{U})$ множество строк булевой матрицы, эквивалентной троичной матрице $\boldsymbol{U}$, сформулируем следующее утверждение: множество $M(\boldsymbol{U})$ является объединением всех интервалов, соответствующих строкам матрицы $\boldsymbol{U}$ $\left(M(\boldsymbol{U}) = \bigcup_i I(\boldsymbol{u}_i)\right)$.

Напрашивается уже знакомая нам по булеву случаю задача: как выяснить, являются ли две троичные матрицы $\boldsymbol{U}$ и $\boldsymbol{V}$ эквивалентными? Как и раньше, эту задачу можно решить путем приведения заданных матриц к некоторой канонической форме, в которой эквивалентные матрицы становятся равными между собой. Простейшей из таких форм может служить α-каноническая форма булевой матрицы, получаемой при замене каждой строки троичной матрицы совокупностью поглощаемых ею булевых векторов. Другой тип канонической формы троичной матрицы будет рассмотрен в следующем параграфе.

Простейшие равносильные преобразования. Всякую замену троичной матрицы $\boldsymbol{U}$ эквивалентной ей матрицей $\boldsymbol{V}$ *назовем равносильным преобразованием матрицы* $\boldsymbol{U}$. Рассмотрим простейшие виды таких преобразований.

Если некоторые строки $\boldsymbol{u}_i$ и $\boldsymbol{u}_j$ матрицы $\boldsymbol{U}$ находятся в отношении соседства по k-й компоненте ($\boldsymbol{u}_i$ nei_k $\boldsymbol{u}_j$), то обе эти строки могут быть заменены одной строкой, имеющей значение «—» в k-й компоненте и совпадающей со строками $\boldsymbol{u}_i$ и $\boldsymbol{u}_j$ во всех остальных компонентах. Назовем эту операцию *склеиванием* строк $\boldsymbol{u}_i$ и $\boldsymbol{u}_j$.

Например,

$$\begin{bmatrix} 0\,1\,-\,-\,1\,0\,0\,1 \\ 0\,1\,-\,-\,1\,1\,0\,1 \end{bmatrix} \text{eqv } [0\,1\,-\,-\,1\,-\,0\,1].$$

Если строки u_i и u_j матрицы U находятся в отношении поглощения (u_i abs u_j), то строка u_j может быть удалена из матрицы U. Назовем эту операцию *поглощением* строки u_i строкой u_i.

Например,

$$\begin{bmatrix} 1\,-\,0\,0\,1\,-\,1\,- \\ 1\,1\,0\,0\,1\,-\,1\,0 \end{bmatrix} \text{eqv } [1\,-\,0\,0\,1\,-\,1\,-].$$

Если строки u_i и u_j матрицы U смежны по k-й компоненте (u_i adj$_k$ u_j), возможна операция *обобщенного склеивания* этих строк. Данная операция выражается включением в матрицу U новой строки u, k-я компонента которой получает значение «—», а значение любой другой l-й компоненты определяется по правилу, задаваемому следующей таблицей:

u_i^l	0	1	0	1	—	—	—
u_j^l	0	1	—	—	0	1	—
u^l	0	1	0	1	0	1	—

Например,

$$\begin{bmatrix} 1\,-\,0\,-\,-\,1\,0\,0 \\ 1\,1\,-\,-\,0\,0\,-\,0 \end{bmatrix} \text{eqv} \begin{bmatrix} 1\,-\,0\,-\,-\,1\,0\,0 \\ 1\,1\,-\,-\,0\,0\,-\,0 \\ 1\,1\,0\,-\,0\,-\,0\,0 \end{bmatrix}.$$

Наконец, произвольную строку u_i матрицы U, имеющую значение «—» в k-й компоненте, можно заменить парой строк, одна из которых получается из строки u_i путем присвоения k-й компоненте значения 0, другая — значения 1. Эта операция называется операцией *разложения* строки u_i по k-й компоненте.

Например,

$$[1\,-\,0\,0\,-\,1\,1\,0] \text{eqv} \begin{bmatrix} 1\,-\,0\,0\,0\,1\,1\,0 \\ 1\,-\,0\,0\,1\,1\,1\,0 \end{bmatrix}.$$

Нетрудно заметить, что операция разложения обратна операции склеивания. Очевидно также, что после-

довательное разложение строки u_i по некоторой группе из m компонент, обладающих значением «—», приводит к получению 2^m новых строк, в которых значения указанных компонент образуют всевозможные комбинации из нулей и единиц, а значения остальных компонент совпадают со значениями соответствующих компонент строки u_i.

§ 6. Сжатие троичных матриц

Постановка задачи. Для каждой троичной матрицы U существует некоторая эквивалентная ей троичная матрица V, минимальная по числу строк. Назовем эту матрицу *кратчайшей формой* матрицы U, а задачу ее нахождения — задачей *максимального сжатия* матрицы U.

Нахождение кратчайшей формы представляет интерес хотя бы потому, что она позволяет наиболее компактным способом отобразить тот объект, который представляется исходной матрицей U. Задача оказывается, однако, значительно более сложной, чем это может показаться на первый взгляд, в связи с чем заслуживают внимания как точные, так и приближенные методы ее решения.

Ограничимся рассмотрением того важного случая, когда не требуется находить все кратчайшие формы заданной матрицы U, а достаточно получить лишь одну из них, неважно, какую именно.

Исходная матрица U может, в частности, оказаться булевой. Исследуем пока эту ситуацию, тем более, что к ней всегда можно перейти, поскольку любую троичную матрицу нетрудно трансформировать в булеву равносильными преобразованиями разложения ее строк.

Применение операции склеивания. Одним из способов сжатия булевой матрицы может служить последовательное применение равносильного преобразования, обратного операции разложения, а именно, операции склеивания соседних строк. При выполнении этой операции склеиваемые строки могут как удаляться из преобразуемой матрицы, так и оставаться в ней — в том и в другом случае преобразование будет равносильным.

Например, булеву матрицу

$$\left[\begin{array}{c|c} 011110 & 1 \\ 110111 & 2 \\ 010111 & 3 \\ 110011 & 4 \\ 110110 & 5 \\ 111110 & 6 \\ 111111 & 7 \\ 010101 & 8 \\ 010110 & 9 \\ 110101 & 10 \\ 101001 & 11 \\ 110010 & 12 \end{array}\right]$$

можно сжать, реализовав последовательность операций склеивания, представленную таблицей 2.6.1, в которой показаны, в частности, новые строки, вводимые в таблицу при склеивании, и присваиваемые им номера.

Таблица 2.6.1

Шаг	Склеиваемые строки	Новая строка	Ее номер	Удаляемые строки
1	1, 9	0 1 — 1 1 0	13	1, 9
2	5, 6	1 1 — 1 1 0	14	6
3	13, 14	— 1 — 1 1 0	15	13
4	5, 12	1 1 0 — 1 0	16	5, 12
5	2, 4	1 1 0 — 1 1	17	4
6	16, 17	1 1 0 — 1 —	18	16, 17
7	2, 7	1 1 — 1 1 1	19	7
8	14, 19	1 1 — 1 1 —	20	14, 19
9	3, 8	0 1 0 1 — 1	21	3, 8
10	2, 10	1 1 0 1 — 1	22	2, 10
11	21, 22	— 1 0 1 — 1	23	21, 22

В результате выполнения этой последовательности операций получается следующая троичная матрица, эквивалентная исходной и составленная из строк с услов-

ными номерами 11, 15, 18, 20 и 23:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ - & 1 & - & 1 & 1 & 0 \\ 1 & 1 & 0 & - & 1 & - \\ 1 & 1 & - & 1 & 1 & - \\ - & 1 & 0 & 1 & - & 1 \end{bmatrix}$$

Результат оказывается оптимальным, то есть полученная матрица состоит действительно из минимально возможного количества строк. Однако рассмотренный пример не дает ясного представления о том, как же все-таки решается данная задача. В самом деле, на этом примере была продемонстрирована некоторая последовательность довольно простых операций, но не было показано, как она строится, то есть как выбирается очередная операция. А именно эти правила выбора и составляют обычно основное содержание алгоритма решения задачи в целом.

Нахождение одной из кратчайших форм. Напомним, что через $M(\boldsymbol{U})$ мы обозначили множество строк булевой матрицы, эквивалентной троичной матрице $\boldsymbol{U}$, и что $M(\boldsymbol{U}) \subseteq M$.

Будем говорить, что интервал I_i булева пространства M является в то же время *интервалом множества* $M(\boldsymbol{U})$, если $I_i \subseteq M(\boldsymbol{U})$. Если же $I_i \subseteq M(\boldsymbol{U})$ и не существует такого интервала I_j, что $I_i \subset I_j \subseteq M(\boldsymbol{U})$, то интервал I_i называется *максимальным в множестве* $M(\boldsymbol{U})$. В дальнейшем мы будем называть такой интервал просто *максимальным*, подразумевая максимальность в множестве $M(\boldsymbol{U})$. Если некоторая строка троичной матрицы $\boldsymbol{U}$ соответствует максимальному интервалу, будем называть ее также *максимальной*.

Очевидно, что если $\boldsymbol{u}_i$ — максимальная строка матрицы $\boldsymbol{U}$, то $I(\boldsymbol{u}_i) \subseteq M(\boldsymbol{U})$, но это условие будет нарушаться при замене любого символа 0 или 1 в данной строке на «—», так как получаемая при этом новая строка $\boldsymbol{u}_i'$ будет характеризоваться свойством $I(\boldsymbol{u}_i) \subset I(\boldsymbol{u}_i')$.

Будем называть матрицу $\boldsymbol{U}$ *безызбыточной*, если все ее строки максимальны и если удаление любой из них является неравносильным преобразованием, то есть приводит к матрице, которая не будет эквивалентна матрице $\boldsymbol{U}$.

Покажем, что для любой троичной матрицы U существует такая кратчайшая форма, которая является в то же время безызбыточной.

Пусть V — некоторая произвольная кратчайшая форма матрицы U. Допустим, что она имеет некоторую немаксимальную строку v_i. Нетрудно показать, что у этой строки должны иметься компоненты со значениями 0 или 1. Присваивая некоторым из них значение «—», можно преобразовать строку v_i в некоторую максимальную строку v_i', удовлетворяющую условию

$$I(v_i) \subset I(v_i') \subseteq M(V).$$

В результате этой замены матрица V преобразуется в некоторую матрицу V', причем это преобразование является равносильным, поскольку $M(V) = M(V')$. Преобразуя аналогичным образом все немаксимальные строки матрицы V, мы получим некоторую матрицу V^*, которая будет эквивалентна матрице V (то есть кратчайшей форме матрицы U) и будет равна ей по количеству строк. Следовательно, матрица V^* также будет являться кратчайшей формой матрицы U, в то же время она безызбыточна: все ее строки максимальны и ни одну из них нельзя выбросить (поскольку матрица является кратчайшей формой).

Отметим; что могут существовать и такие кратчайшие формы, не все строки которых максимальны, то есть которые не являются безызбыточными.

Например, каждая из следующих эквивалентных матриц

$$A = \begin{bmatrix} 1 & - \\ - & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & - \\ 0 & 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \\ - & 1 \end{bmatrix}$$

является кратчайшей, однако лишь матрица A состоит только из максимальных строк.

Из сказанного следует, что при поиске одной из кратчайших форм (но не всех!) некоторой троичной матрицы U можно ограничиться рассмотрением множества лишь максимальных интервалов, и, соответственно, максимальных строк, которые используются как «кирпичи», из которых строится искомое решение.

Введем специальные обозначения для рассмотренных множеств: $\mathrm{Int}\, M(U)$ — множество всех интервалов множества $M(U)$, $\sup \mathrm{Int}\, M(U)$ — множество максимальных

интервалов множества $M(\boldsymbol{U})$, $R(M(\boldsymbol{U}))$ — множество соответствующих им максимальных строк. В тех случаях, когда ясно, какое именно подразумевается множество $M(\boldsymbol{U})$, будем пользоваться сокращениями: Int, sup Int и R. Будем также обозначать множество $M(\boldsymbol{U})$ через M^1.

Рассмотрим отношение поглощения между элементами множеств R и M^1, которое можно представить булевой матрицей $\|R \operatorname{abs} M^1\|$. Каждый столбец этой матрицы поглощений представляет собой перечень тех максимальных троичных векторов, которые поглощают соответствующий данному столбцу элемент множества M^1. Необходимо, чтобы по крайней мере один из этих векторов был включен в искомую кратчайшую форму, так как в противном случае полученная троичная матрица не будет эквивалентна матрице $\boldsymbol{U}$: остальные элементы множества R, из которых строится решение, не поглощают элемент множества M^1, соответствующий рассматриваемому столбцу. С другой стороны, достаточно, если это условие будет выполнено для каждого столбца матрицы $\|R \operatorname{abs} M^1\|$.

Поскольку при этом требуется еще минимизировать общее число включаемых в решение троичных векторов, нетрудно видеть, что после получения булевой матрицы $\|R \operatorname{abs} M^1\|$ рассматриваемая задача сводится к нахождению ее кратчайшего покрытия — уже знакомой нам задаче.

Однако, прежде всего нужно найти само множество R, содержащее все максимальные строки. Это множество удобно представить троичной матрицей $\boldsymbol{R}$, составленной из максимальных строк — элементов множества R. Назовем матрицу $\boldsymbol{R}$ *расширением* матрицы $\boldsymbol{U}$. Нетрудно показать, что эта матрица может служить второй канонической формой троичной матрицы $\boldsymbol{U}$. Действительно, множество строк матрицы $\boldsymbol{R}$ определяется однозначно, а упорядочение этих строк представляет уже тривиальную задачу.

Получение множества максимальных строк. Итак, задана некоторая матрица $\boldsymbol{U}$. Как же найти множество sup Int максимальных интервалов множества $M(\boldsymbol{U})$ или, что эквивалентно, множество R соответствующих им максимальных строк?

Если исходная матрица U является булевой, то можно воспользоваться следующим простым способом решения данной задачи.

Сначала в множестве $M_0 \equiv M(U)$ строк матрицы U отыскиваются и склеиваются пары соседних элементов, в результате чего образуется некоторое множество M_1 продуктов склеивания — строк, каждая из которых содержит ровно один символ «—». После этого из множества M_0 удаляются те элементы, которые поглощаются какими-либо из строк множества M_1, а также проводится «приведение подобных» в множестве M_1. Затем аналогичной процедуре склеивания подвергаются элементы множества M_1, и таким образом возникает множество M_2 некоторых строк, содержащих по два символа «—», после чего производится удаление поглощаемых этими строками элементов множества M_1 и «приведение подобных» в множестве M_2. Процесс продолжается аналогичным образом и заканчивается, когда очередное множество M_k оказывается пустым, то есть когда в множестве M_{k-1} не будет соседних строк.

Будем называть троичный вектор *вектором m-го ранга*, если ровно m его компонент имеют значение 0 или 1 (число компонент со значением «—» может быть произвольным). Таким образом, если размерность рассматриваемых векторов равна n, то элементами множества M_0 служат строки ранга n, элементами множества M_1 — строки ранга $n-1$, элементами множества M_i — строки ранга $n-i$. Поскольку в отношении соседства могут находиться лишь строки одинакового ранга, описанная процедура реализует все возможные склеивания между всеми исходными и появляющимися в ходе вычислений строками. Поэтому объединение полученных описанным образом множеств $M_0, M_1, \ldots, M_{k-1}$, из которых удалены поглощаемые строки, и будет представлять собой искомое множество всех максимальных строк.

Другой метод получения множества R основан на применении операции обобщенного склеивания и годится для того случая, когда матрица U не обязательно является булевой. Предполагается, однако, что в матрице U отсутствуют пары строк, находящихся в отношении

поглощения. Метод реализует перебор пар строк этой матрицы и выявление таких из них, которые находятся в отношении смежности. Такие пары склеиваются, а получаемые при этом новые строки добавляются в матрицу U. Каждая новая строка сравнивается со всеми предыдущими и удаляется из матрицы, если она поглощается какой-либо из них. Если же этого не происходит, то новая строка остается в матрице, а удаляются поглощаемые ею строки, если такие найдутся. Перебор пар организуется путем рассмотрения по очереди всех строк матрицы в сочетании с каждой из предыдущих. Таким образом перебираются все пары строк, в том числе и новых. В результате матрица U в ее окончательном виде будет содержать все максимальные строки и только эти строки *).

Примеры. Проиллюстрируем описанные способы получения множества R на примере булевой матрицы, приведенной в начале параграфа.

Реализация первого способа представлена таблицей 2.6.2, разбитой на три секции, соответствующие множествам M_0, M_1 и M_2.

Справа от таблицы знаком «—» отмечены те строки, которые затем удаляются из рассматриваемых множеств в порядке поглощения или «приведения подобных». Таким образом, совокупность найденных максимальных строк образует следующую матрицу:

$$R = \begin{bmatrix} 1\,0\,1\,0\,0\,1 \\ -\,1\,-\,1\,1\,0 \\ -\,1\,0\,1\,1\,- \\ -\,1\,0\,1\,-\,1 \\ 1\,1\,0\,-\,1\,- \\ 1\,1\,-\,1\,1\,- \end{bmatrix}$$

Реализация второго метода представлена на таблице 2.6.3.

*) Описанные алгоритмы являются переложениями известных методов минимизации булевых функций в классе д н ф (см. главу 3), а именно, метода Квайна — МакКласки и метода Блэка — Порецкого. Познакомиться с этими методами в их непосредственной форме можно, например, по книге [40].

Таблица 2.6.2

0 1 1 1 1 0	1	—		
1 1 0 1 1 1	2	—		
0 1 0 1 1 1	3	—		
1 1 0 0 1 1	4	—		
1 1 0 1 1 0	5	—		
1 1 1 1 1 0	6	—		
1 1 1 1 1 1	7	—	M_0	
0 1 0 1 0 1	8	—		
0 1 0 1 1 0	9	—		
1 1 0 1 0 1	10	—		
1 0 1 0 0 1	11			
1 1 0 0 1 0	12	—		

Шаг	Склеиваемые строки	Новая строка	Ее номер		
1	1, 6	— 1 1 1 1 0	13	—	
2	1, 9	0 1 — 1 1 0	14	—	
3	2, 3	— 1 0 1 1 1	15	—	
4	2, 4	1 1 0 — 1 1	16	—	
5	2, 5	1 1 0 1 1 —	17	—	
6	2, 7	1 1 — 1 1 1	18	—	M_1
7	2, 10	1 1 0 1 — 1	19	—	
8	3, 8	0 1 0 1 — 1	20	—	
9	3, 9	0 1 0 1 1 —	21	—	
10	4, 12	1 1 0 0 1 —	22	—	
11	5, 6	1 1 — 1 1 0	23	—	
12	5, 9	— 1 0 1 1 0	24	—	
13	5, 12	1 1 0 — 1 0	25	—	
14	6, 7	1 1 1 1 1 —	26	—	
15	8, 10	— 1 0 1 0 1	27	—	
16	13, 24	— 1 — 1 1 0	28		
17	14, 23	— 1 — 1 1 0	29	—	
18	15, 24	— 1 0 1 1 —	30		
19	15, 27	— 1 0 1 — 1	31		
20	16, 25	1 1 0 — 1 —	32	—	M_2
21	17, 21	— 1 0 1 1 —	33	—	
22	17, 22	1 1 0 — 1 —	34		
23	17, 26	1 1 — 1 1 —	35		
24	18, 23	1 1 — 1 1 —	36	—	
25	19, 20	— 1 0 1 — 1	37	—	

Таблица 2.6.3

0	1	1	1	1	0	1 —
1	1	0	1	1	1	2 —
0	1	0	1	1	1	3 —
1	1	0	0	1	1	4 —
1	1	0	1	1	0	5 —
1	1	1	1	1	0	6 —
1	1	1	1	1	1	7 —
0	1	0	1	0	1	8 —
0	1	0	1	1	0	9 —
1	1	0	1	0	1	10 —
1	0	1	0	0	1	11
1	1	0	0	1	0	12 —

Шаг	Склеиваемые строки	Новая строка	Ее номер	Поглощаемые строки
1	3, 2	— 1 0 1 1 1	13 —	2, 3
2	6, 1	— 1 1 1 1 0	14 —	1, 6
3	9, 5	— 1 0 1 1 0	15 —	5, 9
4	10, 8	— 1 0 1 0 1	16 —	8, 10
5	12, 4	1 1 0 0 1 —	17 —	4, 12
6	13, 7	1 1 — 1 1 1	18 —	7
7	15, 13	— 1 0 1 1 —	19	13, 15, 27
8	18, 14	1 1 1 1 1 —	20 —	
9	18, 16	1 1 0 1 — 1	21 —	
10	18, 17	1 1 0 — 1 1	22 —	
11	19, 14	— 1 — 1 1 0	23	14
12	19, 16	— 1 0 1 — 1	24	16, 21
13	19, 17	1 1 0 — 1 —	25	17, 22
14	20, 19	1 1 — 1 1 —	26	18, 20
15	24, 23	— 1 0 1 1 —	27 —	

Таким образом, совокупность оставшихся строк составляет матрицу

$$\boldsymbol{R} = \begin{matrix} \\ \\ \\ \\ \\ \end{matrix} \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ - & 1 & 0 & 1 & 1 & - \\ - & 1 & - & 1 & 1 & 0 \\ - & 1 & 0 & 1 & - & 1 \\ 1 & 1 & 0 & - & 1 & - \\ 1 & 1 & - & 1 & 1 & - \end{bmatrix} \begin{matrix} \text{А} \\ \text{Б} \\ \text{В} \\ \text{Г} \\ \text{Д} \\ \text{Е} \end{matrix}$$

равную с точностью до перестановки строк той, которая была получена по первому методу.

Построим теперь матрицу $\|R \operatorname{abs} M^1\|$:

1	2	3	4	5	6	7	8	9	10	11	12	
0	0	0	0	0	0	0	0	0	0	1	0	А
0	1	1	0	1	0	0	0	1	0	0	0	Б
1	0	0	0	1	1	0	0	1	0	0	0	В
0	1	1	0	0	0	0	1	0	1	0	0	Г
0	1	0	1	1	0	0	0	0	0	0	1	Д
0	1	0	0	1	1	1	0	0	0	0	0	Е

В этой матрице содержится много столбцов, обладающих только одной единицей, благодаря чему нахождение кратчайшего покрытия матрицы не составляет особого труда. В результате мы получим следующую матрицу кратчайшего покрытия:

0	0 0 0 0 0 0 0 0 0 0 1 0	А
1	0 0 0 1 1 0 0 1 0 0 0	В
0	1 1 0 0 0 0 1 0 1 0 0	Г
0	1 0 1 1 0 0 0 0 0 0 1	Д
0	1 0 0 1 1 1 0 0 0 0 0	Е

Таким образом, искомая кратчайшая форма исходной матрицы образуется из максимальных строк А, В, Г, Д и Е и имеет следующий вид:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ - & 1 & - & 1 & 1 & 0 \\ - & 1 & 0 & 1 & - & 1 \\ 1 & 1 & 0 & - & 1 & - \\ 1 & 1 & - & 1 & 1 & - \end{bmatrix}$$

Детали программирования. Не приводя целиком Л-программу вычисления множества R максимальных интервалов, рассмотрим некоторые из операций, которые должны быть в ней представлены.

Допустим, что матрица U задана в (10, 01, 00)-коде комплексами α и β, выбор отдельных элементов из которых осуществляется индексами a и b. В этом случае операция анализа двух строк матрицы U на смежность может быть записана как

$$\alpha_a \wedge \beta_b \Rightarrow a\, \alpha_b \wedge \beta_a \vee a \Rightarrow a \nabla \oplus 1 \longmapsto$$

где в значении, приобретаемом переменной a, отмечаются единицами компоненты, по которым рассматриваемые строки ортогональны, а переход по оператору $\longmapsto$ осуществляется при условии, что строки оказываются несмежными.

Операция обобщенного склеивания строк, если они оказались смежными, может быть выполнена следующим фрагментом:

$$\alpha_a \vee \alpha_b \oplus a \Rightarrow b\, \beta_a \vee \beta_b \oplus a \Rightarrow c$$

в котором переменные b и c используются для фиксирования результата.

Проверить, не поглощается ли полученная строка некоторой строкой матрицы U, представленной элементами α_c и β_c, можно с помощью следующего выражения:

$$b \urcorner \wedge \alpha_c \longmapsto 5\, c \urcorner \wedge \beta_c \longmapsto 5$$

в котором переход в предложение с условным номером 5 реализуется в случае, когда выявляется отсутствие указанного поглощения. Обратная операция, то есть проверка возможности поглощения полученной строкой некоторой строки матрицы U, может быть реализована аналогично:

$$\alpha_c \urcorner \wedge b \longmapsto 5\, \beta_c \urcorner \wedge c \longmapsto 5$$

Остальные «хитрости» программирования связываются исключительно с распределением информации: при удалении некоторой строки из матрицы целесообразно заполнять образующиеся в комплексах α и β «дыры», перенося в них последние элементы комплексов. Такой прием позволяет существенно ограничить рост мощности комплексов, то есть полезен в тех случаях, когда приходится заботиться об экономии памяти. Правда, при этом несколько усложняется порядок перебора элементов комплексов.

§ 7. О вырожденных троичных матрицах

Задача анализа матрицы на вырожденность. Сжатие троичных матриц может приводить к получению однострочной матрицы, все элементы которой имеют значение «—». Представляемое такой

матрицей множество M^1 строк эквивалентной булевой матрицы совпадает с булевым пространством M.

Условимся называть подобные матрицы *вырожденными* и рассмотрим задачу анализа некоторой заданной матрицы U на вырожденность. Данную задачу можно, конечно, решить с помощью некоторого алгоритма максимального сжатия матрицы U. Например, чередуя операции обобщенного склеивания и поглощения, читатель может убедиться в вырожденности матрицы

$$U = \left[\begin{array}{cccccc|c}
1 & - & - & - & - & 0 & 1 \\
1 & - & 1 & - & 1 & - & 2 \\
1 & 0 & - & - & 1 & 1 & 3 \\
1 & - & - & - & 0 & 1 & 4 \\
1 & 1 & 0 & - & - & 1 & 5 \\
0 & 1 & - & - & - & 1 & 6 \\
0 & - & - & - & 0 & 1 & 7 \\
0 & 0 & - & - & 1 & - & 8 \\
0 & - & - & 0 & 1 & 0 & 9 \\
0 & 1 & - & 1 & 1 & 0 & 10 \\
0 & 1 & - & - & 0 & 0 & 11 \\
0 & - & 1 & - & 0 & 0 & 12 \\
0 & 0 & 0 & - & 0 & 0 & 13
\end{array}\right]$$

получив в результате однострочную матрицу

$$[------].$$

Проводимые при этом вычисления оказываются, однако, довольно громоздкими, требуя больших затрат времени, на которые не всегда можно пойти.

Между тем, решаемая в данном случае задача является более простой, чем задача максимального сжатия: не требуется находить кратчайшую форму заданной троичной матрицы U, а нужно лишь выяснить, будет ли эта форма однострочной, со значением «—» всех ее элементов.

Алгоритм анализа. Воспользовавшись указанным упрощением, предложим более быстродействующий алгоритм решения поставленной задачи, основанный на поиске троичного вектора x, ортогонального каждой из строк заданной матрицы U (будем говорить, что в этом случае *вектор ортогонален матрице*). Если такой вектор

будет найден, то это будет означать, что матрица U является невырожденной, если же будет установлено, что такого вектора не существует, то тем самым будет доказана вырожденность матрицы U: значит, каждый элемент булева пространства M поглощается какой-либо из строк матрицы U.

Предлагаемый алгоритм осуществляет древообразный перебор троичных векторов, сокращаемый на основе следующих довольно очевидных правил.

Правило 1. Если в матрице U имеется строка, лишь одна компонента которой обладает значением, отличным от «—», то это значит, что искомый вектор x, если он существует, должен обладать инверсным значением этой компоненты, поскольку только в этом случае он будет ортогонален упомянутой строке.

Правило 2. Если в матрице U имеется строка, все компоненты которой обладают значением «—», то это значит, что искомого вектора x не существует.

Правило 3. Если в матрице U имеется столбец, все компоненты которого обладают значением «—», то это значит, что вектор x, если он существует, может обладать таким же значением данной компоненты.

Правило 4. Если в матрице U имеется столбец, элементы которого обладают лишь значениями «—» или 1 («—» или 0), то это значит, что вектор x, если он существует, может обладать значением 0 (1) этой компоненты.

Правило 5. Если значения некоторых компонент искомого вектора x уже фиксированы, то из матрицы U можно удалить сначала строки, ортогональные вектору x, а затем столбцы, соответствующие данным компонентам, и рассматривать далее уже эту упрощенную матрицу.

Правило 6. Удаление всех строк матрицы, согласно правилу 5, свидетельствует о нахождении искомого вектора x. Если же в результате применения правила 5 число строк матрицы остается формально отличным от нуля, в то время как число столбцов становится равным нулю, то это является признаком вырожденности рассматриваемого остатка матрицы, то есть говорит об отсутствии искомого вектора x с частично фиксированными значениями компонент.

Пример. Анализируя вышерассмотренную матрицу и пытаясь определить значения двоичных компонент искомого вектора $x=(a, b, c, d, e, f)$, алгоритм моделирует следующие рассуждения.

Исходная матрица U не обладает свойствами, позволяющими сразу же применить одно из сформулированных правил. Поэтому воспользуемся типичным приемом разложения, образовав точку ветвления вычислительного процесса, соответствующую выбору значения компоненты a.

Допустим, что $a=1$. Тогда, согласно правилу 5, задача определения значений остальных компонент вектора x сводится к задаче нахождения вектора, ортогонального матрице

$$\begin{array}{c} b\ c\ d\ e\ f \\ \left[\begin{array}{ccccc} - & - & - & - & 0 \\ - & 1 & - & 1 & - \\ 0 & - & - & 1 & 1 \\ - & - & - & 0 & 1 \\ 1 & 0 & - & - & 1 \end{array}\right] \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \end{array}$$

Следуя правилу 1 (см. строку 1), мы должны приписать компоненте f значение 1, а пользуясь правилом 3 (см. столбец d), мы можем придать компоненте d значение «—». После этого анализируемая матрица сокращается по правилу 5 до следующего вида:

$$\begin{array}{c} b\ c\ e \\ \left[\begin{array}{ccc} - & 1 & 1 \\ 0 & - & 1 \\ - & - & 0 \\ 1 & 0 & - \end{array}\right] \begin{array}{c} 2 \\ 3 \\ 4 \\ 5 \end{array} \end{array}$$

Далее опять применяется правило 1, компонента e получает значение 1, а матрица сокращается до

$$\begin{array}{c} b\ c \\ \left[\begin{array}{cc} - & 1 \\ 0 & - \\ 1 & 0 \end{array}\right] \begin{array}{c} 2 \\ 3 \\ 5 \end{array} \end{array}$$

Согласно правилу 1, мы находим, что $c=0$, а $b=1$. Последовательное же применение правил 5 и 6 приводит к выводу о вырожденности данного остатка матрицы, откуда следует, что при $a=1$ искомое решение отсутствует.

Теперь допустим, что $a=0$. Тогда остаток матрицы будет иметь следующий вид:

b	c	d	e	f	
1	–	–	–	1	6
–	–	–	0	1	7
0	–	–	1	–	8
–	–	0	1	0	9
1	–	1	1	0	10
1	–	–	0	0	11
–	1	–	0	0	12
0	0	–	0	0	13

Поскольку ни одно из упрощающих правил не подходит и к этой матрице, образуем точку ветвления рассуждений, соответствующую выбору значения компоненты b.

Допустим, что $b=1$. Тогда матрица сокращается до

c	d	e	f	
–	–	–	1	6
–	–	0	1	7
–	0	1	0	9
–	1	1	0	10
–	–	0	0	11
1	–	0	0	12

Далее следует, что $f=0$ (по правилу 1) и $c=0$ (по правилу 4). Последующее применение правила 5 приводит к матрице

d	e	
0	1	9
1	1	10
–	0	12

Затем следует вынужденный выбор значения 1 для компоненты e и получение остатка

d	
0	9
1	10

который оказывается вырожденным: компонента d должна получить одновременно значения 1 и 0, что невозможно.

Рассмотрим другой вариант, допустив, что $b=0$. Тогда мы должны найти вектор, ортогональный матрице

$$\begin{array}{c} c\ d\ e\ f \\ \left[\begin{array}{cccc} - & - & 0 & 1 \\ - & - & 1 & - \\ - & 0 & 1 & 0 \\ 1 & - & 0 & 0 \\ 0 & - & 0 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ 7 \\ 8 \\ 9 \\ 12 \\ 13 \end{array}$$

Для этого необходимо положить $e=0$; присвоив d значение 1, можно перейти к рассмотрению матрицы

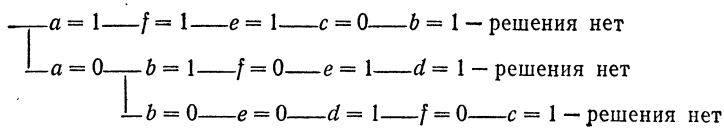
$$\begin{array}{c} c\ f \\ \left[\begin{array}{cc} - & 1 \\ 1 & 0 \\ 0 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ 7 \\ 12 \\ 13 \end{array}$$

После этого следует выбор $f=0$ и получение вырожденного остатка

$$\begin{array}{c} c \\ \left[\begin{array}{c} 1 \\ 0 \end{array}\right] \end{array} \begin{array}{c} \\ 12 \\ 13 \end{array}$$

Поскольку этим завершается перебор возможных комбинаций значений компонент вектора $\boldsymbol{x}$ и при этом не удалось сделать вектор $\boldsymbol{x}$ ортогональным матрице $\boldsymbol{U}$, то тем самым устанавливается вырожденность данной матрицы.

Приведенная последовательность рассуждений может быть представлена следующим деревом поиска вектора $\boldsymbol{x}$:



Л-программы. Переходим к рассмотрению программы, реализующей описанный алгоритм.

Анализ троичной матрицы на вырожденность — вытрома

1. Требуется определить, является ли заданная троичная матрица $\boldsymbol{U}$ вырожденной, и найти в противном случае троичный вектор $\boldsymbol{x}$, ортогональный данной матрице.

2. Внешние операнды:

$\alpha\beta_к^-$ :: матрица $\boldsymbol{U}$ в коде (10, 01, 00),

$\gamma_п$ :: $\Psi(B)$, где B — множество столбцов матрицы $\boldsymbol{U}$,

$\delta\varepsilon_п^+$ :: вектор $\boldsymbol{x}$ в коде (10, 01, 00), при дополнительном выходе ζ,

$\zeta_ч$ — дополнительный выходной полюс, соответствующий невырожденности матрицы $\boldsymbol{U}$,

$\eta_к$ — комплекс памяти, $\sigma(\eta) \leqslant \sigma(B) \cdot (2 + 2\omega(\alpha))$, где $\omega(\alpha)$ — коэффициент размерности комплекса α.

Задаются: a_α, a_β, a_η, b_α, b_β.

Размерность: $\alpha\ \beta\ \gamma\ \delta\ \varepsilon$.

Подпрограммы: разор, наслед.

Внутренние операнды: $\boldsymbol{g}$, c, —.

Примечание: основной выходной полюс достигается при обнаружении вырожденности матрицы $\boldsymbol{U}$.

4.

$$* \ \mathbf{001\ 365}\ (\alpha\, \boldsymbol{e}\, \boldsymbol{f}\, \boldsymbol{g})$$

§ 0 $\quad \circ\delta\circ\varepsilon\circ b_\eta\gamma \Rightarrow \boldsymbol{e}\, b_\alpha \Rightarrow c \to 3$

§ 1 $\quad b_\eta \circ\!\!\to 4\ \eta \Rightarrow (\delta\varepsilon\, b\, c)\, c \Rightarrow b_\alpha c_b \vee \delta \Rightarrow \delta$

§ 2 $\quad *$ разор $\alpha\beta\delta\varepsilon b \mathbin{/\!/} \gamma \oplus \delta \oplus \varepsilon \Rightarrow \boldsymbol{e}\, b \Rightarrow b_\alpha \circ\!\!\to \zeta\, \boldsymbol{e} \circ\!\!\to 1$

§ 3 $\quad *$ наслед $\alpha\beta\, \boldsymbol{e}\, \boldsymbol{f}\, \boldsymbol{g} \mathbin{/\!/} \boldsymbol{f} \wedge \boldsymbol{g} \mapsto 1$

$\boldsymbol{f} \vee \delta \Rightarrow \delta\, \boldsymbol{g} \vee \varepsilon \Rightarrow \varepsilon \boldsymbol{f} \vee \boldsymbol{g} \mapsto 2$

$\boldsymbol{e} \vdash \Rightarrow b\, (\delta\varepsilon\, b\, c) \Rightarrow \eta$

$b_\alpha \Rightarrow c\, c_b \vee \varepsilon \Rightarrow \varepsilon \to 2$

§ 4 .

Разделение строк троичной матрицы по отношению ортогональности к заданному вектору — разор

1. Строки троичной матрицы $\boldsymbol{U}$ должны быть разбиты на два класса, в первый из которых входят строки, пересекающиеся с троичным вектором $\boldsymbol{x}$, во второй — ортогональные ему.

2. Внешние операнды:

$\alpha\beta_к^-$ :: исходная матрица $\boldsymbol{U}$ в коде (10, 01, 00),
$\alpha\beta_к^+$ :: преобразованная матрица $\boldsymbol{U}$ в коде (10, 01, 00),
$\gamma\delta_п$ :: вектор $\boldsymbol{x}$ в коде (10, 01, 00),
$\varepsilon_и^+$ — мощность первого класса, элементы которого собираются в верхней части матрицы $\boldsymbol{U}$ путем некоторых перестановок ее строк.

Задаются: a_α, a_β, b_α, b_β.
Размерность: $\alpha\ \beta\ \gamma\ \delta$.
Внутренние операнды: —, a, —.

4.

* 001 366

§ 0 $\overline{\circlearrowright} a\, b_a \Rightarrow \varepsilon$
§ 1 $\triangle\, a \oplus \varepsilon \circ\!\!\rightarrow 3\,\alpha_a \wedge \delta \longmapsto 2\,\beta_a \wedge \gamma \circ\!\!\rightarrow 1$
§ 2 $\overline{\triangle}\, \varepsilon\, \alpha_a \Leftrightarrow \alpha_\varepsilon\, \beta_a \Leftrightarrow \beta_\varepsilon\, \overline{\triangle}\, a \rightarrow 1$
§ 3 .

Нахождение следствий по правилам поиска ортогонального вектора — н а с л е д

1. Анализируется подматрица, образованная подмножеством Q множества B всех столбцов троичной матрицы $\boldsymbol{U}$ и, согласно правилам 1 и 4, сформулированным в настоящем параграфе, определяются значения некоторых компонент (обязательные или допустимые) вектора $\boldsymbol{x}$, ортогонального данной подматрице. По правилу 2 устанавливается факт отсутствия такого вектора.

2. Внешние операнды:

$\alpha\beta_к$ :: матрица $\boldsymbol{U}$ в коде (10, 01, 00),
$\gamma_п$ :: $\| \{Q\} \ni B \|$,
$\delta\varepsilon_п^+$ :: вектор $\boldsymbol{x}$ в коде (10, 01, 00).

Задаются: a_α, a_β, b_α, b_β.
Размерность: $\alpha\ \beta\ \gamma\ \delta\ \varepsilon$.
Внутренние операнды: $\boldsymbol{d}$, a, —.
Примечание: сигналом отсутствия ортогонального вектора служит условие $\delta^+ \wedge \varepsilon^+ \neq 0$.

3. Работа программы основана на поиске:

а) строк, лишь одна компонента которых обладает значением, отличным от «—»,

б) строк, все компоненты которых имеют значение «—»,

в) столбцов, в каждом из которых элементы принимают значения из множества {1, —} или из множества {0, —}.

4.

$$* \; 001\;367\;(\alpha\, a\, b\, c\, d)$$

$$\S\,0 \quad \bar{\circ}\, a \circ \delta \circ \varepsilon \circ c \circ d$$

$$\S\,1 \quad \triangle\, a \oplus b_{\alpha} \circ\!\!\rightarrow 3$$

$$\alpha_a \Rightarrow a \vee c \Rightarrow c$$

$$\beta_a \Rightarrow b \vee d \Rightarrow d$$

$$a \vee b \wedge \gamma \circ\!\!\rightarrow 2 \bigtriangledown \oplus 1 \mapsto 1$$

$$a \vee \varepsilon \Rightarrow \varepsilon\, b \vee \delta \Rightarrow \delta \rightarrow 1$$

$$\S\,2 \quad \gamma \Rightarrow \varepsilon \Rightarrow \delta.$$

$$\S\,3 \quad c \oplus d \wedge c \vee \varepsilon \wedge \gamma \Rightarrow \varepsilon$$

$$c \oplus d \wedge d \vee \delta \wedge \gamma \Rightarrow \delta.$$

Устранение избыточности в троичной матрице. Если известно, что все строки некоторой троичной матрицы U являются максимальными, то нетрудно получить одну из ее безызбыточных форм, для чего достаточно выбросить из матрицы U некоторые строки.

Рассмотрим один из возможных алгоритмов решения этой задачи, основанный на последовательной проверке строк исходной матрицы на возможность их выбрасывания, причем считается, что строка u_i троичной матрицы U может быть выброшена в том и только в том случае, если каждый из поглощаемых ею булевых векторов поглощается в то же время какой-либо из других строк матрицы U.

Покажем, что необходимым и достаточным условием возможности выбрасывания строки u_i из матрицы U является вырожденность минора U_i, образованного столбцами, в которых строка u_i имеет значение «—», и теми из остальных строк, которые пересекаются со строкой u_i.

Действительно, если минор U_i не является вырожденным, то может быть найден некоторый ортогональ-

ный ему булев вектор. Перенеся значения компонент этого вектора в соответствующие компоненты вектора u_i, обладавшие до этого значениями «—», мы получим булев вектор, поглощаемый вектором u_i и не поглощаемый ни одной из других строк матрицы U. В этом случае строка u_i не может быть выброшена из матрицы U.

Если же минор U_i является вырожденным, то не существует ортогонального ему булева вектора. Любая подстановка значений 1 и 0 на места значений «—» в строке u_i приводит поэтому к получению булева вектора, поглощаемого какой-либо из остальных строк матрицы U.

Например, из матрицы

$$\begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \left[\begin{array}{cccccc} - & 1 & - & 1 & 1 & 1 \\ 0 & - & - & - & - & 0 \\ 0 & 1 & - & - & 1 & - \\ - & 1 & 0 & - & 1 & 1 \\ 1 & - & 1 & 1 & - & 1 \\ 1 & 0 & 1 & - & 1 & - \end{array}\right] \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \end{array}$$

можно выбросить строку 1, поскольку является вырожденным минор

$$\begin{array}{c} \begin{array}{cc} a & c \end{array} \\ \left[\begin{array}{cc} 0 & - \\ - & 0 \\ 1 & 1 \end{array}\right] \begin{array}{c} 3 \\ 4 \\ 5 \end{array} \end{array}$$

но нельзя выбросить строку 3, так как соответствующий ей минор

$$\begin{array}{c} \begin{array}{ccc} c & d & f \end{array} \\ \left[\begin{array}{ccc} - & 1 & 1 \\ - & - & 0 \\ 0 & - & 1 \end{array}\right] \begin{array}{c} 1 \\ 2 \\ 4 \end{array} \end{array}$$

не вырожден: существует булев вектор 1 0 1, ортогональный каждой из строк этого минора. Подставив компоненты этого вектора на места тех компонент строки 3, которые обладают значением «—», мы получим вектор 0 1 1 0 1 1, ортогональный каждой из других строк матрицы U.

Л-программа.

Устранение избыточности в троичной матрице — устиз

1. Заданную троичную матрицу $\boldsymbol{U}$, все строки которой максимальны, требуется привести к безызбыточной путем выбрасывания некоторых из ее строк.

2. Внешние операнды:

$\alpha\beta_к^-$ :: исходное значение матрицы $\boldsymbol{U}$ в коде (10, 01, 00),

$\alpha\beta_к^+$ :: результативное значение матрицы $\boldsymbol{U}$ в коде (10, 01, 00),

$\gamma_к$ — комплекс памяти, $\sigma(\gamma) \leqslant 2\sigma(\alpha) + \sigma(B) \times \times (2 + 2\omega(\alpha))$.

$\delta_п :: \Psi(B)$, где B — множество столбцов матрицы $\boldsymbol{U}$.

Задаются: a_α, a_β, a_γ, b_α, b_β.
Размерность: $\alpha\beta\gamma\delta$.
Подпрограмма: вытрома.
Внутренние операнды: $\boldsymbol{j}$, $\boldsymbol{e}$, $\boldsymbol{Y}$.

4.

$*$ **001 364** $(\alpha\, \boldsymbol{i}\, \boldsymbol{j}\, \boldsymbol{h})$

§ 0 $a_\gamma + b_\alpha \Rightarrow a_{27} + b_\alpha \Rightarrow a_{30}\, \overline{\circ}\, \boldsymbol{d}\, b_\alpha \Rightarrow \boldsymbol{e}$

§ 1 $\triangle\, d \oplus e \circ\!\!\rightarrow 4$

$\alpha_d \Rightarrow \boldsymbol{i}\, \beta_d \Rightarrow \boldsymbol{j}\, \overline{\circ}\, b \circ c$

§ 2 $\triangle\, b \oplus e \circ\!\!\rightarrow 3\, b \oplus d \circ\!\!\rightarrow 2$

$\alpha_b \wedge \boldsymbol{j} \mapsto 2\, \beta_b \wedge \boldsymbol{i} \mapsto 2$

$\alpha_b \Rightarrow \gamma_c\, \beta_b \Rightarrow \boldsymbol{y}_c \triangle c \rightarrow 2$

§ 3 $c \Rightarrow b_\gamma \Rightarrow b_{27}\, \boldsymbol{i} \vee \boldsymbol{j} \urcorner \wedge \delta \Rightarrow \boldsymbol{h}$

$*$ вытрома $\gamma\, \boldsymbol{Y}\, \boldsymbol{h}\, \boldsymbol{i}\, \boldsymbol{j}\, 1\, \boldsymbol{Z}$ //

$\overline{\triangle}\, e\, \alpha_e \Rightarrow \alpha_d\, \beta_e \Rightarrow \beta_d\, \overline{\triangle}\, d \rightarrow 1$

§ 4 $e \Rightarrow b_\alpha \Rightarrow b_\beta$.

§ 8. К максимальному сжатию троичных матриц

Простая матрица поглощений. Любая безызбыточная форма матрицы $\boldsymbol{U}$ соответствует некоторому безызбыточному покрытию булевой матрицы поглощений $\boldsymbol{E} \equiv \|R\ \mathrm{abs}\ M^1\|$, то есть такой совокупности

строк матрицы E, которая покрывает все ее столбцы, но лишается этого свойства при удалении любой строки совокупности. Между тем, матрица E сама может обладать своего рода избыточностью.

В самом деле, каждый столбец этой матрицы задает некоторое элементарное условие, представляя перечень определенных строк матрицы R, из которых в любую безызбыточную форму должна входить по крайней мере одна строка. Но если некоторые столбцы e^i и e^j матрицы E находятся в отношении $e^i \geqslant e^j$, то это означает, что условие, представляемое столбцом e^i, является избыточным, так как оно всегда удовлетворяется при выполнении условия, представляемого столбцом e^j. Следовательно, столбец e^i может быть выброшен из матрицы поглощений.

Например, в знакомой нам по параграфу 2.6 матрице

$$E = \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \left[\begin{array}{cccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ \text{А} \\ \text{Б} \\ \text{В} \\ \text{Г} \\ \text{Д} \\ \text{Е} \end{array}$$

$e^2 > e^3$, $e^5 > e^7$, $e^{10} = e^8$ и т. д., в силу чего матрица может быть сокращена до следующей:

$$\begin{array}{c} \begin{array}{cccccc} 1 & 4 & 7 & 8 & 11 \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ \text{А} \\ \text{Б} \\ \text{В} \\ \text{Г} \\ \text{Д} \\ \text{Е} \end{array}$$

Теперь становится очевидным, что покрытие {А, В, Г, Д, Е} этой матрицы является не только кратчайшим, но и единственным безызбыточным покрытием.

Описанный процесс сокращения матрицы поглощений приводит к матрице, в которой уже не существует таких столбцов e^i и e^j $(i \neq j)$, которые находятся в отношении $e^i \geqslant e^j$. Будем называть такую матрицу *простой матрицей поглощений*, а представляемые столбцами

этой матрицы подмножества строк матрицы R — *простыми совокупностями.*

Вариант задачи о максимальном сжатии. Выше было показано, что задача нахождения кратчайшей формы некоторой троичной матрицы U может быть разложена на две задачи: получение матрицы R, состоящей из максимальных строк, и выбор в ней некоторой минимальной совокупности строк, представляющей собой искомое решение. В свою очередь, вторая из этих задач распадается на две: построение булевой матрицы поглощений $\|R \operatorname{abs} M^1\|$ и нахождение одного из кратчайших покрытий этой матрицы.

Мощность множества M^1 в некоторых случаях может оказаться слишком большой и построение матрицы $\|R \operatorname{abs} M^1\|$ будет поэтому практически невозможным (у нее было бы слишком много столбцов). В то же время для решения поставленной задачи матрица поглощений в целом нам не нужна, а требуется лишь простая матрица поглощений, число столбцов в которой может оказаться достаточно малым. Такая ситуация возникает, например, в тех случаях, когда матрица R обладает значительным числом столбцов, но большинство элементов этой матрицы имеют значение «—».

В связи со сказанным поставим задачу непосредственного получения простой матрицы поглощений из матрицы R. Другими словами, в матрице R требуется выделить все простые совокупности.

Поиск простых совокупностей. Если в матрице R найдется такая строка r_i, для которой $I(r_i) \not\subseteq M(R_i^-)$, где R_i^- — минор, получаемый из матрицы R удалением строки r_i, то это означает, что строка r_i входит в любую безызбыточную форму, то есть существует простая совокупность, содержащая только строку r_i. Очевидно, что в этом случае никакая другая простая совокупность не содержит строку r_i. Если же $I(r_i) \subseteq M(R_i^-)$, то это значит, что существует такая безызбыточная форма, которая не содержит строки r_i и, следовательно, подмножество, которому принадлежит только строка r_i, не будет являться простой совокупностью.

Допустим теперь, что $I(r_i) \subseteq M(R_i^-)$, $I(r_j) \subseteq M(R_j^-)$ и в то же время $I(r_i) \cap I(r_j) \not\subseteq M(R_{ij}^-)$, где через R_{ij}^-

обозначен минор, получаемый из матрицы $\boldsymbol{R}$ удалением строк r_i и r_j. В этом случае нет таких элементов булева пространства M, которые поглощались бы только строкой r_i или только строкой r_j. Существуют, однако, такие элементы, которые поглощаются как строкой r_i, так и строкой r_j, но не поглощаются ни одной из остальных строк матрицы $\boldsymbol{R}$. Следовательно, в этом и только в этом случае подмножество $\{r_i, r_j\}$ должно входить в число простых совокупностей.

Эти правила позволяют найти все простые совокупности, содержащие по одной или по две строки матрицы $\boldsymbol{R}$, то есть найти такие столбцы матрицы поглощений, которые содержат одну или две единицы.

Используя традиционную символику теории множеств, будем обозначать через $\{I(r_k)/r_k \in R_i\}$ множество интервалов, соответствующих тем строкам матрицы $\boldsymbol{R}$, которые принадлежат некоторой совокупности R_i. Далее, пусть $\bigcup[A]$ — объединение всех элементов множества A, а $\bigcap[A]$ — пересечение этих элементов (очевидно, что данные операции имеют смысл только в том случае, когда элементы множества A сами являются подмножествами некоторого множества).

Обобщая предыдущие правила, сформулируем следующий критерий, позволяющий найти все простые совокупности: *подмножество R_i из множества R всех строк троичной матрицы $\boldsymbol{R}$ представляет собой простую совокупность, если*

$$\bigcap[\{I(r_k)/r_k \in R_i\}] \not\subseteq \bigcup[\{I(r_l)/r_l \in R \setminus R_i\}]$$

и если не существует такого подмножества R_j из R, что $R_j \subset R_i$ и

$$\bigcap[\{I(r_k)/r_k \in R_j\}] \not\subseteq \bigcup[\{I(r_l)/r_l \in R \setminus R_j\}].$$

Метод построения простой матрицы поглощений. Теперь становится ясно, как можно решить эту задачу. Сначала следует рассмотреть все строки матрицы $\boldsymbol{R}$ по отдельности и, пользуясь введенным критерием, выяснить, не образует ли какая-нибудь из них простую совокупность. Если такие строки находятся, то построение простой матрицы поглощений начинается с включения в нее соответствующих столбцов с одной единицей каждый. Затем перебираются пары среди

остальных строк матрицы R, отыскиваются двухэлементные простые совокупности и в матрицу поглощений добавляются соответствующие им столбцы, содержащие по две единицы. После этого производится перебор сочетаний по три строки из строк матрицы R, причем не рассматриваются такие сочетания, которые включают в себя простые одно- или двухэлементные совокупности. Нахождение простых трехэлементных совокупностей также сопровождается добавлением соответствующих им столбцов в простую матрицу поглощений.

Процесс продолжается аналогичным образом и заканчивается, когда уже нельзя найти таких сочетаний высшего ранга (то есть содержащих большее число строк), которые не включали бы в себя некоторую простую совокупность низшего ранга.

Таким образом, мы получаем простую матрицу поглощений, представляющую множество всех простых совокупностей. Ее можно рассматривать как некоторую компактную форму, из которой можно получить все безызбыточные троичные матрицы, эквивалентные исходной матрице U. Каждая из этих матриц представляет собой некоторый строчный минор матрицы R — расширения матрицы U. Нахождение кратчайшего покрытия простой матрицы поглощений приводит нас к получению кратчайшей из матричных форм, эквивалентных матрице U.

Допустим теперь, что мы ограничены рассмотрением лишь таких матриц, строки которых выбираются из множества U строк произвольной троичной матрицы U, и среди множества этих матриц ищем такую, которая окажется эквивалентной матрице U и будет при этом обладать минимальным числом строк. Эта задача может быть решена удалением из матрицы U некоторых строк (надо лишь найти, каких именно), в связи с чем назовем процесс ее решения *прямым сжатием матрицы* U.

Поскольку данная задача отличается от рассмотренной выше задачи нахождения кратчайшей формы матрицы U лишь тем, что множество U играет здесь формально роль множества R, то она может быть решена тем же методом, в котором опускается лишь первый этап — получение расширения матрицы U. Описанная выше процедура построения простой матрицы поглощений применяется теперь непосредственно к матрице U,

а не к ее расширению R. В результате этого находится булева матрица, которую мы будем условно называть также простой матрицей поглощений и которая будет представлять множество таких совокупностей из U, которые мы также условно будем называть простыми совокупностями.

Пример. Проиллюстрируем описанный процесс решения на примере следующей матрицы:

$$U = \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \left[\begin{array}{ccccccc} 1 & - & - & 0 & 1 & - & 0 \\ 1 & - & 0 & 0 & 1 & - & 1 \\ 0 & 0 & - & - & - & 0 & - \\ - & - & 1 & - & 1 & 1 & 1 \\ 1 & 1 & - & - & - & - & 0 \\ - & - & - & 0 & 0 & - & 0 \\ - & 0 & - & 0 & - & 0 & 0 \\ 1 & - & - & 0 & 1 & 1 & - \\ 1 & - & 0 & - & - & 0 & 1 \end{array}\right] \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array} \end{array}$$

Рассмотрим первую строку этой матрицы. Согласно сформулированному в предыдущем параграфе правилу, условие $I(u_1) \not\subseteq M(U_1^-)$ будет выполняться в том и только в том случае, когда минор

$$\begin{array}{c} \begin{array}{ccc} 2 & 3 & 6 \end{array} \\ \left[\begin{array}{ccc} 1 & - & - \\ 0 & - & 0 \\ - & - & 1 \end{array}\right] \begin{array}{c} e \\ g \\ h \end{array} \end{array}$$

образованный столбцами 2, 3 и 6, в которых строка a имеет значение «—», и строками e, g и h, пересекающимися со строкой a, будет невырожденным. Пользуясь алгоритмом анализа матрицы на вырожденность, мы легко устанавливаем вырожденность этого минора. Следовательно, строка a не представляет собой простую совокупность.

Аналогичный результат получается при рассмотрении строки b. Однако, уже следующая за ней строка c образует простую совокупность, поскольку соответствующий ей минор

$$\begin{array}{c} \begin{array}{cccc} 3 & 4 & 5 & 7 \end{array} \\ \left[\begin{array}{cccc} - & 0 & 0 & 0 \\ - & 0 & - & 0 \end{array}\right] \begin{array}{c} f \\ g \end{array} \end{array}$$

оказывается, как нетрудно видеть, невырожденным (например, элемент 1 1 1 1 не поглощается ни одной из строк этого минора). Следовательно, одноэлементное множество $\{c\}$ является простой совокупностью.

Продолжая анализ одиночных строк, мы находим следующие простые совокупности: $\{d\}$, $\{e\}$, $\{f\}$ и $\{i\}$. На этом перебор строк по одиночке завершается.

Далее рассматриваются пары, выбираемые теперь среди остальных строк a, b, g и h. Оказывается, что в данном случае существуют только три пары взаимно пересекающихся строк: $\{a, g\}$, $\{a, h\}$ и $\{b, h\}$. Рассмотрим первую из них.

Пересечение строк a и g образует новую строку 1 0—0 1 0 0, соответствующую интервалу $I(a) \cap I(g)$ (пересечение двух интервалов, если это не пустое множество, всегда является интервалом). Поскольку все остальные строки матрицы $\boldsymbol{U}$ ортогональны этой новой строке, нетрудно заключить, что пара $\{a, g\}$ представляет собой простую совокупность. Следующая пара $\{a, h\}$ порождает строку 1— —0 1 1 0, которой соответствует минор

$$\begin{matrix} 2 & 3 \\ [1 & -] \end{matrix} e$$

явно оказывающийся невырожденным, благодаря чему мы должны признать, что совокупность $\{a, h\}$ также является простой. Аналогичным образом мы устанавливаем простоту последней совокупности $\{b, h\}$.

Далее имело бы смысл рассматривать такие тройки строк, которые не включают в себя уже найденных простых совокупностей, содержащих один или два элемента. Но таких троек не существует, поэтому поиск простых совокупностей прекращается, а полученный результат представляется в следующей матричной форме:

$$\left[\begin{array}{c|c} 0\,0\,0\,0\,0\,1\,1\,0 & a \\ 0\,0\,0\,0\,0\,0\,0\,1 & b \\ 1\,0\,0\,0\,0\,0\,0\,0 & c \\ 0\,1\,0\,0\,0\,0\,0\,0 & d \\ 0\,0\,1\,0\,0\,0\,0\,0 & e \\ 0\,0\,0\,1\,0\,0\,0\,0 & f \\ 0\,0\,0\,0\,0\,1\,0\,0 & g \\ 0\,0\,0\,0\,0\,0\,1\,1 & h \\ 0\,0\,0\,0\,1\,0\,0\,0 & i \end{array}\right]$$

Кратчайшее покрытие такой матрицы находится весьма просто. Например, искомым решением может служить множество $\{a, b, c, d, e, f, i\}$.

Следовательно, из исходной матрицы $\boldsymbol{U}$ можно выбросить строки g и h и тем самым максимально сжать эту матрицу:

$$\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \left[\begin{array}{ccccccc} 1 & - & - & 0 & 1 & - & 0 \\ 1 & - & 0 & 0 & 1 & - & 1 \\ 0 & 0 & - & - & - & 0 & - \\ - & - & 1 & - & 1 & 1 & 1 \\ 1 & 1 & - & - & - & - & 0 \\ - & - & - & 0 & 0 & - & 0 \\ 1 & - & 0 & - & - & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \\ i \end{array}$$

О переборе сочетаний строк. При рассмотрении достаточно большой исходной матрицы $\boldsymbol{U}$ становится насущной задача существенного сокращения числа перебираемых сочетаний строк этой матрицы. Решить эту задачу можно на основе следующих соображений.

Во-первых, есть смысл рассматривать только такие сочетания, которые не содержат взаимно ортогональных строк. Во-вторых, среди этих сочетаний следует рассматривать только такие, которые не содержат в качестве подмножества какую-либо простую совокупность.

Учитывая эти соображения, можно предложить алгоритм перебора сочетаний, который работает следующим образом. Сначала он действует так, как это было показано на рассмотренном примере, то есть выявляет все строки, образующие одноэлементные простые совокупности, а среди остальных строк находит пары неортогональных строк и среди них также выявляет простые совокупности. Обозначим множество остальных пар неортогональных строк через S_2. Если в этом множестве найдется тройка пар типа $\{a, b\}$, $\{a, c\}$ и $\{b, c\}$, то это означает, что есть смысл рассмотреть в дальнейшем сочетание из трех строк $\{a, b, c\}$. Удовлетворяющее такому условию сочетание анализируется и, если оно оказывается простой совокупностью, вносится в окончательный результат. В противном случае это сочетание вносится в формируемое таким образом множество S_3 трехэлементных сочетаний. Аналогичным образом множество

S_3 служит базой для получения множества S_4, при этом некоторая совокупность $\{a, b, c, d\}$ рассматривается в том и только в том случае, когда в множестве S_3 присутствуют совокупности $\{a, b, c\}$, $\{a, b, d\}$, $\{a, c, d\}$ и $\{b, c, d\}$, и т. д.

Нахождение ядра безызбыточных форм. Если в булевой матрице поглощений $\boldsymbol{E} \equiv \|R \operatorname{abs} M^1\|$ существует некоторый столбец, содержащий лишь одну единицу, то представляемый этой единицей элемент множества $R \equiv R(M(\boldsymbol{U}))$ будет входить в любую безызбыточную форму матрицы $\boldsymbol{U}$. Совокупность таких элементов множества R назовем *ядром безызбыточных форм* матрицы $\boldsymbol{U}$.

Предварительное выявление ядра может значительно облегчить решение задачи нахождения кратчайшей формы матрицы $\boldsymbol{U}$ и уже в силу этого представляет существенный интерес.

Как мы видели, строка $\boldsymbol{r}_i$ матрицы $\boldsymbol{R}$ образует простую совокупность, если она поглощает некоторый элемент пространства M, не поглощаемый никакой другой строкой матрицы $\boldsymbol{R}$. Именно на проверке этого критерия, называемого далее критерием простоты, и основан рассмотренный ранее алгоритм устиз, который удаляет очередную строку матрицы $\boldsymbol{R}$, если эта строка не образует простую совокупность, и получает таким образом в итоге некоторую безызбыточную форму.

Можно дать и следующую формулировку критерия простоты, эквивалентную предыдущей: строка матрицы $\boldsymbol{R}$ образует простую совокупность, если она поглощает некоторый элемент пространства M, находящийся на расстоянии не ближе чем 1 от любой другой строки матрицы $\boldsymbol{R}$. При этом под расстоянием понимается число компонент, в которых один из сравниваемых векторов имеет значение 0, а другой — значение 1. Например, расстояние между векторами 1 — — 0 0 1 — — и 0 — 1 0 — 0 1 0 равно 2.

Поскольку, по определению, ядро безызбыточных форм содержится в любой безызбыточной форме, представляет интерес его выявление непосредственно там, без предварительного построения множества R. Очевидно, что вся необходимая для этого информация должна содержаться в самой безызбыточной форме, получение же

такой формы, как это было показано ранее, производится относительно легко.

Докажем следующее утверждение: *строка* u_i *произвольной троичной матрицы* U *является элементом ядра безызбыточных форм этой матрицы в том и только в том случае, когда она поглощает некоторый элемент пространства* M, *находящийся на расстоянии не ближе чем* 1 *от любой другой строки матрицы* U, *пересекающейся со строкой* u_i, *и на расстоянии не ближе чем* 2 *от любой из строк матрицы* U, *ортогональных строке* u_i.

Допустим, что указанный элемент существует, и обозначим его через m_j. Введем в рассмотрение множество V всех троичных векторов (обозначаемых через v_p, v_q, v_r, ...), число компонент в каждом из которых совпадает с числом столбцов матрицы U. Разобьем это множество на классы $V(a, b)$, параметры которых a и b принимают значения, равные расстоянию между элементами задаваемого класса и векторами u_i и m_j, соответственно. Обратим особое внимание на классы $V(0, 0)$, $V(0, 1)$, $V(1, 1)$, $V(1, 2)$ и $V(2, 2)$ и введем, кроме того, класс $V^*(0, 0) = V(0, 0) \setminus \{u_i\}$.

Например, если

$$u_i = 0 - - 1 - 0\ 0 - 0,$$
$$m_j = 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0,$$

то

$$\begin{aligned}
&- 0 - 1\ 1\ 0 - - 0 \in V^*(0,0),\\
&0\ 1\ 1 - - - 0\ 1 - \in V\ (0,1),\\
&- - 1\ 0 - 0\ 0 - - \in V\ (1,1),\\
&- - 0 - 1\ 1 - 1\ 0 \in V\ (1,2),\\
&- 0\ 1\ 1 - 1\ 0 - 1 \in V\ (2,2).
\end{aligned}$$

Согласно предположению, в матрице U отсутствуют строки, принадлежащие классам $V^*(0, 0)$ и $V(1, 1)$. В то же время очевидно, что строка u_i входит в ядро безызбыточных форм, если в результате всевозможных обобщенных склеиваний строк матрицы U не появится такая строка, которая поглотит элемент m_j, то есть строка, принадлежащая классу $V^*(0, 0)$. Посмотрим, может ли это произойти.

Для того чтобы вектор v_r, являющийся результатом обобщенного склеивания векторов v_p и v_q, принадлежал классу $V^*(0, 0)$, необходимо, чтобы выполнялось (с точностью до перестановки между индексами p и q) одно из следующих условий:

$$v_p \in V^*(0, 0), \quad v_q \in V(0, 1);$$
$$v_p \in V^*(0, 0), \quad v_q \in V(1, 1);$$
$$v_p \in V(1, 1), \quad v_q = u_l.$$

На доказательстве этого утверждения мы останавливаться не будем, поскольку оно не представляет особых затруднений. Из утверждения следует, что интересующий нас вектор может быть получен только в том случае, когда предварительно будет получен некоторый вектор, принадлежащий классу $V(1, 1)$.

В свою очередь, нетрудно показать, что для того чтобы вектор v_r, являющийся результатом обобщенного склеивания векторов v_p и v_q, принадлежал классу $V(1, 1)$, необходимо, чтобы выполнялось одно из следующих условий (также с точностью до перестановки между индексами p и q):

$$v_p \in V^*(0, 0), \quad v_q \in V(1, 2);$$
$$v_p \in V^*(0, 0), \quad v_q \in V(2, 2);$$
$$v_p \in V(1, 1), \quad v_q \in V(0, 1);$$
$$v_p \in V(1, 1), \quad v_q \in V(1, 1);$$
$$v_p \in V(1, 1), \quad v_q \in V(1, 2);$$
$$v_p \in V(1, 1), \quad v_q \in V(2, 2).$$

Другими словами, необходимо, чтобы по крайней мере один из векторов v_p и v_q принадлежал классу $V^*(0, 0)$ или классу $V(1, 1)$. Это значит, что если таких векторов нет в матрице U с самого начала, то они и не могут появиться при расширении этой матрицы за счет введения продуктов обобщенного склеивания ее строк.

Таким образом, достаточность сформулированного условия можно считать доказанной. Нетрудно доказать и его необходимость.

Действительно, если условие не выполняется, то это означает, что каждый элемент m_j пространства M,

поглощаемый рассматриваемой строкой u_i матрицы U, поглощается некоторой максимальной строкой, отличной от строки u_i: если в матрице U существует строка, пересекающаяся со строкой u_i и находящаяся на расстоянии 0 от элемента m_j, то именно эта строка и поглощает его, если же в матрице U существует строка u_k, ортогональная строке u_i и находящаяся на расстоянии 1 от элемента m_j, то последний будет поглощен продуктом обобщенного склеивания строк u_i и u_k. Отсюда непосредственно следует существование такой безызбыточной формы матрицы U, в которой строка u_i будет отсутствовать.

Таким образом, справедливость критерия простоты при рассмотрении строк произвольной троичной матрицы полностью доказана.

Приведение критерия простоты к более простому виду. В применении к строкам расширенной матрицы R критерий простоты выглядит довольно простым, в случае же произвольной троичной матрицы U он представляется несколько более сложным. Однако оба случая допускают единообразную и весьма простую формулировку.

Объединяя результаты, полученные в последних двух параграфах, сформулируем критерий простоты строк расширенной матрицы R в следующем виде: *для того чтобы строка r_i расширенной матрицы R представляла собой простую совокупность, необходимо и достаточно, чтобы минор R_i матрицы R, образованный столбцами, в которых строка r_i имеет значение «—», и теми из остальных строк, которые пересекаются со строкой r_i, не был вырожденным.*

Обратимся теперь к рассмотрению некоторой строки u_i произвольной троичной матрицы U. Если эта строка поглощает некоторый элемент m_j пространства M, находящийся на расстоянии не ближе чем 1 от любой другой строки матрицы U, пересекающейся со строкой u_i, и на расстоянии не ближе чем 2 от любой из строк матрицы U, ортогональных строке u_i, то это значит, что вектор, образованный из тех компонент вектора m_j, соответствующие которым компоненты строки u_i имеют значение «—», не будет поглощаться ни одной из строк минора U_i^* матрицы U, образованного столбцами, в кото-

рых строка u_i имеет значение «—», и теми из остальных строк матрицы U, которые находятся от строки u_i на расстоянии не далее чем 1. Справедливо и обратное утверждение: если существует некоторый булев вектор, не поглощаемый ни одной из строк вышеуказанного минора U_i^*, то подстановка компонент этого вектора в строку u_i на места, занимаемые символами «—», приводит к получению элемента m_j с указанными свойствами.

Отсюда вытекает следующая формулировка критерия простоты строк произвольной троичной матрицы U: *для того чтобы строка u_i матрицы U представляла собой простую совокупность, необходимо и достаточно, чтобы минор U_i^* матрицы U, образованный столбцами, в которых строка u_i имеет значение «—», и теми из остальных строк, которые пересекаются со строкой u_i или являются смежными с ней, не был вырожденным.*

Таким образом, рассматриваемая задача нахождения элементов ядра безызбыточных форм как в случае расширенной матрицы R, так и в случае безызбыточной матрицы U сводится к знакомой нам по предыдущему параграфу задаче анализа некоторой троичной матрицы (минора) на вырожденность. Решение же этой задачи достаточно эффективно может осуществляться с помощью программы в ы т р о м а.

Следует лишь заметить, что сформулированное правило позволяет отыскивать элементы ядра в любой троичной матрице U, если только они там присутствуют. Однако их там может и не быть, в связи с чем целесообразно предварительно найти некоторую безызбыточную матрицу, эквивалентную матрице U, и искать элементы ядра уже в ней, поскольку там их присутствие гарантировано, если они существуют вообще.

П р и м е р ы. Рассмотрим следующую безызбыточную матрицу:

$$U = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & - & 0 & - & 0 & - & - \\ - & 0 & - & - & 1 & 1 & - \\ - & 1 & 0 & - & - & 0 & 1 \\ - & 0 & - & 1 & - & - & - \\ 1 & - & 1 & 0 & - & - & 0 \\ 1 & 0 & 0 & - & - & - & - \end{array}\right] \end{array} \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix}$$

Посмотрим, входит ли строка a этой матрицы в ядро безызбыточных форм. Чтобы получить однозначный ответ на этот вопрос, нужно рассмотреть минор U_a^*, образованный пересечением столбцов 2, 4, 6 и 7, в которых строка a имеет значение «—», со строками, расстояние от которых до строки a не превышает 1 — такому условию здесь удовлетворяют все остальные строки матрицы.

$$U_a^* = \begin{array}{c} \begin{array}{cccc} 2 & 4 & 6 & 7 \end{array} \\ \left[\begin{array}{cccc} 0 & - & 1 & - \\ 1 & - & 0 & 1 \\ 0 & 1 & - & - \\ - & 0 & - & 0 \\ 0 & - & - & - \end{array}\right] \end{array} \begin{array}{l} b \\ c \\ d \\ e \\ f \end{array}$$

Легко устанавливается невырожденность этого минора. Для этого можно применить описанный в предыдущем параграфе алгоритм поиска вектора, ортогонального каждой из строк минора (программа вытрома). Таким вектором оказывается, например, 1111. Подстановка компонент этого вектора в строку a на места, занимаемые символами «—», приводит к получению элемента 1101011 пространства M, который поглощается строкой a, но не поглощается ни одной из остальных строк рассматриваемой матрицы (при формулировке критерия простоты мы обозначали такой элемент через m_j). Следовательно, строка a образует простую совокупность, то есть принадлежит ядру безызбыточных форм.

Аналогично проводится анализ всех других строк матрицы, которым соответствуют следующие миноры:

$$U_b^* = \begin{array}{c} \begin{array}{cccc} 1 & 3 & 4 & 7 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & - & - \\ - & - & 1 & - \\ 1 & 1 & 0 & 0 \\ 1 & 0 & - & - \end{array}\right] \end{array} \begin{array}{l} a \\ d \\ e \\ f \end{array}, \quad U_c^* = \begin{array}{c} \begin{array}{ccc} 1 & 4 & 5 \end{array} \\ \left[\begin{array}{ccc} 1 & - & 0 \\ - & 1 & - \\ 1 & - & - \end{array}\right] \end{array} \begin{array}{l} a \\ d \\ f \end{array},$$

$$U_d^* = \begin{array}{c} \begin{array}{ccccc} 1 & 3 & 5 & 6 & 7 \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & - & - \\ - & - & 1 & 1 & - \\ - & 0 & - & 0 & 1 \\ 1 & 1 & - & - & 0 \\ 1 & 0 & - & - & - \end{array}\right] \end{array} \begin{array}{l} a \\ b \\ c \\ e \\ f \end{array},$$

$$U_e^* = \begin{array}{c} \begin{matrix} 2 & 5 & 6 \end{matrix} \\ \left[\begin{matrix} - & 0 & - \\ 0 & 1 & 1 \\ 0 & - & - \\ 0 & - & - \end{matrix}\right] \end{array} \begin{matrix} a \\ b \\ d \\ f \end{matrix}, \quad U_f^* = \begin{array}{c} \begin{matrix} 4 & 5 & 6 & 7 \end{matrix} \\ \left[\begin{matrix} - & 0 & - & - \\ - & 1 & 1 & - \\ - & - & 0 & 1 \\ 1 & - & - & - \\ 0 & - & - & 0 \end{matrix}\right] \end{array} \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix}.$$

Миноры U_b^*, U_c^*, U_d^* и U_e^* оказываются невырожденными: например, находится вектор 0000, ортогональный всем строкам минора U_b^*, и векторы 000, 00000 и 111, ортогональные строкам миноров U_c^*, U_d^* и U_e^*, соответственно. Следовательно, строки b, c, d и e входят в ядро безызбыточных форм наравне со строкой a. Исключение составляет строка f: соответствующий ей минор U_f^* оказывается вырожденным.

Очевидно, что просто удалить строку f из матрицы U нельзя, поскольку матрица U является безызбыточной. Может быть, эту строку можно было бы заменить какой-либо другой строкой, однако такая замена не привела бы к сокращению строк в исходной матрице U. Отсюда следует, что матрица U задана в кратчайшей форме.

Заметим, что другой путь решения этой же задачи — проведение последовательности всевозможных обобщенных склеиваний строк исходной матрицы, построение таким образом расширенной матрицы R, затем поиск простых совокупностей строк этой матрицы, построение простой матрицы поглощений и нахождение одного из ее кратчайших покрытий — привел бы к аналогичному результату, но оказался бы в данном случае более громоздким.

Допустим теперь, что

$$U = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \left[\begin{matrix} - & - & - & 1 & 1 & 0 \\ 0 & 1 & - & - & 1 & 0 \\ 1 & - & - & 0 & 0 & - \\ - & - & 1 & - & 1 & 1 \\ - & 1 & 0 & 0 & 1 & - \\ 1 & 1 & - & - & 0 & - \\ - & 1 & 0 & 0 & - & 1 \end{matrix}\right] \end{array} \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix}$$

Предварительно убедившись в безызбыточности строк данной матрицы, найдем элементы ядра безызбыточных форм, для чего проанализируем следующие миноры матрицы U:

$$U_a^* = \begin{matrix} 1\ 2\ 3 \\ \left[\begin{matrix} 0\ 1\ - \\ -\ -\ 1 \\ -\ 1\ 0 \\ 1\ 1\ - \end{matrix}\right] \end{matrix}, \quad U_b^* = \begin{matrix} 3\ 4 \\ \left[\begin{matrix} -\ 1 \\ 1\ - \\ 0\ 0 \\ 0\ 0 \end{matrix}\right] \end{matrix}, \quad U_c^* = \begin{matrix} 2\ 3\ 6 \\ \left[\begin{matrix} -\ 1\ 1 \\ 1\ 0\ - \\ 1\ -\ - \\ 1\ 0\ 1 \end{matrix}\right] \end{matrix},$$

$$U_d^* = \begin{matrix} 1\ 2\ 4 \\ \left[\begin{matrix} -\ -\ 1 \\ 0\ 1\ - \\ 1\ -\ 0 \\ -\ 1\ 0 \\ 1\ 1\ - \\ -\ 1\ 0 \end{matrix}\right] \end{matrix}, \quad U_e^* = \begin{matrix} 1\ 6 \\ \left[\begin{matrix} -\ 0 \\ 0\ 0 \\ 1\ - \\ -\ 1 \\ 1\ - \\ -\ 1 \end{matrix}\right] \end{matrix},$$

$$U_f^* = \begin{matrix} 3\ 4\ 6 \\ \left[\begin{matrix} -\ 1\ 0 \\ -\ 0\ - \\ 1\ -\ 1 \\ 0\ 0\ - \\ 0\ 0\ 1 \end{matrix}\right] \end{matrix}, \quad U_g^* = \begin{matrix} 1\ 5 \\ \left[\begin{matrix} 0\ 1 \\ 1\ 0 \\ -\ 1 \\ -\ 1 \\ 1\ 0 \end{matrix}\right] \end{matrix}.$$

Матрицы U_b^* и U_e^* оказываются вырожденными, остальные — невырожденными. Соответствующие последним строки образуют ядро безызбыточных форм $\{a, c, d, f, g\}$. Под сомнением остаются строки b и e, однако их только две, и разрешить эти сомнения нетрудно.

С помощью алгоритма, воплощенного в программе вытрома, мы находим булев вектор 011010, поглощаемый строкой b и только этой строкой матрицы U, а также вектор 110010, находящийся в аналогическом отношении со строкой e. Если бы существовала строка, поглощающая оба этих элемента из M^1, но не поглощающая ни одного элемента из $M \setminus M^1$, то можно было бы заменить ею обе строки b и e. Однако легко убедиться, что такой строки не существует, поскольку любой интервал пространства M, содержащий элементы 011010 и 110010, будет одновременно содержать и элемент 111010, принадлежащий множеству $M \setminus M^1$.

Следовательно, рассматриваемая форма является кратчайшей.

Нахождение квазиядра кратчайшей формы. При поиске одной из кратчайших форм всегда можно ограничиться рассмотрением лишь максимальных строк. Действительно, то, что вообще существует какая-то кратчайшая форма, сомнений не вызывает. Теперь допустим, что не все ее строки максимальны. Заменив каждую из этих строк той максимальной строкой, которая ее поглощает, мы получим матрицу, все строки которой максимальны и которая тоже будет кратчайшей, поскольку число строк матрицы при этом не изменяется.

Рассмотрим теперь безызбыточную матрицу $\boldsymbol{U}$, некоторые строки которой принадлежат ядру безызбыточных форм. Совокупность элементов булева пространства M, поглощаемых по крайней мере одной из этих строк, обозначим через M_c. Допустим, что $M_c \neq M^1$, то есть множество принадлежащих ядру строк еще не образует решения. В этом случае задача сводится к нахождению некоторой минимальной совокупности троичных векторов, образующих совместно с ядром матрицу, эквивалентную исходной матрице $\boldsymbol{U}$.

Очевидно, что, решая эту последнюю задачу, уже не нужно беспокоиться о поглощении элементов множества M_c, а можно ограничиться поиском минимального числа строк, поглощающих в совокупности остаток $M^1 \setminus M_c$ (но, разумеется, не поглощающих ни одного элемента из множества $M \setminus M^1$). При этом разумно скорректировать понятие максимальности, определяя отношение поглощения между строками лишь на непоглощенном остатке множества M^1, а именно, на множестве $M^1 \setminus M_c$.

Пусть, например, известно, что строки $\boldsymbol{u}_i$ и $\boldsymbol{u}_j$ матрицы $\boldsymbol{U}$ максимальны и, естественно, не находятся в отношении поглощения. Допустим, однако, что найдено такое ядро безызбыточных форм, что совокупность непоглощаемых им элементов из M^1 образует множество $M^1 \setminus M_c$. Если любой элемент этого множества, поглощаемый строкой $\boldsymbol{u}_i$, будет поглощаться и строкой $\boldsymbol{u}_j$, будем говорить, что строка $\boldsymbol{u}_j$ поглощает строку $\boldsymbol{u}_i$ на множестве $M^1 \setminus M_c$. Решая дальше задачу нахождения

кратчайшей формы, можно в этом случае выбросить из рассмотрения строку u_i: если существует кратчайшая форма, содержащая данную строку, то другая форма, в которой строка u_i заменена на u_j, также будет кратчайшей.

В частности, если некоторая строка не поглощает ни одного элемента из множества $M^1 \setminus M_c$, то нет смысла пытаться использовать ее в решении: можно считать, что она поглощается на множестве $M^1 \setminus M_c$ любой другой строкой. Можно назвать такую строку элементом *антиядра безызбыточных форм.* Впрочем, выявление таких строк особого значения не имеет, поскольку они, конечно, не будут привлекаться для поглощения элементов множества $M^1 \setminus M_c$.

Несомненный практический интерес представляет нахождение *квазиядра кратчайшей формы*, как мы назовем такую совокупность строк матрицы U, которая будет входить по крайней мере в одну из кратчайших форм этой матрицы. Таких квазиядер может существовать много, так как непосредственно из определения следует, что квазиядром кратчайшей формы может служить любая кратчайшая форма и любая ее часть.

Процесс нахождения квазиядра кратчайшей формы может носить цепной характер, поскольку нахождение очередного элемента квазиядра облегчает поиск следующего его элемента. Так как ядро безызбыточных форм является, очевидно, также квазиядром кратчайшей формы, этот процесс может начинаться с нахождения этого ядра. Более подробно этот процесс будет рассматриваться в следующей главе.

§ 9. Согласование частичных двублочных разбиений

Частичные двублочные разбиения. *Двублочным (полным) разбиением* множества A называется неупорядоченная пара его подмножеств A_i и A_j (называемых *блоками разбиения*), удовлетворяющая условиям

$$A_i \cap A_j = \varnothing \quad \text{и} \quad A_i \cup A_j = A.$$

Обобщая это понятие, введем в рассмотрение *частичные двублочные разбиения*, то есть такие пары подмно-

жеств A_i и A_j некоторого множества A, которые удовлетворяют первому из перечисленных условий ($A_i \cap A_j = \varnothing$), но не обязательно удовлетворяют второму.

Частичное двублочное разбиение может быть представлено троичным вектором, в котором значением 0 отмечены компоненты, соответствующие элементам одного из блоков, а значением 1 — другого.

Например, если $A = \{a, b, c, d, e, f, g, h\}$, то вектор

$$u = 1 - 00 - 11 -$$

представляет частичное разбиение множества A на блоки $\{a, f, g\}$ и $\{c, d\}$. Заметим, что при этом безразлично, какой из этих блоков мы обозначим через A_i, а какой — через A_j, так как, по определению, разбиение представляет собой неупорядоченную пару подмножеств. По этой же причине вектор

$$u' = 0 - 11 - 00 -,$$

инверсный вектору u, то есть полученный из него путем замены значений 0 на 1, а значений 1 — на 0, представляет то же самое частичное двублочное разбиение, что и вектор u. Иначе говоря, в данном употреблении векторы u и u' *эквивалентны*.

Конечно, можно считать, что троичный вектор задает некоторое трехблочное разбиение, то есть компоненты вектора, имеющие значение «—», соответствуют элементам третьего блока разбиения. Однако такой интерпретацией мы здесь пользоваться не будем. Вместо этого будем полагать, как и раньше, что символ «—» означает некоторую произвольность или неопределенность.

Будем рассматривать частичное двублочное разбиение, представляемое троичным вектором u, как *условие*, предъявляемое к некоторому искомому полному двублочному разбиению множества A. Это условие налагает определенные ограничения на элементы множества A, соответствующие компонентам вектора u, обладающим значениями 0 или 1. Считается, что элементы, соответствующие таким компонентам с различными значениями, должны принадлежать различным блокам

разбиения. Однако это условие не касается элементов множества A, соответствующих компонентам со значением «—»: такие элементы могут принадлежать любому блоку двублочного разбиения.

Договоримся называть далее в этом параграфе просто *разбиениями* как частичные, так и полные двублочные разбиения, которые, очевидно, можно рассматривать как частный случай частичных. Будем обозначать их теми же символами, что и представляющие их троичные векторы (оказывающиеся булевыми при представлении полных разбиений).

Будем также говорить, что полное разбиение *реализует* некоторое частичное, если оно отвечает всем содержащимся в данном частичном разбиении условиям.

Отношение импликации. Пусть a и b — некоторые абстрактные условия, предъявляемые к некоторому классу объектов. Если любой объект, удовлетворяющий условию a, удовлетворяет в то же время и условию b, будем говорить, что условие a *имплицирует* условие b, и записывать это в виде $a \Rightarrow b$.

С целью иллюстрации обратимся к задаче нахождения безызбыточного покрытия некоторой булевой матрицы $\boldsymbol{A}$. Решая эту задачу, мы ищем объект (некоторое подмножество строк матрицы $\boldsymbol{A}$), удовлетворяющий совокупности условий, каждое из которых задается одним из столбцов матрицы $\boldsymbol{A}$. Если столбцы $\boldsymbol{a}^i$ и $\boldsymbol{a}^j$ находятся в отношении $\boldsymbol{a}^i < \boldsymbol{a}^j$, то любая совокупность строк, покрывающая столбец $\boldsymbol{a}^i$, покроет также, как нетрудно видеть, и столбец $\boldsymbol{a}^j$. Следовательно, в данном случае условие, представляемое столбцом $\boldsymbol{a}^i$, имплицирует условие, представляемое столбцом $\boldsymbol{a}^j$. Именно поэтому последнее из них может быть опущено без ущерба для постановки задачи (что и делалось нами при решении указанной задачи: из матрицы удалялся столбец $\boldsymbol{a}^j$).

Вернемся теперь к теме настоящего параграфа. Пусть $\boldsymbol{u}_i$ и $\boldsymbol{u}_j$ — некоторые частичные двублочные разбиения множества A. Если любое полное разбиение этого множества, реализующее частичное разбиение $\boldsymbol{u}_i$, будет в то же время реализовывать и частичное разбиение $\boldsymbol{u}_j$, будем говорить, что разбиение $\boldsymbol{u}_i$ имплицирует разбиение $\boldsymbol{u}_j$ $(\boldsymbol{u}_i \Rightarrow \boldsymbol{u}_j)$.

Обозначив через A_{i1} и A_{i2} блоки разбиения u_i, а через A_{j1} и A_{j2} — блоки разбиения u_j, сформулируем следующее правило: разбиение u_i имплицирует разбиение u_j, если выполняется одно из условий:

а) $A_{j1} \subseteq A_{i1}$ и $A_{j2} \subseteq A_{i2}$,

б) $A_{j1} \subseteq A_{i2}$ и $A_{j2} \subseteq A_{i1}$.

Если $u_i \Rightarrow u_j$ и $u_j \Rightarrow u_i$, то разбиения u_i и u_j будем называть *эквивалентными* ($u_i \Leftrightarrow u_j$).

Например,

$$0-11-0 \Rightarrow 0--1-0,$$
$$00111\,0 \Rightarrow 1-0-01,$$
$$0-1101 \Leftrightarrow 0-1101,$$
$$0-1101 \Leftrightarrow 1-0010.$$

Матрицы разбиений. Так мы будем называть троичные матрицы, строки которых интерпретируются как некоторые частичные двублочные разбиения.

Будем говорить, что матрица U *имплицирует* некоторое разбиение w ($U \Rightarrow w$), если последнее имплицируется хотя бы одной из строк матрицы U. Будем также считать, что матрица U *имплицирует* матрицу V ($U \Rightarrow V$) в том и только в том случае, когда каждая из строк матрицы V, интерпретируемая как некоторое частичное разбиение, имплицируется матрицей U. Если же $U \Rightarrow V$ и $V \Rightarrow U$, будем считать, что матрицы U и V *эквивалентны* ($U \Leftrightarrow V$).

Матрицы разбиений легко канонизируются, то есть приводятся к такой форме, в которой эквивалентные матрицы становятся равными.

Каноническая форма матрицы разбиений может быть получена следующим образом. Сначала из матрицы удаляются все такие строки, которые имплицируются остающимися, затем инверсируются те строки, в которых первый слева символ 0 находится левее первого слева символа 1, наконец, производится упорядочение строк по некоторому общему для всех канонизируемых матриц правилу.

Например, канонизация матрицы разбиений

$$\left[\begin{array}{cccccccc} 1 & 1 & - & - & - & 0 & - & 0 \\ - & 0 & - & - & - & 1 & - & - \\ - & 1 & 0 & 0 & - & - & - & - \\ 0 & 0 & - & - & - & - & - & 1 \\ - & 0 & - & 1 & - & - & - & - \\ - & - & - & 0 & 0 & - & 1 & 1 \end{array}\right] \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array}$$

приводит нас к следующей форме:

$$\left[\begin{array}{cccccccc} 1 & 1 & - & - & - & 0 & - & 0 \\ - & 1 & 0 & 0 & - & - & - & - \\ - & - & - & 1 & 1 & - & 0 & 0 \end{array}\right] \begin{array}{c} a \\ c \\ f' \end{array}$$

поскольку строка a имплицирует строки b и d, строка c имплицирует строку e, а строка f' является инверсией строки f.

Нетрудно показать, что каноническая форма является кратчайшей среди эквивалентных форм матрицы разбиений, то есть не существует такой матрицы разбиений $\boldsymbol{V}$, которая была бы эквивалентна матрице разбиений $\boldsymbol{U}$ и содержала бы меньшее число строк, чем в канонической форме матрицы $\boldsymbol{U}$.

Таким образом, задача нахождения кратчайшей эквивалентной формы заданной матрицы разбиений решается довольно легко. Значительно сложнее оказывается найти *кратчайшую среди имплицирующих форм.* К рассмотрению этой задачи мы сейчас и перейдем.

Нахождение кратчайшей имплицирующей формы. Рассмотрим следующую задачу. Задана некоторая матрица разбиений $\boldsymbol{U}$. Требуется найти троичную матрицу $\boldsymbol{V}$, имплицирующую матрицу $\boldsymbol{U}$ и содержащую минимально возможное число строк.

Назовем *общей импликантой* строк $\boldsymbol{u}_i, \boldsymbol{u}_j, \ldots, \boldsymbol{u}_k$ матрицы $\boldsymbol{U}$ такой троичный вектор, который имплицирует каждую из этих строк. Совокупность строк, для которой можно найти общую импликанту, назовем *совместимой.* Назовем *максимальной совместимой совокупностью* строк матрицы $\boldsymbol{U}$ такую совокупность, которая перестает быть совместимой при добавлении в нее любой из

остальных строк этой же матрицы. Общие импликанты максимальных совместимых совокупностей, причем такие, в которых ни один из символов 1 или 0 не может быть заменен символом «—» (без того, чтобы данный вектор не перестал быть общей импликантой рассматриваемой совокупности), назовем *главными импликантами* матрицы разбиений $\boldsymbol{U}$.

Последующие рассуждения весьма аналогичны тем, которые проводились при решении ряда других задач этой главы. Поэтому мы не будем излагать их подробно, а скажем лишь, что при поиске кратчайшей имплицирующей формы некоторой матрицы разбиений $\boldsymbol{U}$ можно ограничиться рассмотрением множества D ее главных импликант. После получения этого множества решаемая задача сводится к нахождению одного из кратчайших покрытий булевой матрицы

$$\boldsymbol{C} \equiv \|D \Rightarrow U\|,$$

задающей отношение импликации между главными импликантами и строками матрицы $\boldsymbol{U}$, множество которых обозначено здесь через U (элемент c_i^j матрицы $\boldsymbol{C}$ принимает значение 1, если $d_i \Rightarrow u_j$, и принимает значение 0 в противном случае). Совокупность главных импликант, соответствующих элементам найденного кратчайшего покрытия матрицы $\boldsymbol{C}$, образует искомую кратчайшую имплицирующую форму.

В некоторых ситуациях может представить интерес и следующая задача: для заданной матрицы разбиений $\boldsymbol{U}$ требуется найти кратчайшую булеву матрицу $\boldsymbol{W}$, имплицирующую матрицу $\boldsymbol{U}$. Иначе говоря, нужно найти матрицу полных двублочных разбиений, удовлетворяющую всем условиям, представленным в троичной матрице $\boldsymbol{U}$, и содержащую при этом минимально возможное число строк.

Однако эта задача легко сводится к предыдущей, поскольку отношение импликации между матрицами разбиений является транзитивным. Действительно, матрица $\boldsymbol{W}$ может быть получена из кратчайшей (троичной) имплицирующей формы $\boldsymbol{V}$ матрицы разбиений путем произвольной подстановки символов 1 и 0 на места символов «—» в матрице $\boldsymbol{V}$.

Пример. Пусть матрица U обладает следующим значением:

$$\left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & - & a \\ 1 & 0 & - & 0 & - & b \\ - & 1 & - & 1 & 0 & c \\ 1 & 0 & - & - & 1 & d \\ - & 1 & - & 0 & 0 & e \\ 1 & 0 & 0 & - & - & f \\ - & 1 & 1 & - & 0 & g \\ 1 & - & - & 0 & 0 & h \\ - & 1 & 0 & - & 0 & i \\ - & - & 1 & 0 & - & j \end{array}\right]$$

Прежде всего нужно найти множество D всех главных импликант. Сделать это можно по-разному. Например, если в матрице U много столбцов, но мало строк, то целесообразно организовать некоторый сокращенный перебор совокупностей строк (для чего можно использовать программу перебор), выделить из них максимальные совместимые совокупности и получить соответствующие им главные импликанты. Если же число столбцов в матрице U невелико, как, например, в рассматриваемой сейчас матрице, то более удобным может оказаться другой путь: можно реализовать перебор различных булевых векторов соответствующей размерности (в рассматриваемом примере — пятикомпонентных), выделение из них булевых импликант заданной матрицы и последующее нахождение искомых главных импликант.

Главные импликанты получаются из булевых путем их преобразования следующим образом. Если в некоторой импликанте v существует такая компонента со значением 0 или 1, присвоение которой значения «—» не приводит к сокращению множества имплицируемых строк матрицы U, то такое присвоение производится и полученный вектор вновь подвергается анализу. Если же компоненты с указанным свойством не существует и если в своем первоначальном виде рассматриваемый вектор был булевым, то он является главной импликантой.

Заметим, что главные импликанты получаются с точностью до инверсии, поскольку взаимно инверсные векторы в данном смысле эквивалентны.

Для рассматриваемого примера мы получаем следующую матрицу D главных импликант:

11000	1
10100	2
10001	3
10011	4
10101	5
11100	6
10000	7

все строки которой в данном случае оказываются булевыми, и следующую матрицу C отношения импликации:

a b c d e f g h i j	
1000100110	1
0100000101	2
0111011000	3
0001111001	4
0111000011	5
0000101101	6
0100010100	7

После этого осталось найти кратчайшее покрытие матрицы C (для этого можно обратиться к программам типа окрапок). В данном случае оно находится без труда:

a b c d e f g h i j	
1000100110	1
0111011000	3
0001111001	4

Образующим найденное покрытие строкам соответствуют главные импликанты с номерами 1, 3 и 4, выписывая которые мы получаем окончательный результат — кратчайшую имплицирующую форму матрицы U:

11000
10001
10011

Поиск приближения к кратчайшей имплицирующей форме. При большом числе строк в матрице U (порядка сотни и больше) получение точного решения поставленной задачи путем нахождения

кратчайшего покрытия матрицы C становится затруднительным. Предложим в связи с этим алгоритм нахождения приближенного решения, осуществляющий последовательное построение искомой имплицирующей формы V, причем таким образом, чтобы минимальные приращения этой матрицы «съедали» как можно бо́льшие участки матрицы U, позволяя быстро сокращать рассматриваемый далее остаток U^- этой матрицы, образуемый теми ее строками, которые не имплицируются уже построенной частью V^+ матрицы V.

Построение матрицы V начинается с нуля, а ее приращения могут быть двух типов: добавление новой строки или доопределение одной из уже имеющихся строк, выражающееся в замене некоторых символов «—» символами 0 или 1. Новая строка добавляется лишь в случае очевидной необходимости — если в матрице U^- находится строка, несовместимая ни с одной из строк матрицы V^+. Если же каждая строка матрицы U^- совместима по крайней мере с одной из строк матрицы V^+, то матрица V^+ получает приращение второго типа, причем по возможности минимальное, то есть такое, при котором символами 0 или 1 заменяется минимальное число символов «—». Разумеется, при каждом из производимых приращений матрицы V^+ должна имплицироваться какая-то часть матрицы U^-, то есть эта матрица должна сокращаться. Полное решение задачи будет получено в момент исчезновения матрицы U^-.

Дополнительным критерием выбора очередного шага может служить степень симметричности строк матрицы V: считается, что они тем более симметричны, чем меньше разница в числе нулей и единиц в строке. Здесь мы ограничимся ссылкой на статистику решенных задач, показывающую, что, как правило, выбор более симметричных строк (при прочих равных условиях) приводит к лучшим результатам.

Если же на каком-то шаге алгоритма встречается несколько равноценных и «лучших» по данным критериям вариантов, то, как всегда в таких случаях, будем выбирать первый из них. Положим, что варианты перебираются в следующем порядке: просматриваются подряд все строки матрицы V^+ и для каждой из них осуществляется аналогичный перебор строк матрицы U^-.

Пример. Проиллюстрируем работу описанного алгоритма на примере уже знакомой нам матрицы

$$\boldsymbol{U}=\begin{array}{|cccccc|l|}\hline 1&1&0&0&-&-&a\\ 1&0&-&0&-&&b\\ -&1&-&1&0&&c\\ 1&0&-&-&1&&d\\ -&1&-&0&0&&e\\ 1&0&0&-&-&&f\\ -&1&1&-&0&&g\\ 1&-&-&0&0&&h\\ -&1&0&-&0&&i\\ -&-&1&0&-&&j\\ \hline\end{array}$$

Построение матрицы $\boldsymbol{V}$ начинается с «передачи» ей первой строки a матрицы $\boldsymbol{U}$, затем к ней добавляется строка b, поскольку они взаимно несовместимы. Каждая из остальных строк матрицы $\boldsymbol{U}$ оказывается совместима по крайней мере с одной из двух строк (a или b) матрицы $\boldsymbol{V}^+$, но не имплицируется ни одной из них. Поэтому они образуют текущий остаток $\boldsymbol{U}^-$ исходной матрицы $\boldsymbol{U}$, не имплицируемый построенной частью $\boldsymbol{V}^+$ искомой матрицы $\boldsymbol{V}$. Этот промежуточный результат показан на следующей таблице, в которой наряду со строками матрицы $\boldsymbol{V}^+$ показаны символы имплицируемых ими строк матрицы $\boldsymbol{U}$:

$$\begin{array}{l}\boldsymbol{U}^+\\ \boldsymbol{U}^-\end{array}\left\{\begin{array}{|cccccc|l|}\hline 1&1&1&0&0&-&a\\ 2&1&0&-&0&-&b\\ \hline -&1&-&1&0&&c\\ 1&0&-&-&1&&d\\ -&1&-&0&0&&e\\ 1&0&0&-&-&&f\\ -&1&1&-&0&&g\\ 1&-&-&0&0&&h\\ -&1&0&-&0&&i\\ -&-&1&0&-&&j\\ \hline\end{array}\right.$$

В соответствии с используемыми в алгоритмах критериями производится «согласование» строки 2 матрицы $\boldsymbol{V}^+$ со строкой c матрицы $\boldsymbol{U}$: первая из них получает значение

$$1\,0-0\,1$$

и имплицирует вторую, которая поэтому выбрасывается из матрицы U^-. Вместе с ней удаляется и строка d, также импилицируемая теперь строкой 2 матрицы V^+. Затем строка 2 получает новое приращение, принимая значение

10001

и имплицируя строки f и g матрицы U^-, после чего возникает ситуация, изображенная на следующей таблице:

$$U^+\begin{cases} 1\ 1\ 1\ 0\ 0 - a \\ 2\ \underline{1\ 0\ 0\ 0\ 1}\ b,\ c,\ d,\ f,\ g \end{cases}$$

$$U^-\begin{cases} -\ 1 - 0\ 0\ e \\ 1 - -\ 0\ 0\ h \\ -\ 1\ 0 - 0\ i \\ - -\ 1\ 0 - j \end{cases}$$

Теперь строка j матрицы U^- становится несовместимой ни с одной из строк матрицы V^+, поэтому она вводится в матрицу V^+ в качестве ее новой строки. Наконец, получает приращение строка 1: имеющийся в ней единственный символ «—» заменяется символом 0, после чего она имплицирует все оставшиеся строки матрицы U^-. На этом работа алгоритма заканчивается, а полученный им результат представляется следующей имплицирующей матричной формой, которая оказывается для рассмотренного примера кратчайшей:

$$V=\begin{bmatrix} 1\ 1\ 0\ 0\ 0 \\ 1\ 0\ 0\ 0\ 1 \\ -\ -\ 1\ 0 - \end{bmatrix}$$

Л-программа.

Минимизация матрицы разбиений — матрамин

1. Для заданной матрицы U частичных двублочных разбиений требуется найти имплицирующую ее троичную матрицу V, по возможности кратчайшую.

2. Внешние операнды:

$\alpha\beta_к^-$::матрица U в коде (10, 01, 00),

$\alpha\beta_к^+$::матрица V в коде (10, 01, 00).

Задаются: a_α, a_β, b_α, b_β.

Внутренние операнды: g, j, —.

4.

* **001 137** ($\alpha\, a\, b\, c\, d\, e\, f\, g$)

§ 0 $\circ f$

§ 1 $f \Rightarrow b \,\overline{\circ}\, e$

§ 2 $\alpha_b \Rightarrow c\, \beta_b \Rightarrow d \,\overline{\circ}\, a \circ g$

§ 3 $\triangle\, a \oplus f \circ\!\!\rightarrow 12\, \alpha_a \Rightarrow a\, \beta_a \Rightarrow b \circ h$

§ 4 $a \wedge d \mapsto 6\, b \wedge c \mapsto 6$

$a \vee c \bigtriangledown \Rightarrow j\, b \vee d \bigtriangledown \Rightarrow i - j \mapsto 5\, j - i$

§ 5 $\Rightarrow i \,\overline{\circ}\, g a \urcorner \wedge c \Rightarrow g\, b \urcorner \wedge d \vee g \bigtriangledown \circ\!\!\rightarrow 11 + i - e \mapsto 6 + e \Rightarrow e\, a \Rightarrow d\, b \Rightarrow c\, c \Rightarrow e\, d \Rightarrow f$

§ 6 $h \mapsto 3 \,\overline{\circ}\, h\, c \Leftrightarrow d \rightarrow 4$

§ 7 $\triangle\, b - b_a \circ\!\!\rightarrow 2\, e \vee \alpha_d \Rightarrow \alpha_d f \vee \beta_d \Rightarrow \beta_d c \Rightarrow h$

§ 10 $\overline{\triangle}\, b_a \Rightarrow a \oplus f \circ\!\!\rightarrow 13\, \alpha_a \Rightarrow \alpha_h \beta_a \Rightarrow \beta_h$

$a - b \circ\!\!\rightarrow 1 \rightarrow 7$

§ 11 $b \Rightarrow h \rightarrow 10$

§ 12 $g \mapsto 7\, \alpha_f \Leftrightarrow \alpha_b \beta_f \Leftrightarrow \beta_b \triangle f \oplus b_a \mapsto 1$

§ 13 $b_a \Rightarrow b_\beta$.

Оценка эффективности программы. С помощью программы матрамин была минимизирована серия матриц разбиений, число столбцов в которых равнялось 32, а число строк a варьировалось в пределах от 25 до 800. Изменялась также относительная доля μ элементов матрицы со значениями, отличными от «—». Остальные характеристики рассматриваемых матриц разбиений выбирались каждый раз случайным образом.

Таблица 2.9.1

Число строк в имплицирующей матрице

a \ μ	1/16	1/8	2/8	3/8	4/8	5/8	6/8
25	3	4	8	13	21	25	25
50	3	5	16	25	39	48	50
100	4	10	24	46	72	93	99
200	7	14	35	71	122	174	194
400	9	23	65	136	226	332	398
800	13	31	—	—	—	—	779

Результатами выполненного эксперимента послужили число строк в полученных имплицирующих матрицах, а также время решения каждой задачи. Эти данные сведены в таблицы 2.9.1 и 2.9.2.

Таблица 2.9.2

Время решения (на машине М-20)

a \ μ	1/16	1/8	2/8	3/8	4/8	5/8	6/8
25	3″	6″	6″	6″	4″	1″	1″
50	12″	17″	30″	20″	25″	7″	3″
100	40″	1′45″	2′45″	2′10″	2′20″	16″	7″
200	1′30″	6′32″	10′	13′	11′	4′15″	40″
400	5′20″	19′	57′	85′	87′	13′	1′50″
800	21′	61′	—	—	—	—	54′

Как видно, быстродействие программы достаточно удовлетворительное при a не более 100. С дальнейшим увеличением числа строк обрабатываемой матрицы время решения резко возрастает.

ГЛАВА 3

БУЛЕВЫ ФУНКЦИИ, ЛОГИЧЕСКИЕ УРАВНЕНИЯ, ГРАФЫ

§ 1. Булевы функции

Другая интерпретация булева пространства. В предыдущей главе булевым пространством было названо множество всех булевых векторов заданной размерности n. Нетрудно видеть, что это множество может рассматриваться как множество всех подмножеств некоторого множества $X \equiv \{x_1, x_2, \ldots, x_n\}$ мощности n. Такая возможность обеспечивается следующей знакомой нам формулой, устанавливающей взаимно однозначное соответствие между булевыми векторами и подмножествами из X:

$$\boldsymbol{x} = \| \{X_i\} \ni X \|$$

— в данном случае булев вектор $\boldsymbol{x}$ *представляет* подмножество X_i множества X.

Итак, пусть X — некоторое конечное множество, а M — множество всех его подмножеств, включая и пустое подмножество $\varnothing$. В соответствии с принятой в теории множеств символикой множество M может быть определено выражением $M \equiv \tilde{X}$, где верхний знак «тильда» служит оператором взятия всех подмножеств некоторого множества, в данном случае — множества X. Назовем множество M *булевым пространством над* X. Произвольный элемент α множества M может быть представлен булевым вектором $\|\{\alpha\} \ni X\|$, а подмножество M_i множества M — булевой матрицей $\|M_i \ni X\|$.

Например, если $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$, $\alpha = \{x_2, x_3, x_6, x_8\}$ и $M_i = \{\{x_3, x_6, x_7\}, \{x_1, x_4, x_7, x_8\}, \{x_2, x_3, x_6\}, \{x_1, x_4, x_5, x_6, x_7\}\}$, то

$$\| \{\alpha\} \ni X \| = 0\,1\,1\,0\,0\,1\,0\,1,$$

$$\| M_i \ni X \| = \begin{bmatrix} 0\,0\,1\,0\,0\,1\,1\,0 \\ 1\,0\,0\,1\,0\,0\,1\,1 \\ 0\,1\,1\,0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,1\,1\,1\,0 \end{bmatrix}.$$

Булевы функции. Во взаимно однозначное соответствие элементам $x_1, x_2, \ldots, x_n$ множества X поставим двоичные переменные, обозначаемые теми же буквами $x_1, x_2, \ldots, x_n$ и принимающие значения из множества $\{0, 1\}$, а в соответствие элементам булева пространства $M \equiv \bar{X}$ поставим наборы значений этих переменных, считая, что переменная x_i $(i \in \{1, 2, \ldots, n\})$ принимает значение 1 в некотором наборе, если элемент x_i множества X принадлежит соответствующему этому набору элементу пространства M, и принимает значение 0 в противном случае. Очевидно, что при таком определении соответствующий элементу α пространства M набор значений переменных $x_1, x_2, \ldots, x_n$ задается булевым вектором $\|\{\alpha\} \ni X\|$.

Таким образом, упорядоченную совокупность двоичных переменных $x_1, x_2, \ldots, x_n$ можно рассматривать как некоторый переменный вектор $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, принимающий значения из множества M всех постоянных n-компонентных булевых векторов.

Рассмотрим некоторую функцию

$$y = \varphi(\boldsymbol{x}) \equiv \varphi(x_1, x_2, \ldots, x_n),$$

где y — также двоичная переменная, принимающая значения из $\{0, 1\}$, а φ — символ функциональной зависимости между булевым вектором $\boldsymbol{x}$ и булевым скаляром y (или, что то же самое, между совокупностью переменных $x_1, x_2, \ldots, x_n$, с одной стороны, и переменной y, с другой стороны). Такая функция называется *булевой*. Если значения этой функции определены для всех 2^n значений вектора $\boldsymbol{x}$, она называется *полностью определенной*, в противном случае — *не полностью определенной*. Пока что мы ограничимся рассмотрением полностью определенных булевых функций.

Совокупность значений вектора $\boldsymbol{x}$, на которых y принимает значение 1, обозначим через M^1, а совокупность значений, на которых функция обращается в 0, — через M^0. Очевидно, что $M^1 \cup M^0 = M$ (для полностью определенной булевой функции), поэтому можно задать булеву функцию, перечислив лишь элементы множества M^1.

Непосредственное перечисление этих элементов можно осуществить с помощью булевой матрицы $\|M^1 \ni X\|$, каждая строка которой задает один из элементов мно-

жества M^1. Назовем такую форму задания булевой функции *элементарной.*

Возможна и другая форма задания булевой функции, посредством 2^n-компонентного булева вектора $\|\{M^1\}\ni M\|$. При этом мы считаем, что упорядочение элементов множества M задается позиционным двоичным кодом, то есть, например, при $n=3$ $M=\{000, 001, 010, 011, 100, 101, 110, 111\}$. Такая форма, называемая *векторной,* представляет известные удобства, когда n невелико.

Например, булева функция $\varphi\{x_1, x_2, x_3, x_4\}$, принимающая значение 1 на наборе 0011 (то есть когда $x_1=0$, $x_2=0$, $x_3=1$ и $x_4=1$), а также на наборах 1010 и 0110 и принимающая значение 0 на всех других наборах, в элементарной форме представляется матрицей

$$\begin{bmatrix} 0011 \\ 1010 \\ 0110 \end{bmatrix}$$

а в векторной форме — вектором

$$0001001000100000.$$

Две булевы функции будем считать равными в том и только в том случае, когда они принимают одинаковые значения на любом произвольном элементе пространства M.

Нетрудно подсчитать, что число различных булевых функций n переменных равно 2^{2^n}.

Алгебраические формы булевых функций. Среди различных простейших (от небольшого числа переменных) булевых функций особый интерес заслуживает одноместная функция *инверсия,* обозначаемая через $\bar{x}$, а также двуместные функции *дизъюнкция* $(x_1 \vee x_2)$, *конъюнкция* $(x_1 \wedge x_2$, или $x_1x_2)$ и *дизъюнкция с исключением,* или *сумма по модулю два* $(x_1 \oplus x_2)$. Все эти функции оказываются уже хорошо знакомыми нам, поскольку они были использованы при формулировке языка ЛЯПАС, и могут быть представлены таблицей 3.1.1.

Известно, что эта система функций является *функционально полной,* то есть любую булеву функцию можно представить в форме суперпозиции данных функций.

Таблица 3.1.1

x	$\bar{x}$
0	1
1	0

x_1	x_2	$x_1 \vee x_2$	$x_1 x_2$	$x_1 \oplus x_2$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Эту суперпозицию мы будем называть *алгебраической формой* булевой функции.

Обратим внимание на то, что $\bar{x} = x \oplus 1$ и что операции дизъюнкции и конъюнкции могут быть интерпретированы следующим образом:

$$x_1 \vee x_2 = \max\{x_1, x_2\}, \quad x_1 x_2 = \min\{x_1, x_2\}.$$

Наличие у рассматриваемых операций свойства ассоциативности:

$$(x_1 \vee x_2) \vee x_3 = x_1 \vee (x_2 \vee x_3),$$
$$(x_1 \wedge x_2) \wedge x_3 = x_1 \wedge (x_2 \wedge x_3),$$
$$(x_1 \oplus x_2) \oplus x_3 = x_1 \oplus (x_2 \oplus x_3)$$

облегчает введение соответствующих многоместных операций, которые можно представить как операции над вектором $\boldsymbol{x}$:

$$\vee \boldsymbol{x} \equiv x_1 \vee x_2 \vee \ldots \vee x_n = \max\{x_1, x_2, \ldots, x_n\},$$
$$\wedge \boldsymbol{x} \equiv x_1 \wedge x_2 \wedge \ldots \wedge x_n = \min\{x_1, x_2, \ldots, x_n\},$$
$$\oplus \boldsymbol{x} \equiv x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

В дальнейшем нами будут широко использоваться *элементарные конъюнкции*, как называются многоместные конъюнкции переменных, некоторые из которых могут быть заданы в инверсной форме ($x_1 x_2 x_5$, $x_2 \bar{x}_4 \bar{x}_5 \bar{x}_7$, x_1 и т. п.)

Элементарная конъюнкция имеет простую содержательную интерпретацию: она принимает значение 1 на тех и только тех наборах значений переменных $x_1, x_2, \ldots, x_n$, i-я компонента которых имеет значение 1, если в конъюнкцию входит x_i, и значение 0, если в конъюнк-

цию входит $\bar{x}_i$. Нетрудно видеть, что совокупность этих наборов является интервалом булева пространства M.

Будем говорить, что элементарная конъюнкция и определенный таким образом интервал *соответствуют* друг другу.

Если в выражение элементарной конъюнкции входят все переменные, безразлично, в прямой или инверсной форме, то она называется *полной*. Полная элементарная конъюнкция обращается в единицу лишь на одном элементе булева пространства M (пусть этим элементом служит некоторый n-компонентный булев вектор — константа $\boldsymbol{m}_j$) и может быть представлена векторным выражением

$$\boldsymbol{x} \oplus \boldsymbol{m}_j \oplus \mathbf{1} = ((x_1 \oplus m_j^1 \oplus 1), (x_2 \oplus m_j^2 \oplus 1), \ldots, (x_n \oplus m_j^n \oplus 1)),$$

где через $\mathbf{1}$ обозначен вектор, все n компонент которого имеют значение 1.

Например, если $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$, то

$$x_1 \bar{x}_2 \bar{x}_3 x_4 x_5 \bar{x}_6 = \boldsymbol{x} \oplus 100110 \oplus \mathbf{1},$$
$$x_1 x_2 \bar{x}_3 x_4 \bar{x}_5 x_6 = \boldsymbol{x} \oplus 110101 \oplus \mathbf{1}$$

и т. п.

При этом подразумевается, как и раньше, что компоненты рассматриваемых векторов связаны оператором $\wedge$, то есть более точным было бы представление конъюнкции в виде скаляра $\vee(\boldsymbol{x} \oplus \boldsymbol{m}_j \oplus \mathbf{1})$, задающего ее *значение*. Например,

$$x_1 \bar{x}_2 \bar{x}_3 x_4 x_5 \bar{x}_6 = \wedge(\boldsymbol{x} \oplus 100110 \oplus \mathbf{1}),$$
$$x_1 x_2 \bar{x}_3 x_4 \bar{x}_5 x_6 = \wedge(\boldsymbol{x} \oplus 110101 \oplus \mathbf{1})$$

и т. п.

По аналогии с элементарными конъюнкциями определяются элементарные дизъюнкции, примерами которых могут служить выражения x_2, $x_1 \vee \bar{x}_2$, $x_1 \vee \bar{x}_3 \vee \bar{x}_5 \vee x_6$ и т. п.

Многоместная дизъюнкция элементарных конъюнкций называется *дизъюнктивной нормальной формой*, или д н ф. Многоместная конъюнкция элементарных дизъюнкций называется *конъюнктивной нормальной формой*, или к н ф.

Нас будут интересовать в первую очередь д н ф.

Дизъюнктивные нормальные формы. Существует определенное соответствие между булевыми

функциями *дизъюнкция*, *конъюнкция* и *инверсия*, с одной стороны, и теоретико-множественными операциями *объединение*, *пересечение* и *дополнение*, с другой стороны. На это соответствие мы уже обращали внимание в начале книги, здесь же дадим его более общую формулировку.

Пусть $\varphi_1(x)$ и $\varphi_2(x)$ — произвольные булевы функции переменных $x_1, x_2, \ldots, x_n$, а M_1^1 и M_2^1 — представляющие эти функции подмножества из M. Тогда дизъюнкция этих функций $\varphi_1(x) \vee \varphi_2(x)$ образует функцию, представляемую множеством $M_1^1 \cup M_2^1$, их конъюнкция $\varphi_1(x) \wedge \varphi_2(x)$ образует функцию, представляемую множеством $M_1^1 \cap M_2^1$, а инверсия любой из этих функций, то есть функция $\bar{\varphi}_1(x)$ (или $\bar{\varphi}_2(x)$) представляется множеством $M \setminus M_1^1$ (или $M \setminus M_2^1$).

Опираясь на это соответствие, нетрудно показать, что любая булева функция может быть представлена в дизъюнктивной нормальной форме. Для этого достаточно, отправившись от булевой матрицы $\boldsymbol{A} \equiv \|M^1 \ni X\|$ (а в такой форме можно представить любую булеву функцию), поставить в соответствие каждой строке $\boldsymbol{a}_i$ матрицы $\mathbf{A}$ элементарную конъюнкцию $\boldsymbol{k}_i = \boldsymbol{x} \oplus \boldsymbol{a}_i \oplus \mathbf{1}$, а затем взять их дизъюнкцию. В данном случае конъюнкция $\boldsymbol{k}_i$ рассматривается как вектор, значение которого вычисляется покомпонентно с помощью оператора $\oplus$: j-я компонента принимает значение x_j, если $a_i^j = 1$, и принимает значение $\bar{x}_j$, если $a_i^j = 0$. Возможность такого представления свидетельствует, в частности, о функциональной полноте системы из трех рассматриваемых функций (дизъюнкция, конъюнкция и инверсия).

В полученной таким образом д н ф содержатся только полные элементарные конъюнкции. Такая д н ф называется *совершенной* и соответствует заданию функции в элементарной форме $\|M^1 \ni X\|$.

Например, если

$$\|M^1 \ni X\| = \begin{bmatrix} 011001 \\ 110011 \\ 010100 \end{bmatrix},$$

то представляемая множеством M^1 булева функция может быть задана в следующей совершенной дизъюнктив-

ной нормальной форме:

$$\bar{x}_1x_2x_3\bar{x}_4\bar{x}_5x_6 \vee x_1x_2\bar{x}_3\bar{x}_4x_5x_6 \vee \bar{x}_1x_2\bar{x}_3x_4\bar{x}_5\bar{x}_6.$$

В общем случае элементарная конъюнкция может быть представлена троичным вектором со следующей интерпретацией: j-я компонента вектора принимает значение 1, если в конъюнкцию входит x_j, принимает значение 0, если в конъюнкцию входит $\bar{x}_j$, и принимает значение «—», если в конъюнкцию не входит ни x_j, ни $\bar{x}_j$. Иначе говоря, элементарная конъюнкция представляется тем же троичным вектором, что и соответствующий ей интервал булева пространства.

Отсюда следует, что произвольная д н ф может быть задана троичной матрицей, каждая строка которой представляет один из членов д н ф, то есть одну из элементарных конъюнкций, входящих в рассматриваемую д н ф.

Например, дизъюнктивная нормальная форма

$$\bar{x}_1x_4 \vee x_2\bar{x}_4\bar{x}_5x_6 \vee x_3x_4x_6$$

может быть представлена следующей троичной матрицей:

$$\begin{bmatrix} 0 & - & - & 1 & - & - \\ - & 1 & - & 0 & 0 & 1 \\ - & - & 1 & 1 & - & 1 \end{bmatrix}$$

Установленное таким образом взаимно однозначное соответствие между троичными матрицами и дизъюнктивными нормальными формами булевых функций наполняет новым содержанием полученные в главе 2 результаты.

Допустим, что булева функция $\varphi(\boldsymbol{x})$ задается в некоторой дизъюнктивной нормальной форме, которая представляется троичной матрицей $\boldsymbol{U}$. Равносильные преобразования этой матрицы не изменяют множество M^1, откуда следует и неизменность представляемой булевой функции $\varphi(\boldsymbol{x})$; в то же время соответствующая матрице $\boldsymbol{U}$ д н ф будет изменяться вместе с матрицей $\boldsymbol{U}$, причем число членов в д н ф будет всегда равно числу строк матрицы $\boldsymbol{U}$, а число букв, входящих в д н ф, будет равно числу элементов матрицы $\boldsymbol{U}$, имеющих значения, отличные от «—».

Это означает, что, сокращая с помощью равносильных преобразований матрицу U, мы тем самым упрощаем соответствующую этой матрице д н ф. В частности, минимизируя число строк в матрице, мы получаем д н ф с минимальным числом членов, называемую *кратчайшей*. Минимизация же числа таких элементов матрицы U, значения которых отличны от «—», приводит нас к д н ф, называемой *минимальной*. Если матрица U задана в безызбыточной форме, то ей соответствует *безызбыточная* д н ф, то есть такая д н ф, в которой нельзя выбросить ни одного члена и нельзя удалить ни одной буквы в каком-либо члене без того, чтобы не получить д н ф, представляющую уже другую булеву функцию, не равную исходной. Дизъюнктивную нормальную форму, соответствующую расширению некоторой троичной матрицы (то есть матрице, содержащей все максимальные строки), принято называть *сокращенной* (хотя и не совсем понятно, почему).

Таким образом, рассмотренные в главе 2 методы сжатия троичных матриц могут по существу трактоваться как методы минимизации дизъюнктивных нормальных форм булевых функций.

Известен ряд методов минимизации д н ф непосредственно в алгебраической форме, однако мы не будем на них останавливаться, поскольку все они могут быть интерпретированы как некоторые равносильные преобразования над троичными матрицами.

О п е р а ц и и и м п л и к а ц и и и р а в е н с т в а. Существенную роль в теории булевых функций играют двуместные операции *импликация* ($x_1 \to x_2$) и *равенство* ($x_1 = x_2$), определяемые следующей таблицей:

Таблица 3.1.2

x_1	x_2	$x_1 \to x_2$	$x_1 = x_2$
0	0	1	1
0	1	1	0
1	0	0	0
1	1	1	1

Эти операции легко выражаются через другие (например, операция импликации может быть записана как $\bar{x}_1 \vee x_2$, а операция равенства — как $\bar{x}_1 \oplus x_2$), однако в некоторых случаях предпочтительно пользоваться именно ими. Особые удобства они представляют при формулировке и рассмотрении булевых уравнений, связывающих некоторые группы переменных.

Собственно говоря, поскольку символ равенства $=$ входит в число булевых операторов, уравнение по своей записи уже ничем не отличается от некоторой функции. Различие в интерпретации: если некоторое выражение рассматривается просто как *булева функция*, то считается, что все фигурирующие в нем переменные являются свободными, то есть могут независимо друг от друга принимать любые значения, а сама функция в зависимости от этого будет принимать значение 1 или 0. Если же выражение рассматривается как *булево уравнение*, то считается, что указанные переменные не являются уже совершенно свободными, а так должны принимать свои значения, чтобы представленная выражением функция всегда имела значение 1.

Другими словами, говоря о функции $\varphi(x)$, мы подразумеваем некоторую переменную y, которая принимает значение 1, если вектор x принимает значение из M^1, и принимает значение 0 в противном случае. Говоря об уравнении $\varphi(x)$, мы ограничиваемся рассмотрением (или, может быть, поиском) тех значений вектора x, которые принадлежат множеству M^1. Выражение типа $y = \varphi(x)$ обычно интерпретируется как уравнение, связывающее функцию $\varphi(x)$ с переменной y. Однако иногда его можно трактовать как функцию, принимающую значение 1 при совпадении значений $\varphi(x)$ и y и принимающую значение 0 в противном случае. В первом случае знак $=$ выступает как символ *бинарного отношения*, во втором — как *бинарный оператор*. Аналогичную двойную интерпретацию допускают знаки $\to$ и $\neq$, последний из которых может рассматриваться и как символ отношения неравенства и как оператор, эквивалентный оператору $\oplus$.

Импликанты булевых функций. Пусть выражение $\varphi(x) \to \psi(x)$ представляет уравнение, то есть функция $\psi(x)$ принимает значение 1 на всех тех (но, может быть, не только тех) значениях вектора x, которые

обращают в 1 функцию $\varphi(x)$. В этом случае функция $\varphi(x)$ называется *импликантой булевой функции* $\psi(x)$.

Каждая из функций $\varphi(x)$ и $\psi(x)$ может рассматриваться как некоторое условие, которое выполняется в том и только в том случае, когда функция принимает значение 1. При этом функция $\varphi(x)$ играет роль *достаточного условия* обращения в единицу функции $\psi(x)$ (то есть выполнения представленного этой функцией условия), а функция $\psi(x)$ играет роль *необходимого условия* обращения в 1 функции $\varphi(x)$ (выполнения условия, представленного этой функцией).

Отношение импликации *транзитивно*, то есть если $\varphi(x) \to \psi(x)$ и $\psi(x) \to \xi(x)$, то и $\varphi(x) \to \xi(x)$. Если при этом $\varphi(x) \neq \psi(x)$, то будем говорить, что функция $\psi(x)$ *ближе* к функции $\xi(x)$, чем функция $\varphi(x)$ (можно говорить, что функция $\psi(x)$ представляет более необходимое условие обращения в единицу функции $\xi(x)$, чем функция $\varphi(x)$; можно рассмотреть цепочку отношений и с другой стороны, утверждая, что функция $\psi(x)$ является более достаточным условием обращения в единицу функции $\varphi(x)$, чем функция $\xi(x)$). Очевиден предельный случай: если $\varphi(x) = \psi(x)$, то как $\varphi(x) \to \psi(x)$, так и $\psi(x) \to \varphi(x)$, то есть каждая из этих двух функций представляет и необходимое и достаточное условие обращения в 1 другой функции.

Учитывая эквивалентность отношений $\varphi(x) \to \psi(x)$ и $M^1(\varphi) \subseteq M^1(\psi)$, легко подсчитать число различных импликант заданной функции $\psi(x)$: оно оказывается равным $2^{\sigma(M^1(\psi))}$. Одной из этих импликант оказывается сама функция $\psi(x)$.

Если $\varphi(x) \to \xi(x)$ и $\psi(x) \to \xi(x)$, то $\varphi(x) \vee \psi(x) \to \to \xi(x)$, то есть дизъюнкция импликант некоторой функции также является импликантой этой функции (дизъюнкция достаточных условий представляет собой достаточное условие). С другой стороны, если $\varphi(x) \to \psi(x)$ и $\varphi(x) \to \xi(x)$, то и $\varphi(x) \to \psi(x)\xi(x)$ (конъюнкция необходимых условий представляет собой необходимое условие).

Говоря об импликантах булевых функций, обычно накладывают ограничение на тип импликанты, считая, что ею может служить только элементарная конъюнк-

ция. Присоединившись к этому соглашению, обратим внимание на ближние к рассматриваемой функции импликанты, то есть на такие элементарные конъюнкции, которые перестают быть импликантами при удалении из них любой буквы. Такие импликанты принято называть *простыми*. Нетрудно видеть, что они соответствуют максимальным строкам троичных матриц и что дизъюнкция всех простых импликант функции $\psi(x)$ образует сокращенную дизъюнктивную нормальную форму этой функции.

Здесь можно продолжить аналогию с взаимоотношениями достаточного и необходимого условий, заметив, что система всех простых импликант заданной функции $\psi(x)$ является полной в том смысле, что их дизъюнкция всегда равна функции $\psi(x)$ (то есть дизъюнкция всех достаточных условий, образующих полную систему, всегда является необходимым условием).

§ 2. Минимизация дизъюнктивных нормальных форм

Обсуждение постановки задачи. Выше было показано, что задача минимизации дизъюнктивной нормальной формы булевой функции $\varphi = \varphi\ (x_1, x_2, \ldots, x_n)$ может трактоваться как задача максимального сжатия троичной матрицы $\boldsymbol{U}$, представляющей исходную днф. При этом множеству B членов совершенной днф рассматриваемой функции будет соответствовать множество $M^1 = M(\boldsymbol{U})$ строк булевой матрицы, эквивалентной троичной матрице $\boldsymbol{U}$, а множеству A всех простых импликант функции φ ставится в соответствие множество R строк расширения $\boldsymbol{R}$ матрицы $\boldsymbol{U}$. Задача нахождения кратчайшей днф сводится таким образом к получению кратчайшего покрытия булевой матрицы $\|R \operatorname{abs} M^1\|$. Поскольку отношение поглощения между троичными векторами, представляющими некоторые элементарные конъюнкции, является обратным отношению импликации между представляемыми конъюнкциями, матрица $\|R \operatorname{abs} M^1\|$ оказывается тождественной булевой матрице $\|A \leftarrow B\|$, задающей бинарное отношение импликации между элементами совершенной днф и простыми импликантами рассматриваемой булевой функции φ.

Задача решается достаточно просто, когда число элементов в множествах A и B невелико, например порядка двух-трех десятков. Однако при решении многих практических задач приходится рассматривать булевы функции, число переменных у которых исчисляется десятками, а число элементов в множестве B — тысячами. Построение матрицы $\|A \leftarrow B\|$, а тем более нахождение ее кратчайшего покрытия в этом случае становится нереальным.

К счастью, эти функции оказываются, как правило, простыми в другом отношении: они задаются в виде некоторой дизъюнктивной нормальной формы, содержащей относительно небольшое число членов, порядка нескольких десятков или сотен. На учете простоты такого рода и построен алгоритм минимизации булевых функций, описываемый в настоящем параграфе и избегающий непосредственного построения множества B и, при определенных условиях, множества A. В основу алгоритма положены некоторые соображения, изложенные в предыдущей главе в терминологии троичных матриц. Здесь они перелагаются на алгебраический язык и уточняются.

Исходная форма. Будем считать, что исходная днф рассматриваемой булевой функции является произвольной, то есть не гарантируется, например, ее безызбыточность. В качестве иллюстрации предложим следующую днф, выбранную (с помощью вычислительной машины) так, чтобы она оказалась достаточно удобной для демонстрации всех этапов излагаемого алгоритма минимизации и в то же время не была излишне громоздкой:

$$\varphi(a, b, c, d, e, f, g) = a\bar{b}c\bar{d}\bar{g} \vee abd\bar{f}g \vee \bar{a}\bar{b}cd\bar{g} \vee$$

$$\vee \bar{a}c\bar{d}\bar{e}\bar{f}\bar{g} \vee ac\bar{e}\bar{f} \vee a\bar{b}\bar{c}d\bar{e}\bar{f} \vee b\bar{c}\bar{e}\bar{f}\bar{g} \vee \bar{c}\bar{d}e\bar{f}g \vee$$

$$\vee abc\bar{e}\bar{f} \vee \bar{a}\bar{b}d\bar{f}g \vee acd \vee acd\bar{g} \vee \bar{a}b\bar{c}\bar{f}g \vee \bar{a}b\bar{d}e\bar{g} \vee$$

$$\vee \bar{a}bcd\bar{e}fg \vee \bar{a}b\bar{c}d\bar{e}\bar{g} \vee \bar{a}c\bar{d}fg \vee \bar{a}\bar{b}cd\bar{e}\bar{f} \vee ab\bar{c}d\bar{f} \vee$$

$$\vee ab\bar{e}\bar{g} \vee a\bar{b}\bar{d}\bar{g} \vee b\bar{c}\bar{d}e\bar{f}g \vee ab\bar{c}d\bar{f}g \vee a\bar{b}cd\bar{e}\bar{f}g \vee$$

$$\vee \bar{b}cf \vee \bar{a}cef\bar{g} \vee \bar{b}\bar{c}\bar{d}f\bar{g} \vee \bar{a}\bar{b}de\bar{f}\bar{g} \vee \bar{a}\bar{b}cefg \vee \bar{b}\bar{c}\bar{e}\bar{f}\bar{g} \vee$$

$$\vee a\bar{b}\bar{c}\bar{d}efg \vee a\bar{b}e\bar{f}g.$$

Перейдем к матричному представлению этой формы:

$$U = \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left\| \begin{array}{ccccccc} 1 & 0 & 1 & 0 & - & - & 0 \\ 1 & 1 & - & 1 & - & 1 & 1 \\ 0 & 0 & 1 & 1 & - & - & 0 \\ 0 & - & 1 & 0 & 0 & 0 & 0 \\ 1 & - & 1 & - & 0 & 1 & - \\ 1 & 0 & 0 & 1 & 0 & 0 & - \\ - & 1 & 0 & - & 0 & 1 & 0 \\ - & - & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & - & 0 & 1 & - \\ 0 & 0 & - & 1 & - & 1 & 1 \\ 1 & - & 1 & 1 & - & - & - \\ 1 & - & 1 & 1 & - & - & 0 \\ 0 & 1 & 0 & - & - & 0 & 1 \\ 0 & 1 & - & 0 & 1 & - & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & - & 0 \\ 0 & - & 1 & 0 & - & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & - \\ 1 & 1 & 0 & 0 & - & 1 & - \\ 1 & 1 & - & - & 0 & - & 0 \\ 1 & 0 & - & 0 & - & - & 0 \\ - & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & - & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ - & 0 & 1 & - & - & 1 & - \\ 0 & - & 1 & - & 1 & 1 & 0 \\ - & 0 & 0 & 0 & - & 1 & 0 \\ 0 & 0 & - & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & - & 1 & 1 & 1 \\ - & 0 & 0 & - & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & - & - & 1 & 0 & 1 \end{array} \right\| \end{array}$$

Получение безызбыточной днф. Первым естественным шагом на пути минимизации булевой функции является устранение избыточности в исходной форме. Эта задача решается описываемой ниже программой, реализующей модификацию метода, предложенного Квайном [21] и заключающегося в последовательном удалении (если это окажется возможным) отдельных членов

в исходной д н ф или отдельных букв в них. При этом из д н ф можно выбросить некоторый член, если он имплицирует дизъюнкцию остальных членов, и можно выбросить в рассматриваемом члене некоторую букву, если д н ф имплицируется элементарной конъюнкцией, получаемой из рассматриваемого члена путем изменения «фазы отрицания» переменной, представляемой данной буквой (например, из члена $ab\bar{c}$ можно выбросить букву $\bar{c}$, если исходная д н ф имплицируется элементарной конъюнкцией abc, и можно выбросить букву b, если д н ф имплицируется конъюнкцией $a\bar{b}\bar{c}$).

Как видно, основная тяжесть вычислений при решении задачи устранения избыточности в исходной д н ф приходится на операции проверки отношения импликации между некоторой элементарной конъюнкцией k, с одной стороны, и некоторой дизъюнктивной нормальной формой $D = k_1 \vee k_2 \vee \ldots \vee k_m$, с другой стороны.

Отношение $k \to D$ эквивалентно условию: D обращается в 1 на всех тех наборах значений переменных, которые обращают в 1 элементарную конъюнкцию k. Множество всех таких наборов образует интервал, на котором переменные, входящие в k под знаком инверсии, принимают значение 0, переменные, входящие в k без знака инверсии, принимают значение 1, а значения остальных переменных остаются произвольными. Соответствующая фиксация значений указанных переменных в д н ф D трансформирует последнюю в

$$\bigvee_{k_i \in D^*} (k_i : k),$$

где D^* — множество членов из D, неортогональных конъюнкции k, а $k_i : k$ — элементарная конъюнкция, получаемая из k_i путем удаления всех букв, входящих в конъюнкцию k (заметим, что если из какой-либо конъюнкции удаляются все буквы, то в ней остается 1). Отсюда следует, что необходимым и достаточным условием выполнения отношения $k \to D$ является тождество

$$\bigvee_{k_i \in D^*} (k_i : k) \equiv 1.$$

Пусть д н ф представляется троичной матрицей $\boldsymbol{U}$. Тогда проверка полученного тождества сводится к ана-

лизу на вырожденность минора матрицы U, образованного пересечением столбцов, в которых вектор, представляющий конъюнкцию k, принимает значение «—», и строк, неортогональных этому вектору.

Рассмотрим, например, третью строку матрицы U, приведенной на стр. 281, а именно строку

$$0\,0\,1\,1\,-\,-\,0.$$

Остальные из строк матрицы U, неортогональные данной строке, образуют строчный минор

$$\begin{bmatrix} -&0&1&-&-&1&- \\ 0&-&1&-&1&1&0 \end{bmatrix}$$

а пересечение его со столбцами, в которых рассматриваемая строка имеет значение «—», дает минор

$$\begin{bmatrix} -&1 \\ 1&1 \end{bmatrix}$$

оказывающийся невырожденным: существует вектор — 0, ортогональный этому минору. Следовательно, конъюнкцию $\bar{a}\bar{b}cd\bar{g}$, представляемую строкой 0 0 1 1 — — 0, нельзя удалить из д н ф D, представляемой матрицей U. Можно, однако, выбросить из этой конъюнкции букву d, поскольку д н ф D имплицируется элементарной конъюнкцией $\bar{a}\bar{b}c\bar{d}$. Действительно, конъюнкция $\bar{a}\bar{b}c\bar{d}\bar{g}$ представляется вектором

$$0\,0\,1\,0\,-\,-\,0,$$

совокупность неортогональных этому вектору строк матрицы U образует строчный минор

$$\begin{array}{c} a\;b\;c\;d\;e\;f\;g \\ \begin{bmatrix} 0&-&1&0&0&0&0 \\ 0&0&1&0&0&0&- \\ -&0&1&-&-&1&- \\ 0&-&1&-&1&1&0 \\ 0&0&-&0&1&0&0 \end{bmatrix} \end{array}$$

а соответствующее выделение столбцов приводит нас к минору

$$\begin{bmatrix} 0&0 \\ 0&0 \\ -&1 \\ 1&1 \\ 1&0 \end{bmatrix}$$

который оказывается вырожденным.

Описанный алгоритм устранения избыточности в д н ф, заданной троичной матрицей, реализуется следующей программой.

Приведение троичной матрицы к эквивалентной безызбыточной форме — б е з ы з б ы т о ч н а я

1. Для заданной произвольной троичной матрицы U находится эквивалентная ей безызбыточная форма $U_б$, то есть такая троичная матрица $U_б$, эквивалентная матрице U, которая перестает быть эквивалентной ей при удалении из $U_б$ любой строки или присвоении значения «—» любому из элементов матрицы $U_б$, имеющих значение 0 или 1.

2. Внешние операнды:

$\alpha\beta_к^- :: U$ в коде (10, 01, 00),

$\alpha\beta_к^+ :: U_б$ в коде (10, 01, 00),

$\gamma_к$ — комплекс памяти, достаточно $\sigma(\gamma) = 128$,

$\delta_п :: \Psi(B)$, где B — множество столбцов матрицы U

$[\varepsilon_и]$ — максимальная длина памяти, отводимой для каждого из комплексов α и β, на продолжении которых представляются промежуточные результаты; всегда достаточно $[\varepsilon] = 2\sigma(\alpha^-)$,

$\tau^+ = 0$, если решение получено,
$= f_{10}$, если $[\varepsilon]$ оказалась недостаточной.

Задаются: a_α, a_β, a_γ, b_α, b_β.
Подпрограммы: н е о р т о, в ы т р о м а.
Внутренние операнды: p, j, Y.

3. Упрощение матрицы U производится в два цикла. Сначала перебираются подряд все строки матрицы U, причем очередная строка удаляется, если это возможно, в противном случае исследуется возможность присвоения значения «—» тем из компонент строки, которые обладают значением 0 или 1, и, если это возможно, такое присвоение реализуется. Затем вновь перебираются подряд и выбрасываются, если это окажется возможным, оставшиеся в матрице строки (такая возможность может появиться в результате присвоения некоторым элементам матрицы значений «—» в первом цикле упрощения).

4.

* **001 700**

§ 0 $\bar{\bigcirc}\, d\, b_\alpha \Rightarrow e + a_\alpha \Rightarrow a_{27}\, e + a_\beta \Rightarrow a_{30} \bar{\bigcirc}\, g$

§ 1 $\triangle\, d \Rightarrow f \oplus e \circ\!\!\rightarrow 5\, \alpha_d \Rightarrow \boldsymbol{i}\, \beta_d \Rightarrow \boldsymbol{j} \vee \boldsymbol{i} \Rightarrow \boldsymbol{k}$
$\oplus\, \delta \Rightarrow \boldsymbol{h} \twoheadrightarrow 6 \mapsto 10\, j \circ\!\!\rightarrow 4\, g \circ\!\!\rightarrow 1\, \bar{\bigcirc} f$

§ 2 $\boldsymbol{k}\, \dot{\mathrm{X}}\, 3\, h\, \boldsymbol{i} \Rightarrow \boldsymbol{l} \wedge c_h \oplus \boldsymbol{i} \Rightarrow \boldsymbol{i}\, \boldsymbol{j} \Rightarrow \boldsymbol{m} \wedge c_h \oplus \boldsymbol{j} \Rightarrow \boldsymbol{j}$
$\vee \boldsymbol{i} \oplus \delta \Rightarrow \boldsymbol{h} \twoheadrightarrow 6 \mapsto 10\, j \circ\!\!\rightarrow 2\, \boldsymbol{l} \Rightarrow \boldsymbol{i}\, \boldsymbol{m} \Rightarrow \boldsymbol{j} \rightarrow 2$

§ 3 $\boldsymbol{i} \Rightarrow \alpha_d\, \boldsymbol{j} \Rightarrow \beta_d \rightarrow 1$

§ 4 $\bar{\triangle}\, e \Rightarrow b_\alpha \Rightarrow b_\beta\, \alpha_e \Rightarrow \alpha_d\, \beta_e \Rightarrow \beta_d\, \bar{\triangle}\, d \rightarrow 1$

§ 5 $g \circ\!\!\rightarrow 10 \bigcirc g\, \bar{\bigcirc}\, d \rightarrow 1$

§ 6 $\varepsilon - e \Rightarrow b_{27}$ * неорто $\alpha\, \beta\, \boldsymbol{i}\, \boldsymbol{j}\, f\, \boldsymbol{Y}\, \boldsymbol{Z}\, /\!/ \Rightarrow i \mapsto 7\, \bar{\bigcirc}\, \boldsymbol{j}$
$b_{27} \circ\!\!\rightarrow 7 \bigcirc j\, \boldsymbol{h} \circ\!\!\rightarrow 7$
$\bar{\bigcirc}\, j$ * вытрома $\boldsymbol{Y}\, \boldsymbol{Z}\, \boldsymbol{h}\, \boldsymbol{n}\, \boldsymbol{p}\, 7\, \gamma \,\|\bigcirc j$

§ 7 $i\,!$

§ 10.

Построение минора из неортогональных строк — неорто

1. Из троичной матрицы $\boldsymbol{U}$ выделить минор $\boldsymbol{U}'$, образованный из строк, неортогональных троичному вектору $\boldsymbol{m}$. В частности, вектором $\boldsymbol{m}$ может служить одна из строк матрицы $\boldsymbol{U}$, в этом случае она не включается в минор $\boldsymbol{U}'$.

2. Внешние операнды:

$\alpha\beta_к :: \boldsymbol{U}$ в коде (10,01,00),

$\gamma\delta_п :: \boldsymbol{m}$ в коде (10,01,00),

$[\varepsilon_и] = i$, если вектор $\boldsymbol{m}$ является i-й строкой матрицы $\boldsymbol{U}$,
$= [f_{10}]$ в противном случае,

$\zeta\eta^+_к ::$ минор $\boldsymbol{U}'$ в коде (10,01,00); максимальная длина памяти, отводимой для каждого из комплексов ζ и η, задается значением b_ζ; достаточно $b_\zeta = \sigma(\alpha)$,

$\tau^+ = 0$, если решение получено,
$= f_{10}$, если $[b_\zeta]$ оказалась недостаточной.

Задаются: a_α, a_β, a_ζ, a_η, b_α, b_β, b_ζ.
Совмещение: α_ζ, $\beta\eta$.
Внутренние операнды: $\boldsymbol{a}$, b, —

4.

* **001 703**

§ 0 $\overline{\circ} a \circ b$

§ 1 $\triangle a \oplus b_{\alpha} \circ\!\!\rightarrow 2\, a \oplus \varepsilon \circ\!\!\rightarrow 1$

$\alpha_a \wedge \delta \Rightarrow \boldsymbol{a}\, \beta_a \wedge \gamma \vee \boldsymbol{a} \mapsto 1$

$\alpha_a \Rightarrow \zeta_b \beta_a \Rightarrow \eta_b \triangle b \oplus b_{\zeta} \mapsto 1\, f_{10} \rightarrow 3$

§ 2 $b \Rightarrow b_{\zeta} \Rightarrow b_{\eta} 0$

§ 3 .

Реализация описанной программы приводит в рассматриваемом случае к получению следующей безызбыточной формы:

a	*b*	*c*	*d*	*e*	*f*	*g*
1	0	–	–	1	0	1
1	–	0	0	1	–	1
–	0	1	–	–	–	0
–	–	1	0	0	0	0
–	0	–	–	0	0	0
1	0	–	1	0	0	–
–	1	0	–	0	1	0
–	–	0	0	1	0	1
1	–	1	–	0	1	–
0	0	–	1	–	1	1
1	–	1	1	–	–	–
–	0	–	0	–	–	0
0	1	0	–	–	0	1
0	–	–	0	1	–	0
0	–	1	–	1	1	0
–	1	0	1	0	–	0
0	–	1	0	–	1	1
0	0	1	0	0	–	–
1	1	0	0	–	1	–
1	1	–	–	0	–	0
–	0	1	–	–	1	–
1	1	–	1	–	1	1

Заметим, что при выполняемом данной программой преобразовании строки исходной матрицы подвергаются некоторому перемешиванию (это связано с повышением эффективности вычислительного процесса), что следует

иметь в виду при анализе примера. Аналогичной чертой характеризуются и некоторые из программ, излагаемых ниже.

Нахождение ядра. Поиск кратчайшей (или минимальной) днф заданной булевой функции может быть существенно облегчен предварительным нахождением *ядра безызбыточных днф*, то есть множества простых импликант, входящих в любую безызбыточную днф. Очевидно, что в матричном виде это множество будет представляться ядром безызбыточных форм, рассмотренным нами в § 8 главы 2, где был изложен метод извлечения данного ядра из произвольной безызбыточной днф. Переформулировка этих результатов на алгебраический язык приводит нас к следующему утверждению:

член k днф D принадлежит ядру безызбыточных днф в том и только в том случае, когда не будет выполняться тождество

$$\bigvee_{k_i \in D^{**}} (k_i : k) \equiv 1,$$

где D^{**} — множество таких отличных от k членов из D, которые неортогональны конъюнкции k или смежны с нею.

Как видно, проверка этого условия сводится к анализу на вырожденность такого минора троичной матрицы, задающей исследуемую днф, который образуется множеством строк, неортогональных или смежных строке, представляющей элементарную конъюнкцию k (разумеется, исследуемая строка в этот перечень не входит), и столбцов, в которых эта строка имеет значение «—». Решение этой задачи поручается программе вытрома.

Нахождение ядра в целом производится следующей программой.

Нахождение ядра безызбыточных форм — ядро

1. В троичной матрице $\boldsymbol{U}$ выделить строки, принадлежащие ядру безызбыточных форм, то есть входящие в любую безызбыточную матрицу, эквивалентную матрице $\boldsymbol{U}$, и поместить их в начале матрицы $\boldsymbol{U}'$, получаемой из $\boldsymbol{U}$ путем соответствующей перестановки строк. Если матрица $\boldsymbol{U}$ является безызбыточной, то ядро будет получено целиком.

2. Внешние операнды:

$\alpha\beta_к^-$ $::U$ в коде (10,01,00),

$\alpha\beta_к^+$ $::U'$ в коде (10,01,00),

$[\gamma_и^+]$ — число найденных элементов ядра,

$\delta_п$ $::\Psi(B)$, где B — множество столбцов матрицы U,

$[\varepsilon_и]$ — максимальная длина памяти, отведенной для каждого из комплексов α и β, на продолжении которых представляются промежуточные результаты; всегда достаточно $[\varepsilon] = 2\sigma(\alpha)$,

$\zeta_к$ — комплекс памяти, достаточно $\sigma(\zeta) = 128$,

τ^+ $= 0$, если решение получено,

$= f_{10}$, если $[\varepsilon]$ оказалась недостаточной.

Задаются: a_α, a_β, a_ζ, b_α, b_β.
Подпрограммы: неортосмеж, вытрома.
Внутренние операнды: j, f, Y.

4.

* **001 706**

§ 0 $\bar{0}\, d \circ \gamma\, b_\alpha + a_\alpha \Rightarrow a_{27}\, b_\alpha + a_\beta \Rightarrow a_{30}$

§ 1 $\Delta\, d \Rightarrow f \oplus b_\alpha \circ\!\!\rightarrow 3\alpha_a \Rightarrow i\,\beta_d \Rightarrow j \vee i \oplus \delta \Rightarrow h$

$\varepsilon - b_\alpha \Rightarrow b_{27}$ * неортосмеж $\alpha\,\beta\, i\, j\, f\, Y\, Z\ //\!\!\rightarrow 3$

$b_{27} \circ\!\!\rightarrow 2$ * вытрома $Y\, Z\, h\, i\, j\, 2\zeta\ //\!\!\rightarrow 1$

§ 2 $\alpha_d \Leftrightarrow \alpha_\gamma\, \beta_d \Leftrightarrow \beta_\gamma\ \Delta\, \gamma \rightarrow 1$

§ 3 .

Построение минора из неортогональных и смежных строк — неортосмеж

1. Из троичной матрицы U выделить минор U', образованный из строк, неортогональных или смежных троичному вектору m. В частности, вектором m может служить одна из строк матрицы U, в этом случае она не включается в минор U'.

2. Внешние операнды:
$\alpha\beta_к :: \boldsymbol{U}$ в коде (10,01,00),
$\gamma\delta_п :: \boldsymbol{U}'$ в коде (10,01,00),
$[\varepsilon_и] = i$, если вектор $\boldsymbol{m}$ является i-й строкой матрицы $\boldsymbol{U}$,
$= [f_{10}]$ в противном случае,
$\zeta\eta_к^+ ::$ минор $\boldsymbol{U}'$ в коде (10,01,00); максимальная длина памяти, отводимой для каждого из комплексов ζ и η, задается значением b_ζ; всегда достаточно $[b_\zeta] = \sigma(\alpha)$,
$\tau^+ = 0$, если решение получено,
$= f_{10}$, если $[b_\zeta]$ оказалась недостаточной.

Задаются: a_α, a_β, a_ζ, a_η, b_α, b_β, b_ζ.
Совмещение: $\alpha\zeta$, $\beta\eta$.
Внутренние операнды: $\boldsymbol{a}$, b, —.

4.

* **001 705**

§ 0 $\overline{\bigcirc} a \bigcirc b$

§ 1 $\triangle\, a \oplus b_\alpha \circ\!\!\rightarrow 2\, a \oplus \varepsilon \circ\!\!\rightarrow 1\, \alpha_a \wedge \delta \Rightarrow \boldsymbol{a}\beta_a \wedge \gamma \vee \boldsymbol{a}$
$\nabla - 2 \mapsto 1\, \alpha_a \Rightarrow \zeta_b \beta_a \Rightarrow \eta_b \triangle\, b \oplus b_\zeta \mapsto 1\, f_{10} \rightarrow 3$

§ 2 $b \Rightarrow b_\zeta \Rightarrow b_\eta 0$

§ 3 .

Реализация описанного алгоритма на рассматриваемом примере приводит к получению следующего ядра безызбыточных форм:

a	b	c	d	e	f	g
–	0	1	–	–	–	0
0	0	–	1	–	1	1
1	–	1	1	–	–	–
0	1	0	–	–	0	1
0	0	1	0	0	–	–

Получение множества простых импликант. Возможно, что в ядро попадут все члены рассматриваемой безызбыточной днф и задача ее минимизации окажется таким образом полностью решенной. На оценке вероятности такого события мы остановимся несколько

ниже, пока же ограничимся замечанием, что эта вероятность быстро растет с увеличением числа переменных при заданных ограничениях на сложность исходной д н ф.

В нашем примере, однако, такое событие не наступило: оказалось, что бо́льшая часть членов рассматриваемой безызбыточной д н ф в ядро не входит. В этом случае следует перейти к нахождению множества A простых импликант данной функции, которое может быть задано матрицей $\boldsymbol{R}$, представляющей собой расширение полученной перед этим безызбыточной троичной матрицы.

Для решения этой задачи предлагается следующая программа, в основу которой положен метод Блека — Порецкого (см. § 6 главы 2), дополненный некоторыми соображениями, позволяющими сократить производимые вычисления.

Расширение троичной матрицы — р а с ш и р е н и е

1. Находится расширение $\boldsymbol{U}_p$ троичной матрицы $\boldsymbol{U}$, строки которой не находятся в отношении поглощения. Возможная интерпретация: если $\boldsymbol{U}$ представляет д н ф некоторой булевой функции, то $\boldsymbol{U}_p$ будет представлять множество всех простых импликант этой функции.

2. Внешние операнды:

$\alpha\beta_к^-::\boldsymbol{U}$ в коде (10,01,00),

$\alpha\beta_к^+::\boldsymbol{U}_p$ в коде (10,01,00),

$[\gamma_и]$ — максимальная длина памяти, отведенной для каждого из комплексов α и β,

$\tau^+=0$, если решение получено,

$=f_{10}$, если $[\gamma]$ оказалась недостаточной.

Задаются: a_α, a_β, b_α, b_β.

Внутренние операнды: $\boldsymbol{e}$, i, $-$.

4.

* **001 533**

§ 0 $\circ b\,\bar{\circ}\,c\,b_\alpha \Rightarrow a$

§ 1 $\triangle b - a \mapsto 11\ \triangle c\,\bar{\circ}\,d$

§ 2 $\triangle d - b \mapsto 1$

$\alpha_d \wedge \beta_b \Rightarrow c\,\alpha_b \wedge \beta_d \vee c \Rightarrow c\ \nabla \oplus 1 \mapsto 2$

$\alpha_b \vee \alpha_d \oplus c \Rightarrow a\,\beta_b \vee \beta_d \oplus c \Rightarrow b\,\bar{\circ}\,e\,\bar{\circ}\,e\,\circ\,d$

§ 3 $\triangle e - a \mapsto 7\, \boldsymbol{d} \mapsto 4$
$\boldsymbol{a} \urcorner \wedge \alpha_e \mapsto 4\, \boldsymbol{b} \urcorner \wedge \beta_e \multimap 2$

§ 4 $\alpha_e \urcorner \wedge \boldsymbol{a} \mapsto 3 \beta_e \urcorner \wedge \boldsymbol{b} \mapsto 3 \bar{\circ} \boldsymbol{d} \bar{\triangle} a\, 4 \Rightarrow i$
$1 \Rightarrow f\, b - e \multimap 6 \triangle f\, b \oplus e \mapsto 5 \circ e$

§ 5 $\triangle f\, d - e \multimap 6 \triangle f$

§ 6 $\bar{\triangle} f \oplus f_{10} \multimap 3\, h_i \Rightarrow h\, h_f \Rightarrow g$
$\alpha_g \Rightarrow \alpha_h \beta_g \Rightarrow \beta_h \bar{\triangle} h_i f \Rightarrow i \to 6$

§ 7 $\boldsymbol{a} \Rightarrow \alpha_a \boldsymbol{b} \Rightarrow \beta_a \triangle a \oplus \gamma \multimap 10\, \boldsymbol{e} \multimap 1 \to 2$

§ 10 $f_{10} \to 12$

§ 11 $a \Rightarrow b_\alpha \Rightarrow b_\beta 0$

§ 12 .

Реализация программы расширение приводит к расширению множества простых импликант, содержащихся в рассматриваемой безызбыточной днф (22 импликанты), путем дополнения его восемнадцатью элементарными конъюнкциями, представляемыми строками следующей матрицы:

a	b	c	d	e	f	g
1	–	–	–	0	0	0
1	0	–	1	–	0	1
–	0	0	0	1	0	–
1	–	0	0	1	1	–
1	1	0	–	–	1	1
1	0	–	0	1	–	–
1	0	1	–	1	–	–
–	–	0	0	–	1	0
1	–	–	0	0	–	0
0	–	0	0	1	0	–
0	–	1	0	–	0	0
–	–	1	1	1	1	0
0	1	0	1	0	0	–
0	–	1	0	1	1	–
–	–	1	0	0	1	1
–	–	0	1	0	0	0
1	–	1	–	0	–	0
1	1	–	–	0	1	–

Нахождение циклического остатка. Далее мы учтем специфику решаемой нами задачи,

выражающуюся в том, что нас интересует нахождение лишь одной из кратчайших д н ф, неважно какой именно.

Известно, что производимые при минимизации некоторой булевой функции вычисления можно сократить, предварительно разбив множество A всех простых импликант на три класса: я д р о, которому принадлежат те элементы из A, которые входят в любую безызбыточную д н ф, *антиядро*, состоящее из таких простых импликант, которые не входят ни в одну из безызбыточных д н ф (такие элементарные конъюнкции должны имплицировать дизъюнкцию элементов ядра), и *циклическую часть*, образуемую из остальных элементов множества A. Объем последующих вычислений зависит, прежде всего, от мощности циклической части, в пределах которой организуется некоторый перебор, производимый при поиске решения.

Очевидна целесообразность «сжатия» этой части множества A путем некоторых не слишком трудоемких операций. Соответствующая методика уже описывалась в § 8 главы 2, и здесь мы ограничимся ее переформулировкой на алгебраический язык.

Введем понятие *квазиядра кратчайшей д н ф* (полностью соответствующее понятию квазиядра кратчайшей формы троичной матрицы, рассмотренному в предыдущей главе), определив его как такое множество простых импликант, которое содержится по крайней мере в одной из кратчайших д н ф (то есть в множестве членов этой д н ф). По аналогии определим *квазиантиядро кратчайшей* д н ф, называя так такое множество простых импликант, которое не пересекается с множеством членов по крайней мере одной кратчайшей д н ф.

Рассматриваемое ниже преобразование множества A будет заключаться в последовательном сокращении его циклической части до так называемого циклического остатка путем передачи некоторых ее элементов в квазиядро или в квазиантиядро. Для рассмотрения этого преобразования введем символы текущих значений квазиядра A', квазиантиядра A'' и циклического остатка $A''' = A \setminus A' \cup A''$.

За исходные значения множеств A' и A''' принимаются ядро и его дополнение $A \setminus A'$, соответственно. Затем организуется чередование следующих двух операций:

а) Пусть k_i, $k_j \in A'''$ и $i \neq j$. Тогда k_i передается в A'', если $k_i \to D' \vee k_j$, где D' — дизъюнкция всех элементов из A'.

б) Элемент k_i из A''' передается в A', если он не имплицирует дизъюнкцию остальных элементов множества $A' \cup A'''$.

Интересно отметить цепной характер описанного преобразования, выражающийся в том, что передача некоторых простых импликант в квазиантиядро может открыть возможности нахождения новых элементов квазиядра. В связи с этим преобразование в целом производится за несколько циклов и завершается, когда уже ни одна из простых импликант, оставшихся в циклическом остатке, не может быть передана ни в множество A', ни в множество A''.

Описанное преобразование может приводить к существенному сокращению множества A'''.

Нахождение циклического остатка — циклостат

1. Троичной матрицей $\boldsymbol{A}$ задано множество A всех простых импликант некоторой булевой функции, причем m начальных строк этой матрицы представляют ядро, а остальные рассматриваются как начальное значение циклического остатка. Требуется выделить в множестве A квазиядро A' и квазиантиядро A'' и удалить последнее из A, представив результат матрицей $\boldsymbol{A}^*$, n начальных строк которой будут представлять квазиядро A', остальные — циклический остаток A'''.

2. Внешние операнды:

$\alpha\beta_к^- :: \boldsymbol{A}$ в коде (10,01,00),

$\alpha\beta_к^+ :: \boldsymbol{A}^*$ в коде (10,01,00),

$\gamma_к$ — комплекс памяти; достаточно $\sigma(\gamma) = 128$,

$\delta_п :: \Psi(B)$, где B — множество столбцов матрицы $\boldsymbol{A}$,

$[\varepsilon_и^-] = m$,

$[\varepsilon_и^+] = n$,

$[\zeta_и]$ — максимальная длина памяти, отведенной для каждого из комплексов α и β, на продолжении которых представляются промежуточные результаты; всегда достаточно $[\zeta] = 2\sigma(\alpha^-)$,

$\tau^+ = 0$, если решение найдено,

$= f_{10}$, если $[\zeta]$ оказалась недостаточной.

Задаются: a_α, a_β, a_γ, b_α, b_β.

Подпрограммы: квазиантиядро, квазиядро.

Внутренние операнды: j, j, Y.

3. Реализуется описанный выше алгоритм, причем поочередно для каждого $k_j \in A'''$ перебираются все $k_i \in A'''$ и для пар k_i, k_j выполняется операция а), затем поочередно для всех $k_j \in A'''$ выполняется операция б). Этот процесс повторяется, пока множества A' и A''' не стабилизируются.

4.

```
        * 001 773
§ 0
§ 1  bα⇒j * квазиантиядро αβγδεζ //
     ⟼ 3 j ⊕ bα ○⟼ 3
     ε⇒i * квазиядро αβεδζγ // ⟼ 3
     ε + 1 ⊕ bα ⟼ 2 Δ̄ bα Δ̄ bβ 0 ⟼ 3
§ 2  ε⊕i ⟼ 1
§ 3  .
```

Удаление элементов квазиантиядра — квазиантиядро

1. Троичной матрицей A задана совокупность простых импликант некоторой булевой функции, причем m начальных строк этой матрицы представляют квазиядро A'. Из числа остальных строк требуется удалить такие, которые представляют элементы квазиантиядра (имплицирующие дизъюнкцию всех элементов квазиядра и одного из остальных элементов, представленных в матрице A, за исключением рассматриваемого элемента). Результат представляется соответствующим сокращением матрицы A до A^*.

2. Внешние операнды:

$\alpha\beta_к^- :: A$ в коде (10,01,00),

$\alpha\beta_к^+ :: A^*$ в коде (10,01,00),

$\gamma_к$ — комплекс памяти; достаточно $\sigma(\gamma) = 128$,

$\delta_п :: \Psi(B)$, где B — множество столбцов матрицы A,

$[\varepsilon_и] = m$,

$[\zeta_и]$ — максимальная длина памяти, отведенной для каждого из комплексов α и β, на продолжении которых представляются промежуточные результаты; всегда достаточно $[\zeta] = 2\sigma(\alpha^-)$,

$\tau^+ = 0$, если решение получено,

$= f_{10}$, если $[\zeta]$ оказалась недостаточной.

Задаются: a_α, a_β, a_γ, b_α, b_β.
Подпрограмма: неорто.
Внутренние операнды: *j*, *h*, ***Y***.

4.

* **001 711**

§ 0 $b_\alpha \Rightarrow e\,\varepsilon - 1 \Rightarrow h + 2 \Rightarrow b_\alpha \bar{\circ} f$

§ 1 $\Delta\, h - e \longmapsto 4\,\alpha_h \Leftrightarrow \alpha_\varepsilon\,\beta_h \Leftrightarrow \beta_\varepsilon\,\varepsilon \Rightarrow d$

§ 2 $\Delta\, d - e \longmapsto 1\,\alpha_d \Rightarrow \boldsymbol{i}\,\beta_d \Rightarrow \boldsymbol{j} \vee \boldsymbol{i} \oplus \delta \Rightarrow \boldsymbol{h}$

$a_\alpha + e \Rightarrow a_{27}\,a_\beta + e \Rightarrow a_{30}\,\zeta - e \Rightarrow b_{27}$

* неорто $\alpha\,\beta\,\boldsymbol{i}\,\boldsymbol{j}\,f\,\boldsymbol{Y}\,\boldsymbol{Z}$ // $\longmapsto 5\,b_{27} \circ\!\!\rightarrow 2$

* вытрома $\boldsymbol{Y}\,\boldsymbol{Z}\,\boldsymbol{h}\,\boldsymbol{i}\,\boldsymbol{j}\,2\,\gamma$ //

$\bar{\Delta}\, e\, h - d \longmapsto 3\,\alpha_e \Rightarrow \alpha_d\,\beta_e \Rightarrow \beta_d\,\bar{\Delta}\, d \rightarrow 2$

§ 3 $\alpha_h \Rightarrow \alpha_d\,\alpha_e \Rightarrow \alpha_h\,\beta_h \Rightarrow \beta_d\,\beta_e \Rightarrow \beta_h\,\bar{\Delta}\, d\,\bar{\Delta}\, h \rightarrow 2$

§ 4 $e \Rightarrow b_\alpha \Rightarrow b_\beta\, 0$

§ 5 .

Расширение квазиядра — квазиядро

1. Троичной матрицей $\boldsymbol{A}$ задана совокупность простых импликант некоторой булевой функции, причем m начальных строк этой матрицы представляют элементы квазиядра. Требуется расширить квазиядро, включив в него каждую такую строку матрицы $\boldsymbol{A}$, которая представляет элементарную конъюнкцию, не имплицирующую дизъюнкцию конъюнкций, представляемых остальными строками данной матрицы. Результат будет представлен матрицей $\boldsymbol{A}^*$, получаемой из $\boldsymbol{A}$ некоторой перестановкой строк; полученное квазиядро будет представлено n начальными строками этой матрицы.

2. Внешние операнды:

$\alpha\beta_к^- :: \boldsymbol{A}$ в коде (10, 01, 00),

$\alpha\beta_к^+ :: \boldsymbol{A}^*$ в коде (10, 01, 00),

$[\gamma_и^-] = m$,
$[\gamma_и^+] = n$,
$\delta_п :: \Psi(B)$, где B — множество столбцов матрицы $\boldsymbol{A}$,
$[\varepsilon_и]$ — максимальная длина памяти, отведенной для каждого из комплексов α и β, на продолжении которых представляются промежуточные результаты; всегда достаточно $[\varepsilon] = 2\sigma(\alpha)$,
$\zeta_к$ — комплекс памяти; достаточно $\sigma(\gamma) = 128$,
$\tau^+ = 0$, если решение получено,
$= f_{10}$, если $[\varepsilon]$ оказалась недостаточной.

Задаются: a_α, a_β, a_ζ, b_α, b_β.
Подпрограммы: неорто, вытрома.
Внутренние операнды: j, f, $\boldsymbol{Y}$.
4.

* **001 722**
§ 0 $\gamma - 1 \Rightarrow d\, b_\alpha \Rightarrow e + a_\alpha \Rightarrow a_{27}\, e + a_\beta \Rightarrow a_{30}$
§ 1 $\triangle d \Rightarrow f \oplus e \circ\!\!\rightarrow 3\, \alpha_d \Rightarrow \boldsymbol{i}\, \beta_d \Rightarrow \boldsymbol{j} \vee \boldsymbol{i} \oplus \delta \Rightarrow \boldsymbol{h}$
$\varepsilon - e \Rightarrow b_{27}$ * неорто $\alpha\, \boldsymbol{\beta\, i\, j\, f\, Y\, Z}\, //\, \mapsto 3\, b_{27} \circ\!\!\rightarrow 2$
* вытрома $\boldsymbol{Y\, Z\, h\, i\, j}\, 2\, \zeta\, // \rightarrow 1$
§ 2 $\alpha_d \Leftrightarrow \alpha_\gamma\, \beta_d \Leftrightarrow \beta_\gamma \triangle \gamma \rightarrow 1$
§ 3 .

Однако вернемся к рассмотрению нашей конкретной задачи. Покажем, например, что простую импликанту $a\bar{b}ce$ (эта импликанта представлена строкой 1 0 1 — 1 — —, полученной в результате расширения безызбыточной матрицы) можно отправить в квазиантиядро A''. Для этого достаточно взять пятую строку расширенной матрицы 1 0 — 0 1 — —, соответствующую импликанте $a\bar{b}\bar{d}e$, и проверить, имеет ли место отношение $a\bar{b}ce \rightarrow D' \vee a\bar{b}\bar{d}e$, где $D' \vee a\bar{b}\bar{d}e$ представляется троичной матрицей

a	b	c	d	e	f	g
—	0	1	—	—	—	0
0	0	—	1	—	1	1
1	—	1	1	—	—	—
0	1	0	—	—	0	1
0	0	1	0	0	—	—
1	0	—	0	1	—	—

Проверка сводится к анализу на вырожденность матрицы

$$\begin{bmatrix} - & - & 0 \\ 1 & - & - \\ 0 & - & - \end{bmatrix}$$

которая действительно оказывается вырожденной, откуда и следует, что простая импликанта $a\bar{b}ce$ может быть выброшена из дальнейшего рассмотрения, то есть включена в квазиантиядро.

В конце концов в квазиантиядро попадают простые импликанты, представляемые строками матрицы

$$\begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & - & 1 & - \\ 1 & 1 & - & 1 & - & 1 & 1 \\ 1 & 0 & 1 & - & 1 & - & - \\ 0 & - & 0 & 0 & 1 & 0 & - \\ - & - & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & - \\ - & - & 1 & 0 & 0 & 1 & 1 \\ 1 & - & 1 & - & 0 & - & 0 \end{bmatrix} \end{array}$$

а ядро расширяется до квазиядра путем дополнения его элементарными конъюнкциями $\bar{a}ce\bar{f}\bar{g}$, $ab\bar{c}fg$ и $\bar{a}c\bar{d}fg$ и представляется следующей матрицей:

$$\begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} - & 0 & 1 & - & - & - & 0 \\ 0 & 0 & - & 1 & - & 1 & 1 \\ 1 & - & 1 & 1 & - & - & - \\ 0 & 1 & 0 & - & - & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & - & - \\ 0 & - & 1 & - & 1 & 1 & 0 \\ 1 & 1 & 0 & - & - & 1 & 1 \\ 0 & - & 1 & 0 & - & 1 & 1 \end{bmatrix} \end{array}$$

Что касается полученного циклического остатка, то он

будет представлен следующей матрицей:

a	b	c	d	e	f	g
0	–	1	0	–	0	0
–	1	0	–	0	1	0
–	–	0	0	1	0	1
1	–	1	–	0	1	–
1	–	0	0	1	–	1
1	0	–	–	1	0	1
–	0	–	0	–	–	0
–	–	1	0	0	0	0
0	–	–	0	1	–	0
–	–	0	1	0	0	0
–	1	0	1	0	–	0
0	–	0	0	1	0	–
–	0	–	–	0	0	0
1	1	–	–	0	1	–
1	1	–	–	0	–	0
–	0	1	–	–	1	–
1	–	0	0	1	1	–
1	–	–	–	0	0	0
1	0	–	1	–	0	1
–	0	0	0	1	0	–
1	0	–	1	0	0	–
1	–	–	0	0	–	0
1	0	–	0	1	–	–
–	–	0	0	–	1	0

Построение простой матрицы Квайна. После получения множеств A' и A''' задача нахождения кратчайшей днф сводится к выделению из A''' некоторого подмножества A_i, такого, чтобы дизъюнкция элементов множества $A' \cup A_i$ была равносильна исходной днф, и обладающего при этом минимальной мощностью.

Продолжая переформулировку материала параграфа 8 главы 2, назовем *простой совокупностью* такое подмножество A_k из множества A всех простых импликант, которое обладает свойством: «в любую безызбыточную днф заданной функции входит по крайней мере один из элементов множества A_k», однако теряет это свойство при удалении из A_k любого элемента. Множество всех простых совокупностей достаточно удобно представляется уже знакомой нам простой матрицей поглощений, ко-

торую в алгебраической интерпретации мы будем называть *простой матрицей Квайна*. Построение этой матрицы эквивалентно нахождению всех простых совокупностей путем некоторого перебора подмножеств из A и их анализа, в результате которого выясняется, является ли рассматриваемое подмножество простой совокупностью.

Подмножество A_k из A оказывается простой совокупностью в том и только в том случае, когда выполняются следующие два условия:

а) конъюнкция элементов из A_k не имплицирует дизъюнкцию элементов из $A \setminus A_k$,

б) не существует такого $A_j \subset A_k$, для которого выполнялось бы условие а).

Разложив множество A всех простых импликант на квазиядро A', квазиантиядро A'' и циклический остаток A''', мы существенно сократили перебор подмножеств из A, ограничившись перебором подмножеств в циклическом остатке A'''. Заметим, что такая замена может привести к потере некоторых решений — кратчайших д н ф, в которые могут входить элементы квазиантиядра и которые уже нельзя будет поэтому найти при дальнейших вычислениях. Однако для нас такая потеря несущественна, поскольку нахождение одной из кратчайших д н ф гарантируется. Что касается квазиядра A', то оно играет роль множества всех одноэлементных простых совокупностей (а точнее говоря, таких простых совокупностей, каждой из которых принадлежит ровно один элемент из $A' \cup A'''$).

Перебор подмножеств из A''' организуется следующим образом. Сначала строится множество пар взаимно неортогональных элементов из A''', затем из него выделяются пары, оказывающиеся после проверки простыми совокупностями. Эти пары фиксируются путем достройки простой матрицы Квайна соответствующими столбцами, а остальные из неортогональных пар образуют множество S_2, используемое в качестве своеобразного источника трехэлементных подмножеств из A''', заслуживающих последующего рассмотрения. А именно, некоторое трехэлементное подмножество из A''' будет рассматриваться в том и только в том случае, когда все содержащиеся в нем двухэлементные подмножества являются элементами множества S_2.

Аналогичным образом организуется перебор и анализ трехэлементных подмножеств, четырехэлементных и т. д. Этот процесс завершается, когда получаемое на очередном шаге множество k-элементных множеств оказывается пустым.

Переходя к рассмотрению реализующих описанный процесс программ, заметим, что простая матрица Квайна находится в несколько иной форме и обладает двумя характерными особенностями. Во-первых, в ней не представляются одноэлементные простые совокупности, поскольку они уже найдены — ими служат элементы полученного квазиядра. Во-вторых, матрица строится в транспонированном виде, так как это упрощает построение программы.

Получение простых совокупностей — п р о с о

1. Троичной матрицей $\boldsymbol{A}$ задано множество $A = A' \cup \cup A'''$ элементов ядра A' (представленных m начальными строками матрицы) и циклического остатка A'''. Требуется получить множество P простых совокупностей из A''', то есть все подмножества $A_k \subseteq A'''$, удовлетворяющие следующим двум условиям:

а) конъюнкция элементов из A_k не имплицирует дизъюнкцию элементов из $A \setminus A_k$,

б) не существует такого $A_j \subset A_k$, для которого выполнялось бы условие а).

2. Внешние операнды:

$\alpha\beta_к :: \boldsymbol{A}$ в коде (10, 01, 00),

$\gamma_к$ — комплекс памяти; достаточно $\sigma(\gamma) = 128$,

$\delta_п :: \Psi(B)$, где B — множество столбцов матрицы $\boldsymbol{A}$,

$\varepsilon_к^+ :: \| P \ni A''' \|$,

$[\xi_и] = m$,

$\tau^+ = 0$, если решение получено,

$= f_{10}$, если оказалась недостаточной $[b_\varepsilon]$, задающая максимальную длину памяти, отводимой для комплекса ε.

Задаются: a_α, a_β, a_γ, a_ε, b_α, b_β, b_ε.

Подпрограммы: п а п е р е с, с е п а р а т о р, п е р е к, к о н д е н с а т о р.

Внутренние операнды: j, h, $\boldsymbol{W}$.

Примечание: на продолжении комплексов α и β представляется промежуточная информация, в связи с чем память для каждого из них отводится вдвое большая, чем $\sigma(A)$.

3. Реализуемый алгоритм содержит две основные операции: перебор и анализ некоторых подмножеств из A'''. Операция перебора представляется программой конденсатор, организующей построение из имеющихся k-элементных совокупностей $k+1$-элементных, удовлетворяющих условию б). Операция анализа полученных $k+1$-элементных совокупностей производится программой сепаратор, выделяющей из них простые совокупности (проверкой условия а)); остальные служат базой для построения $k+2$-элементных совокупностей. Процесс перебора подмножеств из A''' начинается с получения двухэлементных совокупностей, находимых программой паперес, и кончается, когда очередное множество k-элементных совокупностей оказывается пустым.

4.

* **001 713**

§ 0 $a_\varepsilon \Rightarrow h + b_\varepsilon - 40 \Rightarrow a_{25}$

$a_\alpha + \zeta \Rightarrow a_{27}\, a_\beta + \zeta \Rightarrow a_{30}\, b_\alpha - \zeta \Rightarrow b_{27}$

* паперес $\boldsymbol{Y}\,\boldsymbol{Z}\,\varepsilon$ // $\mapsto 3$

§ 1 * сепаратор $\alpha\,\beta\,\zeta\,\varepsilon\, g\,\delta\,\gamma$ // $a_\varepsilon + g \Rightarrow a_\varepsilon \Rightarrow a_{26}$

$b_\varepsilon - g \Rightarrow b_{26} \circ\!\!\rightarrow 2\, a_{25} - b_{26} \Rightarrow c - a_\varepsilon \Rightarrow b_\varepsilon$

* перек $\boldsymbol{X}\,c$ // * конденсатор $\boldsymbol{X}\,\boldsymbol{W}\varepsilon$ // $\mapsto 3 \rightarrow 1$

§ 2 $a_\varepsilon - h \Rightarrow b_\varepsilon\, h \Rightarrow a_\varepsilon\, 0$

§ 3 .

Нахождение пар пересекающихся строк — паперес

1. Для заданной троичной матрицы $\boldsymbol{U}$ требуется найти множество W всех пар строк, находящихся в отношении пересечения.

2. Внешние операнды:

$\alpha\beta_к :: \boldsymbol{U}$ в коде (10, 01, 00),

$\gamma_к^+ :: \| W \ni U \|$, где U — множество строк матрицы $\boldsymbol{U}$,

$\tau^+ = 0$, если решение получено,

$= f_{10}$, если оказалась недостаточной $[b_\gamma]$, задающая максимальную длину памяти, отводимой для комплекса γ.

Задаются: a_α, a_β, a_γ, b_α, b_β, b_γ.
Внутренние операнды: —, c, —.

4.

* **001 702**

§ 0 $\bar{\circ} a \circ c$

§ 1 $\triangle a \Rightarrow b \oplus b_\alpha \circ\!\!\rightarrow 3$

§ 2 $\triangle b \oplus b_\alpha \circ\!\!\rightarrow 1\, \alpha_a \wedge \beta_b \longmapsto 2\, \alpha_b \wedge \beta_a \longmapsto 2$
$c_a \vee c_b \Rightarrow \gamma_c \triangle c \oplus b_\gamma \longmapsto 2\, f_{10} \rightarrow 4$

§ 3 $c \Rightarrow b_\gamma 0$

§ 4 .

Выделение элементов с заданным свойством — сепаратор

1. Троичной матрицей $\boldsymbol{A}$ задано множество $A = A' \cup A''' \cup A'''$ элементов ядра A' (представленных m начальными строками матрицы) и циклического остатка A''', а булевой матрицей $\|K \ni A'''\|$ задано множество K некоторых подмножеств из A'''. Требуется выделить в K элементы A_i со свойством «конъюнкция элементов из A_i не имплицирует дизъюнкцию элементов из $A \setminus A_i$» и представить результат булевой матрицей $\|K^* \ni A'''\|$, в которой выделенные элементы будут представлены n начальными строками и которая получается из матрицы $\|K \ni A'''\|$ путем соответствующей перестановки строк (то есть множество K^* получается некоторой перестановкой элементов в множестве K).

2. Внешние операнды:
$\alpha\beta_к :: \boldsymbol{A}$ в коде (10, 01, 00),
$[\gamma_и] = m$,
$\delta_к^- :: \|K \ni A'''\|$,
$\delta_к^+ :: \|K^* \ni A'''\|$,
$[\varepsilon_и^+] = n$,
$\zeta_п :: \Psi(B)$, где B — множество столбцов матрицы $\boldsymbol{A}$,
$\eta_к$ — комплекс памяти; достаточно $\sigma(\eta) = 128$.

Задаются: a_α, a_β, a_δ, a_η, b_α, b_β, b_δ.
Подпрограммы: неорто, вытрома.
Внутренние операнды: j, f, $\boldsymbol{Y}$.
Примечание: на продолжении комплексов α и β представляется промежуточная информация, в связи с чем

память для каждого из них отводится вдвое большая, чем $\sigma(A)$.

4.

* **001 712**

§ 0 $b_\delta \Rightarrow \varepsilon\, b_\alpha \Rightarrow e \bar{\circ} d \bar{\circ} f\, a_\alpha + e \Rightarrow a_{27}\, a_\beta + e \Rightarrow a_{30}$

§ 1 $\triangle d \oplus \varepsilon \circ\!\!\rightarrow 6 \circ \boldsymbol{i} \circ \boldsymbol{j} \bar{\circ} \boldsymbol{a} \circ b$

§ 2 $\triangle a \oplus e \circ\!\!\rightarrow 4\, a - \gamma \circ\!\!\rightarrow 3 \Rightarrow c$

$\delta_d \wedge c_c \circ\!\!\rightarrow 3\, \alpha_a \vee \boldsymbol{i} \Rightarrow \boldsymbol{i}\, \beta_a \vee \boldsymbol{j} \Rightarrow \boldsymbol{j} \rightarrow 2$

§ 3 $\alpha_a \Rightarrow \boldsymbol{y}_b \beta_a \Rightarrow \boldsymbol{z}_b \triangle b \rightarrow 2$

§ 4 $b \Rightarrow b_{27} \boldsymbol{i} \vee \boldsymbol{j} \oplus \zeta \Rightarrow \boldsymbol{h}$ * неорто $\boldsymbol{Y\,Z\,i\,j}\, f\, \boldsymbol{Y\,Z}\, /\!/\, b_{27}$ $\circ\!\!\rightarrow 1\, \boldsymbol{h} \circ\!\!\rightarrow 5$ * вытрома $\boldsymbol{Y\,Z\,h\,i\,j}\, 1\, \eta\, /\!/$

§ 5 $\bar{\triangle}\, \varepsilon\, \delta_d \Leftrightarrow \delta_\varepsilon\, \bar{\triangle}\, d \rightarrow 1$

§ 6 .

Построение матрицы $\boldsymbol{U}(k+1)$ *из* $\boldsymbol{U}(k)$ — конденсатор

1. Задана булева матрица $\boldsymbol{U}(k)$, все строки которой различны и содержат по k единиц. Требуется построить матрицу $\boldsymbol{U}(k+1)$, включив в нее все строки, содержащие по $k+1$ единице и такие, что векторы, получаемые из этих строк удалением любой одной из единиц, находятся среди строк матрицы $\boldsymbol{U}(k)$.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{U}(k)$,

$\beta_к$ — комплекс памяти; достаточно $\sigma(\beta) = 32$,

$\gamma_к^+ :: \boldsymbol{U}(k+1)$,

$\tau^+ = 0$, если решение получено,

$= f_{10}$, если оказалась недостаточной $[b_\gamma]$, задающая максимальную длину памяти, отводимой для комплекса γ.

Задаются: a_α, a_β, a_γ, b_α, b_β.
Подпрограмма: очистка.
Внутренние операнды: $\boldsymbol{b}$, f, —.

4.

* **001 701**

§ 0 $\bar{\circ}\, b \circ e\, \alpha_0 \bigtriangledown \Rightarrow d\, 40 \Rightarrow b_\beta$

§ 1 $\triangle b \Rightarrow f \oplus b_\alpha \circ\!\!\rightarrow 3\, \alpha_b \urcorner \Rightarrow \boldsymbol{a}$

* очистка $\beta\, /\!/$

§ 2 $\triangle f \oplus b_\alpha \circ\!\!\rightarrow 1\, a_f \wedge a \Rightarrow b \vdash \Rightarrow c$
$c_c \oplus b \mapsto 2 \triangle \beta_c \oplus d \mapsto 2$
$\alpha_b \vee b \Rightarrow \gamma_e \triangle e \oplus b_\gamma \mapsto 2\, f_{10} \rightarrow 4$
§ 3 $e \Rightarrow b_\gamma 0$
§ 4 .

Перепись комплекса — перек

1—2. Комплекс α перемещается в оперативном комплексе на место, задаваемое значением индекса β (указывающим новое начало комплекса α в оперативном комплексе).

Задаются: a_α, b_α.

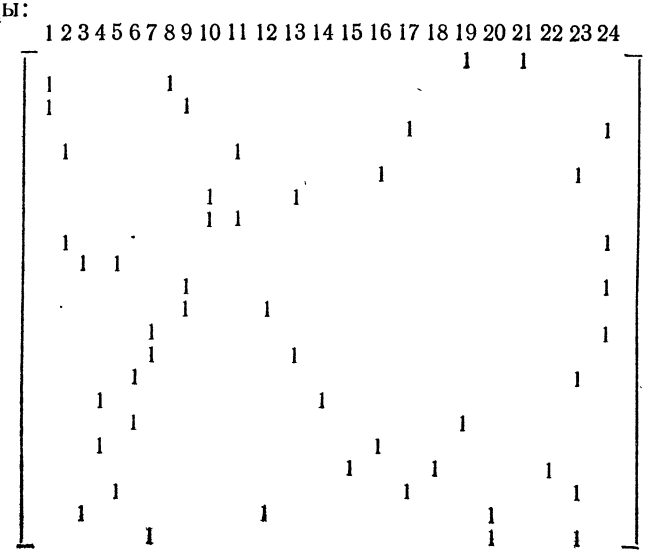
Внутренние операнды: —, b, —.

4.

* 001 704

§ 0 $\overline{\bigcirc} a \beta \Rightarrow b - a_\alpha \circ\!\!\rightarrow 2 \circ\!\!\rightarrow 3\, b_\alpha \Rightarrow a + \beta \Rightarrow b$
§ 1 $\overline{\triangle}\, a\, \overline{\triangle}\, b - \beta \circ\!\!\rightarrow 3\, \alpha_a \Rightarrow k_b \rightarrow 1$
§ 2 $\triangle a \oplus b_\alpha \circ\!\!\rightarrow 3 \alpha_a \Rightarrow k_b \triangle b \rightarrow 2$
§ 3 $\beta \Rightarrow a_\alpha$.

Применение программы проco в рассматриваемом примере приводит к получению следующей булевой матрицы:



Получение кратчайшей днф. Поскольку простые совокупности представляются строками (а не столбцами) булевой матрицы, полученной программой просо, далее следует искать «столбцовое» покрытие этой матрицы. Эта задача может решаться программой окрапок5, если только число столбцов в матрице не превышает 32. В данном случае это условие выполняется, и применение указанной программы приводит нас к кратчайшему покрытию, представляемому булевым вектором

$$101100001010101000100011.$$

Отыскивая в циклическом остатке импликанты, соответствующие тем компонентам этого вектора, которые имеют значение 1, и объединяя их с элементами квазиядра, мы получим окончательное решение — одну из кратчайших днф. Она будет представлена троичной матрицей

a	b	c	d	e	f	g
—	0	1	—	—	—	0
0	0	—	1	—	1	1
1	—	1	1	—	—	—
0	1	0	—	—	0	1
0	0	1	0	0	—	—
0	—	1	—	1	1	0
1	1	0	—	—	1	1
0	—	1	0	—	1	1
0	—	1	0	—	0	0
—	—	0	0	1	0	1
1	—	1	—	0	1	—
0	—	—	0	1	—	0
—	1	0	1	0	—	0
—	0	—	—	0	0	0
1	1	—	—	0	—	0
1	0	—	1	—	0	1
1	0	—	0	1	—	—
—	—	0	0	—	1	0

Соответствующая алгебраическая форма будет выглядеть следующим образом:

$$\varphi(a, b, c, d, e, f, g) = \bar{b}c\bar{g} \vee \bar{a}\bar{b}df\bar{g} \vee acd \vee$$
$$\vee \bar{a}b\bar{c}\bar{f}g \vee \bar{a}\bar{b}cd\bar{e} \vee \bar{a}cef\bar{g} \vee ab\bar{c}fg \vee$$
$$\vee \bar{a}cd\bar{f}g \vee \bar{a}cd\bar{f}\bar{g} \vee \bar{c}de\bar{f}g \vee ac\bar{e}f \vee$$
$$\vee \bar{a}de\bar{g} \vee b\bar{c}d\bar{e}\bar{g} \vee \bar{b}\bar{e}\bar{f}\bar{g} \vee ab\bar{e}\bar{g} \vee a\bar{b}d\bar{f}g \vee$$
$$\vee a\bar{b}de \vee \bar{c}d\bar{f}\bar{g}.$$

§ 3. Минимизация слабо определенных булевых функций

Булевы функции, неполностью и слабо определенные. Если значения булевой функции заданы не на всех элементах пространства M, то она называется *не полностью определенной* булевой функцией. Неполнота задания функции обычно интерпретируется как произвольность ее значений на некоторых наборах. Считается, что если на некотором элементе булева пространства M значение функции не задано, то на данном элементе функции может быть приписано любое значение.

Не полностью определенная булева функция соответствует разбиению пространства M уже не на два, а на три подмножества: M^1, на котором функция принимает значение 1; M^0, на котором функция принимает значение 0; и M^-, на котором функция не определена. Легко видеть, что существует $2^{\sigma(M^-)}$ различных способов доопределения этой функции, при которых получаются различные полностью определенные функции, совпадающие между собой и с исходной функцией на элементах множеств M^1 и M^0. Таким образом, рассматривая не полностью определенную булеву функцию, мы как бы оперируем одновременно с множеством из $2^{\sigma(M^-)}$ указанных функций.

Возможность произвольного доопределения не полностью определенной булевой функции может быть использована при решении задачи минимизации, когда среди $2^{\sigma(M^-)}$ допустимых функций разумно выбрать такую, которой соответствовала бы более простая днф.

Особенно богатые в этом отношении возможности открываются при минимизации *слабо определенных* булевых функций, как мы будем называть такие булевы функции, которые заданы на относительно небольшом числе элементов булева пространства. Рассмотрим реализующий эти возможности метод, позволяющий минимизировать функции с достаточно большим числом аргументов (порядка нескольких десятков), если только число наборов, на которых функция определена, сильно ограничено.

Задача нахождения кратчайшей днф. Будем называть *интервально-поглощаемыми* такие подмножества M_i множества M^1, для каждого из которых может быть найден интервал I_j пространства M, поглощающий M_i и не пересекающийся с M^0, то есть удовлетворяющий условиям

$$M_i \subseteq I_j \quad \text{и} \quad I_j \cap M^0 = \varnothing.$$

Обозначим через M^* множество всех интервально-поглощаемых подмножеств из M^1, а через $\sup M^*$ — *верхнюю границу* множества M^*, то есть совокупность всех таких его элементов, каждый из которых не поглощается никаким другим элементом этого же множества.

Суть рассматриваемого метода заключается в нахождении множества $\sup M^*$ и в решении задачи кратчайшего покрытия матрицы $\|\sup M^* \ni M^1\|$. Найдя это покрытие, мы тем самым выделим некоторое подмножество из $\sup M^*$. Каждый из элементов этого подмножества является интервально-поглощаемым подмножеством из M^1, а это значит, что в пространстве M может быть построен интервал, содержащий все элементы данного подмножества из M^1 и не содержащий ни одного элемента множества M^0. Выполнив подобные построения для всех элементов покрытия, мы получим множество интервалов, содержащих в совокупности все элементы множества M^1 и не пересекающихся с множеством M^0. Очевидно, найденная совокупность интервалов будет соответствовать искомой днф заданной слабо определенной булевой функции, и эта днф будет содержать минимально возможное число конъюнкций, то есть будет кратчайшей.

Рассмотрим в качестве примера процесс минимизации слабо определенной булевой функции $f(x) \equiv f(a, b, c, d,$

e, f, g, h, i, j, k), заданной матрицами

$$\|M^1 \ni X\| = \begin{array}{c} a\,b\,c\,d\,e\,f\,g\,h\,i\,j\,k \\ \left[\begin{array}{l} 10111001001 \\ 00100101011 \\ 01100010010 \\ 01011001101 \\ 10100010100 \\ 00001011011 \\ 01100100100 \\ 01011001111 \\ 10101100010 \\ 00100011100 \end{array}\right] \end{array} \begin{array}{l} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array}$$

$$\|M^0 \ni X\| = \left[\begin{array}{l} 11101101101 \\ 00010010111 \\ 01010110011 \\ 01001010010 \\ 10110101001 \\ 01100011000 \\ 10100101010 \\ 00101100101 \end{array}\right]$$

Нахождение множества $\sup M^*$. Прежде всего, требуется получить множество $\sup M^*$, а точнее, матрицу $\|\sup M^* \ni M^1\|$. Это можно сделать, перебрав каким-то образом подмножества из M^1 и выяснив для каждого из них, является ли оно интервально-поглощаемым. Поскольку множество M^* является выпуклым, перебор всех подмножеств из M^1 необязателен и может быть сокращен с помощью программы перебор, описанной в первой главе. Анализ каждого из рассматриваемых подмножеств сводится к построению соответствующего ему минимального поглощающего интервала и проверке этого интервала на пересечение с множеством M^0.

Минимальным поглощающим интервалом, соответствующим подмножеству M_i булева пространства M, называется такой интервал пространства M, который содержит все элементы подмножества M_i, являясь в то же время минимальным (то есть не существует отличный от него и поглощаемый им интервал, также содержащий все элементы подмножества M_i). Можно сказать, что минимальный поглощающий интервал для подмножества M_i есть пересечение всех интервалов, включающих под-

множество M_i. Найти такой интервал весьма просто. Для этого достаточно сравнить соответствующие компоненты булевых векторов, представляющих элементы множества M_i, и следующим образом определить значения компонент троичного вектора, представляющего искомый интервал: если исходные векторы совпадают по значению данной компоненты, то это же значение присваивается и соответствующей компоненте троичного вектора, в противном случае рассматриваемая компонента получает значение «—».

Например, если множество M_i будет состоять из трех элементов, отмеченных в вышеприведенной матрице $\|M^1 \ni X\|$ номерами 0, 1 и 2, то операция получения минимального поглощающего интервала производится следующим образом:

	1	0	1	1	1	0	0	1	0	0	1
	0	0	1	0	0	1	0	1	0	1	1
	0	1	1	0	0	0	1	0	0	1	0
Результат	—	—	1	—	—	—	—	—	0	—	—

Совершенно аналогично решается задача нахождения минимального поглощающего интервала для системы интервалов. В этом случае результатом служит минимальный среди интервалов, поглощающих все интервалы, входящие в заданную систему. Например,

	—	0	1	1	0	—	1	0	1	—	0
	1	0	—	1	1	0	1	—	1	—	0
	0	0	—	0	—	—	1	0	—	—	0
	1	0	0	1	0	1	1	1	0	1	0
Результат	—	0	—	—	—	—	1	—	—	—	0

Проверка некоторого интервала на пересечение с заданным множеством M^0 также довольно проста: необходимым и достаточным условием такого пересечения является наличие в множестве M^0 элемента, вектор которого совпадает с троичным вектором интервала по всем компонентам, значения которых отличны от «—». Например, полученный выше интервал, представленный вектором

— — 1 — — — — — 0 — —,

пересекается с фигурирующим выше множеством M^0, поскольку в последнем содержится элемент, заданный вектором

$$1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1.$$

Вспомогательные Л-программы. Описанные операции выполняются следующими подпрограммами.

Получение минимального поглощающего интервала— мипин

1. Находится минимальный поглощающий интервал I_i для подмножества M_i булева пространства $M \equiv \bar{X}$, причем $M_i \subseteq M^1 \subset M$.

2. Внешние операнды:

$\alpha_к :: \| M^1 \ni X \|$,

$\beta_п :: \| \{M_i\} \ni M^1 \|$,

$\gamma\delta_п^+ ::$ полученный интервал в коде (01, 11, —0),

$\varepsilon_п :: \Psi(X)$.

Задаются a_α, b_α.

Размерность: $\alpha\gamma\delta\varepsilon$.

Внутренние операнды: $\boldsymbol{a}$, a, —.

4.

$$* \ \mathbf{001\ 401}\ (\beta\boldsymbol{a})$$

$$\S\ 0 \quad \beta \Rightarrow \boldsymbol{a}\, a\, \dot{X}\, 2\, a\, \alpha_a \Rightarrow \gamma \circ \delta$$

$$\S\ 1 \quad \boldsymbol{a}\, \dot{X}\, 2\, a\, \alpha_a \oplus \gamma \vee \delta \Rightarrow \delta \to 1$$

$$\S\ 2 \quad \delta \urcorner \wedge \varepsilon \Rightarrow \delta.$$

5. Пример:

$$\alpha :: \begin{bmatrix} 0\,1\,0\,1\,0\,1\,1\,0\,1\,1 \\ 0\,1\,1\,1\,1\,0\,1\,1\,0\,1 \\ 1\,0\,1\,0\,1\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,0\,1\,0\,1\,1 \\ 0\,1\,1\,0\,1\,1\,0\,1\,0\,1 \end{bmatrix}$$

$$\beta :: [\,0\,1\,0\,1\,1\,0\,0\,0\,0\,0]$$

$$\gamma^+ :: [0\,1\,1\,1\,1\,0\,1\,1\,0\,1]$$

$$\delta^+ :: [1\,0\,0\,0\,1\,0\,0\,0\,0\,1]$$

$$\varepsilon = 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0 \ldots 0$$

Анализ интервала на пересечение с подмножеством из M — анапер

1. Требуется определить, пересекается ли заданный интервал I булева пространства M с некоторым подмножеством M_i из M.

2. Внешние операнды:

α — дополнительный выходной полюс, достигаемый при отсутствии пересечения,

$\beta_к$ :: $\| M_i \supseteq X \|$,

$\gamma\delta_п$ :: интервал I в коде (01, 11, —0).

Задаются: a_α, b_β.
Размерность: $\beta\gamma\delta$.
Внутренние операнды: —, a, —.

4.

* 001 402

§ 0 $\bar{\supset} a$

§ 1 $\Delta a \oplus b_\beta \circ\!\!\rightarrow \alpha\beta_a \oplus \gamma \wedge \delta \mapsto 1$.

Л-программа получения матрицы Квайна. Приведем полную программу получения матрицы $\|\sup M^* \supseteq M^1\|$, которую можно назвать *матрицей Квайна*, поскольку задача нахождения оптимальных покрытий булевых матриц была впервые исследована Квайном. В эту программу, кроме только что рассмотренных Л-операторов мипин и анапер, входят также подпрограммы перебор и включмак, описанные в первой главе.

Получение матрицы Квайна для слабо определенной булевой функции — квасобу

1. Требуется найти множество $\sup M^*$, являющееся верхней границей множества M^* всех интервально-поглощаемых подмножеств из M^1 для булевой функции, заданной подмножествами M^1 и M^0 булева пространства $M \equiv \bar{X}$, на которых она принимает значения 1 и 0, соответственно. Результат нужно представить в форме булевой матрицы $\|\sup M^* \supseteq M^1\|$.

2. Внешние операнды:

α — дополнительный выходной полюс, достигаемый при нехватке памяти, отводимой для представления результата,

$\beta_к :: \| M^1 \ni X \|$,

$\gamma_к :: \| M^0 \ni X \|$,

$\delta_к^+ :: \| \sup M^* \ni M^1 \|$,

$\varepsilon_п :: \Psi(X)$.

Задаются: a_β, a_γ, a_δ, b_β, b_γ, b_δ, причем значением b_δ ограничивается допустимый размер получаемой матрицы.

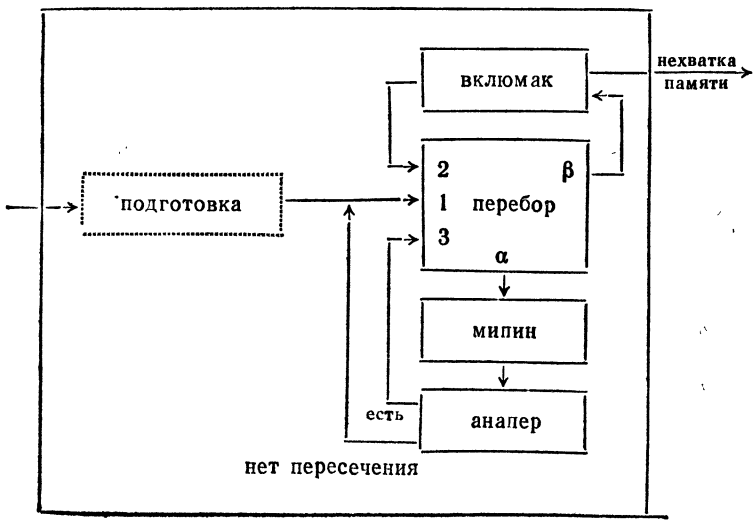
Размерность: $\beta\gamma$.

Подпрограммы: м и п и н, а н а п е р, в к л ю м а к, п е р е б о р.

Внутренние операнды: d, c, —.

Примечание: мощность матрицы $\|\sup M^* \ni M^1\|$ быстро растет с увеличением $\sigma(M^1)$, поэтому описываемая программа практически пригодна лишь для слабо определенных булевых функций.

3. Динамику функционирования данной программы можно отобразить следующей блок-схемой, роль дирижера в которой играет Л-оператор п е р е б о р:



4.

* **001 404**

§ 0 $\circ \boldsymbol{d}\, b_\delta \Rightarrow b \circ b_\delta\, b_\beta \Rightarrow c \rightarrow 4$

§ 1 * мипин $\beta\, \boldsymbol{d}\, \boldsymbol{b}\, \boldsymbol{c}\, \varepsilon$ // * анапер $4\, \gamma\, \boldsymbol{bc}$ // $\rightarrow$ * перебор 3

§ 2 * включак $3\, \delta\, \boldsymbol{d}\, b$ // $\rightarrow$ * перебор 2

§ 3 $\rightarrow \alpha$

§ 4 * перебор $1\, 2\, \boldsymbol{d}\, c$ //.

5. Продолжим рассмотрение приведенного в начале данного параграфа примера слабо определенной булевой функции, у которой $\sigma(X) = 11$, $\sigma(M^1) = 10$ и $\sigma(M^0) = 8$. Полученное подпрограммой квасобу и представленное конечным значением комплекса δ решение будет иметь следующий вид:

$$\| \sup M^* \ni M^1 \| :: \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix} \\ \left[\begin{array}{cccccccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Дальнейший путь решения задачи. Кратчайшее покрытие матрицы $\| \sup M^* \ni M^1 \|$ может быть найдено с помощью программы окрапок4, а приближение к нему — с помощью программы прикрапок1.

Например, применение программ окрапок4 приводит в данном случае к следующему результату:

$$\| U \ni M^1 \| = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix} \\ \left[\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}\right] \end{array},$$

где через U обозначено полученное решение — совокупность интервально-поглощаемых подмножеств из M^1, являющаяся кратчайшим покрытием множества M^1.

Переход от этого покрытия к искомой д н ф сводится к построению минимальных поглощающих интервалов для каждого из элементов покрытия и к возможно большему их расширению на области $M \setminus M^0$, которому соответствует выбрасывание некоторых букв из конъюнкций, представляющих данные интервалы. Например, первым элементом полученного покрытия служит подмножество из M^1, содержащее элементы с номерами 0, 5 и 9. Получив минимальный поглощающий интервал для этого подмножества:

a	b	c	d	e	f	g	h	i	j	k
1	0	1	1	1	0	0	1	0	0	1
0	0	0	0	1	0	1	1	0	1	1
0	0	1	0	0	0	1	1	1	0	0
—	0	—	—	—	0	—	1	—	—	—

соответствующий конъюнкции $\bar{b}\bar{f}h$, мы убеждаемся в невозможности его расширения: ни одна из букв конъюнкции $\bar{b}\bar{f}h$ не может быть удалена без того, чтобы интервал не вышел из области $M \setminus M^0$, то есть не стал пересекаться с множеством M^0. Следовательно, конъюнкция $\bar{b}\bar{f}h$ включается в искомую д н ф.

Аналогично получаются два остальных интервала, представляемые следующими троичными векторами:

$$0 — — — — — — 1 — — 1,$$
$$— — 1\ 0 — — — 0 — — 0$$

и соответствующие конъюнкциям $\bar{a}hk$ и $c\bar{d}\bar{h}\bar{k}$. В последней из этих конъюнкций может быть выброшена буква $\bar{d}$, в результате чего окончательное решение окажется представленным матрицей

$$\begin{bmatrix} — & 0 & — & — & 0 & — & 1 & — & — & — \\ 0 & — & — & — & — & — & — & 1 & — & 1 \\ — & — & 1 & — & — & — & — & 0 & — & 0 \end{bmatrix}$$

которой соответствует д н ф $\bar{b}\bar{f}h \vee \bar{a}hk \vee c\bar{h}\bar{k}$.

Л-программы. Описанный переход от кратчайшего покрытия булевой матрицы $\|\sup M^* \supseteq M^1\|$ к крат-

чайшей днф заданной булевой функции реализуется следующей программой.

Получение днф булевой функции по заданному покрытию — диноф

1. Для булевой функции $f(\boldsymbol{x})$, заданной множествами M^1 и M^0 и, в общем случае, неполностью определенной, требуется получить дизъюнктивную нормальную форму, соответствующую покрытию множества M^1 заданной совокупностью M^* подмножеств из M^1, относительно которых известно, что они являются интервально-поглощаемыми, то есть для каждого из них можно найти интервал пространства M, поглощающий данное подмножество и не пересекающийся с M^0.

2. Внешние операнды:

$\alpha_к :: \| M^* \ni M^1 \|$,

$\beta_к :: \| M^1 \ni X \|$,

$\gamma_к :: \| M^0 \ni X \|$,

$\delta_п :: \Psi(X)$,

$\varepsilon\zeta_к^+ ::$ полученная днф в коде $(10, 01, 00)$.

Задаются: a_α, a_β, a_γ, a_ε, a_ζ, b_α, b_β, b_γ.

Размерность: $\beta\gamma\delta\zeta$.

Подпрограммы: мипин, слурин.

Внутренние операнды: $\boldsymbol{c}$, c, —.

Примечание: $\sigma(\varepsilon) = \sigma(\alpha)$, $\sigma(\zeta) = \sigma(\alpha)$. Данная программа стремится минимизировать число букв в каждой конъюнкции, однако получение строго минимального в этом смысле решения не гарантируется.

4.

$*$ **001 411** $(\delta \boldsymbol{bc})$

§ 0 $\circ\, c$

§ 1 $*$ мипин $\beta(\alpha_c)\, \boldsymbol{b}\, \boldsymbol{c}\, \delta$ // $*$ слурин $\gamma\, \boldsymbol{b}\, \boldsymbol{c}$ //

$\boldsymbol{b} \wedge \boldsymbol{c} \Rightarrow \zeta_c \oplus \boldsymbol{c} \Rightarrow \varepsilon_c \triangle c \oplus b_\alpha \mapsto 1 b_\alpha \Rightarrow b_\varepsilon \Rightarrow b_\zeta$.

Кроме уже знакомой нам подпрограммы мипин, в рассмотренной выше программе диноф используется следующая подпрограмма.

Случайное расширение интервала — слурин

1. Интервал I булева пространства $M \equiv \bar{X}$ требуется расширить (случайным образом) до максимального в множестве $M \setminus M^0$. Предполагается, что $I \cap M^0 = \varnothing$.

2. Внешние операнды:

$\alpha_к$ :: $\| M^0 \ni X \|$,

$\beta\gamma_п^-$ :: исходный интервал в коде (01, 11, —0),

$\beta\gamma_п^+$ :: расширенный интервал в коде (01, 11, —0).

Задаются: a_α, b_α.
Размерность: $\alpha\beta\gamma$.
Подпрограммы: анапер.
Внутренние операнды: a, b, —.

4.

```
        * 001 122 (αa)
§ 0     γ⇒a
§ 1     aẌ2bγ⊕c_b⇒γ
        *анапер1αβγ//γ∨c_b⇒γ→1
§ 2     .
```

Программы нахождения кратчайших днф. Рассмотренные в данном параграфе алгоритмы минимизации слабо определенных булевых функций реализуются следующими программами.

Получение кратчайшей днф слабо определенной булевой функции — крафсо

1. Слабо определенная булева функция $f(x)$ задана подмножествами M^1 и M^0 булева пространства $M \equiv \bar{X}$, на которых она принимает значения 1 и 0, соответственно. Требуется найти одну из кратчайших днф этой функции.

2. Внешние операнды:

α — дополнительный выходной полюс, достигаемый в случае нехватки памяти,

$\beta_к$ :: $\| M^1 \ni X \|$,

$\gamma_к$ :: $\| M^0 \ni X \|$,

$\delta_п$ :: $\Psi(X)$,

$\varepsilon\zeta_к^+$ :: полученная днф в коде (10, 01, 00),

$\eta_к$ — комплекс памяти.

Задаются: a_β, a_γ, a_ε, a_ζ, a_η, b_β, b_γ, b_η.
Размерность: $\beta\gamma\delta\varepsilon\zeta$.
Подпрограммы: квасобу, окрапок4, диноф.
Внутренние операнды: e, i, X.

Примечание: необходимо выполнение условия $\sigma(M^1) \leqslant 32$.

4.

```
        * 001 410
§ 0   a_η + b_β ⇒ a_26 b_η — b_β ⇒ b_26 ⇒ i
        * квасобу 1 β γ X δ // b_β — 1 ⇒ b 0 — c_b ⇒ e
        * окрапок4 1 X i e η // * диноф η β γ δ ε ζ // →2
§ 1   →α
§ 2   .
```

Аналогично строится программа приближенного решения поставленной задачи, называемая прикрафсо и отличающаяся от программы крафсо только тем, что вместо подпрограммы окрапок4 в ней используется подпрограмма прикрапок1, находящая лишь приближение к кратчайшему покрытию.

Получение приближения к кратчайшей днф слабо определенной булевой функции — прикрафсо

1. Слабо определенная булева функция $f(x)$ задана подмножествами M^1 и M^0 булева пространства $M \equiv \bar{X}$, на которых она принимает значения 1 и 0, соответственно. Требуется найти приближение к кратчайшей днф этой функции.

2. Внешние операнды те же, что и в крафсо.

Подпрограммы: квасобу, прикрапок1, диноф.

Внутренние операнды: e, g, Z.

Примечание: необходимо выполнение условия $\sigma(M^1) \leqslant 32$.

4.

```
        * 001 415
§ 0   a_η + b_β ⇒ a_30 b_η — b_β ⇒ b_30
        * квасобу1 β γ Z δ // b_β — 1 ⇒ b 0 — c_b ⇒ e
        * прикрапок1 Z e η // * диноф η β γ δ ε ζ // →2
§ 1   →α
§ 2   .
```

§ 4. Метод конкурирующих интервалов

Постановка задачи. Изложенные выше алгоритмы минимизации слабо определенных булевых функций теряют свою эффективность и становятся практически нереализуемыми, когда число элементов в каж-

дом из множеств M^1 и M^0 достигает порядка сотен. В данном параграфе описывается алгоритм, разработанный специально для этого случая. Будучи некритичен к объему используемой памяти, он обладает довольно высоким быстродействием, в чем можно убедиться, ознакомившись с приведенными в следующем параграфе результатами статистических испытаний. Качество получаемых решений вполне удовлетворительно с практической точки зрения.

Итак, рассмотрим задачу получения кратчайшей (или близкой к ней) дизъюнктивной нормальной формы слабо определенной булевой функции, у которой число переменных $\sigma(X)$ может достигать порядка десятков, а число $\sigma(M^1)$ единичных значений функции и число $\sigma(M^0)$ нулевых значений функции становятся равными нескольким сотням каждое.

Как уже было показано, решение поставленной задачи сводится к нахождению некоторой системы интервалов булева пространства $M \equiv X$, не пересекающихся с множеством M^0 и содержащих в совокупности все элементы множества M^1. При этом считается, что качество полученной системы тем выше, чем она проще, то есть чем меньше она содержит интервалов и чем ниже ранги этих интервалов.

Метод решения. Существенной особенностью предлагаемого алгоритма является *параллельное* построение искомой системы интервалов, с постепенным расширением отдельных интервалов (понижением их рангов). Алгоритм реализует итеративный процесс, на i-м шаге которого анализируется и улучшается *частичное решение*, определяемое в данной задаче как некоторая совокупность U_i интервалов пространства M, не пересекающихся с множеством M^0 и содержащих *некоторые* из элементов множества M^1.

Введем оператор $*$ бинарного отношения совместимости между элементом m булева пространства M и интервалом I этого же пространства M для булевой функции, заданной множествами M^1 и M^0. Будем считать, что отношение $m * I$ имеет место тогда и только тогда, когда в пространстве M существует такой интервал, который содержит элемент m, поглощает интервал I и не пересекается с множеством M^0.

В алгоритме используется булева матрица $\|M_i^1 * U_i\|$ данного бинарного отношения, где U_i — очередное частичное решение, а M_i^1 — не покрытый им остаток множества M^1, то есть совокупность всех таких элементов множества M^1, ни один из которых не принадлежит ни одному из элементов данного частичного решения U_i.

На каждом шаге итерации имеющееся частичное решение может быть преобразовано или путем добавления некоторого нового интервала, или путем расширения одного из уже имеющихся интервалов, чему соответствует удаление некоторых букв из представляющей данный интервал конъюнкции. Алгоритм должен выбрать такую последовательность указанных преобразований частичного решения, которая приводила бы, в конечном счете, к полному решению (полностью покрывающему множество M^1), являющемуся в то же время хорошим приближением к кратчайшей д н ф. Делается это следующим образом.

Во-первых, если матрица $\|M_i^1 * U_i\|$, соответствующая очередному частичному решению U_i, содержит строку без единиц, то это означает, что ни один из уже входящих в решение интервалов не может быть расширен так, чтобы он смог покрыть соответствующий данной строке элемент из множества M_i^1, который мы обозначим через m_k. В этом случае необходимо ввести новый интервал. Поскольку в текущий момент времени неизвестно, какие другие из элементов множества M_i^1 целесообразно покрыть данным интервалом, он строится только из одного элемента m_k, что позволяет зарезервировать все возможности его расширения на будущее. Полученный интервал вводится в множество U_i, а покрываемый им элемент m_k удаляется из M_i^1. Соответственно преобразуется и матрица $\|M_i^1 * U_i\|$: из нее выбрасывается одна строка и добавляется один столбец, который представляет свойства вновь введенного интервала.

Во-вторых, если в матрице $\|M_i^1 * U_i\|$ имеется столбец, не содержащий ни одной единицы, то это значит, что соответствующий этому столбцу интервал уже не может быть расширен так, чтобы покрыть какой-либо

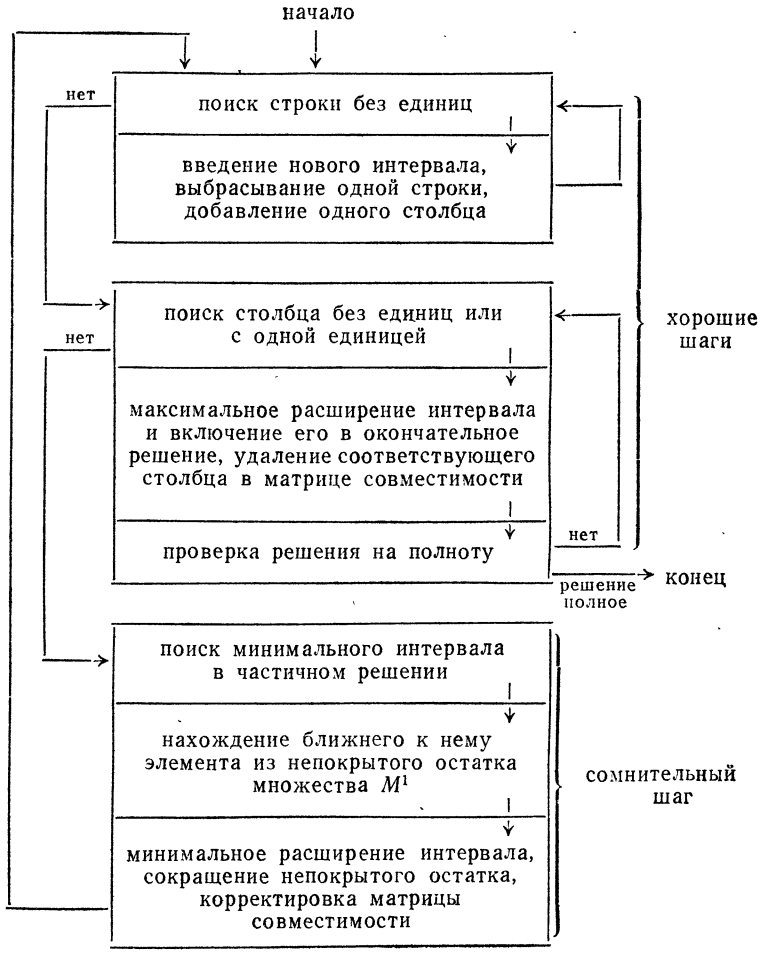
из элементов множества M_i^1. Поэтому его можно включить в окончательное решение, предварительно максимально расширив его на множестве $M \setminus M^0$ (с тем, чтобы минимизировать число букв в конъюнкции, представляющей этот интервал), и удалить упомянутый столбец из матрицы $\|M_i^1 * U_i\|$. Если же в этой матрице имеется столбец, содержащий ровно одну единицу, то реализуется именно тот вариант расширения интервала, соответствующего данному столбцу, при котором покрывается отмеченный единицей элемент множества M_i^1. Очевидно, что это единственный способ использования рассматриваемого интервала для покрытия элементов множества M_i^1. Полученный интервал, расширенный на множестве $M \setminus M^0$, также включается в окончательное решение, а соответствующий столбец матрицы $\|M_i^1 * U_i\|$ вычеркивается.

В-третьих, если в матрице $\|M_i^1 * U_i\|$ отсутствуют строки и столбцы с указанными свойствами, то в частичном решении U_i отыскивается некоторый минимальный интервал (то есть такой, который представляется конъюнкцией с максимальным числом букв и состоит поэтому из минимального числа элементов пространства M). Затем среди элементов множества M_i^1, совместимых с данным интервалом, находится ближний к нему. Напомним, что расстояние между троичными векторами мы измеряем числом одноименных компонент, в которых один из векторов имеет значение 0, а другой — значение 1. После этого производится минимальное расширение выбранного интервала, обеспечивающее покрытие указанного элемента, в результате чего данный интервал заменяется поглощающим его минимальным интервалом, покрывающим также упомянутый ближний элемент. Описанный процесс сопровождается удалением из множества M_i^1 всех элементов, покрываемых новым интервалом, и корректировкой столбца матрицы $\|M_i^1 * U_i\|$, соответствующего преобразованному интервалу — некоторые из единиц в этом столбце заменяются нулями (очевидно, что обратная замена невозможна).

Этот алгоритм весьма напоминает рассмотренный ранее метод прямого размещения отдыхающих по лодкам.

Так же, как и тот, он реализует некоторую последовательность из «хороших» и «сомнительных» шагов. Легко видеть, что в данном случае к «хорошим» шагам относятся действия, выполняемые при нахождении в матрице $\|M_i^1 * U_i\|$ строки без единиц или столбца, содержащего не более одной единицы.

Схема алгоритма. Чередование описанных операций производится согласно следующей схеме.



Полагается, что в начале алгоритма частичное решение представляется пустым множеством, в соответствии с чем число столбцов исходной матрицы совместимости равно нулю. На первом же шаге начинается построение первого интервала — для начала он будет содержать лишь первый из элементов множества M^1. Матрица совместимости становится одностолбцовой. Дальнейший процесс уже зависит от значения этого столбца.

При поиске объектов с экстремальными значениями некоторых параметров (например, при поиске минимальных интервалов, ближних элементов и т. д.) может быть обнаружено несколько «лучших» в данном смысле объектов. В порядке уточнения алгоритма заметим, что в этом случае из них всегда будет выбираться первый встреченный. Очевидно, что производимый выбор будет зависеть, таким образом, от порядка перебора объектов. Будем считать, что в приводимом ниже примере этот порядок совпадает с порядком перечисления объектов. Заметим также, что, несмотря на несущественность сделанного уточнения, оно необходимо для полной определенности алгоритма.

Работа алгоритма завершается, когда множество M^1 окажется полностью покрытым совокупностью построенных интервалов и когда все построенные интервалы будут включены в окончательное решение.

Пример. Пусть

$$\|M^1 \ni X\| = \begin{matrix} \\ 0\,0\,1\,1\,0\,0 \\ 0\,1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1\,0 \\ 1\,0\,1\,0\,0\,0 \\ 1\,1\,0\,0\,0\,1 \\ 1\,1\,0\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1 \\ 1\,1\,0\,1\,0\,0 \\ 0\,1\,1\,1\,1\,1 \end{matrix} \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \qquad \|M^0 \ni X\| = \begin{matrix} 1\,0\,0\,0\,0\,1 \\ 1\,0\,1\,1\,1\,1 \\ 0\,1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,1\,1\,0 \\ 0\,1\,1\,0\,0\,1 \\ 0\,1\,0\,1\,0\,1 \end{matrix}$$

На первом шаге алгоритма строится интервал 0 0 1 1 0 0, содержащий лишь один элемент, а именно, 1-й. Обозначим этот интервал через А. Он оказывается несовместимым со следующим, 2-м элементом множества M^1, так как интервал 0 — 1 — — —, являющийся минимальным покрывающим интервалом для элементов 1 и 2, пе-

ресекается с множеством M^0, имея общие элементы 011000 и 011001. Поэтому элемент 2 используется для построения следующего интервала Б. Среди остальных элементов множества M^1 лишь один оказывается несовместимым с каждым из построенных интервалов — это элемент 5, с которого начинается построение следующего интервала В. Три описанных шага алгоритма запишем в виде следующей таблицы реализации алгоритма, в каждой строке которой представлен номер очередного шага, символическая форма выполняемой операции и код получаемого интервала:

1. А = 1 001100
2. Б = 2 011011
3. В = 5 110001

Матрица совместимости принимает к этому времени следующий вид:

	А	Б	В
3	0	1	1
4	1	0	0
6	0	1	1
7	0	1	0
8	1	0	1
9	1	1	0

Поскольку в ней отсутствуют строки, не содержащие единиц, а также столбцы, содержащие менее двух единиц, подготавливается расширение одного из уже имеющихся в частичном решении интервалов. Для этого отыскивается минимальный из них (поскольку на данном этапе ранги всех трех интервалов равны, выбирается первый из них, интервал А), затем находится ближний к нему элемент непокрытого остатка {3, 4, 6, 7, 8, 9} множества M^1 (таким элементом оказывается элемент 4). Минимальное расширение интервала А, необходимое для покрытия элемента 4, приводит к смене значения этого интервала с 001100 на —01—00. Символически данное действие представляется следующей строкой таблицы реализации алгоритма:

4. $4 \Rightarrow A$ —01—00

После этого интервал А становится уже несовместимым с элементами 8 и 9, благодаря чему в матрице совместимости появляется столбец, в котором нет единиц:

	А	Б	В
3	0	1	1
6	0	1	1
7	0	1	0
8	0	0	1
9	0	1	0

Использование интервала А для покрытия остающихся элементов множества M^1 становится невозможным, поэтому он максимально расширяется на множестве $M \setminus M^0$ и включается в окончательное решение. Этот шаг записывается в следующей форме:

5. А $-0\,1-00$

$-0--00$ в решение

Будучи продолженной аналогичным образом, таблица реализации алгоритма принимает следующий окончательный вид:

1.	А=1	0 0 1 1 0 0	
2.	Б=2	0 1 1 0 1 1	
3.	В=5	1 1 0 0 0 1	
4.	4⇒А	− 0 1 − 0 0	
5.	А	− 0 1 − 0 0	
		− 0 − − 0 0	в решение
6.	9⇒Б	0 1 1 − 1 1	
7.	Г=7	1 0 0 0 1 1	
8.	6⇒В	1 1 0 − − 1	
9.	3⇒Б	− 1 1 − 1 −	
		− 1 − − 1 −	в решение
10.	Г	1 0 0 0 1 1	
		− − − 0 1 1	в решение
11.	8⇒В	1 1 0 − − −	
		1 1 − − − −	в решение

Полученное решение содержит четыре интервала и в случае, например, когда $X = \{a, b, c, d, e, f\}$, соответствует дизъюнктивной нормальной форме

$$\bar{b}\,\bar{e}\,\bar{f} \vee b\,e \vee \bar{d}\,e\,f \vee a\,b.$$

Уточнения алгоритма. Внесем в алгоритм некоторые уточнения, облегчающие его машинную реализацию.

Совокупность элементов булева пространства M, образующая множество M_i^1, задается некоторым комплексом, каждый элемент которого представляет один из элементов множества M_i^1. В процессе вычислений множество M_i^1 сокращается путем выбрасывания элементов, покрываемых расширяемыми и вновь вводимыми интервалами частичного решения. Положим, что каждое элементарное сокращение множества M_i^1, заключающееся в выбрасывании одного элемента, сопровождается перестановкой последнего элемента множества на место выбрасываемого элемента и присвоением ему номера удаляемого элемента. Нумерация остальных элементов множества M_i^1 при этом не меняется. Данная операция легко осуществляется передачей значения последнего элемента упомянутого комплекса тому из его элементов, который представляет выбрасываемый элемент множества M_i^1, и сокращением мощности комплекса на единицу.

Аналогично производится сокращение матрицы совместимости — ее строки переставляются синхронно с элементами множества M_i^1.

Столбцы матрицы совместимости представляются соответствующими разрядами булевых векторов — элементов некоторых комплексов. Эти разряды заполняются слева направо, по мере введения в матрицу новых столбцов, и освобождаются при выбрасывании строк из матрицы.

Пример для уточненного алгоритма. Рассмотрим более сложный пример, решенный описанным уточненным алгоритмом. Пусть $\sigma(X) = 11$, $\sigma(M^1) = 48$ и $\sigma(M^0) = 16$, матрица $\|M^1 \ni X\|$ имеет значение

1	1	1	1	0	0	1	0	1	0	1	1	1
1	1	0	0	0	1	1	0	1	0	1		2
0	1	1	1	1	1	1	1	1	1	1	1	3
1	0	1	1	1	0	0	1	0	0	1		4
0	0	0	1	1	0	0	1	1	0	0		5
1	1	0	0	0	0	1	1	1	1	1		6
0	1	1	0	1	1	0	1	0	1	0		7
0	1	0	0	0	0	1	0	1	0	1		8
0	1	1	0	1	1	1	1	1	0	1		9
0	0	0	1	0	0	0	0	0	1	1		10
0	0	0	0	1	0	1	1	0	1	1		11
1	1	0	0	0	0	0	0	0	1	0		12
1	0	1	0	0	1	0	0	0	0	1		13
1	1	1	0	1	0	0	1	0	1	0		14
0	1	0	1	1	0	1	0	1	0	0		15
1	0	1	0	1	1	1	0	0	0	1		16
1	1	0	1	0	0	1	0	0	1	0		17
1	0	0	1	0	0	1	1	0	1	0		18
0	0	1	1	1	0	1	0	0	1	1		19
0	0	0	1	0	1	0	0	1	0	0		20
0	1	0	1	1	0	0	1	1	1	0		21
1	1	1	0	1	0	0	0	0	1	1		22
0	0	1	0	1	0	0	0	0	1	1		23
0	0	1	1	0	1	1	0	0	1	1		24
0	0	1	1	0	0	1	1	1	0	0		25
0	1	1	0	0	1	1	1	0	1	0		26
1	1	0	0	1	1	0	1	1	0	1		27
1	1	1	1	1	1	1	0	0	1	1		28
0	1	1	1	1	0	0	0	0	1	0		29
1	1	1	0	0	1	1	0	0	0	1		30
1	1	0	0	0	0	1	1	1	0	0		31
0	0	1	1	1	0	1	0	1	0	0		32
1	0	0	1	1	1	1	0	0	1	0		33
0	1	0	0	1	0	1	1	1	1	0		34
1	1	1	0	1	0	1	1	0	0	0		35
0	1	0	0	1	1	0	0	1	1	1		36
1	1	1	0	0	0	0	1	0	1	1		37
0	1	1	1	1	1	1	0	0	0	0		38
1	1	0	1	0	0	1	0	0	0	1		39
0	0	1	1	0	0	1	0	0	0	0		40
0	1	0	0	1	0	1	1	0	1	0		41
1	0	1	0	0	1	1	0	0	1	1		42
0	0	0	0	0	0	1	1	1	1	0		43
0	0	0	1	1	1	0	1	0	0	1		44
1	1	1	0	0	1	1	1	1	0	1		45
0	0	0	1	1	1	1	0	0	0	0		46
0	0	0	0	1	1	0	0	1	0	0		47
1	1	0	0	1	0	1	1	0	1	1		48

и матрица $\|M^0 \ni X\|$ — значение

$$\begin{bmatrix}
0&0&0&0&0&1&1&1&0&0&1\\
1&0&1&1&1&1&1&1&1&0&0\\
1&1&1&1&1&1&0&0&0&1&1\\
1&0&1&0&0&1&1&0&0&1&0\\
0&1&0&1&0&1&0&0&1&1&1\\
1&1&1&0&0&1&0&1&0&1&1\\
1&0&1&1&1&1&1&1&1&0&1\\
1&1&1&1&1&0&0&1&1&0&1\\
1&0&1&1&0&0&0&1&1&1&1\\
0&1&0&0&0&0&0&1&1&1&0\\
1&0&1&1&1&0&1&1&0&1&1\\
0&0&0&0&0&0&1&0&1&1&0\\
1&1&1&0&1&0&0&0&1&1&1\\
1&0&0&1&1&1&0&0&1&1&0\\
1&1&0&0&0&0&1&0&1&0&1\\
0&0&1&1&1&1&0&1&0&1&1
\end{bmatrix}$$

(эти матрицы получены с помощью генератора случайных значений булева вектора). Процесс минимизации данной слабо определенной булевой функции представляется следующей таблицей реализации описанного алгоритма:

1.		А = 1	1	1	1	1	0	0	1	0	1	0	1
2.		Б = 48	1	1	0	0	1	0	1	1	0	1	1
3.		В = 2	1	1	0	0	0	1	1	0	1	0	1
4.		Г = 4	1	0	1	1	1	0	0	1	0	0	1
5.		Д = 8	0	1	0	0	0	0	1	0	1	0	1
6.	39⇒А		1	1	–	1	0	0	1	0	–	0	1
7.	6⇒Б		1	1	0	0	–	0	1	1	–	1	1
8.		Е = 7	0	1	1	0	1	1	0	1	0	1	0
9.	45⇒В		1	1	–	0	0	1	1	–	1	0	1
10.	44⇒Г		–	0	–	1	1	–	0	1	0	0	1
11.	15⇒Д		0	1	0	–	–	0	1	0	1	0	–
12.	14⇒Е		–	1	1	0	1	–	0	1	0	1	0
13.	17⇒А		1	1	–	1	0	0	1	0	–	–	–
14.	41⇒Б		–	1	0	0	–	0	1	1	–	1	–
15.	34⇒Б		–	1	0	0	–	0	1	1	–	1	–

16.		Ж = 37	1	1	1	0	0	0	0	1	0	1	1	
17.	22 ⇒	Ж	1	1	1	0	–	0	0	–	0	1	1	
18.	30 ⇒	В	1	1	–	0	0	1	1	–	–	0	1	
19.		З = 36	0	1	0	0	1	1	0	0	1	1	1	
20.	47 ⇒	З	0	–	0	0	1	1	0	0	1	–	–	
21.	29 ⇒	Е	–	1	1	–	1	–	0	–	0	1	0	
22.	12 ⇒	Ж	1	1	–	0	–	0	0	–	0	1	–	
23.	42 ⇒	В	1	–	–	0	0	1	1	–	–	–	1	
24.	16 ⇒	В	1	–	–	0	–	1	1	–	–	–	1	в решение
25.	5 ⇒	Г	–	0	–	1	1	–	0	1	–	0	–	
26.	32 ⇒	Д	0	–	–	–	–	0	1	0	1	0	–	
27.	27 ⇒	З	–	–	0	0	1	1	0	–	1	–	–	
28.	28 ⇒	А	1	1	–	1	–	–	1	0	–	–	–	
29.	43 ⇒	Б	–	–	0	0	–	0	1	1	–	1	–	
30.	11 ⇒	Б	–	–	0	0	–	0	1	1	–	1	–	
31.	9 ⇒	З	–	–	–	0	1	1	–	–	1	–	–	
			–	–	–	0	1	1	–	–	–	–	–	в решение
32.	21 ⇒	Е	–	1	–	–	1	–	0	–	–	1	0	
33.	35 ⇒	Ж	1	1	–	0	–	0	–	–	0	–	–	
34.		И = 10	0	0	0	1	0	0	0	0	0	1	1	
35.	38 ⇒	Е	–	1	–	–	1	–	–	–	–	–	0	в решение
36.	23 ⇒	Ж	–	–	–	0	–	0	–	–	0	–	–	в решение
37.	24 ⇒	И	0	0	–	1	0	–	–	0	0	1	1	
38.	19 ⇒	И	0	0	–	1	–	–	–	0	0	1	1	
39.	46 ⇒	И	0	0	–	1	–	–	–	0	0	–	–	
40.	40 ⇒	И	0	0	–	1	–	–	–	0	0	–	–	
41.	31 ⇒	Б	–	–	0	0	–	0	1	1	–	–	–	
42.		К = 26	0	1	1	0	0	1	1	1	0	1	0	
43.	18 ⇒	Б	–	–	0	–	–	0	1	1	–	–	–	в решение
44.	3 ⇒	К	0	1	1	–	–	1	1	–	–	1	–	
45.	33 ⇒	А	1	–	–	1	–	–	1	0	–	–	–	
			–	–	–	1	–	–	1	0	–	–	–	в решение
46.	20 ⇒	И	0	0	–	1	–	–	–	0	–	–	–	в решение
47.	13 ⇒	Г	–	0	–	–	–	–	0	–	–	0	–	в решение
48.	25 ⇒	Д	0	–	–	–	–	0	1	–	1	0	–	
			0	–	–	–	–	–	–	–	1	0	–	в решение
49.		К	0	1	1	–	–	1	1	–	–	1	–	
			0	1	1	–	–	–	–	–	–	–	–	в решение

Окончательный результат представляется матрицей

$$\begin{bmatrix}
1 & - & - & 0 & - & 1 & 1 & - & - & - & 1 \\
- & - & - & 0 & 1 & 1 & - & - & - & - & - \\
- & 1 & - & - & 1 & - & - & - & - & - & 0 \\
- & - & - & 0 & - & 0 & - & - & 0 & - & - \\
- & - & 0 & - & - & 0 & 1 & 1 & - & - & - \\
- & - & - & 1 & - & - & 1 & 0 & - & - & - \\
0 & 0 & - & 1 & - & - & - & 0 & - & - & - \\
- & 0 & - & - & - & - & 0 & - & - & 0 & - \\
0 & - & - & - & - & - & - & - & 1 & 0 & - \\
0 & 1 & 1 & - & - & - & - & - & - & - & -
\end{bmatrix}$$

и, если $X=\{a, b, c, d, e, f, g, h, i, j, k\}$, соответствует днф

$$a\bar{d}fgk \vee \bar{d}ef \vee be\bar{k} \vee \bar{d}\bar{f}\bar{i} \vee \bar{c}\bar{f}gh \vee dg\bar{h} \vee \bar{a}\bar{b}d\bar{h} \vee$$
$$\vee \bar{b}\bar{g}\bar{j} \vee \bar{a}i\bar{j} \vee \bar{a}bc.$$

Л-программа. Описанный алгоритм минимизации слабо определенной булевой функции, заданной подмножествами M^1 и M^0 булева пространства $M \equiv \bar{X}$, реализуется следующей подпрограммой, выдающей решение в виде некоторого множества U интервалов пространства M.

Минимизация слабо определенной булевой функции — минсоб

1. Постановка задачи дана выше.

2. Внешние операнды:

$\alpha_к^- :: \| M^1 \ni X \|$,

$\beta_к :: \| M^0 \ni X \|$,

$\gamma_п :: \Psi(X)$,

$\delta\varepsilon_к^+ ::$ полученная днф в коде (10, 01, 00),

$\zeta_к$ — память, $\sigma(\zeta) = \sigma(\alpha) + 100_8$.

Задаются: a_α, a_β, a_δ, a_ε, a_ζ, b_α, b_β.

Подпрограммы: очистка, новички, элер, выбор, ассимил.

Внутренние операнды: f, g, $\boldsymbol{Y}$.

3. Реализуемый алгоритм описан выше.

Комплекс памяти ζ разбивается на три комплекса: собственно комплекс ζ представляет матрицу совместимости $\|M_i^1 * U_i\|$, а выделяемые из него комплексы $\boldsymbol{Y}$

и Z представляют текущее частичное решение U_i. Матрица $\|M_i^! * U_i\|$ может размещаться в произвольной совокупности разрядов элементов комплекса ζ, выделяемых единичными значениями компонент булева вектора $\Psi(U_i)$. Например, если вектор $\Psi(U_i)$ имеет значение 10010011 ..., то это значит, что левый столбец матрицы представлен нулевыми компонентами элементов комплекса, а остальные — компонентами с номерами 3, 6, 7 и т. д., в порядке их следования. Этот же вектор выделяет совокупность элементов комплексов Y и Z, используемых для представления элементов текущего частичного решения U_i — интервалов булева пространства M, представляемых в коде (01, 11, —0).

Поиск в матрице совместимости $\|M_i^! * U_i\|$ строк без единиц и соответствующие преобразования частичного решения в случае нахождения таких строк производятся подпрограммой новички. Вместе с частичным решением U_i изменению подвергаются значения величин $M_i^!$, $\|M_i^! * U_i\|$ и $\Psi(U_i)$ (инверсное значение булева вектора $\Psi(U_i)$ задается переменной d).

Подпрограмма элер ищет в матрице совместимости столбец без единиц или с одной единицей и, в случае нахождения такого столбца, также преобразует частичное решение и зависящие от него величины. При этом формируется очередной элемент полного решения — интервал пространства M, представляемый в коде (01, 11, —0) переменными e и f. Если в матрице совместимости отсутствуют столбцы с указанными свойствами, начинается реализация подпрограммы выбор, в противном случае полученный элемент полного решения фиксируется в комплексах δ и ε, уже в коде (10, 01, 00), и вновь реализуется подпрограмма элер.

Подпрограмма выбор находит в текущем частичном решении минимальный интервал и ближний к нему элемент непокрытого остатка, представляя их номера значениями индексов f и e, соответственно. Нумерация элементов частичного решения не является сплошной и соответствует нумерации тех элементов комплексов Y и Z, которые используются для их представления.

Наконец, подпрограмма ассимил минимально расширяет найденный интервал, обеспечивая покрытие им

указанного ближнего элемента. Она же корректирует непокрытый остаток множества M^1, выбрасывая из него все элементы, покрываемые полученным расширенным интервалом, и вносит соответствующие изменения в матрицу $\|M_i^1 * U_i\|$. После этого вновь реализуется подпрограмма новички и т. д.

4.

* **001 303**

§ 0 $b_\alpha + 100 \Rightarrow b_\zeta$ * очистка ζ //
$b_\alpha \Rightarrow b_\zeta + a_\zeta \Rightarrow a_{27} + 40 \Rightarrow a_{30} \circ g \bar{\circ} \underline{d}$

§ 1 * новички ζ $\boldsymbol{Y Z} \alpha \beta \boldsymbol{d} \gamma$ //

§ 2 * элер ζ $\boldsymbol{Y Z} \alpha \beta \boldsymbol{d e f} 3$ //
$\boldsymbol{e} \urcorner \wedge \boldsymbol{f} \Rightarrow \delta_g \boldsymbol{e} \wedge \boldsymbol{f} \Rightarrow \varepsilon_g \triangle g \boldsymbol{d} \urcorner \circ\!\!\rightarrow 4 \rightarrow 2$

§ 3 * выбор ζ $\boldsymbol{Y Z} \alpha e f$ //
* ассимил ζ $\boldsymbol{Y Z} \alpha \beta e f$ // $\rightarrow 1$

§ 4 $b_\alpha \mapsto 1\, g \Rightarrow b_\delta \Rightarrow b_\varepsilon$.

Вспомогательные программы. Перейдем к рассмотрению используемых подпрограмм, ограничиваясь пунктами 2 и 4 стандартного описания, поскольку постановки решаемых данными подпрограммами задач были изложены выше, а реализуемые алгоритмы достаточно просты.

Очистка комплекса — очистка

1—2. Оператор очистка присваивает нулевые значения всем элементам комплекса α.

Задаются: a_α, b_α.

Внутренние операнды: —, a, —.

4.

* **001 061**

§ 0 $\bar{\circ} a$

§ 1 $\triangle a \oplus b_a \circ\!\!\rightarrow 2 \circ \alpha_a \rightarrow 1$

§ 2 .

Поиск новых элементов частичного решения — новички

2. Внешние операнды:

$\alpha_к :: \| M_i^1 * U_i \|$,

$\beta\gamma_к ::$ пара миноров размерности $\Psi(U_i)$ на $\Psi(X)$, задающих частичное решение U_i в коде (01, 11, —0),

$\delta_к :: \| M_i^1 \Rightarrow X \|$,

$\varepsilon_к :: \| M^0 \Rightarrow X \|$,

$\zeta_п :: \overline{\Psi}(U_i)$ — инверсное значение булева вектора $\Psi(U_i)$,

$\eta_п :: \Psi(X)$.

Задаются: a_α, a_β, a_γ, a_δ, a_ε, b_α, b_δ, b_ε.
Подпрограммы: анапер.
Внутренние операнды: b, e, —.

4.

* 001 305

§ 0 $\overline{\bigcirc}\, c\, b_\alpha \Rightarrow d$

§ 1 $\triangle\, c \oplus d \circ\!\!\rightarrow 4\, \alpha_c \mapsto 1$

$\zeta\, \dot{X}\, 4\, e\, \delta_c \Rightarrow \beta_e \Rightarrow \boldsymbol{a}\, \eta \Rightarrow \gamma_e$

$\overline{\triangle}\, d\, \delta_d \Rightarrow \delta_c\, \alpha_d \Rightarrow \alpha_c\, \overline{\triangle}\, c\, \overline{\bigcirc}\, b$

§ 2 $\triangle\, b \oplus d \circ\!\!\rightarrow 1\, \delta_b \oplus \boldsymbol{a}\, \urcorner \wedge \eta \Rightarrow \boldsymbol{b}$

* анапер $3\, \varepsilon\, \boldsymbol{a}\, \boldsymbol{b}\, // \rightarrow 2$

§ 3 $\alpha_b \vee c_e \Rightarrow \alpha_b \rightarrow 2$

§ 4 $d \Rightarrow b_\alpha \Rightarrow b_\delta$.

Формирование элемента полного решения — элер

2. Внешние операнды:

$\alpha_к :: \| M_i^1 * U_i \|$,

$\beta\gamma_к ::$ пара миноров размерности $\Psi(U_i)$ на $\Psi(X)$, задающих частичное решение U_i в коде (01, 11, —0),

$\delta_к :: \| M_i^1 \Rightarrow X \|$,

$\varepsilon_к :: \| M^0 \Rightarrow X \|$,

$\zeta_п :: \Psi(U_i)$,

$\eta\vartheta_п^+ ::$ сформированный интервал в коде (01, 11, —0),

$\varkappa_ч$ — дополнительный выходной полюс, соответствующий отсутствию искомого элемента.

Задаются: a_α, a_β, a_γ, a_δ, a_ε, b_α, b_δ, b_ε.
Подпрограммы: слурин.
Внутренние операнды: c, d, —.
4.

* **001 304**

§ 0 $\bar{\circ} a \circ b \circ c\, b_\alpha \Rightarrow d$

§ 1 $\triangle\, a \oplus d \circ\!\!\rightarrow 2$

$a_a \wedge b \vee c \Rightarrow c\, a_a \vee b \Rightarrow b \rightarrow 1$

§ 2 $b \vee \zeta \urcorner \circ\!\!\rightarrow 3 \vdash \Rightarrow c\gamma_c \Rightarrow \vartheta \rightarrow 5$

§ 3 $c \vee \zeta \urcorner \circ\!\!\rightarrow \varkappa \vdash \Rightarrow c \bar{\circ} b$

§ 4 $\triangle\, b\, \alpha_b \wedge c_c \circ\!\!\rightarrow 4$

$\delta_b \oplus \beta_c \urcorner \wedge \gamma_c \Rightarrow \vartheta\ \overline{\triangle}\ d \Rightarrow b_\alpha$

$\Rightarrow b_\delta\, \delta_d \Rightarrow \delta_b\, \alpha_d \Rightarrow \alpha_b$

§ 5 $\beta_c \Rightarrow \eta$ * слурин $\varepsilon\, h\, \eta$ //

$\zeta \vee c_c \Rightarrow \zeta \circ \beta_c \circ \gamma_c$.

Выбор пары интервал — элемент — выбор

2. Внешние операнды:

$\alpha_к :: \| M_i^! * U_i \|$,

$\beta\gamma_к ::$ пара миноров размерности $\Psi(U_i)$ на $\Psi(X)$, задающих частичное решение U_i в коде (01, 11, —0),

$\delta_к :: \| M_i^! \Rightarrow X \|$,

$[\varepsilon_и^+]$ — номер найденного элемента из $M_i^!$,

$[\zeta_и^+]$ — номер найденного интервала из M.

Задаются: a_α, a_β, a_γ, a_δ, b_α, b_δ.
Внутренние операнды: c, c, —.
Примечание: множество U_i не может быть пустым.
4.

* **001 307**

§ 0 $\bar{\circ} a \circ b$

§ 1 $\triangle\, a \oplus 40 \circ\!\!\rightarrow 2$

$\gamma_a \bigtriangledown \Rightarrow c\, b - c \mapsto 1\, c \Rightarrow b\, a \Rightarrow \zeta \rightarrow 1$

§ 2 $c_\zeta \Rightarrow a\, \beta_\zeta \Rightarrow b\, \gamma_\zeta \Rightarrow c \bar{\circ} a \circ b$

§ 3 $\triangle\, a \oplus b_\delta \circ\!\!\rightarrow 4\, \alpha_a \wedge a \circ\!\!\rightarrow 3$

$\delta_a \oplus b \urcorner \wedge c \bigtriangledown \Rightarrow c\, b - c \mapsto 3\, c \Rightarrow b\, a$

$\Rightarrow \varepsilon \rightarrow 3$

§ 4 .

Расширение интервала и ассимиляция покрываемых элементов — а с с и м и л

2. Внешние операнды:

$\alpha_к :: \| M_i^1 * U_i \|$,

$\beta\gamma_к ::$ пара миноров размерности $\Psi(U_i)$ на $\Psi(X)$, задающих частичное решение U_i в коде (01, 11, —0),

$\delta_к :: \| M_i^1 \Rightarrow X \|$,

$\varepsilon_к :: \| M^0 \Rightarrow X \|$,

$[\zeta_и]$ — номер покрываемого элемента из M_i^1,

$[\eta_и]$ — номер расширяемого интервала из U_i.

Задаются: a_α, a_β, a_γ, a_δ, a_ε, b_α, b_δ, b_ε.

Подпрограммы: а н а п е р.

Внутренние операнды: $\boldsymbol{c}$, c, —.

4.

* **001 306**

§ 0 $\delta_\zeta \oplus \beta_\eta \urcorner \wedge \gamma_\eta \Rightarrow \gamma_\eta \Rightarrow \boldsymbol{c}$

$\beta_\eta \Rightarrow \boldsymbol{b} \,\overline{\circ}\, b\ b_\alpha \Rightarrow c$

§ 1 $\triangle\, b \oplus c \circ\!\!\rightarrow 3\,\alpha_b \wedge c_\eta \circ\!\!\rightarrow 1$

$\delta_b \oplus \boldsymbol{b} \wedge \boldsymbol{c} \longmapsto 2$

$\overline{\triangle}\, c\ \delta_c \Rightarrow \delta_b\ \alpha_c \Rightarrow \alpha_b\ \overline{\triangle}\, b \rightarrow 1$

§ 2 $\oplus\, \boldsymbol{c} \Rightarrow \boldsymbol{a}$ * анапер 1 ε $\boldsymbol{b}\,\boldsymbol{a}$ // $\alpha_b \oplus c_\eta \Rightarrow \alpha_b \rightarrow 1$

§ 3 $c \Rightarrow b_\alpha \Rightarrow b_\delta$.

§ 5. Эксперименты над алгоритмами

О б э ф ф е к т и в н о с т и а л г о р и т м о в. Как мы видели, алгоритмы решения логических задач можно разбить на два класса: *точные* алгоритмы, гарантирующие получение оптимального в каком-то смысле решения, и *приближенные* алгоритмы, позволяющие получать лишь некоторое приближение к оптимальному решению.

Использование приближенных алгоритмов оправдывается в тех случаях, когда нахождение точных решений оказывается весьма трудоемким или даже практически невозможным. Однако приближенные алгоритмы полезны не только в этих случаях.

При решении практических задач типична ситуация, когда требуется, с одной стороны, получить достаточно качественное решение, а с другой стороны, получить его достаточно дешево. Приходится учитывать то обстоятельство, что решение логических задач сопряжено, как правило, со значительным перебором анализируемых вариантов, влекущим существенные затраты машинного времени при решении задач на вычислительных машинах. Эти затраты машинного времени естественно рассматривать как заплаченную за решение *цену*. *Качество* же решения удобно измерять степенью приближения решения к оптимальному.

Иногда требования к качеству искомого решения бывают настолько высокими, что за него можно заплатить большую цену: затратить на процесс решения много машинного времени. В других же случаях такая трата не будет оправдана, и значительно более целесообразным будет согласиться на получение некоторого приближения, если только его качество может быть признано удовлетворительным и если оно получается достаточно экономным путем.

В связи со сказанным возникает весьма важная задача *оптимизации процесса решения* логической задачи. Как правило, требуемый оптимум нельзя достигнуть, ограничиваясь рамками, в которых рассматривается лишь решаемая логическая задача (процесс решения которой нужно оптимизировать). Необходимо знать, для чего решается данная задача и где будет использовано получаемое решение, чтобы разумно сформулировать требования к качеству решения и оценить допустимые затраты на получение решения. Другими словами, критерий оптимальности следует искать в постановке той «внешней», более общей задачи, которая содержит данную логическую задачу как некоторую часть.

Выше уже говорилось, что чем шире круг задач, в которых можно использовать некоторый алгоритм, тем больший интерес он (алгоритм) представляет и тем более оправдывается его детальная разработка. Вместе с тем следует учитывать, что алгоритм, достаточно хороший при решении одних задач указанного круга, будет не столь уж хорош при решении других задач. Конечно, коль скоро мы рассматриваем задачи того же круга,

данный алгоритм может быть применен, но насколько это будет целесообразно? Возможно, что в этом случае лучшим окажется некоторый другой алгоритм.

Поэтому развитая система алгоритмов должна содержать серии *конкурирующих* алгоритмов, решающих по существу одну и ту же задачу, но отличающихся друг от друга по своим характеристикам. Нас будут интересовать, прежде всего, *количественные оценки эффективности алгоритмов.*

Такие оценки должны указывать на цену решения (выражаемую, как правило, через машинное время) как функцию некоторых параметров, характеризующих сложность решаемой задачи и называемых *определяющими*, и на качество решения — для приближенных алгоритмов. Данные величины являются случайными и могут принимать различные значения для одних и тех же значений определяющих параметров. Однако при разумном выборе определяющих параметров разброс значений этих величин может быть достаточно мал, что позволяет довольно точно прогнозировать ожидаемые затраты времени на получение решения и его качество и, следовательно, планировать процесс вычисления. Можно говорить, что качество оценок эффективности алгоритмов тем выше, чем более точный прогноз возможен на основе этих оценок. Именно наличие оценок эффективности и позволяет в каждой конкретной ситуации производить надлежащий выбор среди конкурирующих алгоритмов.

Исследование эффективности алгоритмов получения кратчайшего покрытия. Получение оценок эффективности алгоритма аналитическим путем весьма сложно и, за исключением тривиальных случаев, оказывается практически невозможным. Однако эти оценки во многих случаях могут быть получены экспериментальным путем — при решении на вычислительной машине некоторой серии задач заданного типа. Проиллюстрируем этот путь на примере исследования алгоритмов окрапок4 и прикрапок1, решающих задачу нахождения кратчайшего покрытия булевой матрицы (точно или приближенно).

Прежде всего требуется выбрать набор определяющих параметров, удобный для рассматриваемой задачи. В данном случае в такой набор естественно включить

следующие величины: m — число строк булевой матрицы, n — число столбцов и p — плотность единиц. Уточним смысл величины p: считается, что каждый из элементов булевой матрицы может, независимо друг от друга, принять значение 1 с вероятностью p, в результате чего примерно pmn элементов матрицы будут иметь значение 1, остальные — значение 0.

Для проведения эксперимента над алгоритмом подготавливается генератор псевдослучайного потока задач типа, соответствующего данному алгоритму. Псевдослучайным (а не случайным) его приходится называть потому, что генератор оформляется как программа, в которой используется генератор псевдослучайных булевых векторов, то есть используется последовательность значений переменной *я*. Эта последовательность удовлетворяет всем требованиям, предъявляемым обычно к случайным последовательностям, когда последние используются в вычислениях, однако она не является случайной в строгом смысле этого слова, поскольку последующий член последовательности однозначно определяется предыдущим. Важно то, что закономерность, связывающая члены последовательности, никак не отражается в решаемой задаче, то есть с точки зрения решаемой задачи последовательность практически ничем не отличается от строго случайной. Она оказывается даже более удобной в том смысле, что ее всегда можно воспроизвести, что может быть полезно при сравнении различных алгоритмов: легко организовать сравнение на одном и том же потоке задач.

Результат решения каждого конкретного примера условимся характеризовать двумя числами: h — мощность полученного покрытия и t — время решения (на машине М-20), измеряемое в часах (^{h}), минутах ($'$) и секундах ($''$).

Результаты исследования. Если каждый отдельный пример решается довольно быстро, в ходе эксперимента можно решить достаточно много различных конкретных задач и накопить данные для последующей статистической обработки. Если же цена решения отдельной задачи высока, можно удовлетвориться приведением данных в форме таблицы, отражающей результаты решения каждого отдельного примера. По крайней

Таблица 3.5.1

Результаты испытаний программы окрапок4

($n = 16$)

	$p = 1/16$		$p = 1/8$		$p = 1/4$	
m	h	t	h	t	h	t
50	11	5″	6	3″	4	10″
100	7	7″	5	35″	3	15″
150	6	12″	5	1′ 30″	3	20″
200	6	35″	5	3′ 10″	3	47″
250	6	1′ 35″	5	5′ 52″	3	40″
300	6	2′ 47″	4	9′ 05″	3	55″
350	6	4′ 48″	4	5′ 08″	3	1′ 12″
400	6	4′ 53″	4	5′ 30″	3	1′ 07″
450	6	6′ 45″	4	13′ 00″	3	1′ 30″
500	5	9′ 47″	4	9′ 30″	3	1′ 40″
550	5	11′ 00″	4	11′ 45″	3	2′ 01″
600	5	14′ 03″	4	13′ 20″	3	2′ 15″
650	5	18′ 12″	4	16′ 15″	3	2′ 32″
700	5	20′ 43″	4	21′ 20″	3	2′ 53″
750	5	23′ 53″	4	27′ 20″	3	3′ 17″
800	5	30′ 40″	4	33′ 00″	3	3′ 42″
850	5	35′ 10″	4	39′ 00″	3	5′ 44″
900	5	40′ 20″	—	—	3	3′ 37″
950	—	—	—	—	3	3′ 25″

Таблица 3.5.2

Результаты испытаний программы окрапок4

($n = 32$)

	$p = 1/16$		$p = 1/8$		$p = 1/4$	
m	h	t	h	t	h	t
50	17	3″	7	35″	5	15″
100	13	4′ 00″	6	8′ 25″	4	30″
150	12	26′ 44″	6	34′ 30″	4	4′ 20″
200	8	$1^h24'45''$	6	$2^h28'30''$	4	6′ 37″
250	—	—	—	—	4	11′ 39″
300	—	—	—	—	4	19′ 15″

мере при этом не будет потерь информации, неизбежных при любом способе статистической обработки информации.

В данном случае был выбран второй путь, причем было признано целесообразным, рассматривая некоторую последовательность комбинаций значений определяющих параметров, ограничиться решением для каждой из этих комбинаций по одному примеру.

Результаты исследований программы окрапок4 представлены в таблицах 3.5.1, 3.5.2 и 3.5.3.

Таблица 3.5.3

Результаты испытаний программы окрапок4

($n = 11$, $p = 1/8$)

m	h	t	m	h	t	m	h	t	m	h	t	m	h	t
10	5	0,5″	160	3	10″	310	3	11″	460	3	28″	610	3	1′29″
20	5	1″	170	3	19″	320	3	1′09″	470	3	1′58″	620	3	1′03″
30	5	1″	180	4	12″	330	3	17″	480	3	3′34″	630	3	31″
40	5	1″	190	3	23″	340	3	20″	490	3	6″	640	3	22″
50	4	2″	200	4	3″	350	3	1′35″	500	3	1′26″	650	3	3′49″
60	5	2,5″	210	3	13″	360	3	22″	510	3	1′12″	660	3	4′12″
70	4	3″	220	4	46″	370	3	12″	520	3	22″	670	3	55″
80	5	5″	230	3	25″	380	4	3′32″	530	3	1′32″	680	3	1′53″
90	4	5″	240	3	29″	390	3	32″	540	3	15″	690	3	1′30″
100	3	6″	250	3	5″	400	3	32″	550	3	1′48″	700	3	1′05″
110	4	4″	260	3	14″	410	3	1′00″	560	3	1′20″	710	3	1′17″
120	4	12″	270	3	4″	420	3	34″	570	3	33″	720	3	4′05″
130	4	7″	280	3	6″	430	3	1′03″	580	3	25″			
140	3	6″	290	3	1′17″	440	3	50″	590	3	1′08″			
150	3	8″	300	3	46″	450	3	2′16″	600	3	3′26″			

Представленная в таблицах информация позволяет оценивать целесообразность применения программы окрапок4 в различных конкретных ситуациях. Интересно заметить, что время решения колеблется в довольно широких пределах в зависимости от индивидуальных особенностей решаемых задач (то есть от особенностей рассматриваемых булевых матриц). Значительно более устойчивой величиной является h — мощность кратчайшего покрытия.

Эксперименты, выполненные с программой прикрапок1, показали, что, обладая значительно более

высоким быстродействием, эта программа позволяет получать довольно хорошие приближения к кратчайшему покрытию. Например, на одной и той же серии псевдослучайных булевых матриц, для которой $n=11$, $p=1/8$, а m пробегает ряд значений 10, 20, 30, ..., 250, программами окрапок4 и прикрапок1 получены покрытия со следующими мощностями, обозначаемыми через h_0 и $h_п$, соответственно:

$$h_0=5,5,5,5,4,5,4,5,4,3,4,4,4,3,3,3,3,3,4,3,4,3,4,3,3,3,$$
$$h_п=5,5,6,5,4,5,4,5,4,5,4,3,5,4,4,3,3,3,3,3,4,3,4,3,4,3,4,3.$$

Как видно, в данной серии задач находимое программой прикрапок1 покрытие лишь в трех случаях оказывается неоптимальным, причем во всех этих случаях мощность найденного покрытия на единицу больше мощности кратчайшего покрытия, найденного программой окрапок4.

Организация эксперимента над программой минимизации днф булевых функций. Целью этого эксперимента являлось получение количественных оценок эффективности различных процедур, составляющих в совокупности алгоритм минимизации булевых функций, описанный в параграфе 2 данной главы.

Эксперимент был организован следующим образом. Генерировалась последовательность псевдослучайных булевых функций в дизъюнктивной нормальной форме, сложность которой задавалась параметрически через число аргументов n, число членов в днф k и средний ранг конъюнкций r (допускаются небольшие отклонения рангов различных конъюнкций). Получение таких псевдослучайных форм производится достаточно легко путем построения двух комплексов мощностью k каждый, в элементах которых выделяются левые n компонент, примерно r из которых (выбираемые случайно) принимают значение 1 в том или в другом комплексе, но не вместе, с равной вероятностью. Значение такой пары комплексов интерпретируется как дизъюнктивная нормальная форма булевой функции, заданная в (10, 01, 00)-коде.

Полученные таким образом днф обрабатывались упомянутым выше алгоритмом минимизации, с реги-

страцией результатов работы различных подпрограмм. Эти результаты оформлялись в виде выводимых на печать таблиц, каждая из строк которых представляла данные, полученные при обработке одной булевой функции. В строке последовательно перечисляются значения r и k для исходной, генерируемой днф, значения r и k для безызбыточной днф, получаемой из исходной программой безызбыточная, число членов в ядре, находимом программой ядро, значения r и k в расширенной форме (сокращенной днф), получаемой программой расширение. Затем следует число q циклов упрощения, производимого программой циклостат, и результат этого упрощения: число k' членов в квазиядре и число k'' членов в циклическом остатке. Далее приводится число m простых совокупностей, находимое программой просо в циклическом остатке и, наконец, характеристики найденной в конце концов кратчайшей днф — соответствующие значения r и k.

В таблице 3.5.4 приводится фрагмент таких данных, полученных при $n=7$.

Таблица 3.5.4

	Генерируемая днф		Безызбыточная днф		Ядро	Расширение		Циклы упрощения			Простые совокупности	Кратчайшая днф	
	r	k	r	k	k	r	k	q	k'	k''	m	r	k
1	5	16	5	12	8	5	28	2	12	0		5	12
2	5	32	4	22	5	5	40	3	8	24	22	4	18
3	5	48	3	17	6	4	39	6	16	0		4	16
4	5	64	2	7	6							2	7
5	5	80	2	8	3	2	15	1	3	9	7	2	7
6	5	96	0	1	1							0	1
7	5	112	0	1	1							0	1

Пустые клетки в таблице означают, что процесс минимизации завершается раньше, чем могла бы понадобиться программа, соответствующая данному столбцу. Например, при рассмотрении днф, соответствующей строке 1, кратчайшая днф находилась уже на этапе упрощения множества простых импликант, когда число элементов в циклическом остатке оказывалось равным

нулю и, следовательно, решение полностью представлялось получаемым на этом этапе квазиядром. Аналогичная ситуация отражена в строке 3, с той лишь разницей, что в этом случае квазиядро оказалось на один элемент меньше, чем полученная ранее безызбыточная д н ф, в то время как в строке 1 такого сокращения нет.

Условимся обозначать через y ситуацию, нашедшую отражение в строке 1, а через y_1 — в строке 3 (нижний индекс указывает на степень сокращения полученной безызбыточной д н ф при переходе к кратчайшей д н ф). Рассмотрим также другие ситуации, представленные в таблице 3.5.4. В строке 2 отражен процесс минимизации, включающий нахождение множества простых совокупностей в циклическом остатке и построение соответствующей матрицы Квайна. Обозначим представленную здесь ситуацию через κ_4 — кратчайшая д н ф на 4 элемента меньше безызбыточной. Аналогичная ситуация, представленная в строке 5, будет обозначаться через κ_1. При минимизации исходной д н ф, соответствующей строке 4, процесс минимизации завершается построением ядра, поскольку число элементов в нем оказывается лишь на единицу меньше числа элементов в безызбыточной д н ф. В этом случае становится очевидным, что последняя является также кратчайшей д н ф. Такую ситуацию обозначим через $я_1$. Наконец, в последних двух строках таблицы отражены ситуации, в которых исходная д н ф оказывается вырожденной, то есть тождественно равной единице. Такая ситуация, обозначаемая через 1, выявляется на этапе построения безызбыточной д н ф: в этом случае полученная безызбыточная д н ф будет содержать лишь одну элементарную конъюнкцию нулевого ранга. Заметим, что для выявления такой ситуации можно использовать также программу в ы т р о м а.

Э к с п е р и м е н т а л ь н ы е д а н н ы е. В ходе эксперимента было минимизировано около 200 д н ф, различающихся значениями параметров n, r и k. Не приводя здесь подробных данных по каждому элементарному эксперименту (как мы будем называть процесс минимизации одной д н ф), ограничимся классификацией элементарных экспериментов по ситуациям, представив соответствующую информацию в таблице 3.5.5.

Таблица 3.5.5

n	r \ k	16	32	48	64	80	96	112	128	144	160
6	3	1	1	1	1	1	1	1	1		
	4	$к_1$	$я$	$я$	1	1	1	1	1		
	5	$у$	$к_2$	$у_1$	$к_1$	$я$	$я$	$я$	1		
7	3	1	1	1	1	1	1	1	1		
	4	$я$	$я$	$я$	1	1	1	1	1		
	5	$у$	$к_1$	$у$	$я_1$	$к_1$	1	1	1		
	6	$у$	$к$	$К$	$к_3$	$к$	$к_1$	$К$	$К$	$у$	$к_1$
8	5	$у$	$у_2$	$К$	$к$	1	1	1	1	1	1
	6		$у$		$К$		$к_2$		$у_1$		$к_1$
10	3	1	1	1	1	1	1				
	4	$я$									
	5		$у_1$	$у$	$у$	$у$	$к$				1
	6	$я$	$у$	$у_1$	$у_1$						
	7		$у$	$у$	$у$	$к_2$					
	8	$я$		$у$	$у_1$	$у$	$к_5$	$К$			
	9	$я''$	$у$	$у$	$у$	$у$	$у_1$	$у$	$у_3$	$к_4$	
12	5	$у$	$у$	$у$	$К$						
	7	$я$	$у$	$у$	$у$						
	9	$я_1$	$у$	$у$	$у$	$у$	$у$	$у_1$	$у$		
	10	$я''$									
	11		$я''$	$у$	$у$	$у$	$у$	$у_1$	$у_1$		
14	8		$я''$								
	9				$у''$						
	11		$я_1$		$я_1''$		$у$		$у$		$у$
16	12					$у''$	$у$	$у$	$у''$	$у''$	$у$

Каждая непустая клетка таблицы соответствует определенной комбинации значений параметров n, r и k, при которой был выполнен один элементарный эксперимент, а символ, находящийся в клетке, показывает тип ситуации, реализованной при данном эксперименте. Кроме уже знакомых нам символов, в таблице встречаются также символы $к$, $К$, $я$, $я''$, $у''$, интерпретируемые следующим образом: $к$ — ситуация, в которой реализуется программа окрапок5, причем оказывается, что число членов в кратчайшей днф совпадает с числом

членов в полученной ранее безызбыточной днф; *К* — ситуация, в которой число простых совокупностей, находимых в циклическом остатке, оказывается бо́льшим, чем 32 (в этом случае программа окрапок5, находящая кратчайшее покрытие соответствующей булевой матрицы, должна быть заменена программой окрапок6, не принимавшей участия в данном эксперименте); *я* — ситуация, в которой все члены безызбыточной днф попадают в ядро; *я″* — ситуация, в которой исходная днф оказывается безызбыточной и все ее члены также попадают в ядро; *у″* — ситуация, в которой исходная днф оказывается безызбыточной, а число членов в ней оказывается равным числу членов в кратчайшей днф, получаемой на этапе упрощения множества простых импликант.

Интерпретация результатов. При анализе таблицы 3.5.5 бросается в глаза, что в большинстве случаев (когда у символов отсутствуют нижние индексы) получаемая программой безызбыточная днф оказывается кратчайшей. Это говорит о том, что упомянутая программа может с успехом использоваться при решении многих практических задач, когда требуется получить лишь хорошее приближение к кратчайшей форме. Обращает на себя внимание область вырождения генерируемых функций, достигаемая в данном эксперименте при малых значениях r и достаточно больших значениях k.

Следует отметить высокую эффективность процедуры упрощения множества простых импликант, приводящей к существенному сокращению матрицы Квайна, получаемой на следующем этапе. С ростом n увеличивается область, в которой это упрощение приводит непосредственно к кратчайшей днф, в связи с чем в выполненном эксперименте необходимость рассмотрения матрицы Квайна возникала лишь при $n \leqslant 12$. При дальнейшем возрастании n в заданных пределах изменения r и k процесс минимизации постепенно вырождается в реализацию первых двух этапов: получение бызыбыточной днф и извлечение из нее ядра. Более того, в дальнейшем оказывается, что вероятность встречи такой днф, которая не была бы уже кратчайшей, достаточно мала — по крайней мере в проведенном эксперименте такие днф не встречались уже при $n > 20$.

Представленную в таблице 3.5.5 информацию можно использовать для рационального выбора того или иного алгоритма минимизации в различных ситуациях, характеризующихся конкретными значениями параметров n, r и k. Для этого могут также оказаться полезными сведения о затратах машинного времени на реализацию различных этапов описанного алгоритма. Такие данные, полученные на машине М-220, приводятся в таблице 3.5.6.

Каждый элементарный эксперимент характеризуется здесь последовательностью чисел, показывающих затраты времени на реализацию следующих этапов (если они имели место):

1) получение безызбыточной днф из исходной,
2) нахождение ядра,
3) получение сокращенной днф (расширение),
4) нахождение квазиядра и циклического остатка,
5) получение простой матрицы Квайна и нахождение ее кратчайшего покрытия.

Время показано в минутах и измерялось весьма грубо; 0 означает, что затраты времени на соответствующем этапе были существенно меньше одной минуты.

Эксперименты с программой квасобу. Эта программа, описанная в параграфе 3 настоящей главы, ориентирована на использование в программе крафсо (нахождения кратчайшей днф слабо определенной булевой функции, заданной множествами M^1 и M^0). Программа квасобу реализует первый этап общего процесса вычисления, то есть получает матрицу Квайна $\|\sup M^* \supseteq M^1\|$. Последующее нахождение кратчайшего покрытия этой матрицы, составляющее второй этап вычислений и производимое программой окрапок4, приводит к решению задачи в целом.

Предварительные эксперименты показали, что программа квасобу оказывается значительно более быстродействующей, чем работающая вслед за ней программа окрапок4, то есть программа квасобу не является «узким местом» программы крафсо. Поэтому мы не будем приводить здесь оценки быстродействия программы квасобу, а обратим внимание на другой вопрос.

Таблица 3.5.6

n	r \ k	16	32	48	64	80	96	112	128	144	160
6	3	0	0	0	0	0	0	0	0	0	0
	4	00000	00	00	0	0	0	0	0	0	0
	5	0000	0004	00 0 0	00 0 02	00	00	00	0		
7	3	0	0	0	0	0	0	0	0		
	4	00	00	00	0	0	0	0	0		
	5	0000	00010	00 0 0	00	00 0 00	0	0	0		
	6		00000	00 0 0?	00 0 11	10 0 10	10 0 17	10 0 1?	10 0 0?	100 0	10 0 10
8	5	0000	0000	00 0 1?	00 0 00	0	0	0	0	0	0
	6		0001		00 1 7?		00 1 20		10 0 1		10 0 00
10	3	0	0	0	0	0	0				
	4	00									
	5		0012	00 0 0	00 0 1	00 0 0	00 0 00				
	6	00	0011	10 7 11	10 23 37						
	7	00	0010	00 2 2	10 5 18	20 12 40	10 16 46 1				
	8	00	0000	10 0 1	1 1 2	10 4 6	20 4 11 0	20 9 90?			
	9	00	0000	10 0 0	20 0 0	20 1 4	40 5 0	40 3 12	50 2 34	50 3 56 0	
12	5	0000	0021	00 0 1	10 6 36?						
	7	00	0011	10 20 2	10 30 8						
	9	00	00	00 0 1	10 0 2	20 7 6	30 17 11	40 25 31	40 18 51		
	10	00									
	11		00	10 0 0	20 0 0	30 0 0	40 1 0	61 0 3	120 1 3		
14	8		00								
	9				20 14 1						
	11		00		20		40 1 6		21 6 3		10 1 16 17
16	12					30 1 1	50 3 1	60 5 2	81 7 2	10 17 2	12 122 8

Поскольку трудности реализации второго этапа программы к р а ф с о в сильной степени определяются размерами матрицы Квайна (это видно из таблиц 3.5.1 и 3.5.2), представляет интерес определить характер статистической зависимости размеров этой матрицы от таких параметров задаваемой булевой функции, как мощности множеств X, M^1 и M^0. Именно с этой целью на машине М-20 был выполнен статистический эксперимент над программой к в а с о б у.

Эксперимент в целом, занявший около 10 часов машинного времени, состоял примерно из 1800 элементарных экспериментов; каждый из них заключался в реализации программы к в а с о б у, информация для которой задавалась с помощью генератора псевдослучайных булевых векторов (встроенного в механизм выдачи очередных значений переменной $я$). Каждому элементарному эксперименту соответствовала одна, вполне определенная комбинация значений параметров задаваемой булевой функции, варьируемых в следующих пределах: $\sigma(X)=8, 16, 24, 32$, $\sigma(M^1)=5, 6, \ldots, 24$, $\sigma(M^0)=5, 6, \ldots$ $\ldots, 32$. В соответствии с выбранной комбинацией генерировались множества M^1 и M^0 путем равновероятного выбора элементов булева пространства в требуемом количестве (с дополнительным условием неповторения) и, посредством реализации программы к в а с о б у, определялась мощность результативного множества $\sup M^*$, равная числу строк получаемой таблицы Квайна.

Обработка получаемой информации была выполнена с достаточной точностью визуальным методом, суть которого заключается в следующем.

Совокупность результатов, соответствующих какой-либо комбинации значений $\sigma(X)$ и $\sigma(M^1)$, представляется на графике в виде семейства точек в координатах $\sigma(M^0)$, $\sigma(\sup M^*)$, как это показано на рис. 3.5.1. Затем визуальным путем проводится плавная кривая, представляющая, с некоторым приближением, совокупность математических ожиданий значения $\sigma(\sup M^*)$ для различных значений $\sigma(M^0)$.

Заметим, что традиционный метод получения указанных математических ожиданий по отдельности требует выполнения достаточно большой серии элементарных экспериментов для каждого из значений $\sigma(M^0)$

(при фиксированных значениях $\sigma(X)$ и $\sigma(M^1)$), что влечет большие затраты машинного времени. Используемый же здесь визуальный метод применим и в том случае, когда каждому значению $\sigma(M^0)$ соответствует лишь одно полученное экспериментально значение $\sigma(\sup M^*)$, а информация об искомых математических ожиданиях извлекается из совокупности экспериментальных данных

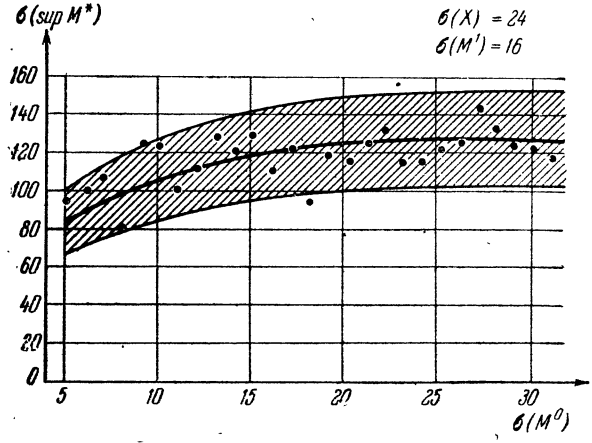


Рис. 3.5.1.

путем интерполяции и сглаживания в пространстве $\sigma(M^0) \times \sigma(\sup M^*)$.

Попутно определяется степень статистической зависимости $\sigma(\sup M^*)$ от $\sigma(M^0)$, выражаемая величиной отклонений экспериментальных точек от сглаженной кривой интерполяции. Например, в рассматриваемом случае эта степень находит следующее количественное выражение: число экспериментальных точек, отстоящих по ординате от кривой интерполяции далее чем на 20% абсолютной величины соответствующей ординаты кривой, составляет не более 10% общего числа экспериментальных точек. Определяемая описанным образом зона «допустимых отклонений» показана на рис. 3.5.1 штриховкой.

Семейство полученных кривых, соответствующих одному значению $\sigma(X)$ и различным значениям $\sigma(M^1)$,

наносится на один график (пример которого, соответствующий значению $\sigma(X)=24$, представлен на рис. 3.5.2) и подвергается операции согласования, эквивалентной, в определенном смысле, интерполяции и сглаживанию по координате $\sigma(\sup M^*)$. Пример согласованного таким образом семейства кривых показан на рис. 3.5.3.

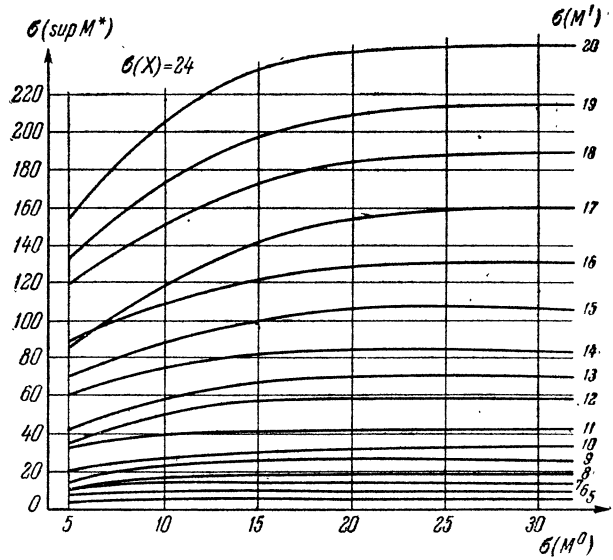


Рис. 3.5.2.

Очевидно, число экспериментальных точек, выходящих из зоны «допустимых отклонений», при согласовании кривых в семействе несколько возрастает. В данном случае это число возрастает до 15%.

На рис. 3.5.4, 3.5.5 и 3.5.6 показаны полученные аналогичным образом семейства для $\sigma(X)=8$, 16 и 32, соответственно. Интересно отметить слабую зависимость $\sigma(\sup M^*)$ от $\sigma(M^0)$, особенно для случаев $\sigma(X)=8$ и $\sigma(X)=16$. При $\sigma(X)=32$ $\sigma(\sup M^*)$ практически перестает зависеть от $\sigma(M^0)$, когда $\sigma(M^0)>\sigma(M^1)$. Соответствующие плавные участки кривых, начала которых показаны на рис. 3.5.6 пунктиром, продолжаются вправо достаточно долго. Однако нетрудно показать, что при

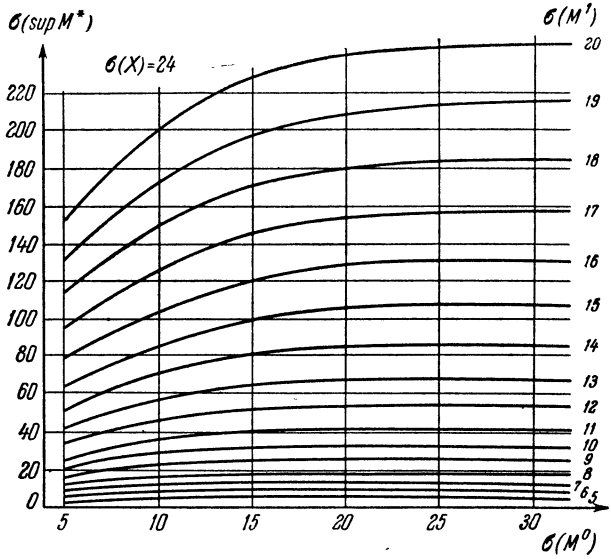


Рис. 3.5.3.

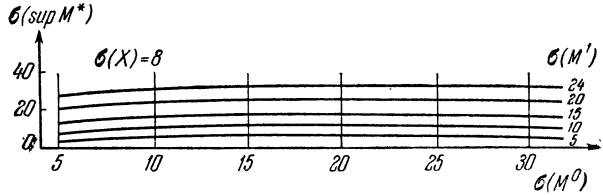


Рис. 3.5.4.

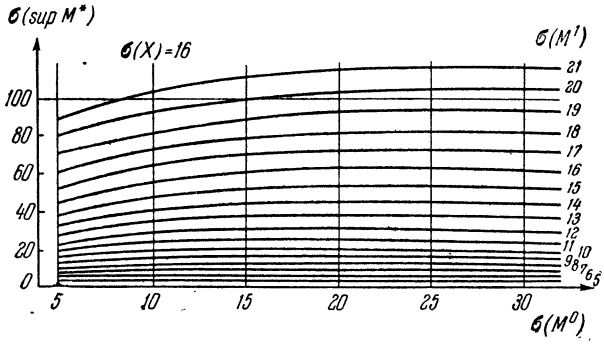


Рис. 3.5.5.

стремлении $\sigma(M^0)$ к $\sigma(M \setminus M^1)$ величина $\sigma(\sup M^*)$ должна стремиться статистически к значению $\sigma(M^1)$.

Сравнительный анализ эффективности программ минимизации слабо определенных булевых функций. Результаты эксперимента

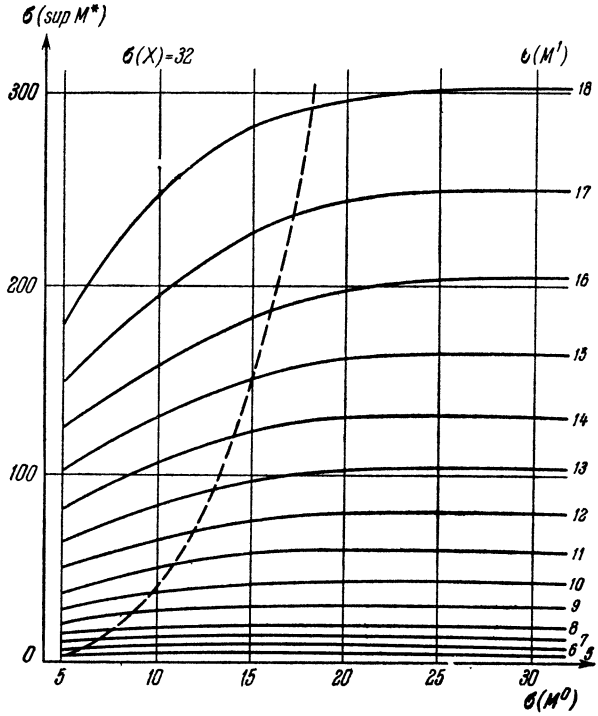


Рис. 3.5.6.

с программой крафсо в целом приводятся в таблице 3.5.7, в которой показаны также результаты решения тех же задач программами прикрафсо и минсоб, находящими приближения к кратчайшей днф.

Результаты испытаний говорят сами за себя. Программа крафсо, находящая кратчайшие днф, может служить эталоном для оценки качества решений, находимых приближенными алгоритмами. Однако

Таблица 3.5.7

Результаты испытаний программ минимизации слабо определенных булевых функций

$(\sigma(X) = 20)$

a — число конъюнкций в решении, b — число букв в решении, t — время решения.

Алгоритм		минсоб			прикрафсо			крафсо		
$\sigma(M^1)$	$\sigma(M^0)$	a	b	t	a	b	t	a	b	t
8	20	3	9	1″	2	7	1″	2	7	1″
	40	3	14	1″	3	14	4″	3	15	2″
	60	3	13	2″	3	12	6″	3	12	3″
	80	3	15	1″	3	16	3″	3	14	3″
	100	3	16	7″	3	17	2″	3	17	5″
16	20	4	13	1″	4	13	40″	4	12	32″
	40	5	21	7″	5	22	10″	5	20	1′03″
	60	6	30	5″	5	24	20″	5	24	52″
	80	5	24	3″	5	25	27″	4	20	32″
	100	5	26	7″	5	25	28″	5	24	1′10″
24	20	6	21	5″	5	15	2′58″	5	18	42′31″
	40	7	33	5″	5	22	2′09″	5	25	32′10″
	60	7	32	8″	7	33	1′54″	6	29	1^{h}50′15″
	80	8	39	9″	7	36	1′44″	5	24	3^{h}05′10″
	100	8	40	10″	7	38	1′48″	6	29	2^{h}07′10″
32	20	7	24	5″	6	22	18′04″			
	40	7	30	8″	7	28	9′31″			
	60	8	41	7″	8	39	7′37″			
	80	9	46	16″	8	41	6′29″			
	100	9	50	15″	8	42	5′30″			

потолок этой программы довольно низок — например, при $\sigma(X) = 20$, $\sigma(M^1) = 24$ и $\sigma(M^0) = 80$ на реализацию данной программы уходит 3 часа машинного времени. Существенно более быстродействующей является программа прикрафсо, дающая неплохие приближения к кратчайшей днф. Еще быстрее работает программа минсоб, однако находимые ею решения несколько дальше отходят от оптимальных. Заметим, что критерием оптимальности получаемых днф в данном случае считается минимум числа членов в днф, число же букв в днф, оптимальных в указанном смысле, может и от-

личаться от минимально возможного. Поэтому решения, получаемые приближенными алгоритмами, могут в некоторых случаях содержать меньшее число букв, чем решения, находимые программой крафсо.

В табл. 3.5.8 и 3.5.9 приведены результаты дополнительных экспериментов, выполненных над программой минсоб.

Таблица 3.5.8

Результаты испытаний программы минсоб
$(\sigma(X) = 20)$

$\sigma(M^1)$	$\sigma(M^0)$	a	b	t	$\sigma(M^1)$	$\sigma(M^0)$	a	b	t
20	100	6	30	3″	60	400	18	128	1′ 57″
	200	7	44	7″		500	21	155	2′ 35″
	300	7	45	26″	80	100	18	96	1′ 08″
	400	8	58	38″		200	22	126	1′ 44″
	500	8	58	58″		300	25	174	2′ 26″
40	100	10	56	19″		400	25	179	4′ 14″
	200	10	59	35″		500	26	195	2′ 50″
	300	14	98	54″	100	100	21	112	1′ 30″
	400	15	110	1′ 14″		200	25	162	2′ 30″
	500	14	109	1′ 32″		300	26	174	3′ 29″
60	100	15	77	43″		400	27	188	4′ 21″
	200	17	110	1′ 04″		500	28	204	5′ 17″
	300	17	114	1′ 32″					

Таблица 3.5.9

Результаты испытаний программы минсоб
$(\sigma(X) = 32)$

$\sigma(M^1)$	$\sigma(M^0)$	a	b	t	$\sigma M^1)$	$\sigma(M^0)$	a	b	t
100	100	18	97	16″	300	100	38	205	13′ 20″
	200	19	123	56″		200	44	277	25′ 05″
	300	22	148	4′ 24″		300	48	328	36′ 10″
	400	22	155	5′ 58″		400	51	362	44′ 12″
	500	24	181	6′ 07″		500	54	393	54′ 57″
200	100	29	149	7′ 44″	400	100	47	255	25′ 23″
	200	34	214	6′ 41″		200	55	342	49′ 52″
	300	35	232	12′ 54″			—	—	
	400	37	261	28′ 21″	500	100	53	282	34′ 54″
	500	37	271	37′ 25″		200	61	375	59′ 15″

Отметим довольно хорошую равномерность распределения букв по различным конъюнкциям находимых программой д н ф. Например, при минимизации булевой функции с $\sigma(X)=32$, $\sigma(M^1)=300$ и $\sigma(M^0)=200$ получена д н ф, в которой одна конъюнкция обладает рангом 8, пятнадцать — рангом 7, двадцать четыре — рангом 6 и четыре — рангом 5.

Рандомизация программы минсоб. При реализации программы минсоб часто возникают критические ситуации, делаемый в которых шаг не является заведомо хорошим. Иногда в таких ситуациях выбирается тот вариант продолжения процесса вычислений, который является наилучшим согласно используемым критериям, в других случаях лучших в этом смысле вариантов оказывается несколько, и тогда выбор падает на первый из них. Насколько он окажется хорошим, показывает дальнейший процесс вычислений.

Можно *рандомизировать* указанный выбор, то есть производить его случайно среди конкурирующих по используемым критериям вариантов. Эту рандомизацию можно производить непосредственно в момент выбора варианта, используя для этого очередное значение переменной $я$, но это не единственный путь.

Мы вынесем рандомизацию в начало процесса вычислений, совершая некоторую случайную перестановку элементов комплекса, представляющего множество M^1. Действительно, если в некоторой ситуации несколько элементов этого комплекса являются равноценными, то случайный выбор среди них вполне можно заменить случайным перемешиванием элементов с последующим выбором первого из них. Это позволяет сохранить без изменений программу минсоб, работающую в данном случае вслед за некоторой программой, производящей случайную перестановку элементов указанного комплекса.

Следует подчеркнуть, что варианты, среди которых производится случайный выбор, эквивалентны лишь с точки зрения используемых критериев. В действительности же одни из них приводят в конечном счете к лучшим решениям, другие — к худшим. Эти колебания качества решения можно использовать для его

улучшения, достигаемого путем многократной реализации программы минсоб.

Был поставлен следующий эксперимент: посредством множеств M^1 и M^0 задается некоторая слабо определенная булева функция, затем элементы множества M^1 подвергаются случайной перестановке, реализуется программа минсоб и фиксируется качество полученного решения (число конъюнкций и число букв в полученной днф), затем элементы множества M^1 снова переставляются случайным образом и снова реализуется программа минсоб, находящая некоторое новое решение, качество которого опять фиксируется, и т. д. Таким образом получается некоторая последовательность решений, из которой уже нетрудно выбрать наилучшее.

Например, для заданной случайным образом булевой функции со следующими значениями определяющих параметров: $\sigma(X)=20$, $\sigma(M^1)=32$, $\sigma(M^0)=20$ получена последовательность решений, представленная в таблице 3.5.10.

Таблица 3.5.10

1. Номер решения	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19
2. Число членов	7	7	7	7	7	7	6	7	7	7	7	6	7	7	7	8	7	3
3. Число букв	24	24	28	27	24	28	21	24	24	25	26	23	26	26	27	30	25	33

1.	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
2.	6	7	7	8	7	7	7	7	7	7	7	7	8	7	6	8	7	7	8	7	7	7	7
3.	20	27	24	30	24	26	24	25	24	26	31	23	29	26	22	30	25	24	32	24	27	24	25

1.	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
2.	7	8	8	8	7	7	7	7	8	7	7	7	7	7	7	7	7	6	8	7	7	7	7
3.	25	28	30	30	24	23	25	25	32	28	26	26	26	24	26	27	23	20	30	24	25	26	24

Из всех 64 решений особый интерес представляют три решения с номерами 1, 7 и 19, поскольку только эти решения оказываются лучше всех предыдущих. Образуемую этими решениями последовательность можно записать в следующей компактной форме:

$$1—7.24,\quad 7—6.21,\quad 19—6.20.$$

Интересно сравнить полученные результаты с данными экспериментов над программой прикрафсо,

находящей для этого же примера решение с 6 конъюнкциями и 22 буквами и затрачивающей на это 18 минут (на машине М-20). На получение каждого из 64 решений программа минсоб тратит около 5 секунд, откуда следует, что уже в течение первой минуты она находит решение лучшее, чем это делает в данном случае программа прикрафсо за 18 минут. Более полное представление об эффективности рандомизации программы минсоб можно составить на основе таблицы 3.5.11, отражающей результаты решения тех же задач, которые были использованы при составлении таблицы 3.5.7. Для каждой из этих задач была получена серия из 64 решений, из которых в таблице 3.5.11 представлены лишь лучшие по сравнению с предыдущими.

Таблица 3.5.11

$\sigma(X)$	$\sigma(M^1)$	$\sigma(M^2)$	Последовательность улучшений
20	8	20	1—3.9, 6—2.7
		40	1—3.14, 3—3.13, 7—3.12, 56—3.11
		60	1—3.13, 13—3.11, 41—3.11
		80	1—3.15
		100	1—3.16, 4—3.15, 11—3.14
	16	20	1—4.13, 3—4.12
		40	1—5.21, 27—5.20
		60	1—6.30, 2—5.24, 7—5.23
		80	1—6.27, 5—6.24
		100	1—5.26, 16—5.25, 38—5.24
	24	20	1—6.21, 4—5.19, 9—5.18
		40	1—7.33, 2—6.22
		60	1—7.32, 4—7.31, 6—6.27
		80	1—8.39, 9—7.37, 19—7.36, 23—7.35, 39—7.34
		100	1—8.40, 2—7.36, 16—7.35, 29—6.34
	32	20	1—7.24, 7—6.21, 19—6.20
		40	1—7.30, 6—7.28, 19—7.27
		60	1—8.41, 4—8.38, 23—8.37, 25—8.33
		80	1—9.46, 5—9.44, 6—9.41, 9—8.40, 31—8.39, 54—8.38
		100	1—9.50, 2—9.47, 3—9.46, 5—8.42, 7—8.40, 11—8.39

Сопоставление данной таблицы с таблицей 3.5.7 показывает, что по качеству получаемых решений рандомизированная программа минсоб вполне может конкурировать не только с программой прикрафсо, но

Таблица 3.5.12

$\sigma(X)$	$\sigma(M^1)$	$\sigma(M^0)$	Последовательность улучшений					
20	20	50	1—5.22,	2—5.21,	12—5.20			
		100	1—7.34,	3—6.32,	8—6.31,	12—6.30,	29—6.28	
		150	1—7.41,	2—7.40,	16—7.39			
		200	1—6.37,	6—6.36				
		250	1—7.47,	11—7.46				
	40	50	1—9.37,	54—8.34				
		100	1—10.54,	17—10.52,	18—10.51,	21—10.50		
		150	1—12.73,	3—11.67,	4—11.61,	15—11.60,	16—10.57,	23—10.56
		200	1—12.74,	2—12.73,	7—12.72,	9—11.68		
		250	1—13.83,	4—13.82,	5—12.78,	12—12.76,	14—12.75,	25—11.73
	60	50	1—12.56,	12—12.53,	17—12.52,	20—12.51,	25—11.52,	57—11.48
		100	1—14.76,	2—13.69,	4—13.66,	10—12.61		
		150	1—15.90,	2—15.86,	3—14.79,	43—14.78		
		200	1—17.109,	2—16.104,	3—16.99,	6—15.91		
		250	1—18.117,	3—17.111,	4—17.107,	6—16.107,	9—16.105,	60—16.104
	80	100	1—18.100,	2—18.99,	5—18.96,	7—18.95,	8—17.95,	17—17.87
		150	1—20.120,	2—20.115,	3—19.113,	6—19.110,	12—18.105	
		200	1—18.111,	18—18.96				
		250	1—21.138,	3—20.131,	5—20.129,	14—20.126		
	100	50	1—15.65,	4—15.64,	62—15.63			
		100	1—22.120,	2—21.119,	3—21.115,	5—21.112,	10—21.111,	15—20.105
		200	1—19.115,	7—19.114,	56—19.113,	59—18.108		
		400	1—22.160,	2—22.157,	3—21.152,	5—21.148,	37—21.146	
	200	200	1—34.212,	2—33.207,	3—32.197,			
		400	1—38.274,	3—38.266				

и с программой крафсо, получающей всегда минимальные по числу членов днф. В то же время быстродействие программы минсоб остается довольно высоким даже при ее многократной реализации.

Заметим, что рандомизированная программа представляет определенные удобства в том отношении, что становится возможным регулировать отпускаемое на процесс решения время. Учитывая особенности внешней программы, в которой рассматриваемая рандомизированная программа используется как подпрограмма, можно решить в каждой конкретной ситуации, сколько времени разумно предоставить данной подпрограмме, и удовлетворяться тем лучшим из решений, которые она за это время сумеет найти.

В таблице 3.5.12 приведены результаты дополнительных исследований эффективности рандомизации программы минсоб. Так же, как и ранее, каждая строка таблицы представляет собой выборку из 64 решений, полученных программой минсоб при решении одной и той же задачи. Исключением являются две последние строки — в этих случаях на поиск решений уходило много времени, в связи с чем было получено лишь по 6 решений.

§ 6. Решение системы логических уравнений

Постановка задачи. Пусть система уравнений $F(x)$ задана в виде

$$f_1(x),\ f_2(x),\ \ldots,\ f_m(x),$$

где каждое из выражений $f_i(x)$ представляет собой некоторую суперпозицию операций $\wedge$, $\vee$, $-$, $\oplus$, $=$ и $\rightarrow$. Будем считать, что решить эту систему — значит найти ее *корни* — значения вектора x, удовлетворяющие всей системе в целом, то есть каждому из входящих в нее уравнений $f_i(x)$.

Иногда условия решаемой задачи требуют получения всех корней системы, иногда же достаточно найти лишь некоторые из них, в частности — один. В первом из этих случаев совокупность решений удобно представить в форме булевой функции $f(x)$, принимающей значе-

ние 1 на тех и только тех значениях вектора x, которые являются корнями системы, то есть свести систему к виду $f(x)=1$. Во втором случае полученные частные решения можно представить в виде некоторой булевой функции $g(x)$, являющейся импликантой функции $f(x)$, и выразить результат уравнением $g(x)=1$, все корни которого являются в то же время корнями рассматриваемой системы $F(x)$.

Каждое из уравнений системы $F(x)$ представляет некоторое условие, налагающее ограничения на допустимые значения вектора x. Поскольку все эти условия должны выполняться одновременно, система $F(x)$ эквивалентна одному уравнению

$$f_1(x) \wedge f_2(x) \wedge \ldots \wedge f_m(x),$$

то есть множество корней этого уравнения совпадает с множеством корней системы $F(x)$.

Как видно, записать такое уравнение несложно. В чем же заключаются трудности решения системы $F(x)$?

Задачу решения данной системы можно сформулировать как задачу реализации логического перемножения функций $f_1(x)$, $f_2(x)$, ..., $f_m(x)$. Существенными факторами, определяющими сложность решения этой задачи, являются форма представления булевых функций $f_1(x)$, $f_2(x)$, ..., $f_m(x)$ и форма, в которой требуется выразить результат.

При рассмотрении систем с малым числом переменных, то есть с малым числом n компонент в векторе x (например, при $n\leqslant 10$), задача решается довольно просто, однако с ростом n она становится далеко не тривиальной, требуя громадных затрат времени на решение, избежать которых в общем случае, по-видимому, невозможно.

Один из самых общих подходов к решению системы $F(x)$ связан с перебором всех значений вектора x и с вычислением соответствующих им значений для каждой из функций $f_1(x)$, $f_2(x)$, ..., $f_m(x)$. Если каждая из этих функций принимает значение 1 для некоторого значения вектора, то данное значение является корнем системы, в противном случае это значение отбрасывается.

Так последовательно отыскиваются все корни системы. Однако при больших n этот простой метод становится практически нереализуемым.

Следует учесть, что при любых практических постановках задач логического характера объем исходной информации, как правило, невелик. Это значит, например, что системы логических уравнений, решение которых представляет практический интерес, задаются булевыми функциями ограниченного класса, учитывая особенности которого, можно разработать достаточно эффективные алгоритмы решения систем.

Рассмотрим следующую частную, но достаточно важную постановку задачи.

Пусть система $F(x)$ задана в дизъюнктивной нормальной форме:

$$K_1 \equiv k_1^1 \vee k_1^2 \vee \ldots \vee k_1^{l_1} = 1,$$
$$K_2 \equiv k_2^1 \vee k_2^2 \vee \ldots \vee k_2^{l_2} = 1,$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$K_m \equiv k_m^1 \vee k_m^2 \vee \ldots \vee k_m^{l_m} = 1,$$

где через $k_i^j \equiv k_i^j(x)$ обозначена некоторая элементарная конъюнкция. Искомую функцию $f(x)$, представляющую все корни системы, требуется представить также в виде д н ф:

$$K \equiv k^1(x) \vee k^2(x) \vee \ldots \vee k^l(x) = 1.$$

Приведение системы $F(x)$ к дизъюнктивной нормальной форме обычно не связано с особыми затруднениями. Основные трудности возникают при раскрытии скобок в выражении

$$(k_1^1 \vee k_1^2 \vee \ldots \vee k_1^{l_1}) \wedge (k_2^1 \vee k_2^2 \vee \ldots \vee k_2^{l_2}) \wedge \ldots$$
$$\ldots \wedge (k_m^1 \vee k_m^2 \vee \ldots \vee k_m^{l_m})$$

то есть при логическом перемножении дизъюнктивных нормальных форм $K_1, K_2, \ldots, K_m$, приводящем к требуемой д н ф K.

Алгоритм последовательного раскрытия скобок. Пусть нам нужно решить следующую систему уравнений, заданных в дизъюнктивной нормальной форме:

$$K_1 \equiv a e \bar{j} \vee \bar{b} g \vee c \bar{h} \vee \bar{b} \bar{f} j = 1,$$
$$K_2 \equiv \bar{d} h \vee b e \vee a d \bar{g} h = 1,$$
$$K_3 \equiv \bar{b} e \bar{j} \vee c i = 1,$$
$$K_4 \equiv f j \vee \bar{a} \bar{c} d i \vee b g \bar{h} \vee c \bar{h} \vee a \bar{d} f = 1,$$
$$K_5 \equiv \bar{e} i \vee b f \bar{h} \vee d j \vee \bar{d} \bar{f} h = 1,$$
$$K_6 \equiv a \bar{c} \bar{i} \vee d g \bar{j} \vee \bar{g} j = 1.$$

Можно просто рассмотреть все комбинации из шести элементарных конъюнкций, взятых по одной из каждого уравнения, логически перемножить их, выбросить те, которые обращаются тождественно в нуль, привести подобные и произвести всевозможные поглощения и склеивания с целью упрощения результата. Результатом здесь будет служить днф искомой булевой функций $f(x)$, представляющей все корни заданной системы уравнений.

Однако общее число рассматриваемых при этом комбинаций может оказаться весьма большим, будучи равно произведению чисел членов в перемножаемых днф. В данном примере оно равно $4 \cdot 3 \cdot 2 \cdot 5 \cdot 4 \cdot 3 = 1440$, что оказывается не меньше, чем 1024 — число различных значений вектора x, которые пришлось бы перебирать по упомянутому выше простейшему методу нахождения корней системы.

Можно воспользоваться тем обстоятельством, что многие пары конъюнкций, принадлежащих различным уравнениям системы, находятся в отношении ортогональности, и существенно сократить перебор, производимый при логическом перемножении днф.

Рассмотрим в связи с этим следующий простой, но достаточно эффективный алгоритм, представив для удобства исходную систему уравнений в матричной

форме:

$$
\begin{array}{c}
\quad a\ b\ c\ d\ e\ f\ g\ h\ i\ j \\
K_1\left[\begin{array}{cccccccccc}
1 & - & - & - & 1 & - & - & - & - & 0 \\
- & 0 & - & - & - & - & 1 & - & - & - \\
- & - & 1 & - & - & - & - & 0 & - & - \\
- & 0 & - & - & - & 0 & - & - & - & 1
\end{array}\right]\begin{array}{c}1\\2\\3\\4\end{array} \\
K_2\left[\begin{array}{cccccccccc}
- & - & - & 0 & - & - & - & 1 & - & - \\
- & 1 & - & - & 1 & - & - & - & - & - \\
1 & - & - & 1 & - & - & 0 & 1 & - & -
\end{array}\right]\begin{array}{c}1\\2\\3\end{array} \\
K_3\left[\begin{array}{cccccccccc}
- & 0 & - & - & 1 & - & - & - & - & 0 \\
- & - & 1 & - & - & - & - & - & 1 & -
\end{array}\right]\begin{array}{c}1\\2\end{array} \\
K_4\left[\begin{array}{cccccccccc}
- & - & - & - & - & 1 & - & - & - & 1 \\
0 & - & 0 & 1 & - & - & - & - & 1 & - \\
- & 1 & - & - & - & - & 1 & 0 & - & - \\
- & - & 1 & - & - & - & - & 0 & - & - \\
1 & - & - & 0 & - & 1 & - & - & - & -
\end{array}\right]\begin{array}{c}1\\2\\3\\4\\5\end{array} \\
K_5\left[\begin{array}{cccccccccc}
- & - & - & - & - & 0 & - & - & 1 & - \\
- & 1 & - & - & - & 1 & - & 0 & - & - \\
- & - & - & 1 & - & - & - & - & - & 1 \\
- & - & - & 0 & - & 0 & - & 1 & - & -
\end{array}\right]\begin{array}{c}1\\2\\3\\4\end{array} \\
K_6\left[\begin{array}{cccccccccc}
1 & - & 0 & - & - & - & - & - & 0 & - \\
- & - & - & 1 & - & - & 1 & - & - & 0 \\
- & - & - & - & - & - & - & 0 & - & 1
\end{array}\right]\begin{array}{c}1\\2\\3\end{array}
\end{array}
$$

Реализуемый алгоритмом процесс эквивалентен последовательному раскрытию скобок в выражении

$$(a e \bar{j} \vee \bar{b} g \vee c \bar{h} \vee \bar{b} \bar{f} j) \wedge (\bar{d} h \vee b e \vee a d \bar{g} h) \wedge$$
$$\wedge (\bar{b} e \bar{j} \vee c i) \wedge (f j \vee \bar{a} \bar{c} d i \vee b g \bar{h} \vee \bar{c} h \vee a \bar{d} f) \wedge$$
$$\wedge (\bar{e} i \vee b f \bar{h} \vee d j \vee \bar{d} \bar{f} h) \wedge (a \bar{c} \bar{i} \vee d g \bar{j} \vee \bar{g} j).$$

Сначала перемножаются первые две д н ф, обозначенные нами через K_1 и K_2. Эта процедура распадается на $4 \cdot 3 = 12$ элементарных актов перемножения конъюнкций, принадлежащих данным д н ф. В векторном представлении эти операции выполняются чрезвычайно про-

сто, например,

$$\begin{array}{l} 1\ ---\ 1\ ----\ 0 \quad k_1^1 \\ ---\ 0\ ---\ 1\ --\ \ k_2^1 \\ \hline 1\ --\ 0\ 1\ --\ 1\ -\ 0 \quad k_1^1 \wedge k_2^1 \end{array}$$

$$\begin{array}{l} 1\ ---\ 1\ ----\ 0 \quad k_1^1 \\ -\ 1\ --\ 1\ ------ \quad k_2^2 \\ \hline 1\ 1\ --\ 1\ ----\ 0 \quad k_1^1 \wedge k_2^2 \end{array}$$

Некоторые пары дают при перемножении тождественный нуль, например, пара

$$\begin{array}{l} -\ 0\ ----\ 1\ ---\ k_1^2 \\ -\ 1\ --\ 1\ ------\ k_2^2 \end{array}$$

В результате получается следующая матрица, каждая строка которой представляет элементарную конъюнкцию, полученную при перемножении указанных справа членов перемножаемых д н ф K_1 и K_2:

a	b	c	d	e	f	g	h	i	j	K_1	K_2
1	—	—	0	1	—	—	1	—	0	1	1
1	1	—	—	1	—	—	—	—	0	1	2
1	—	—	1	1	—	0	1	—	0	1	3
—	0	—	0	—	—	1	1	—	—	2	1
—	1	1	—	1	—	—	0	—	—	3	2
—	0	—	0	—	0	—	1	—	1	4	1
1	0	—	1	—	0	0	1	—	1	4	3

Как видно, относительное число компонент со значением «—» уменьшается. Это обстоятельство является благоприятным для дальнейших вычислений, так как при перемножении полученного результата на следующую д н ф относительное число пар, не дающих при перемножении тождественный нуль, также будет уменьшаться.

Не приводя в целом результат очередного этапа (перемножение полученной только что д н ф на K_3), перечислим те трехэлементные комбинации, по одной конъюнкции из д н д K_1, K_2 и K_3, логическое произведение

которых отлично от нуля:

K_1 1 1 1 1 1 2 2 3 4 4
K_2 1 1 2 3 3 1 1 2 1 3
K_3 1 2 2 1 2 1 2 2 2 2

На дальнейших этапах выделяются следующие комбинации:

K_1 1 1 1 1 1 2 2 2 3 3 3
K_2 1 1 2 2 2 1 1 1 2 2 2
K_3 1 2 2 2 2 1 2 2 2 2 2
K_4 5 5 3 4 5 5 1 5 1 3 4 5

K_1 1 1 1 2 2 3 3 3 3 3 3
K_2 2 2 2 1 1 2 2 2 2 2 2
K_3 2 2 2 2 2 2 2 2 2 2 2
K_4 3 4 5 1 5 1 1 3 3 4 5
K_5 2 2 2 1 1 2 3 2 3 2 2

и, наконец,

K_1 1 1 3 3 3 3 3 3 3
K_2 2 2 2 2 2 2 2 2 2
K_3 2 2 2 2 2 2 2 2 2
K_4 3 4 1 1 3 4 4 4 5
K_5 2 2 2 3 2 2 2 3 2
K_6 2 2 3 3 2 2 3 3 3

Девять перечисленных комбинаций порождают матрицу

$$
\begin{array}{c}
\begin{matrix} a & b & c & d & e & f & g & h & i & j \end{matrix} \\
\left[\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
- & 1 & 1 & - & 1 & 1 & 0 & 0 & 1 & 1 \\
- & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
- & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
- & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
- & 1 & 1 & - & 1 & 1 & 0 & 0 & 1 & 1 \\
- & 1 & 1 & 1 & - & 0 & 0 & 1 & 1 & \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1
\end{array}\right]
\end{array}
$$

упрощение которой приводит нас к окончательному результату:

$$\begin{matrix} a & b & c & d & e & f & g & h & i & j \end{matrix}$$
$$\begin{bmatrix} - & 1 & 1 & - & 1 & 1 & 0 & 0 & 1 & 1 \\ - & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ - & 1 & 1 & 1 & 1 & - & 0 & 0 & 1 & 1 \end{bmatrix}$$

Теперь легко перейти к алгебраической форме найденной функции $f(x)$:

$$f(x) = bcef\bar{g}\bar{h}ij \vee bcdef g\bar{h}i\bar{j} \vee bcde\bar{g}\bar{h}ij =$$
$$= bce\bar{h}i(f\bar{g}j \vee dfg\bar{j} \vee d\bar{g}j),$$

представляющей в компактном виде все восемь корней рассмотренной системы уравнений.

Л-программа. Приведем Л-программу, реализующую описанный процесс последовательного раскрытия скобок при перемножении нескольких днф (но не проводящую последующие возможные упрощения внутри полученной днф — для этого может быть использована какая-либо из других, уже известных нам программ).

Решение системы логических уравнений путем последовательного раскрытия скобок в перемножаемых днф — пораскоб

1. Задана система логических уравнений в виде секционированной троичной матрицы $\boldsymbol{A}$, секции которой представляют днф отдельных уравнений. Требуется найти множество всех корней системы и выразить его в виде функции $f(x)$, представив ее также в дизъюнктивной нормальной форме, посредством некоторой троичной матрицы $\boldsymbol{B}$.

2. Внешние операнды:

$\alpha\beta_к :: \boldsymbol{A}$ в (10,01,00)-коде,

$\gamma_к$ — разделительный комплекс, i-й элемент которого задает конечную границу i-х секций в комплексах α и β посредством номера тех первых элементов этих комплексов, которые следуют за выделяемыми секциями, представляющими i-е уравнение системы,

$\delta e_к^+ :: \boldsymbol{B}$ в (10,01,00)-коде,

$\zeta_ч$ — дополнительный выходной полюс, соответствующий нехватке памяти при реализации программы.

Задаются: a_α, a_β, a_γ, a_δ, a_ε, b_α, b_β, b_γ, b_δ, b_ε.

Внутренние операнды: —, f, —.

Примечание: следует учитывать, что фиксация промежуточных результатов осуществляется в комплексах δ и ε, начальный объем которых задается условно значением b_δ — это значение должно быть поэтому достаточно большим, в противном случае памяти для хранения промежуточных результатов может не хватить и будет реализован аварийный выход по дополнительному выходу ζ.

4.

* 001 300

§ 0 $\overline{\bigcirc}\, d \bigcirc \delta_0 \bigcirc \varepsilon_0\, 1 \Rightarrow c \bigcirc e$

§ 1 $c \circ\!\!\rightarrow 5 \;\triangle\; d \oplus b_\gamma \circ\!\!\rightarrow 5$

$e - 1 \Rightarrow f\; \gamma_d + 1 \Rightarrow e\; b_\delta \Rightarrow a$

§ 2 $\overline{\triangle}\, c\; \overline{\triangle}\, a\; \delta_c \Rightarrow \delta_a\, \varepsilon_c \Rightarrow \varepsilon_a\, c \mapsto 2\, \overline{\triangle}\, a$

§ 3 $\triangle\; a \oplus b_\delta \circ\!\!\rightarrow 1\, f \Rightarrow b$

§ 4 $\triangle\; b \oplus e \circ\!\!\rightarrow 3$

$\delta_a \vee \alpha_b \Rightarrow \delta_c\, \varepsilon_a \vee \beta_b \Rightarrow \varepsilon_c \wedge \delta_c \mapsto 4$

$\triangle\; c \oplus a \circ\!\!\rightarrow \zeta \rightarrow 4$

§ 5 $c \Rightarrow b_\delta \Rightarrow b_\varepsilon$.

Лексикографический перебор комбинаций. Укажем на один из недостатков предложенного метода перемножения днф: может оказаться, что на некотором этапе раскрытия очередной пары скобок число членов промежуточной днф будет слишком большим и произойдет так называемое переполнение памяти. Предложим в связи с этим видоизмененный метод перемножения днф, полезный также в тех случаях, когда требуется нахождение не всех корней системы, а лишь нескольких из них или когда их удобно получать последовательно, один за другим.

Суть предлагаемого метода заключается в *лексикографическом переборе* комбинаций из конъюнкций, принадлежащих различным днф системы. Как именно производится этот перебор, лучше всего показать на при-

мере решения уже знакомой нам задачи, представив начало реализуемого при этом процесса таблицей 3.6.1.

Таблица 3.6.1

Перебираемые комбинации

K_1	1																			
K_2	1										2									
K_3	1					2					1	2								
K_4	1	2	3	4	5	1	2	3	4	5	1	2	3				4			
K_5			1	2	3	4				1	2	3	4		1	2		3	4	1	2
K_6														1	2	3			1	2
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	0	0	0	0	+

Каждая очередная комбинация представляется одним из столбцов данной таблицы. Например, первая комбинация образуется так: сначала берется конъюнкция k_1^1, затем она умножается на k_2^1. Результат отличен от нуля, поэтому он умножается на k_3^1, а затем на k_4^1. Здесь получается тождественный нуль, что отмечается в нижней строке таблицы. Расширение данной комбинации путем добавления в нее некоторых элементов из K_5 и K_6 уже смысла не имеет, поэтому мы переходим к построению следующей комбинации, варьируя выбор элемента из K_4. Конъюнкция $k_1^1 \wedge k_2^1 \wedge k_3^1 \wedge k_4^2$ также равна тождественно нулю; она фиксируется вторым столбцом таблицы, также отмечаемым нулем в нижней строке. Чтобы не загромождать таблицу, мы задаем ее в дифференциальном виде, обращая внимание лишь на изменения в текущей комбинации.

На участке вычислительного процесса, отраженном данной таблицей, находятся две комбинации, соответствующие некоторым корням решаемой системы и отмеченные символом + в нижней строке. Впрочем, оказывается, что эти комбинации соответствуют одному и тому же корню 1 1 1 1 1 1 0 1 0.

Упрощение системы уравнений. Решаемая задача может быть сформулирована как вариант задачи о нахождении некоторых совокупностей взаимно совместимых элементов. Специфичным для этого варианта является условие, что в искомые совокупности

должно входить ровно по одному элементу из каждого множества некоторой заданной системы множеств. В данном случае роль элементов играют элементарные конъюнкции, принадлежащие перемножаемым д н ф, а отношением несовместимости служит отношение ортогональности конъюнкций.

Возвращаясь к рассматриваемому примеру, построим булеву матрицу отношения ортогональности для конъюнкций, принадлежащих различным д н ф системы. Коль скоро это отношение симметрично, представим матрицу в урезанном виде. В этой матрице содержится вся информация, необходимая для нахождения искомых совокупностей взаимно пересекающихся конъюнкций. Пары взаимно ортогональных конъюнкций отмечены в данной матрице единицей, нули соответствуют парам неортогональных конъюнкций.

K_2 123	K_3 12	K_4 12345	K_5 1234	K_6 123		
000	00	11000	1010	001	1	
011	00	00100	0100	001	2	K_1
101	00	01000	0001	100	3	
010	10	10101	0100	010	4	
	00	01110	0110	010	1	
	10	00000	1000	000	2	K_2
	00	01111	0101	010	3	
		10100	1110	001	1	K_3
		01000	0000	100	2	
			0001	010	1	
			0001	100	2	
			0001	001	3	K_4
			0001	100	4	
			0011	010	5	
				100	1	
				000	2	K_5
				010	3	
				101	4	

Сформулируем следующее правило, справедливость которого очевидна: *если некоторая конъюнкция, принадлежащая одной из д н ф системы, ортогональна каждой из конъюнкций, принадлежащих некоторой другой д н ф, этой же системы, то указанная конъюнкция может быть выброшена из содержащей ее д н ф, так как множество корней системы при этом не изменяется.*

В рассматриваемом примере такой конъюнкцией оказывается k_5^4, ортогональная каждой из конъюнкций, образующих д н ф K_4. После выбрасывания этой конъюнкции из д н ф K_5 оказывается, что можно выбросить и конъюнкцию k_3^1, ортогональную теперь каждой из конъюнкций, оставшихся в д н ф K_5. Затем по тем же соображениям выбрасываются конъюнкции k_4^2 и k_6^1, после чего матрица отношения ортогональности принимает следующий вид:

K_2	K_3	K_1	K_5	K_6	
1 2 3	2	1 3 4 5	1 2 3	2 3	
0 0 0	0	1 0 0 0	1 0 1	0 1	1
0 1 1	0	0 1 0 0	0 1 0	0 1	2
1 0 1	0	0 0 0 0	0 0 0	0 0	3 K_1
0 1 0	0	1 1 0 1	0 1 0	1 0	4
	0	0 1 1 0	0 1 1	1 0	1
	0	0 0 0 0	1 0 0	0 0	2 K_2
	0	0 1 1 1	0 1 0	1 0	3
		0 0 0 0	0 0 0	0 0	2 K_3
			0 0 0	1 0	1
			0 0 0	0 1	3
			0 0 0	0 0	4 K_1
			0 0 1	1 0	5
				0 0	1
				0 0	2 K_5
				1 0	3

Здесь сформулированное правило становится неприменимым. Однако можно продолжить процесс упрощения рассматриваемой системы, воспользовавшись следующим правилом, также довольно очевидным: *если все конъюнкции одной из д н ф, неортогональные некоторой конъюнкции k_i^j, принадлежащей другой д н ф,*

в то же время ортогональны некоторой конъюнкции k_p^q, входящей в третью днф рассматриваемой системы, то конъюнкции k_l^l и k_p^q можно условно считать взаимно ортогональными, внеся соответствующие коррекции в матрицу отношения ортогональности.

Например, конъюнкция k_1^4 пересекается с конъюнкциями k_2^1 и k_2^3 из днф K_2, но ортогональна конъюнкции k_2^2. В свою очередь, как k_2^1, так и k_2^3 ортогональны конъюнкциям k_4^3 и k_4^4. Следовательно, если мы будем условно считать, что конъюнкция k_1^4 также ортогональна конъюнкциям k_4^3 и k_4^4, и, исходя уже из этого, продолжать решение задачи, то множество находимых в конце концов корней системы от этого не изменится. С другой стороны, появление новых единиц в матрице отношения ортогональности приводит к новым возможностям дальнейшего упрощения системы. Так, в результате данной операции оказывается, что конъюнкция k_1^4 становится ортогональной каждой конъюнкции из днф K_4 и, следовательно, ее можно исключить из системы.

Рассмотрение отношения между конъюнкцией k_1^1 и днф K_6 аналогичным образом приводит нас к выбрасыванию конъюнкции k_2^3. Затем, после некоторого чередования двух сформулированных правил, последовательно удаляются конъюнкции k_1^2, k_2^1 и k_5^1, в результате чего матрица отношения ортогональности сокращается до

K_2	K_3	K_4	K_5	K_6		
2	2	1345	23	23		
0	0	1001	01	01	1	K_1
0	0	0000	00	00	3	
0		0000	00	00	2	K_2
		0000	00	00	2	K_3
			00	10	1	
			01	01	3	K_4
			00	00	4	
			01	10	5	
				00	2	K_5
				10	3	

где дальнейшие упрощения с помощью тех же правил становятся невозможными.

Важно отметить, что безразлично, в каком порядке производятся упрощения, конечный результат, то есть некоторый неупрощаемый остаток матрицы отношения ортогональности, всегда будет одним и тем же. Дальнейшее решение задачи можно осуществить ранее рассмотренным методом, последовательно раскрывая скобки в выражении

$$(a e \bar{j} \vee c \bar{h}) \wedge b e \wedge c i \wedge (f j \vee b g \bar{h} \vee c \bar{h} \vee a \bar{d} f) \wedge$$
$$\wedge (b f \bar{h} \vee d j) \wedge (d g \bar{j} \vee \bar{g} j),$$

соответствующем упрощенной системе логических уравнений. Производимый при этом перебор оказывается существенно меньшим в сравнении с исходной, неупрощенной системой.

Поиск одного из корней системы. Перед решением этой задачи систему $F(\boldsymbol{x})$ удобно привести к виду:

$$K_1 \equiv k_1^1 \vee k_1^2 \vee \ldots \vee k_1^{l_1} = 0,$$
$$K_2 \equiv k_2^1 \vee k_2^2 \vee \ldots \vee k_2^{l_2} = 0,$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$K_m \equiv k_m^1 \vee k_m^2 \vee \ldots \vee k_m^{l_m} = 0.$$

Очевидно, что корнями этой системы будут служить такие значения вектора $\boldsymbol{x}$, которые будут ортогональны любой конъюнкции любой из д н ф $K_1, K_2, \ldots, K_m$.

Данная система легко приводится к одному уравнению того же типа:

$$K \equiv k^1 \vee k^2 \vee \ldots \vee k^l =$$
$$= k_1^1 \vee k_1^2 \vee k_1^3 \vee \ldots \vee k_1^{l_1} \vee k_2^1 \vee k_2^2 \vee \ldots \vee k_2^{l_2} \vee \ldots$$
$$\ldots \vee k_m^1 \vee k_m^2 \vee \ldots \vee k_m^{l_m} = 0,$$

поскольку операция логического сложения нескольких д н ф производится значительно проще, чем операция логического перемножения д н ф такой же сложности.

Этим самым задача сводится к уже знакомой нам проверке некоторой троичной матрицы на вырожденность. В данном случае эта троичная матрица будет представлять днф K; если матрица окажется вырожденной, то это будет означать, что рассматриваемая система не имеет корней, если же будет найден некоторый вектор, ортогональный каждой из строк матрицы, то он будет представлять собой некоторое частичное решение системы.

Таким образом, здесь с успехом может быть использована программа вытрома.

§ 7. Задачи теории графов

Многие задачи логического характера удобно формулировать в терминах теории графов, позволяющей обозримо представлять сущность решаемых задач. В этом параграфе мы рассмотрим некоторые из таких задач. В дальнейшем мы увидим, что алгоритмы их решения окажутся чрезвычайно полезными в теории синтеза дискретных автоматов.

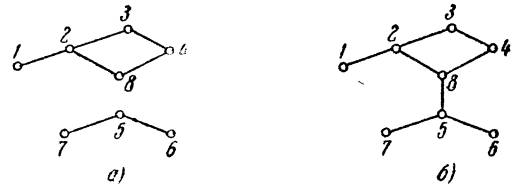


Рис. 3.7.1.

Определение связности графа. На рис. 3.7.1 даны примеры двух симметрических, или неориентированных графов. Один из них (*б*) является *связным*, то есть для любой пары вершин этого графа существует связывающая их цепь. Другой (*а*) *несвязен:* например, не существует цепи, соединяющей вершины *1* и *7*. В данном случае это свойство графа представляется очевидным.

Полезно, однако, иметь алгоритм определения связности графа. Довольно эффективным оказывается последовательный перебор пар *смежных* (то есть соединенных некоторым ребром) вершин i и j, сопровождае-

мый следующей процедурой: в граф добавляются ребра, связывающие вершину j со всеми вершинами, смежными вершине i. Если исходный граф связен, и только в этом случае, в результате реализации данного алгоритма будет получен *полный граф*, то есть такой, в котором каждые две вершины являются смежными.

Информацию о графе удобно задавать в виде матрицы смежности: квадратной булевой матрицы $\boldsymbol{L}$, элемент l_i^j которой обладает значением 1, если вершины i и j оказываются смежными, и обладающий значением 0 — в противном случае.

Например, представленным на рис. 3.7.1 графам будут соответствовать следующие матрицы смежности:

а)
$$\begin{vmatrix} 0\,1\,0\,0\,0\,0\,0\,0 \\ 1\,0\,1\,0\,0\,0\,0\,1 \\ 0\,1\,0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,1\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0\,0\,0 \end{vmatrix}$$

б)
$$\begin{vmatrix} 0\,1\,0\,0\,0\,0\,0\,0 \\ 1\,0\,1\,0\,0\,0\,0\,1 \\ 0\,1\,0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,1\,1 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,1\,0\,1\,1\,0\,0\,0 \end{vmatrix}$$

Добавление в граф новых ребер будет отражаться сменой значений некоторых элементов матрицы смежности с 0 на 1. В случае связности исходного графа реализация алгоритма приведет к тому, что все элементы матрицы примут значение 1, в противном случае в каждой ее строке будут присутствовать нули. В рассматриваемых примерах полученные матрицы будут иметь следующий вид:

а)
$$\begin{vmatrix} 1\,1\,1\,1\,0\,0\,0\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \\ 0\,0\,0\,0\,1\,1\,1\,0 \\ 0\,0\,0\,0\,1\,1\,1\,0 \\ 0\,0\,0\,0\,1\,1\,1\,0 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{vmatrix}$$

б)
$$\begin{vmatrix} 1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 \\ 1\,1\,1\,1\,1\,1\,1\,1 \end{vmatrix}$$

Описанный алгоритм весьма компактно представляется Л-программой.

Анализ симметрического графа на связность — а н с и г р а с

1. Требуется определить, связен ли симметрический граф, заданный матрицей смежности $\boldsymbol{L}$.

2. Внешние операнды:

$\alpha_{ч}$ — дополнительный выходной полюс, соответствующий обнаружению несвязности графа,

$\beta_{к}^{-} :: \boldsymbol{L}$,

$\gamma_{п} :: \Psi(F)$, где F — множество вершин графа.

Задаются: a_{β}, b_{β}.

Размерность: $\beta\gamma$.

Внутренние операнды: $\boldsymbol{b}$, b, —.

4.

* **001 015** $(\beta\, \boldsymbol{a}\, \boldsymbol{b})$

§ 0 $\overline{\bigcirc}\, b$

§ 1 $\triangle\, a \oplus b_{\beta} \circ\!\!\rightarrow 3\, \beta_{b} \Rightarrow \boldsymbol{a} \Rightarrow \boldsymbol{b}$

§ 2 $\boldsymbol{a}\, \dot{\mathrm{X}}\, 1\, \boldsymbol{a}\, \boldsymbol{b} \vee \beta_{a} \Rightarrow \beta_{a} \rightarrow 2$

§ 3 $\beta_{0} \oplus \gamma \mapsto \alpha$.

Анализ графа на двусвязность. Симметрический граф называется *k-связным*, если он остается связным при удалении любых $k-1$ ребер. Решение задачи определения числа k связности произвольного симметрического графа сопряжено с большими вычислительными трудностями. Для общего случая можно предложить перебор подмножеств ребер графа с проверкой, являются ли эти подмножества сечениями, то есть не влечет ли их удаление из графа потерю его связности. Очевидно, что множество всех сечений есть *выпуклое множество по отношению включения*, то есть если некоторое подмножество является сечением, то и любое содержащее его множество тоже есть сечение. Учитывая это обстоятельство, производимый перебор подмножеств можно несколько сократить. Перебор можно сократить еще более, поскольку число связности графа определяется минимальным (по мощности) сечением: можно исключить из рассмотрения все подмножества мощности не менее p, где p — минимальная мощность уже найденных сечений.

Наибольшего сокращения вычислительного процесса удается добиться, решая задачу в частных случаях: анализируя граф на 1-связность (эту задачу мы уже рассмотрели) или на 2-связность. Рассмотрим вторую задачу.

Будем рассматривать симметрический граф G как совокупность вершин, образующих множество F, и ребер, образующих множество $U: G \equiv (F, U)$. Будем говорить, что граф G 2-связен на подмножестве вершин $C \subseteq F$, если 2-связен его подграф $(C, U_{(C)})$, образуемый вершинами из множества C и связывающими их ребрами графа G.

Предложим следующий алгоритм, идея которого заключается в последовательном расширении множества C, на котором устанавливается 2-связность графа (F, U), за счет включения в множество C вершин, входящих в элементарные цепи частичного графа $(F, U \setminus U_{(C)})$, замыкающиеся на C, то есть такие, граничные вершины которых принадлежат C. При этом предполагается, что $\sigma(F) \geqslant 2$.

Для удобства разобьем алгоритм на пять различных этапов. На этапе 1 находится ребро, инцидентное множеству C, то есть связывающее некоторую вершину из множества C с некоторой вершиной из множества $F \setminus C$.

На этапе 2 имитируется движение по некоторой элементарной цепи, началом которой служит найденное на этапе 1 ребро и которая проходит вне C. При этом движении матрица смежности $\boldsymbol{L}$, представляющая исследуемый граф, изменяется так, чтобы исключить возможность обратных движений по уже пройденным ребрам: некоторые ее элементы меняют значение с 1 на 0. Собственно говоря, эта матрица уже перестает быть матрицей смежности, а представляет теперь лишь возможности последующих путешествий по исследуемому графу. Пройденные вершины включаются в множество B — множество вершин графа (F, U), подвергнутых анализу, начинающемуся с рассмотрения нулевой вершины (допустим, что нумерация вершин начата с 0).

Как только продолжение элементарной цепи вне множества C становится невозможным, начинается реализация этапов 3 и 4, на которых выясняется, существует

ли в графе (F, U), в котором запрещены возвратные движения по уже пройденным ребрам (такой граф можно рассматривать уже как ориентированный), путь, связывающий конечную вершину пройденной цепи с множеством C.

При наличии такого пути реализуется этап 5, на котором все вершины, принадлежащие множеству B, включаются в множество C. Если при этом множества

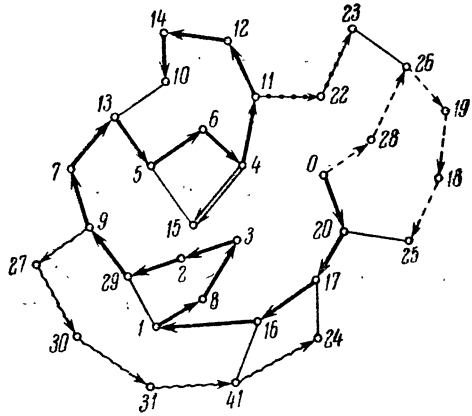


Рис. 3.7.2.

C и F совпадут, двусвязность графа G можно считать установленной и прекратить реализацию алгоритма, в противном случае вновь реализуется этап 1 и т. д. Отсутствие 2-связности выявляется, если при реализации этапа 2 не находится ни одного ребра, инцидентного множеству C, или если при реализации последних этапов устанавливается отсутствие искомого пути.

Работа алгоритма проиллюстрирована на примере анализа графа, представленного на рис. 3.7.2. Она выражается в последовательном включении в множество C тех вершин, через которые проходят элементарные цепи, получаемые при реализации этапа 2 и обозначаемые на рисунке следующими линиями: а) жирной, б) пунктирной, в) двойной, г) волнистой, д) точечной.

Л-программа. Приведем соответствующую Л-программу.

Анализ симметрического графа на двусвязность — анеиграсдва

1. Требуется определить, является ли двусвязным граф, заданный матрицей смежности L.

2. Внешние операнды:

$\alpha_ч$ — дополнительный выходной полюс, соответствующий обнаружению отсутствия двусвязности графа,

$\beta_к^- :: L$,

$\gamma_п :: \Psi(F)$, где F — множество вершин графа.

Задаются: a_β, b_β.

Размерность: $\beta\gamma$.

Внутренние операнды: e, b, —.

3. Номера предложений Л-программы совпадают с номерами описанных выше этапов алгоритма. В предложении 0 начинается построение множества C: в него включается вершина 0. Изменяемые множества B и C представляются текущими значениями переменных b и c: $b :: \| \{B\} \ni F \|$, $c :: \| \{C\} \ni F \|$.

4.

* 001 016 ($\beta\, a\, b\, c\, d\, e$)

§ 0 $c_0 \Rightarrow b \to 5$

§ 1 $a \dot{\mathrm{X}} \alpha c \urcorner \wedge \beta_a \circ\!\!\to 1 \Rightarrow a$

§ 2 $a \Rightarrow b\, a \vdash \Rightarrow a\, c_a \vee b \Rightarrow b$

$c_b \oplus \beta_a \Rightarrow \beta_a\, b \urcorner \wedge \beta_a \Rightarrow a \mapsto 2$

$c_a \Rightarrow e \Rightarrow a \Rightarrow d$

§ 3 $a \dot{\mathrm{X}} 4\, a\, \beta_a \vee d \Rightarrow d \to 3$

§ 4 $d \wedge \mathrm{c} \mapsto 5\, e \urcorner \wedge d \Rightarrow a \circ\!\!\to \alpha$

$\vee\, e \Rightarrow e \to 3$

§ 5 $b \Rightarrow \mathrm{c} \Rightarrow a \oplus \gamma \mapsto 1$.

Нахождение множества достижимых вершин графа. Рассмотрим теперь ориентированный граф $G = (F, U)$, в котором через U обозначено множество дуг графа. Пусть B — некоторое подмножество вершин этого графа: $B \subseteq F$. Вершина p графа G называется *достижимой из* B, если существует путь, ведущий от какой-либо вершины, принадлежащей множе-

ству B, в вершину p. Рассмотрим задачу нахождения множества C всех достижимых из B вершин графа G. Алгоритм, представленный приводимой ниже Л-программой, достаточно прост. Он реализует постепенное расширение области достижимых вершин и прекращает работу, как только дальнейшее расширение становится невозможным.

Нахождение множества достижимых вершин ориентированного графа — экспансия

1. Матрицей смежности $\boldsymbol{L}$ задан ориентированный граф G, в котором выделено некоторое подмножество B его вершин. Требуется найти множество C всех достижимых из B вершин графа G.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{L}$,
$\beta_п :: \| \{B\} \ni F \|$,
$\gamma_п^+ :: \| \{C\} \ni F \|$.

Задаются: a_α, b_α.
Размерность: $\alpha\beta\gamma$.
Внутренние операнды: $\boldsymbol{b}$, a, —.

4.

$* \; 001\,117 \; (\beta \boldsymbol{a}\, \boldsymbol{b})$
§ 0 $\circ\gamma\circ \boldsymbol{b}\,\beta \Rightarrow \boldsymbol{a}$
§ 1 $\boldsymbol{a}\,\dot{\mathrm{X}}\,2\,a\,a_a \vee \gamma \Rightarrow \gamma \to 1$
§ 2 $\boldsymbol{b}\,\neg \wedge \gamma \Rightarrow \boldsymbol{a}\,\gamma \Rightarrow \boldsymbol{b}\,\boldsymbol{a} \mapsto 1.$

С примером работы алгоритма можно познакомиться по рисунку 3.7.3. На изображенном там графе тонкой линией обведено множество вершин B, а пунктиром — находимое данным алгоритмом множество C достижимых из B вершин графа.

Кратчайшие цепи в графе. Цепь, соединяющая две заданные в графе вершины и состоящая из минимального числа дуг, называется *кратчайшей*. Представляют практический интерес задачи, связанные с рассмотрением кратчайших цепей и решаемые следующими алгоритмами.

Локализация кратчайших цепей — локрацеп

1. Матрицей смежности $\boldsymbol{L}$ задан симметрический граф, из множества F вершин которого выделена совокуп-

ность B, а в ней, в свою очередь, две вершины p и q. Требуется найти совокупность вершин, лежащих на кратчайших цепях, связывающих вершины p и q и не проходящих вне B, и разбить их на классы d_i по

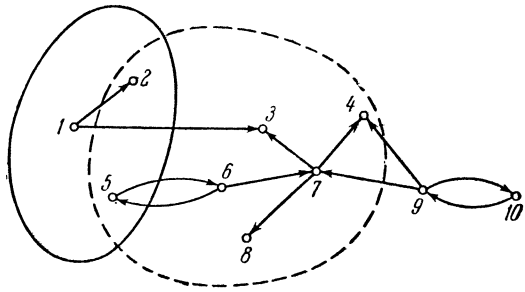


Рис. 3.7.3.

расстоянию i до вершины p, измеряемому числом ребер, получив таким образом множество $D=\{d_0, d_1, \ldots, d_k\}$.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{L}$,

$\beta_п :: \| \{B\} \ni F \|$,

$\gamma_п :: \| \{\{p, q\}\} \ni F \|$,

$\delta_к^+ :: \| D \ni F \|$.

Задаются: a_α, a_δ, b_α.

Размерность: $\alpha\beta\gamma\delta$.

Внутренние операнды: $\boldsymbol{d}$, b, —.

Примечание: признаком отсутствия цепи, лежащей в B и связывающей вершины p и q, служит принятие элементом b_δ значения 0.

3. При реализации алгоритма сначала находится ближняя окрестность вершины p в B, затем следующие по степени удаленности, пока какая-либо из них не окажется содержащей вершину q. Очевидно, что полученная последовательность классов из B мажорирует искомую последовательность $d_0, d_1, \ldots, d_k$. Уточнение искомых классов производится при аналогичном поиске окрестностей вершины q, начиная с ближней, путем пересечения этих окрестностей с соответствующими членами первой последовательности.

4.

```
      * 001 115 (a a b c d)
§ 0   γ ⌝+ 1 ⋀ γ ⇒ b ⊕ γ ⇒ a ○ b β ⇒ c ○ b_δ
§ 1   a ⇒ δ_b ⊕ c ⇒ c △ b ○ d ⇸ 3
      d ∧ c ⇒ a ○→ 5 ∧ b ○→ 1 ⇒ δ_b b + 1 ⇒ b_δ
§ 2   δ_b ⇒ a ⇸ 3 ⊼ b ○→ 5 d ∧ δ_b ⇒ δ_b → 2
§ 3   a X̌ 4 a a_a ∨ d ⇒ d → 3
§ 4   !
§ 5   .
```

5. Рассмотрим граф (рис. 3.7.4), в котором зачернены вершины, принадлежащие совокупности B, и обведены вершины p и q.

Анализируя этот граф, алгоритм локрацеп находит следующее множество D классов вершин, представляемое конечным значением комплекса δ:

$$\|D \ni F\| = \begin{bmatrix} 00000\ 10000\ 00000\ 00000\ 00000\ 00 \\ 00001\ 00000\ 01000\ 00100\ 00000\ 00 \\ 00000\ 01000\ 10000\ 00000\ 10001\ 00 \\ 00000\ 00001\ 00000\ 01001\ 00000\ 00 \\ 00000\ 00000\ 00001\ 00000\ 00000\ 00 \end{bmatrix}$$

Кратчайшие цепи, соединяющие вершины p и q и не проходящие вне B, отмечены на рисунке жирными линиями.

Получение всех кратчайших цепей в подграфе — кращеп

1. Матрицей смежности L задан симметрический граф, в котором длина цепей измеряется числом входящих в них ребер. В этом графе из множества F всех его вершин выделены две вершины p и q и некоторый подграф, заданный в виде множества $D=\{d_0, d_1, \ldots, d_k\}$ классов вершин, равноудаленных от вершин p (в класс d_i входят вершины, удаленные от вершины p на расстояние i) и находящихся на некоторых равновеликих цепях длиной k, соединяющих p и q и образующих множество S. Требуется получить множество S в явном виде, представив каждую из найденных цепей перечнем находящихся на ней вершин.

2. Внешние операнды:

$\alpha_к :: L$,

$\beta_к :: \| D \ni F \|$,

$\gamma_к$ — комплекс памяти, $\sigma(\gamma) = 2k\sigma(\alpha)$.

$\delta_к^+ :: \| S \ni^2 F \|$, где $s_i \ni^2 f_j$ означает, что **вершина** f_j находится на цепи s_i, то есть входит в некоторое ее ребро $\{f_j, f_t\}$.

Задаются: a_α, a_β, a_γ, a_δ, b_α, b_β.
Размерность: $\alpha\beta\gamma\delta$.
Внутренние операнды: b, c, —.

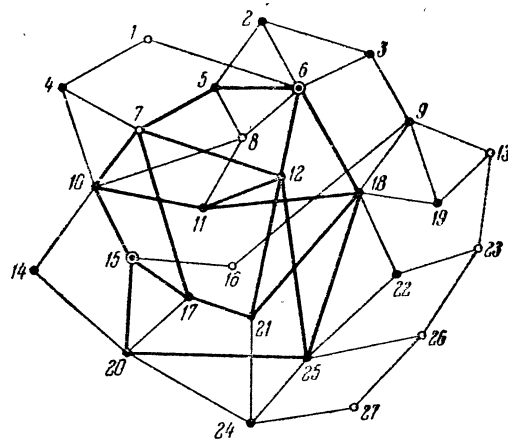


Рис. 3.7.4.

3. При получении искомых цепей производится перебор типа обхода дерева, начинающийся с вершины p.

4.

$$* \ 001 \ 114 \ (\alpha \, a \, b)$$

$$\S \ 0 \quad \circ c \, \beta_0 \Rightarrow b \, \beta_1 \Rightarrow a \, 2 \Rightarrow b \circ b_\gamma$$

$$\S \ 1 \quad a \dot{\vee} 2 \, a \, (a \, b) \Rightarrow \gamma \, b \vee c_a \Rightarrow b$$
$$\alpha_a \wedge \beta_b \Rightarrow a \triangle b \oplus b_\beta \longmapsto 1 \ b \Rightarrow \delta_c \triangle c$$

$$\S \ 2 \quad \gamma \Rightarrow (a \, b) \ \bar{\triangle} \ b \oplus 1 \longmapsto 1 \, c \Rightarrow b_\delta.$$

5. Рассмотрим представленный на рис. 3.7.4 граф, в котором жирными линиями отмечены ребра, принадлежащие интересующим нас цепям.

Множество вершин, инцидентных этим ребрам, распадается на совокупность классов, задаваемую исходным значением комплекса β:

$$\| D \ni F \| = \begin{bmatrix} 00000\ 10000\ 00000\ 00000\ 00000\ 00 \\ 00001\ 00000\ 01000\ 00100\ 00000\ 00 \\ 00000\ 01000\ 10000\ 00000\ 10001\ 00 \\ 00000\ 00001\ 00000\ 01001\ 00000\ 00 \\ 00000\ 00000\ 00001\ 00000\ 00000\ 00 \end{bmatrix}$$

Совокупность S найденных программой к р а ц е п цепей представляется конечным значением комплекса δ:

$$\| S \ni^2 F \| = \begin{bmatrix} 00001\ 11001\ 00001\ 00000\ 00000\ 00 \\ 00001\ 11000\ 00001\ 01000\ 00000\ 00 \\ 00000\ 11001\ 01001\ 00000\ 00000\ 00 \\ 00000\ 11000\ 01001\ 01000\ 00000\ 00 \\ 00000\ 10001\ 11001\ 00000\ 00000\ 00 \\ 00000\ 10000\ 01001\ 01000\ 10000\ 00 \\ 00000\ 10000\ 01001\ 00001\ 00001\ 00 \\ 00000\ 10001\ 10001\ 00100\ 00000\ 00 \\ 00000\ 10000\ 00001\ 01100\ 10000\ 00 \end{bmatrix}$$

Н а х о ж д е н и е м а к с и м а л ь н ы х п о л н ы х п о д г р а ф о в. Полный пограф графа G называется *максимальным*, если он не является подграфом какого-либо другого полного подграфа этого же графа G. Рассмотрим задачу нахождения всех максимальных полных подграфов в заданном симметрическом графе G.

В основе предлагаемого ниже алгоритма решения данной задачи лежат следующие соображения.

Пусть задан граф $G=(F, U)$ и некоторая совокупность S подмножеств из F: $S=\{S_1, S_2, \ldots, S_m\}$. Рассмотрим следующее преобразование множества S.

Выбирается некоторая вершина x графа G, для которой строится множество $Px=\Gamma x \cup \{x\}$, где Γx — множество вершин, смежных с вершиной x, и для каждого из множеств $S_j \in S$ проверяются два условия:

$$x \in S_j \quad \text{и} \quad S_j \cap (F \setminus Px) \neq \varnothing.$$

В случае выполнения этих условий S_j исключается из совокупности S и находятся множества $S'_j = S_j \setminus \{x\}$ и $S''_j = S_j \cap Px$. Каждое из этих множеств включается в S тогда и только тогда, когда оно не содержится ни

в одном из $S_i \in S$. В случае невыполнения хотя бы одного из данных условий множество S_j остается в S без изменений.

Если задать в качестве начального значения $S = \{F\}$ и применить к S указанное преобразование $n-1$ раз (n — число вершин в графе), поочередно перебирая все вершины $x \in F$, кроме последней, в порядке их перечисления, то в результате совокупность S будет представлять семейство подмножеств множества F, образующих всевозможные максимальные полные подграфы графа G. Очевидно, что для последней вершины сформулированные условия никогда не будут выполняться.

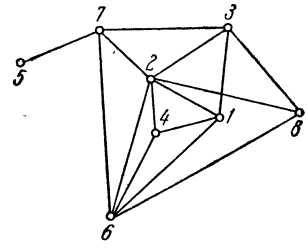


Рис. 3.7.5.

Для иллюстрации работы описанного алгоритма приведем пример выделения максимальных полных подграфов из графа, изображенного на рис. 3.7.5.

Ниже представлена последовательность результатов применения к множеству S описанного преобразования для каждой из вершин 1, 2, 3, 4, 5, 6 и 7.

{1, 2, 3, 4, 5, 6, 7, 8};
1 : {1, 2, 3, 4, 6}, {2, 3, 4, 5, 6, 7, 8};
2 : {2, 3, 4, 6, 7, 8}, {3, 4, 5, 6, 7, 8}, {1, 2, 3, 4, 6};
3 : {1, 2, 3}, {1, 2, 4, 6}, {2, 3, 7, 8}, {2, 4, 6, 7, 8}, {3, 7, 8}, {4, 5, 6, 7, 8};
4 : {1, 2, 3}, {1, 2, 4, 6}, {2, 3, 7, 8}, {2, 4, 6}, {2, 6, 7, 8}, {4, 6}, {5, 6, 7, 8};
5 : {1, 2, 3}, {1, 2, 4, 6}, {2, 3, 7, 8}, {2, 6, 7, 8}, {5, 7}, {6, 7, 8};
6 : {1, 2, 3}, {1, 2, 4, 6}, {2, 3, 7, 8}, {2, 6, 7, 8}, {5, 7};
7 : {1, 2, 3}, {1, 2, 4, 6}, {2, 3, 7}, {2, 3, 8}, {2, 6, 7}, {2, 6, 8}, {5, 7}.

Подчеркнутые множества удаляются как поглощаемые. Последняя строка представляет решение.

Эта же задача может быть решена и другим алгоритмом. Рассмотрим его.

Итак, пусть $F=\{x_1, x_2, \ldots, x_n\}$ — множество вершин графа G. Весь процесс нахождения максимальных полных подграфов можно разбить на n этапов, каждый из которых связан с определенной вершиной $x_i \in F$. На i-м этапе находятся все максимальные полные подграфы, содержащие вершину x_i и не содержащие вершин с меньшими номерами, то есть таких x_j, для которых $j < i$.

Пусть S_i — множество вершин, образующих один из полных подграфов графа G, формируемых на i-м этапе. За начальное значение множества S_i принимается множество, состоящее из единственной вершины x_i. Множество S_i расширяется за счет поочередного включения в него элементов $x_j \in F$, удовлетворяющих следующим условиям:

$$i < j \leqslant n \quad \text{и} \quad x_j \in \bigcap [C],$$

где $C=\{\Gamma x / x \in S_i\} \setminus S_i$, $\bigcap [C]$ — пересечение всех элементов множества C. Каждый раз при соблюдении этих условий выбирается x_j с минимальным j. Это расширение множества S_i продолжается до тех пор, пока $\bigcap [C]$ не станет пустым.

Подграф, образуемый полученным множеством S_i, проверяется на максимальность согласно следующему свойству. Полный подграф, образуемый множеством вершин S, является максимальным в том и только в том случае, когда

$$\bigcup [\{(F \setminus \Gamma x)/x \in S\}] = F.$$

Действительно, вершина, не вошедшая ни в одно из множеств $F \setminus \Gamma x$, образованных для всех $x \in S$, является смежной для всех $x \in S$, и, следовательно, ее можно присоединить к S, не нарушая полноты графа.

Если объединение множеств $F \setminus \Gamma x$ совпадает со всем множеством F, то для любой вершины, не принадлежащей S, найдется в S хотя бы одна не связанная с ней вершина и, следовательно, полный подграф, образованный множеством вершин S, максимален.

Подграф, прошедший такую проверку на максимальность, включается в решение. Используя метод сокращенного перебора, описанный ранее, можно получить все множества S_i, порождающие всевозможные полные подграфы, за n этапов.

Для графа на рис. 3.7.5 множества S_i ($i=1, 2, \ldots, 8$) будут принимать следующие значения:

S_1: {1}, {1, 2}, **{1, 2, 3}**, {1, 2, 4}, **{1, 2, 4, 6}**, {1, 2, 6}, {1, 3}, {1, 4}, {1, 4, 6}, {1, 6};

S_2: {2}, {2, 3}, **{2, 3, 7}**, {2, 3, 8}, {2, 4}, {2, 4, 6}, {2, 6}, **{2, 6, 7}**, **{2, 6, 8}**, {2, 7}, {2, 8};

S_3: {3}, {3, 7}, {3, 8};

S_4: {4}, {4, 6};

S_5: {5}, {5, 7};

S_6: {6}, {6, 7}, {6, 8};

S_7: {7};

S_8: {8}.

Здесь выделены жирным шрифтом те значения S_i, которые образуют подграфы, вошедшие в решение.

Л-программы. Описанные алгоритмы реализуются следующими Л-программами, составленными Ю. В. Поттосиным.

Нахождение максимальных множеств совместимости — сомакс

1. Для симметрического графа $G=(F, U)$, заданного урезанной матрицей смежности $\boldsymbol{K}$ (которая отличается от матрицы смежности $\boldsymbol{L}$ тем, что $k_i^j=0$ при $i>j$), требуется найти все максимальные полные подграфы, представив их порождающими подмножествами из F, образующими в совокупности некоторое множество N.

В названии Л-программы отражена одна из возможных интерпретаций задачи, связанная с рассмотрением графа G как графа совместимости элементов некоторого множества, поставленного в соответствие множеству F вершин графа.

2. Внешние операнды:

$\alpha_к :: \boldsymbol{K}$,

$\beta_к^+ :: \| N \ni F \|$,

$\gamma_п :: \Psi(F)$.

Задаются: a_α, a_β, b_α.

Размерность: $\alpha\beta\gamma$.

Внутренние операнды: $\boldsymbol{e}$, e, —.

4.

* 001 310 ($\alpha\, a\, b\, d\, e$)

§ 0 $1 \Rightarrow e \Rightarrow b_\beta\, \gamma \Rightarrow b_0\, \urcorner \Rightarrow e \circ d$

§ 1 $c_d \vee e \Rightarrow e \oplus \alpha_d \Rightarrow d\, \urcorner \circ\!\!\rightarrow 11 \circ c$

§ 2 $c_d \wedge \beta_c \circ\!\!\rightarrow 7\, c_d \oplus \beta_c \Rightarrow a\, \beta_c \wedge d \Rightarrow b \circ\!\!\rightarrow 7$
$\circ \beta_c \circ a \circ b \circ c$

§ 3 $\beta_a\, \urcorner \wedge b \circ\!\!\rightarrow 4 \triangle a - e \circ\!\!\rightarrow 3\, b \Rightarrow \beta_c \triangle c$

§ 4 $\beta_b\, \urcorner \wedge a \circ\!\!\rightarrow 5 \triangle b - e \circ\!\!\rightarrow 4\, c \mapsto 6\, a \Rightarrow \beta_c \rightarrow 7$

§ 5 $c \mapsto 7\, \bar{\triangle} e \oplus c \circ\!\!\rightarrow 10 \beta_e \Rightarrow \beta_c\, e - b_\beta \mapsto 7 \bar{\triangle} b_\beta \rightarrow 2$

§ 6 $a \Rightarrow \beta_b \triangle e$

§ 7 $\triangle c - b_\beta \circ\!\!\rightarrow 2$

§ 10 $e \Rightarrow b_\beta$

§ 11 $\triangle d - b_a \circ\!\!\rightarrow 1.$

Нахождение максимальных полных подграфов симметрического графа — подграмакс

1. Требуется найти все максимальные полные подграфы симметрического графа $G = (F, U)$, заданного урезанной матрицей смежности K.

Поставим задачу последовательного получения элементов решения: пусть при каждом обращении к описываемой Л-программе находится только один из искомых подграфов, представляемый соответствующим подмножеством F_i вершин графа G. При серии обращений к Л-программе таким образом будет получено все множество максимальных полных подграфов.

2. Внешние операнды:

$\alpha_ч$ — дополнительный выходной полюс, соответствующий получению очередного элемента решения; выход через основной полюс реализуется, когда будет найдено все искомое множество максимальных полных подграфов,

$[\beta_и]$ — число вершин графа G,

$\gamma_к :: L$ — матрица смежности графа G,

$\delta_к$ — комплекс памяти,

$\varepsilon_к$ — комплекс памяти,

$\zeta_к :: K$ — урезанная матрица смежности графа G,

$\eta_{\text{п}}^{+}::\|\{F_i\}\ni F\|$ — очередной элемент решения, выдаваемый при выходе по дополнительному полюсу α, для получения следующего элемента надо обратиться к дополнительному входному полюсу 4.

Задаются a_γ, a_δ, a_ε, a_ζ, b_γ, b_ζ.
Размерность: $\gamma\delta\varepsilon\zeta\eta$.
Внутренние операнды: b, d, —.
4.

* 001 320 ($\gamma\, a\, b$)

§ 0 $\circ d$

§ 1 $\circ c\, c_d \Rightarrow \eta\, \gamma_d \urcorner \Rightarrow \delta_c\, \zeta_d \Rightarrow a \Rightarrow \varepsilon_d \circ b$

§ 2 $a\, \dot{\mathrm{X}}\, 3\, a\, c_a \vee \eta \Rightarrow \eta$
$\zeta_a \wedge a \Rightarrow a \Rightarrow \varepsilon_a \triangle b\, \gamma_a \urcorner \vee \delta_c \Rightarrow \delta_b \triangle c \to 2$

§ 3 $\delta c \urcorner \circ\!\!\to \alpha$

§ 4 $\eta \urcorner + 1 \wedge \eta \Rightarrow b \oplus \eta \circ\!\!\to 5 \Rightarrow \eta \urcorner + 1 \wedge \eta \vdash \Rightarrow a$
$\varepsilon_a \oplus b \Rightarrow a \Rightarrow \varepsilon_a \bar{\triangle} c \bar{\triangle} b \to 2$

§ 5 $\triangle d - \beta \circ\!\!\to 1.$

Сравнение алгоритмов. Выполненные на машине М-20 эксперименты показали, что первый из описанных алгоритмов (сомакс) предпочтительнее использовать при относительно большом числе ребер в рассматриваемом графе, второй (подграмакс) оказывается лучшим в противоположной ситуации, когда число ребер в графе относительно невелико.

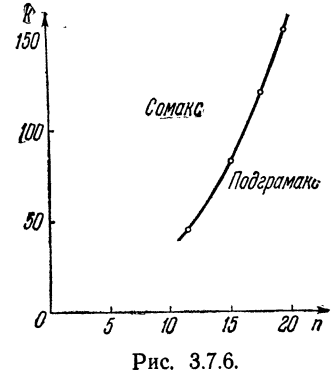


Рис. 3.7.6.

Некоторое представление об областях предпочтительного применения того или иного алгоритма дает рис. 3.7.6. Рассматриваемая область задается совокупностью следующих параметров: n — число вершин в графе и k — число ребер в графе.

ГЛАВА 4

СИНТЕЗ ДИСКРЕТНЫХ АВТОМАТОВ

§ 1. Функциональные модели автоматов

Релейные устройства и дискретные автоматы. Поведение любого технического устройства описывается, как известно, в терминах некоторых *физических переменных*, примерами которых могут служить координата некоторой перемещающейся части, напряжение на заданном участке электрической цепи, угол поворота вала, давление на поршень, освещенность некоторой поверхности и т. д. Особую роль играют две группы переменных, а именно, входные и выходные переменные. *Входными* называются те переменные, значения которых задаются извне и не определяются самим устройством, а напротив, влияют на его поведение. *Выходными* являются те переменные, для выработки значений которых и построено, по существу, рассматриваемое устройство. Эти значения определяются как некоторые функции входных переменных, *реализуемые* устройством. Те физические переменные, которые не являются ни входными, ни выходными, но тем не менее оказываются существенными при описании поведения устройства, называются *внутренними*. Они играют вспомогательную роль, подчиненную задаче реализации заданной функциональной зависимости между входными и выходными переменными.

Все большее распространение в современной технике получают такие устройства, у которых значения существенных переменных оказываются, грубо говоря, проквантованными. Выражаясь точнее, оказывается, что из области значений каждой существенной физической переменной можно выделить несколько взаимно непересекающихся интервалов, причем таким образом, что можно будет с достаточной полнотой изучать поведение устройства, не интересуясь при этом точными значениями физических переменных, а учитывая эти значе-

ния лишь с точностью до интервалов, к которым они принадлежат в рассматриваемый момент времени. Иначе говоря, можно пренебречь различиями между значениями, принадлежащими одному и тому же интервалу, считая их несущественными. Можно отказаться также от рассмотрения «промежуточных» значений, не принадлежащих ни одному из выделенных интервалов и пробегаемых физической переменной при переходе от одного интервала к другому. Здесь существенным оказывается лишь результат такого перехода, то есть достаточно знать, в какой интервал переходит значение переменной.

Упомянутые технические устройства, при изучении которых допустим описанный упрощенный подход, называются *релейными устройствами*, или *устройствами дискретного действия*. Область их применения весьма широка, а арсенал физических явлений, положенных в основу их действия, отличается большим разнообразием. Тем не менее отмеченные особенности релейных устройств позволяют изучать их с единой точки зрения, заменяя их непосредственное рассмотрение анализом абстрактной модели, называемой *дискретным автоматом*.

Переход от реального релейного устройства к его абстрактной модели — дискретному автомату — совершается путем замены каждой физической переменной с бесконечным числом значений на дискретную переменную, число значений которой конечно и равно числу выделенных интервалов в области значений рассматриваемой физической переменной. При этом считается, что если физическая переменная релейного устройства принимает значение в некотором интервале, то дискретная переменная принимает значение, соответствующее данному интервалу.

Далее мы ограничимся исследованием того важнейшего как с теоретической, так и с практической точки зрения случая, когда число интервалов, выделяемых в области значений каждой физической переменной, равно двум. Это значит, что каждой физической переменной мы ставим в соответствие некоторое элементарное событие, считая, что оно наступает, если физическая переменная принимает значение из одного интервала, и не наступает, если значение данной переменной

принадлежит другому интервалу. Для представления такого события естественно использовать *логическую* (двоичную) *переменную*, принимающую значение 0 или 1 в зависимости от того, к какому из интервалов принадлежит значение соответствующей ей физической переменной.

Например, в некотором релейном устройстве одной из существенных физических переменных может служить напряжение между некоторыми двумя точками электрической схемы. Допустим, что оно может принимать значения из двух интервалов, показанных на рис. 4.1.1. В этом случае можно условиться, что соответствующая логическая переменная принимает значение 0, если напряжение не выходит из интервала от 0 до 2 вольт, и принимает значение 1, если напряжение находится в пределах от 5 до 7 вольт.

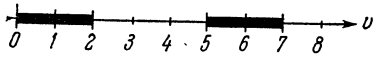


Рис. 4.1.1.

Можно рассмотреть отношение и с другой стороны, говоря о *физическом представлении*, или *физической реализации* логических переменных. Так, в данном примере значение 0 логической переменной реализуется выбором напряжения в интервале от 0 до 2 вольт, в то время как значение 1 реализуется напряжением от 5 до 7 вольт.

Если для реализации различных логических переменных используются однородные физические переменные с одинаковым образом выбираемыми интервалами их значений, то такую реализацию будем называть *однородной*. Например, с таким случаем мы сталкиваемся при представлении различных логических переменных напряжениями на различных участках электрической схемы, если на каждом из этих участков значение 0 логической переменной будет представлено интервалом от 0 до 2 вольт, а значение 1 — интервалом от 5 до 7 вольт.

Логические переменные дискретного автомата, соответствующие входным и выходным физическим переменным релейного устройства, мы будем называть также входными и выходными. В дальнейшем будет удобно рассматривать их как *входные* и *выходные полюсы* дискретного автомата, считая, что каждый из этих полюсов

может находиться в одном из двух состояний, обозначаемых через 0 и 1.

Комбинационный автомат. Так называется дискретный автомат, удовлетворяющий следующему условию: *каждой комбинации состояний входных полюсов автомата должна соответствовать некоторая вполне определенная комбинация состояний выходных полюсов.*

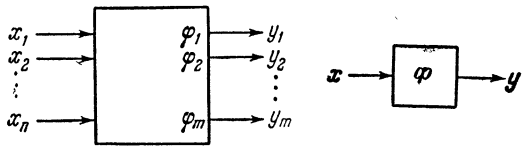


Рис. 4.1.2.

Отсюда непосредственно следует, что каждая двоичная переменная, представляющая состояние некоторого выходного полюса комбинационного автомата, является булевой функцией двоичных переменных, представляющих состояния входных полюсов. Другими словами, комбинационный автомат *реализует* некоторую систему булевых функций

$$\begin{aligned} y_1 &= \varphi_1(x_1, x_2, \ldots, x_n), \\ y_2 &= \varphi_2(x_1, x_2, \ldots, x_n), \\ &\ldots\ldots\ldots\ldots \\ y_m &= \varphi_m(x_1, x_2, \ldots, x_n), \end{aligned}$$

где символами $x_1, x_2, \ldots, x_n$ представлены входные логические переменные, а символами $y_1, y_2, \ldots, y_m$ — выходные логические переменные.

Рассматривая упорядоченные совокупности этих переменных как булевы векторы-переменные $\boldsymbol{x}$ и $\boldsymbol{y}$, а также представляя упорядоченную совокупность операторов $\varphi_1, \varphi_2, \ldots, \varphi_m$ посредством буквы Φ, данную систему булевых функций можно выразить в более компактной векторной форме:

$$\boldsymbol{y} = \Phi(\boldsymbol{x}).$$

Соответствующие этим формам графические представления комбинационных автоматов показаны на рис. 4.1.2.

Реализуемая комбинационным автоматом система булевых функций представляет функциональные свойства автомата и может рассматриваться как его функциональная модель (или функциональное описание). Разумеется, пользуясь этой моделью, мы допускаем некоторую идеализацию реальных релейных устройств. В самом деле, последние обладают инерционностью, отображая которую, мы должны были бы исходить из того, что изменение комбинации состояний входных полюсов автомата не приводит к мгновенному образованию соответствующей комбинации состояний выходных полюсов, так как для этого требуется некоторое время. Однако во многих практически важных случаях учетом влияния этого явления можно пренебречь, считая, что рассматриваемое устройство практически безынерционно.

Совокупность входных полюсов комбинационного автомата будем называть в дальнейшем *входом* автомата, совокупность выходных полюсов — его *выходом*. Легко подсчитать, что существует 2^n различных состояний входа, или *входных состояний*, и 2^m *выходных состояний*. Эти состояния представляются соответствующими комбинациями значений входных переменных $x_1, x_2, \ldots$ $\ldots, x_n$ и выходных переменных $y_1, y_2, \ldots, y_m$. Следует заметить, что для заданного комбинационного автомата некоторые выходные состояния (может случиться, что и большинство из них) могут оказаться *нереализуемыми* ни при каком из входных состояний.

Элементарные преобразователи. Пусть некоторому релейному устройству соответствует комбинационный автомат, который обладает одним входным и одним выходным полюсом и функционирует таким образом, что значения двоичных переменных, представляющих состояния полюсов, будут всегда совпадать. Назовем такое релейное устройство *элементарным преобразователем*.

Назначением элементарных преобразователей является преобразование физического представления логических переменных. В частности, к числу элементарных преобразователей можно отнести широко применяемые в технике датчики и исполнительные механизмы. Как правило, *датчики* используются для получения та-

кой физической реализации логических переменных, которая оказывается удобной для последующей автоматической обработки информации некоторым техническим устройством. *Исполнительные механизмы* обеспечивают приведение получаемых при этой обработке результатов к требуемой физической форме. Впрочем, деление элементарных преобразователей на датчики и исполнительные механизмы носит несколько условный характер. Не менее широкий класс образуется повторителями и усилителями, также являющимися элементарными преобразователями.

Можно сказать, что каждый элементарный преобразователь обеспечивает связь между некоторыми двумя *элементарными* событиями, то есть такими, которые мы уже не разлагаем на более простые в каком-то смысле события и не представляем их как некоторую комбинацию этих более простых событий. Например, элементарным преобразователем можно назвать электрический звонок. Событие, заключающееся в появлении напряжения на обмотке звонка, автоматически влечет за собой другое событие — раздается звонок.

Конъюнкции элементарных событий и связь между ними. Пусть двоичные переменные $x_1, x_2, \ldots, x_n$, образующие множество X, представляют некоторые элементарные события: будем считать, что переменная x_i принимает значение 1 при наступлении соответствующего события и принимает значение 0 в противном случае. *Конъюнкцией элементарных событий*, образующих некоторое подмножество X_j из X, назовем событие, наступающее в том и только в том случае, когда наступает каждое из элементарных событий, принадлежащих множеству X_j, и не наступает ни одно из элементарных событий, принадлежащих множеству $X \setminus X_j$.

Очевидно, что множество всех конъюнкций элементарных событий из X образует булево пространство $M(X)$ над X и состоит из 2^n элементов, которые мы будем также называть событиями из $M(X)$.

Аналогичным образом построим пространство $M(Y)$ над некоторым другим множеством Y элементарных событий $y_1, y_2, \ldots, y_m$. Допустим теперь, что требуется реализовать заданную каким-то образом связь между событиями в этих пространствах, обеспечив для каждого

события из $M(X)$ наступление некоторого вполне определенного события из $M(Y)$. Нетрудно видеть, что такая связь может быть реализована автоматически, с помощью соответствующего комбинационного автомата (точнее выражаясь, с помощью релейного устройства, абстрактной моделью которого служит некоторый комбинационный автомат), а также серии элементарных преобразователей.

Действительно, в чем бы ни заключались элементарные события, представляемые двоичными переменными $x_1, x_2, \ldots, x_n$, их можно отобразить состояниями входных полюсов некоторого комбинационного автомата, использовав для этого соответствующие датчики. Комбинационный автомат должен реализовать именно ту систему булевых функций, которая характеризует требуемую связь между интересующими нас событиями. Состояния выходных полюсов автомата должны быть автоматически связаны с событиями из множества Y, для чего следует применить некоторые исполнительные механизмы.

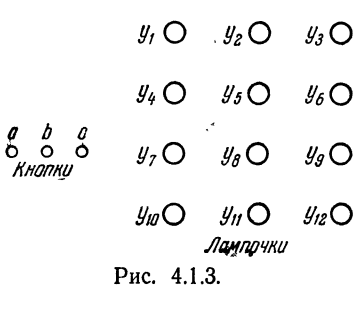


Рис. 4.1.3.

Обратимся к конкретному примеру.

Пример. Пусть в нашем распоряжении имеются три кнопки и двенадцать электрических лампочек, расположенных в форме матрицы размером 4 на 3, как это показано на рис. 4.1.3.

Представим состояния кнопок двоичными переменными a, b и c (значение 1 соответствует нажатию кнопки), а состояния лампочек — двоичными переменными $y_1, y_2, \ldots, y_{12}$ (значение 1 здесь поставим в соответствие светящейся лампочке). Сформулируем следующую задачу: пусть между событиями в множествах $M(X)$ и $M(Y)$ (где $X \equiv \{a, b, c\}$) требуется реализовать такую связь, чтобы комбинация светящихся лампочек всегда образовывала цифру, двоичный код которой задан состояниями кнопок. Эта связь показана на рис. 4.1.4, на котором зачернены нажатые кнопки и светящиеся лампочки.

Анализируя этот рисунок, найдем систему булевых функций, связывающих две группы переменных. Сначала представим ее в виде следующей таблицы, довольно просто получаемой из рис. 4.1.4:

a	b	c	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}
0	0	0	1	1	1	1	0	1	1	0	1	1	1	1
0	0	1	0	1	0	0	1	0	0	1	0	0	1	0
0	1	0	1	1	1	0	0	1	0	1	0	1	1	1
0	1	1	1	1	1	0	1	0	0	0	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	0	0	1
1	0	1	1	1	1	1	0	0	0	1	1	1	1	1
1	1	0	1	0	0	1	1	1	0	1	1	1	1	1
1	1	1	1	1	0	0	1	0	1	0	0	1	0	0

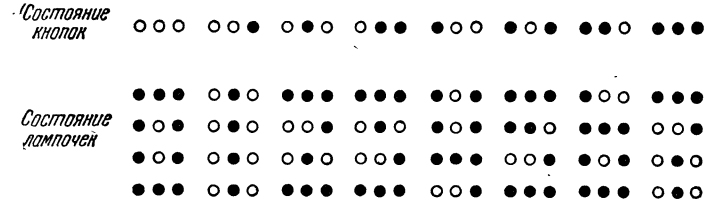


Рис. 4.1.4.

Представляя эту систему в алгебраической форме и по возможности минимизируя ее, получим

$$y_1 = a \vee b \vee \bar{c},$$
$$y_2 = \bar{a} \vee c,$$
$$y_3 = \bar{a}b \vee ac \vee \bar{b}\bar{c},$$
$$y_4 = a\bar{b} \vee a\bar{c} \vee \bar{b}\bar{c},$$
$$y_5 = \bar{a}c \vee \bar{b}c \vee ab\bar{c},$$
$$y_6 = \bar{c} \vee ab,$$
$$y_7 = a\bar{c} \vee \bar{b}\bar{c},$$
$$y_8 = \bar{a}\bar{b}c \vee a\bar{b}\bar{c} \vee \bar{a}b\bar{c} \vee abc,$$
$$y_9 = a\bar{c} \vee a\bar{b} \vee \bar{b}c \vee \bar{a}bc,$$
$$y_{10} = \bar{a}b \vee \bar{a}\bar{c} \vee b\bar{c} \vee a\bar{b}c,$$
$$y_{11} = \bar{a} \vee b \vee c,$$
$$y_{12} = \bar{c} \vee \bar{a}b \vee a\bar{b}.$$

Именно эта система булевых функций и должна быть реализована выбираемым нами комбинационным автоматом. Необходимо договориться о способе физической реализации рассматриваемых логических переменных на входных и выходных полюсах автомата, а также снабдить этот автомат системой элементарных преобразователей, согласующих эту реализацию с заданной системой элементарных событий.

Например, удобным способом представления двоичных переменных в данном случае является использование проводимости отдельных цепей электрической схемы: если некоторая цепь замкнута, то есть обладает высокой проводимостью, можно считать, что она находится в состоянии 1, если же она разомкнута, то есть ее проводимость весьма мала, то цепь находится в состоянии 0. При таком выборе роль датчика может играть сама кнопка, преобразующая нажатие в замыкание некоторой цепи — входного полюса комбинационного автомата, а роль исполнительного механизма играет лампочка, загорающаяся при замыкании другой цепи — выходного полюса автомата.

От внутренней структуры автомата мы пока абстрагируемся.

Последовательностные автоматы. Изменяя во времени состояния входа комбинационного автомата, мы можем получить некоторую последовательность входных состояний, которую можно представить соответствующей последовательностью значений вектора x. На выходе автомата образуется в этом случае последовательность выходных состояний, представляемых соответствующими значениями вектора y. При этом каждый член входной последовательности будет однозначно определять соответствующий член выходной последовательности. Например, если значению 1001 вектора x соответствует значение 011 вектора y, то это соответствие будет реализовываться комбинационным автоматом всегда, вне зависимости от предшествующих значений вектора x. Именно это свойство комбинационных автоматов имеют в виду, называя их иногда *автоматами без памяти*: поведение таких автоматов не зависит от прошлого, а полностью определяется текущим входным состоянием.

Итак, можно сказать, что комбинационный автомат реализует некоторое однозначное отображение множества входных состояний во множество выходных состояний.

Принципиально по-иному ведет себя *автомат с памятью*, или *последовательностный автомат* (мы будем далее пользоваться вторым из этих названий). Само его название говорит о том, что он реализует функциональную связь уже не между отдельными состояниями, а между их последовательностями, точнее — между входной и выходной последовательностью.

Последовательностный автомат может по-разному реагировать на одинаковые входные состояния, ставя им в соответствие различные выходные состояния. Его реакция зависит, вообще говоря, от всей последовательности входных состояний, реализованной к текущему моменту времени, и выражается выдачей некоторой последовательности выходных состояний.

Такое поведение последовательностного автомата становится понятным, если учесть, что он обладает некоторым множеством *внутренних состояний*, в каждом из которых он ведет себя подобно некоторому комбинационному автомату, то есть реализует однозначную функциональную связь между входными и выходными состояниями. Однако наряду с этим его реакция на очередное входное состояние может выразиться также в смене внутреннего состояния, то есть в переходе в некоторое другое внутреннее состояние, в котором автомат уже по-иному будет реагировать на следующее входное состояние.

Заметим, что множество всевозможных значений булева вектора x, используемых для представления входных состояний автомата, принято называть *структурным входным алфавитом*, а множество значений вектора y — *структурным выходным алфавитом*. Такая терминология оправдана, так как значения векторов x и y задают входные и выходные состояния автомата как некоторые комбинации состояний его полюсов, то есть действительно показывают внутреннюю структуру состояний автомата.

Между тем, при рассмотрении ряда задач удобно абстрагироваться от структуры входных и выходных

состояний автомата и представлять их довольно произвольными символами, беспокоясь лишь о том, чтобы разные состояния оказались представленными различными символами. Будем считать, что эти символы выбираются из некоторых множеств S_α и S_β, называемых соответственно *абстрактным входным алфавитом* и *абстрактным выходным алфавитом*. Допустимо также называть эти множества множествами входных и выходных состояний (заданных в абстрактных алфавитах).

Вводя в рассмотрение множество S_γ внутренних состояний автомата, будем пока считать, что эти состояния заданы также в абстрактном алфавите.

Синхронный автомат. Наиболее известными разновидностями последовательностных автоматов являются синхронный автомат и асинхронный автомат.

Поведение *синхронного автомата* рассматривается в *дискретном времени* t, представляющем собой последовательность моментов, обозначаемых через 1, 2, 3, 4 и т. д. Такое рассмотрение обычно связывается с наличием некоторых тактовых генераторов в технических реализациях автомата. Поведение автомата определяется реализуемым им отношением между входными и выходными последовательностями. При этом считается, что члены входной последовательности представляют собой входные состояния автомата в следующие друг за другом моменты дискретного времени. Аналогично определяется выходная последовательность.

Поведение синхронного автомата задается следующей парой функций, называемых соответственно *функцией переходов и функцией выходов*:

$$\gamma(t+1) = \Psi(\alpha(t),\ \gamma(t)),$$
$$\beta(t) = \Phi(\alpha(t),\ \gamma(t)),$$

где $\alpha(t)$ — входное состояние автомата в момент времени t, $\beta(t)$ — выходное состояние в момент t, $\gamma(t)$ — внутреннее состояние в момент t и $\gamma(t+1)$ — внутреннее состояние в последующий момент $t+1$.

Данная система функций может рассматриваться как *аналитическая функциональная модель синхронного автомата* и известна под названием *модели Мили*. Характерной чертой этой модели является зависимость вы-

ходного состояния как от внутреннего, так и от входного состояния, то есть зависимость от *полного состояния* автомата, как принято называть совокупность входного и внутреннего состояний. Если же функция выходов представляет зависимость лишь от внутреннего состояния автомата в текущий момент времени, то есть если поведение автомата описывается системой

$$\beta(t) = \Phi(\gamma(t)),$$
$$\gamma(t+1) = \Psi(\alpha(t), \gamma(t)),$$

то такая модель носит название *модели Мура*.

Очевидно, что для того, чтобы предсказать реакцию синхронного автомата на некоторую входную последовательность, то есть вычислить соответствующую входную последовательность, надо знать *начальное состояние* автомата. Его задание уточняет функциональную модель автомата, что в ряде случаев является необходимым.

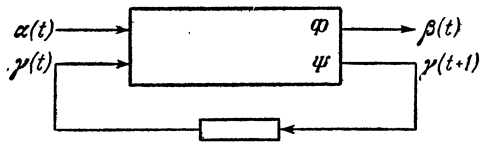


Рис 4.1.5.

Заметим, кстати, что, говоря просто о состоянии автомата, мы всегда подразумеваем внутреннее состояние.

Удобное графическое изображение синхронного автомата показано на рис. 4.1.5. Нижний элемент изображения представляет схематически «память» автомата, которую можно трактовать как некоторое *устройство задержки*. Это устройство как бы задерживает на один такт вычисляемое в текущий момент t значение функции Ψ: в момент $t+1$ оно появляется на выходе элемента и играет роль аргумента при вычислении следующего значения функций Ψ и Φ.

Матричная модель синхронного автомата. Если число входных и внутренних состояний автомата невелико, функции переходов и выходов удобно представлять в виде матриц Ψ и Φ, строки которых соответствуют значениям аргумента γ, столбцы —

значениям аргумента α. Матрицу Ψ, элементами которой служат значения функции Ψ, будем называть *матрицей переходов*, а матрицу Φ, элементами которой служат значения функции Φ, — *матрицей выходов*.

С целью упрощения обозначений условимся пользоваться сокращенной символикой рассматриваемых состояний. Например, в следующей паре матриц

$$\Psi = \begin{array}{c} a\ b\ c\ d\ e \\ \left[\begin{array}{ccccc} 2 & 2 & 1 & 4 & 4 \\ 2 & 4 & 3 & 3 & 2 \\ 1 & 2 & 2 & 1 & 1 \\ 4 & 3 & 3 & 1 & 2 \end{array}\right] \end{array} \begin{array}{c} 1 \\ 2 \\ 3' \\ 4 \end{array} \qquad \Phi = \begin{array}{c} a\ b\ c\ d\ e \\ \left[\begin{array}{ccccc} 1 & 1 & 3 & 7 & 6 \\ 4 & 4 & 4 & 4 & 4 \\ 3 & 2 & 2 & 5 & 3 \\ 5 & 5 & 6 & 6 & 2 \end{array}\right] \end{array} \begin{array}{c} 1 \\ 2 \\ 3' \\ 4 \end{array}$$

представляющей функциональные свойства синхронного автомата (модель Мили) с множеством входных состояний $S_\alpha = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$, с множеством внутренних состояний $S_\gamma = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$ и с множеством выходных состояний $S_\beta = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7\}$, элементы множеств S_β и S_γ представлены их номерами, а символы входных состояний α_1, α_2, α_3, α_4 и α_5 заменены на a, b, c, d и e, соответственно.

Допустим, что в исходный момент времени автомат находится в состоянии γ_1 (то есть пусть γ_1 — начальное состояние автомата). Найдем реакцию автомата на следующую *входную последовательность* (то есть последовательность входных состояний):

$$\alpha_2\,\alpha_4\,\alpha_4\,\alpha_1\,\alpha_3\,\alpha_5\,\alpha_3\,\alpha_2\,\alpha_2,$$

которую мы представляем как последовательность *bddacecbb*.

В первый момент времени поведение автомата будет определяться элементами ψ_{12} и φ_{12} матриц Ψ и Φ. Поскольку $\psi_{12}=2$ и $\varphi_{12}=1$, реализуется выходное состояние β_1 и автомат перейдет (к следующему моменту времени) во внутреннее состояние γ_2. Поскольку в следующий момент времени входным состоянием автомата будет $\alpha_4 = d$, а внутренним — γ_2, поведение автомата в этот момент будет задаваться матричными элементами ψ_{24} и φ_{24}: реализуется выходное состояние β_4 и автомат перейдет во внутреннее состояние γ_3. Процесс в целом можно представить следующей таблицей.

Таблица 4.1.1

Время	1 2 3 4 5 6 7 8 9
Входное состояние . .	*b d d a c e c b b*
Внутреннее состояние .	1 2 3 1 2 3 1 1 2
Выходное состояние . .	1 4 5 1 4 3 3 1 4

Из таблицы видно, что входная последовательность *bddacecbb* трансформируется в *выходную последовательность* (последовательность выходных состояний) 145143314.

Аналогичным путем можно при заданном начальном состоянии для любой другой входной последовательности найти соответствующую ей выходную последовательность. Заметим, что если не ограничивать при этом длину рассматриваемых последовательностей, то число получаемых при таком рассмотрении пар типа «входная последовательность и соответствующая ей выходная последовательность» будет бесконечным.

Таким образом, синхронный автомат реализует некоторое *отображение множества входных последовательностей во множество выходных последовательностей*, и это отображение полностью определяется матрицей переходов **Ψ** и матрицей выходов **Φ**.

Отметим два вырожденных случая синхронных автоматов.

Если автомат обладает лишь одним внутренним состоянием, то очевидно, что он должен все время в нем оставаться. В этом случае автомат можно рассматривать как комбинационный, поведение которого полностью представляется однострочной матрицей **Φ**.

Если же автомат обладает лишь одним входным состоянием, то такой автомат называется *автономным*. Его поведение определяется парой одностолбцовых матриц **Ψ** и **Φ** и, следовательно, не зависит от каких-либо внешних воздействий.

Граф поведения автомата. Построим ориентированный граф, вершины которого будут поставлены во взаимно однозначное соответствие с внутренними состояниями синхронного автомата, а дуги — с возможными переходами между ними. Отметим вершины символами соответствующих внутренних состояний,

а дуги — символами входных состояний, при которых реализуется представленный дугой переход, а также символами реализуемых при этом выходных состояний. Назовем этот граф *графом поведения автомата.*

Например, рассмотренной выше паре матриц **Ψ** и **Φ** будет соответствовать следующий граф поведения автомата (рис. 4.1.6).

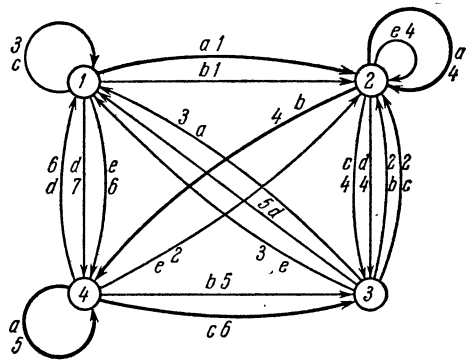


Рис. 4.1.6.

Использование этого графа представляет в ряде случаев известные удобства. Функционирование автомата можно отобразить движением по графу некоторой точки, начинающей свой путь из вершины, соответствующей начальному состоянию, и в каждый такт проходящей по дуге, соответствующей реализуемому в этом такте входному состоянию. Назовем эту точку *фазовой.* Реализуемая автоматом выходная последовательность будет представлена как последовательность цифр, отмечающих проходимые фазовой точкой дуги графа.

Например, на рисунке жирными линиями отмечен путь, проходимый фазовой точкой при реализации входной последовательности *aabac* при начальном состоянии 1. Лежащие на данном пути вершины и дуги задают реализуемую при этом последовательность внутренних состояний 1 2 2 4 4 3 и выходную последовательность 1 4 4 5 6.

Если в графе поведения мы уберем все отметки, относящиеся к выходным состояниям, то получим *граф переходов.*

Частичные автоматы. До сих пор мы рассматривали автоматы с полностью определенными функциональными свойствами: для любого начального состояния и любой входной последовательности (в пределах заданного входного алфавита) однозначно определялась соответствующая выходная последовательность. Такие автоматы будем называть *полными*.

Напомним, что дискретный автомат представляет собой абстрактную модель релейного устройства, отражая функциональные свойства последнего. Может случиться, что эти свойства будут интересовать нас не полностью, а частично. В этом случае в качестве абстрактной модели удобно взять не полный, а частичный автомат, определяемый следующим образом.

Допустим, что некоторые символы матрицы переходов имеют значение «—», символизирующее, как и прежде, неопределенность. Полные состояния, соответствующие этим элементам, назовем *граничными*. Дальнейшее поведение автомата, попавшего в граничное состояние, не определяется, поскольку последующее внутреннее состояние считается неизвестным. Автомат, обладающий граничными состояниями, назовем *частичным*.

Таким образом, поведение частичного автомата определяется не на любых парах «начальное состояние и подаваемая при нем входная последовательность», а лишь на некоторых, образующих в совокупности некоторое множество $P(A)$, называемое областью задания частичного автомата A. Для каждого начального состояния эта область будет задавать множество допустимых при данном состоянии входных последовательностей, при реализации которых автомат A пробегает некоторую последовательность полных состояний, лишь последнее из которых может быть граничным.

Построенная таким образом модель оказывается более абстрактной, чем модель полного автомата. Она отражает уже целую совокупность релейных устройств (а также соответствующих им полных автоматов), функциональные свойства которых могут быть различными, совпадая лишь на области задания рассматриваемого частичного автомата.

Будем считать, что символы неопределенности «—» могут содержаться также в матрице выходов. Здесь они

не нарушают детерминированности процесса изменения внутренних состояний автомата, но могут быть причиной того, что некоторые выходные последовательности, выдаваемые автоматом, также будут содержать в себе символы «—». Будем называть такие выходные последовательности *частичными*. Наличие в них символов «—» означает произвольность значений соответствующих элементов последовательности: считается, что вместо символов «—» могут быть поставлены любые символы из выходного алфавита, и это не будет противоречить исходному заданию свойств автомата.

Например, пусть поведение некоторого частичного автомата задается следующими матрицами переходов и выходов:

$$\Psi = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \left[\begin{matrix} 3 & 2 & 4 \\ 3 & - & - \\ - & - & - \\ - & 3 & 2 \end{matrix}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3' \\ 4 \end{matrix} \qquad \Phi = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \left[\begin{matrix} 1 & - & 2 \\ - & 1 & - \\ 2 & 3 & 3 \\ 2 & 3 & - \end{matrix}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3' \\ 4 \end{matrix}$$

Поведение автомата в данном случае задается на таких входных последовательностях, каждая из которых приводит автомат в некоторое граничное состояние. Множество таких последовательностей оказывается конечным и его можно непосредственно перечислить, задав на нем соответствующие выходные последовательности. Представим это перечисление в виде таблицы 4.1.2, в

Таблица 4.1.2

Начальное состояние	Трансформация последовательностей
1	$aa \to 12$, $ab \to 13$, $ac \to 13$, $baa \to --2$, $bab \to --3$, $bac \to --3$, $bb \to -1$, $bc \to --$, $ca \to 22$, $cba \to 232$, $cbb \to 233$, $cbc \to 233$, $ccaa \to 2--2$, $ccab \to 2--3$, $ccac \to 2--3$, $ccb \to 2-1$, $ccc \to 2--$;
2	$aa \to -2$, $ab \to -3$, $ac \to -3$, $b \to 1$, $c \to -$;
3	$a \to 2$, $b \to 3$, $c \to 3$;
4	$a \to 2$, $ba \to 32$, $bb \to 33$, $bc \to 33$, $caa \to --2$, $cab \to --3$, $cac \to --3$, $cb \to -1$, $cc \to --$.

которой показано, например, что при начальном состоянии 1 автомат реагирует на входную последовательность *aa* выдачей выходной последовательности 12, при начальном состоянии 2 входная последовательность *aa* трансформируется в выходную последовательность — 2, первый символ которой считается произвольным, и т. д.

На данном примере видно, что матричное представление свойств автомата оказывается значительно более компактным, чем табличное.

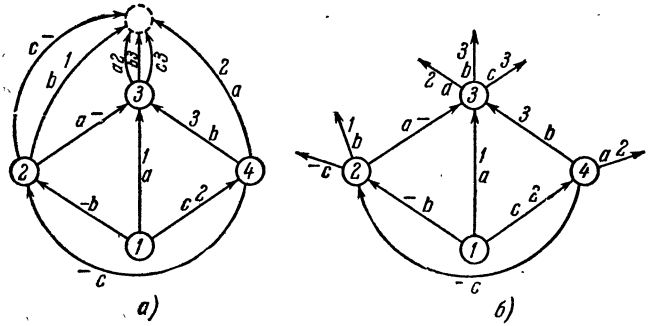


Рис. 4.1.7.

Достаточно удобным оказывается и графическое представление частичного автомата. Положим, что соответствующие граничным полным состояниям дуги графа поведения частичного автомата замыкаются на дополнительную вершину, соответствующую «неопределенному состоянию». Попадание фазовой точки в эту вершину, из которой не исходит ни одна дуга, соответствует выходу из области $P(A)$, на которой задано поведение автомата A.

Пример такого графа, представляющего все тот же частичный автомат, показан на рис. 4.1.7, *а*, где дополнительная вершина изображена пунктирным кружком. В некоторых случаях (когда число дуг, входящих в дополнительную вершину, достаточно велико) удобно упрощать изображение графа, удаляя из него дополнительную вершину и оставляя свободными соответствующие концы входивших в нее дуг (рис. 4.1.7, *б*).

Допустим, что мы доопределим поведение частичного автомата, то есть заменим некоторые из символов «—»

в матрице переходов символами конкретных внутренних состояний. В графическом представлении такой процедуре будет соответствовать «переключение» некоторых из дуг, входящих в дополнительную вершину, на другие вершины графа. В этом случае область задания поведения автомата будет расширена, однако поведение автомата на исходной области, очевидно, не изменится. Аналогичное доопределение матрицы выходов также не

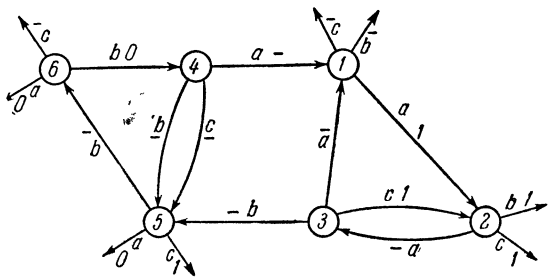


Рис. 4.1.8.

нарушает первоначально заданных свойств автомата, а приводит лишь, как это нетрудно видеть, к уточнению реакций автомата: некоторые из символов «—» в выходных последовательностях заменяются конкретными символами выходного алфавита.

Число входных последовательностей, на которых задается поведение частичного автомата, может быть и бесконечным. Соответствующий такому случаю пример представлен следующей парой матриц:

$$\Psi = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \left[\begin{array}{ccc} 2 & - & - \\ 3 & - & - \\ 1 & 5 & 2 \\ 1 & 5 & 5 \\ - & 6 & - \\ - & 4 & - \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4' \\ 5 \\ 6 \end{matrix} \qquad \Phi = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \left[\begin{array}{ccc} 1 & - & - \\ - & 1 & 1 \\ - & - & 1 \\ - & - & - \\ 0 & - & 1 \\ 0 & 0 & - \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4' \\ 5 \\ 6 \end{matrix}$$

Граф поведения этого же автомата показан на рис. 4.1.8.

Причина указанной бесконечности очевидна: граф поведения содержит контуры.

Асинхронный автомат. Эта модель отражает поведение автомата, внутреннее состояние которого может меняться лишь при изменении входного состояния, причем в результате этого изменения автомат всегда приходит в конечном счете в некоторое *устойчивое полное состояние*, то есть такое полное состояние, в котором автомат остается до тех пор, пока не изменится его входное состояние. Считается также, что новое изменение входного состояния не может производиться до того, как автомат перейдет в устойчивое полное состояние.

В связи с этим моменты дискретного времени, в котором рассматривается поведение автомата, связываются с моментами, в которые происходит изменение входного состояния. Естественно, что при этом теряет смысл рассмотрение таких входных последовательностей, которые содержат одинаковые соседние символы.

По аналогии с синхронным автоматом, асинхронный автомат также может быть представлен матричной моделью — парой матриц Ψ и Φ, однако сформулированные условия налагают в этом случае определенные ограничения на характер матрицы Ψ.

Рассмотрим эти ограничения сначала в следующем усиленном виде: если некоторый элемент ψ_i^j матрицы Ψ имеет значение k, то элемент ψ_k^j также должен обладать значением k. Это означает, что любое полное состояние рассматриваемого автомата связано *прямым переходом* с некоторым устойчивым полным состоянием.

Например, такому условию удовлетворяет следующая матрица переходов:

$$\Psi = \begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} \\ \left[\begin{array}{cccc} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{5} \\ 1 & \mathbf{2} & \mathbf{2} & \mathbf{2} \\ \mathbf{3} & \mathbf{2} & \mathbf{4} & \mathbf{5} \\ 3 & \mathbf{2} & \mathbf{4} & \mathbf{5} \\ \mathbf{3} & \mathbf{5} & \mathbf{4} & \mathbf{5} \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3, \\ 4 \\ 5 \end{matrix}$$

в которой выделены жирным шрифтом элементы, соответствующие устойчивым полным состояниям.

В общем виде указанные ограничения можно сформулировать следующим образом: для любого элемента ψ_i^j матрицы Ψ должна выполняться следующая цепочка

равенств: $\psi_i^l = k_1$, $\psi_{k_1}^l = k_2$, ..., $\psi_{k_p}^l = k_p$. Это означает, что любое полное состояние асинхронного автомата должно быть связано цепочкой переходов с некоторым устойчивым полным состоянием (в противном случае возможно «зацикливание» фазовой точки, в результате которого при некотором входном состоянии автомат никогда не сможет достигнуть устойчивого состояния).

Например, этому условию удовлетворяет матрица Ψ_1, но не удовлетворяет матрица Ψ_2:

$$\Psi_1 = \begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} \\ \left[\begin{array}{cccc} 2 & 1 & 1 & 3 \\ 4 & 2 & 1 & 3 \\ 2 & 2 & 4 & 3 \\ 4 & 3 & 5 & 1 \\ 1 & 4 & 5 & 4 \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3, \\ 4 \\ 5 \end{matrix} \qquad \Psi_2 = \begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} \\ \left[\begin{array}{cccc} 5 & 2 & 2 & 5 \\ 2 & 3 & 1 & 2 \\ 2 & 4 & 1 & 4 \\ 3 & 5 & 2 & 4 \\ 5 & 1 & 3 & 1 \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3. \\ 4 \\ 5 \end{matrix}$$

Какие-либо ограничения на матрицу выходов Φ можно и не налагать. Однако часто при рассмотрении асинхронных автоматов достаточно обращать внимание лишь на те значения функции Φ, которые соответствуют устойчивым полным состояниям. В таких случаях матрице Φ удобно придавать одностолбцовую форму, соответствующую модели Мура.

§ 2. Минимизация числа состояний

Как это уже было показано на некоторых примерах, в ряде случаев мы можем отвлечься от рассмотрения внутренних состояний, их структуры и даже их числа. Дело в том, что основной функциональной характеристикой автомата является зависимость выхода автомата от его входа, то есть отношение между множествами выходных и входных последовательностей. Что же касается внутренних состояний, то они играют вспомогательную роль, обеспечивая лишь некоторую реализацию указанного отношения. Эта роль может выполняться по-разному, благодаря чему и возникает рассматриваемая в данном параграфе задача.

Эквивалентность состояний. Пусть γ_i и γ_j — некоторые два внутренние состояния полного автомата.

Будем называть их *эквивалентными*, если на любую входную последовательность автомат будет отвечать некоторой выходной последовательностью, одинаковой для начальных состояний γ_i и γ_j. Аналогично определяется эквивалентность состояний, принадлежащих различным автоматам: состояние γ_i автомата A_1 считается *эквивалентным* состоянию γ_j автомата A_2, если при данных начальных состояниях автоматы A_1 и A_2 отвечают одинаковой выходной последовательностью на любую входную последовательность.

Определенное таким образом отношение эквивалентности между состояниями некоторого полного автомата оказывается *рефлексивным* (каждое состояние эквивалентно само себе), *симметричным* (если состояние γ_i эквивалентно состоянию γ_j, то и состояние γ_j эквивалентно состоянию γ_i) и *транзитивным* (если состояние γ_i эквивалентно состоянию γ_j, а состояние γ_j эквивалентно состоянию γ_k, то состояние γ_i эквивалентно состоянию γ_k). Отсюда следует, что множество всех состояний полного автомата разбивается на *классы эквивалентности*, образованные из взаимно эквивалентных состояний. Состояния же, принадлежащие различным классам эквивалентности, являются взаимно неэквивалентными.

Из определения эквивалентности состояний нетрудно вывести правило проверки состояний на эквивалентность: во-первых, эквивалентным состояниям должны соответствовать равные между собой строки матрицы выходов $\mathbf{\Phi}$, во-вторых, соответствующие этим состояниям строки матрицы переходов $\mathbf{\Psi}$ должны быть также равны или отличаться символами взаимно эквивалентных состояний. Другими словами, если указанные строки из матрицы $\mathbf{\Psi}$ образуют на пересечении с некоторым столбцом пару различных символов, то это должны быть символы взаимно эквивалентных состояний. Следовательно, проверка эквивалентности заданной пары состояний может влечь за собой проверку эквивалентности внутри некоторых других пар, которые мы будем называть *производными* от заданной пары, и этот процесс может иметь цепной характер. Однако поскольку число состояний практически всегда конечно, рано или поздно процесс проверки завершается и устанавливается, эквивалентны рассматриваемые состояния или нет.

Пример. Рассмотрим следующий автомат:

$$\Psi = \begin{array}{|ccc|c|}\hline a & b & c & \\ \hline 4 & 2 & 5 & 1 \\ 5 & 1 & 4 & 2 \\ 3 & 5 & 4 & 3 \\ 5 & 8 & 4 & 4 \\ 7 & 2 & 1 & 5 \\ 1 & 3 & 4 & 6 \\ 5 & 3 & 7 & 7 \\ 3 & 5 & 6 & 8 \\ \hline \end{array} \qquad \Phi = \begin{array}{|ccc|c|}\hline a & b & c & \\ \hline 1 & 2 & 1 & 1 \\ 2 & 1 & 2 & 2 \\ 2 & 1 & 2 & 3 \\ 1 & 2 & 2 & 4 \\ 1 & 2 & 1 & 5 \\ 1 & 2 & 2 & 6 \\ 1 & 2 & 2 & 7 \\ 2 & 1 & 2 & 8 \\ \hline \end{array}$$

Легко, например, видеть, что состояния 1 и 2 неэквивалентны, поскольку оказываются неравными соответствующие строки матрицы Φ: $\varphi_1 \neq \varphi_2$. Рассмотрим теперь пару состояний 2 и 3. Первое условие эквивалентности этих состояний выполняется: $\varphi_2 = \varphi_3$. Проверим, выполняется ли второе условие. Сравнивая строки ψ_2 и ψ_3 матрицы переходов, мы видим, что задача сводится к проверке эквивалентности состояний 5 и 3, а также состояний 1 и 5. Но состояние 5 оказывается неэквивалентным состоянию 3, так как $\varphi_5 \neq \varphi_3$, следовательно, второе условие эквивалентности состояний 2 и 3 не выполняется и, значит, состояния 2 и 3 неэквивалентны.

К иному результату приходим мы, например, при сопоставлении состояний 4 и 6. Первое условие эквивалентности этих состояний выполняется ($\varphi_4 = \varphi_6$), а второе сводится к проверке эквивалентности состояний 1 и 5, а также эквивалентности состояний 3 и 8. Условием эквивалентности состояний 1 и 5 оказывается эквивалентность состояний 4 и 7, а условием эквивалентности последних, в свою очередь, оказывается эквивалентность состояний 3 и 8. Наконец, условием эквивалентности состояний 3 и 8 является эквивалентность состояний 4 и 6, с которых мы начали. Поскольку все пары состояний, затронутые проверкой, удовлетворяют первому условию эквивалентности и поскольку дальнейший анализ уже не приводит к рассмотрению каких-либо новых пар, можно считать, что условия эквивалентности состояний 4 и 6 выполняются.

Граф условий эквивалентности. Описанный процесс проверки состояний автомата на эквивалентность удобно отобразить ориентированным графом,

вершины которого соответствуют некоторым парам состояний, а дуги представляют условия эквивалентности типа «состояния, входящие в пару $\{\gamma_i, \gamma_j\}$, эквивалентны, если эквивалентны элементы пары $\{\gamma_k, \gamma_l\}$». Условимся при этом, что дуга исходит из вершины, соответствующей паре $\{\gamma_i, \gamma_j\}$, и заходит в вершину, соответствующую паре $\{\gamma_k, \gamma_l\}$. Пара $\{\gamma_i, \gamma_j\}$ обозначается на графе как ij.

Построение этого графа начинается с введения вершин, соответствующих таким парам состояний, которые удовлетворяют первому условию эквивалентности, и заканчивается представлением в виде дуг всех условий второго типа. При этом в граф могут быть введены вершины, соответствующие некоторым парам состояний, не удовлетворяющим первому условию эквивалентности. Эти вершины отмечаются.

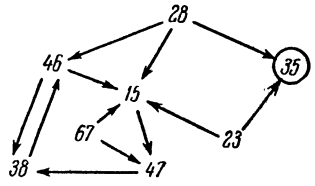


Рис. 4.2.1.

Назовем описанный граф *графом условий эквивалентности состояний автомата*, или просто *графом условий эквивалентности*. Для рассматриваемого нами примера полного автомата он будет иметь следующий вид (рис. 4.2.1).

В данном случае в граф входит лишь одна отмеченная вершина, соответствующая паре состояний 3 и 5, не удовлетворяющих первому условию эквивалентности. На графе хорошо видно, что второму условию эквивалентности не удовлетворяет пара 2 и 8, а также пара 2 и 3, поскольку пара 3 и 5 является производной от этих пар. Остальные же из представленных на графе пар состоят из эквивалентных состояний.

После выявления всех пар взаимно эквивалентных состояний легко находится разбиение множества всех состояний автомата на классы эквивалентности. В данном случае таких классов оказывается четыре: $\{1, 5\}$, $\{2\}$, $\{3, 8\}$, $\{4, 6, 7\}$.

Минимизация числа состояний полного автомата. Два автомата назовем *эквивалентными*, если для любого состояния одного из автоматов найдется эквивалентное ему состояние другого автомата. Очевидно, что число состояний у эквивалентных автоматов

может различаться: если один автомат имеет несколько взаимно эквивалентных состояний, то им достаточно поставить в соответствие одно и то же состояние другого автомата.

Рассмотрим следующую задачу: найти автомат, эквивалентный заданному и вместе с тем обладающий минимально возможным числом состояний. Она называется задачей *минимизации числа внутренних состояний автомата*, или просто задачей *минимизации числа состояний*. Практический интерес к этой задаче произрастает на почве стремления к упрощению автомата, реализующего заданное отображение множества входных последовательностей во множество выходных последовательностей.

Если рассматриваемый автомат является полным, то эта задача решается довольно просто. Достаточно найти разбиение множества состояний на классы эквивалентности, а затем построить новый автомат, состояния которого будут поставлены во взаимно однозначное соответствие с этими классами. Пусть состояние γ_i нового автомата будет при этом соответствовать классу Γ_i старого автомата. В этом случае строка φ_i новой матрицы Φ будет равна каждой из строк старой матрицы Φ, соответствующих состояниям, принадлежащим классу эквивалентности (очевидно, что такие строки равны между собой), а строка ψ_i новой матрицы переходов Ψ может быть получена из входящих в соответствующую группу строк старой матрицы Ψ путем замены в них символов состояний на номера классов эквивалентности, которым эти состояния принадлежат. Таким образом, число состояний нового автомата, эквивалентного старому, будет равно числу классов эквивалентности, на которые разбивается множество состояний исходного автомата. Очевидно, что это решение единственно (с точностью до перенумерации состояний в полученном автомате).

Например, поскольку в рассматриваемом нами автомате множество состояний разбивается на следующие классы эквивалентности:

$$\text{класс } 1 = \{1, 5\},\ \text{класс } 2 = \{2\},\ \text{класс } 3 = \{3, 8\}$$
$$\text{и класс } 4 = \{4, 6, 7\},$$

матрицы переходов и выходов этого автомата преобразуются в более компактные матрицы

$$\Psi = \begin{matrix} a & b & c \end{matrix}\begin{bmatrix} 4 & 2 & 1 \\ 1 & 1 & 4 \\ 3 & 1 & 4 \\ 1 & 3 & 4 \end{bmatrix}\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \qquad \Phi = \begin{matrix} a & b & c \end{matrix}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \\ 2 & 1 & 2 \\ 1 & 2 & 2 \end{bmatrix}\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

путем замены в матрице переходов символа 5 на 1, 8 на 3, 6 на 4 и 7 на 4 и последующего совмещения одинаковых строк.

Реализация частичных автоматов. Задача минимизации числа состояний частичного автомата оказывается значительно сложнее и ставится несколько по-другому.

Пусть A — некоторый частичный автомат, поведение которого задано на области $P(A)$, причем выходные последовательности, которыми автомат отвечает на допустимые входные последовательности при соответствующих начальных состояниях, могут быть также частичными. Чтобы не налагать дополнительные ограничения на матрицу переходов и матрицу выходов, будем по-прежнему считать, что рассматриваемый автомат является синхронным. Матрицы асинхронных автоматов представляют собой более частный случай, в связи с чем получаемые в этом параграфе результаты окажутся применимыми и к асинхронным автоматам.

Здесь полезно расширить понятие эквивалентности автоматов, введя следующие определения.

Будем говорить, что *последовательность d_i реализует последовательность d_j*, если последнюю можно доопределить так, что она совпадет с первой. Очевидно, что определение имеет нетривиальный смысл, если последовательность d_j является частичной. Например, последовательность 5 4 — 4 — 6 2 реализует последовательность 5 4 — — — — 6 —. Далее, будем говорить, что состояние γ_i автомата A_1 реализует состояние γ_j частичного автомата A_2 на области $P(A_2)$ определения последнего, если для любой входной последовательности, допустимой при состоянии γ_j, отвечающая ей выходная последовательность автомата A_2 будет реализоваться выходной последовательностью автомата A_1, отвечающей этой

же входной последовательности и его начальному состоянию γ_i.

Будем говорить, что частичный автомат A, заданный на области $P(A)$, *реализуется* некоторым другим автоматом A_i, если для любого состояния автомата A найдется реализующее его на области $P(A)$ состояние автомата A_i. Очевидно, что это отношение несимметрично: из того, что автомат A реализуется автоматом A_i, не следует, что автомат A_i реализуется автоматом A. В тех конкретных случаях, когда данное отношение оказывается симметричным, мы имеем дело с отношением эквивалентности рассматриваемых автоматов, в общем же случае говорить об эквивалентности не приходится.

Итак, при рассмотрении частичных автоматов задача минимизации числа состояний приобретает следующую формулировку: *для заданного частичного автомата найти реализующий его автомат, обладающий минимально возможным числом состояний.*

Совместимость состояний. Две последовательности назовем *совместимыми*, если существует третья последовательность, реализующая каждую из них. Например, последовательности 1 2 2 — 3 — — 5 и — 2 — 4 3 5 — — совместимы, так как любая из них реализуется последовательностью 1 2 2 4 3 5 — 5. Легко заметить нетранзитивность данного отношения: например, последовательность 2 — — 3 совместима с — 1 — 3, а последняя совместима с 1 1 2 —, однако последовательности 2 — — 3 и 1 1 2 — несовместимы.

Состояния γ_i и γ_j некоторого частичного автомата будем называть *совместимыми*, если при любой входной последовательности, допустимой при начальных состояниях γ_i и γ_j, соответствующие этим состояниям выходные последовательности будут совместимы.

Как определить по заданным матрицам частичного автомата, совместима ли некоторая пара его состояний?

Решение этой задачи напоминает проверку состояний полного автомата на эквивалентность. Два состояния частичного автомата оказываются совместимыми, если, во-первых, совместимы соответствующие этим состояниям строки матрицы Φ (две строки называются совместимыми, если их можно доопределить так, что они совпадут), и если, во-вторых, соответствующие рассмат-

риваемым состояниям строки матрицы Ψ будут равны с точностью до символов совместимых состояний.

Совокупность такого рода условий представляется *графом условий совместимости*, вершины которого ставятся во взаимно однозначное соответствие с парами совместимых состояний, а дуги задают условия совместимости типа «пара состояний $\{\gamma_i, \gamma_j\}$ совместима, если совместима пара $\{\gamma_k, \gamma_l\}$». Заметим, что, в отличие от графа условий эквивалентности, мы не рассматриваем здесь несовместимых пар.

Например, частичный автомат, представленный парой матриц

$$\Psi = \begin{array}{c|ccc} & a & b & c \\ \hline 1 & 2 & - & - \\ 2 & 3 & - & - \\ 3 & 1 & 5 & 2 \\ 4 & 1 & 5 & 5 \\ 5 & - & 6 & - \\ 6 & - & 4 & - \end{array} \qquad \Phi = \begin{array}{c|ccc} & a & b & c \\ \hline 1 & 1 & - & - \\ 2 & - & 1 & 1 \\ 3 & - & - & 1 \\ 4 & - & - & - \\ 5 & 0 & - & 1 \\ 6 & 0 & 0 & - \end{array}$$

будет характеризоваться графом условий совместимости, представленным на рис. 4.2.2.

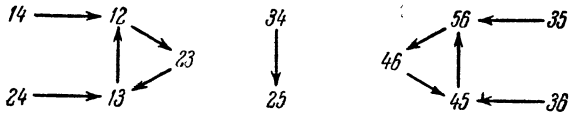


Рис. 4.2.2.

Совокупность состояний, каждое из которых совместимо с каждым, назовем *группой совместимости*. Если совокупность, получаемая из некоторой группы совместимости Г добавлением любого нового состояния, уже не является группой совместимости, то группа совместимости Г называется *максимальной*.

Например, из рис. 4.2.2 можно найти следующие три максимальные группы совместимости: {1, 2, 3, 4}, {3, 4, 5, 6} и {2, 3, 4, 5}.

Совмещение состояний. Пусть состояния γ_i, γ_j, ..., γ_k частичного автомата A образуют группу совместимости. Назовем *совмещением* этих состояний

такое преобразование автомата A к реализующему его автомату A', при котором все перечисленные состояния заменяются одним состоянием, реализующим каждое из них. В общем случае это преобразование оказывается нетривиальным, в связи с чем рассмотрим его более подробно.

В простейшем случае состояниям $\gamma_i, \gamma_j, \ldots, \gamma_k$ будут соответствовать одинаковые строки в матрице Ψ и одинаковые строки в матрице Φ. Очевидно, что если в каждой из матриц мы удалим все эти строки, кроме одной, с одинаковым номером, а также заменим на этот номер символы выброшенных строк всюду в матрице Ψ, то мы получим пару матриц, эквивалентную исходной, то есть представляющую автомат, эквивалентный автомату A.

Несколько сложнее эта операция выполняется, когда соответствующие состояниям $\gamma_i, \gamma_j, \ldots, \gamma_k$ строки матриц оказываются не равными, но совместимыми. В этом случае в каждой из матриц они заменяются одной строкой, которая реализует их, имея при этом максимально возможное число символов неопределенности «—». Например, для строк 2—66——5, 2——68—— и ———68—5 такой строкой будет 2—668—5 (вне зависимости от того, какой именно матрице, Ψ или Φ, принадлежат рассматриваемые строки). Как и в предыдущем случае, символ новой строки всюду в матрице Ψ заменяет символы выброшенных строк. При таком преобразовании, также называемом совмещением состояний $\gamma_i, \gamma_j, \ldots, \gamma_k$, мы получаем автомат, в общем случае не эквивалентный исходному, а *реализующий* его (можно сказать, что новый автомат эквивалентен старому лишь на области определения поведения последнего).

В рассмотренных ситуациях состояния $\gamma_i, \gamma_j, \ldots, \gamma_k$ оказывались совместимыми, причем эта совместимость носила безусловный характер, не будучи связана с совместимостью каких-либо других состояний. В более общем случае, однако, мы сталкиваемся с цепной зависимостью, при которой необходимым условием совмещения состояний одной группы является совмещение состояний внутри некоторых других групп.

Если группа совместимости содержит лишь два состояния γ_i и γ_j, то необходимым условием совмещения

состояний этой группы является совмещение внутри всех пар, производных от $\{\gamma_i, \gamma_j\}$. Это условие хорошо представляется графом условий совместимости.

Например, на графе, изображенном на рис. 4.2.2, видно, в частности, что состояние 3 можно совместить с состоянием 4 только в том случае, если будут совмещены также состояния 2 и 5. Реализуя это совмещение, мы заменим пару $\{3, 4\}$ одним новым состоянием, которому можно присвоить, например, номер 3, и заменим пару $\{2, 5\}$ новым состоянием, которое обозначим через 2. Выражаясь более кратко, будем говорить в данном случае о следующем преобразовании над множеством внутренних состояний:

$$\{1\} \to 1,\ \{2, 5\} \to 2,\ \{3, 4\} \to 3,\ \{6\} \to 4$$

(состояние 6 для порядка перенумеруем на 4).

Соответствующие новым состояниям строки матрицы Ψ получаются согласно этому преобразованию: например, две строки

$$\left|\begin{matrix} 1 & 5 & 2 \\ 1 & 5 & 5 \end{matrix}\right| \begin{matrix} 3 \\ 4 \end{matrix},$$

заменяются одной строкой

$$|1\ 2\ 2|3$$

Строки матрицы Φ получаются как и прежде: каждая новая строка должна реализовать заменяемые ею старые.

В рассматриваемом случае преобразование матриц Ψ и Φ приводит к следующему результату:

$$\Psi = \begin{bmatrix} 2 & - & - \\ 3 & 4 & - \\ 1 & 2 & 2 \\ - & 3 & - \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \qquad \Phi = \begin{bmatrix} 1 & - & - \\ 0 & 1 & 1 \\ - & - & 1 \\ 0 & 0 & - \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Необходимо отметить существенное отличие операции совмещения состояний частичного автомата от соответствующей операции над полным автоматом. Может оказаться, что, вводя новое состояние, реализующее некоторую пару совместимых состояний частичного автомата, нельзя будет одновременно удалить оба элемента пары, поскольку они могут входить также в производные пары.

В рассматриваемом примере (рис. 4.2.2) с такой ситуацией мы встречаемся, пытаясь совместить состояния 1 и 2. Производными от этой пары являются пары {2, 3} и {1, 3}. Это значит, что наряду с состоянием, реализующим пару {1, 2}, мы должны ввести также новые состояния, реализующие и эти производные пары, и лишь после этого сможем удалить реализованные таким образом состояния 1, 2 и 3. В результате данного преобразования число состояний автомата остается прежним. В других случаях совмещение некоторых состояний может привести даже к увеличению общего числа состояний.

Перейдем, наконец, к рассмотрению общего случая, когда группа совместимости $\Gamma_u = \{\gamma_i, \gamma_j, \ldots, \gamma_k\}$ содержит произвольное число состояний. В этом случае необходимым условием совмещения всех состояний данной группы является совмещение внутри всех групп, производных от нее. При этом группа $\Gamma_v = \{\gamma_p, \gamma_q, \ldots, \gamma_r\}$ называется *производной* от группы Γ_u, если выполняется одно из следующих условий:

а) в матрице Ψ существует столбец, пересечение которого со строками $i, j, \ldots, k$ порождает элементы, совокупность значений которых образует множество $\{p, q, \ldots, r\}$ (некоторые из порожденных элементов могут обладать также значением «—»), — в этом случае группа Γ_v называется *непосредственной* производной от группы Γ_u,

б) существует группа Γ_w, производная от Γ_u и такая, что Γ_v оказывается производной от Γ_w.

Очевидно, что совмещение состояний внутри всех производных групп обеспечивается совмещением внутри максимальных из них, то есть таких, которые не являются собственными подмножествами других производных.

Учитывая сказанное, можно следующим образом сформулировать алгоритм совмещения состояний, образующих произвольную группу совместимости Γ_u:

1. Найти множество групп, производных от Γ_u.

2. Выделить среди них совокупность максимальных групп и, добавив в нее группу Γ_u, образовать таким образом множество Γ_u^*.

3. Реализовать задаваемое множеством Γ_u^* преобразование матриц Ψ и Φ, для чего

а) для каждой группы из Γ_u^* ввести в матрицы новую строку, представляющую состояние, реализующее каждое состояние данной группы,

б) удалить строки, соответствующие реализованным таким образом состояниям,

в) всюду в матрице Ψ заменить символы реализуемых состояний символами реализующих.

Правила образований новых строк поясним на следующем примере. Пусть $\Gamma_u = \{1, 5, 6, 9\}$, $\Gamma_u^* = \{\{1, 5, 6, 9\}, \{2, 4\}, \{3, 4, 6\}, \{2, 9\}\}$. Допустим также, что группе Γ_u соответствует следующая совокупность строк матриц Ψ и Φ:

Ψ					Φ					
—	2	4	9	5	2	1	5	—	—	1
3	—	—	2	—	—	—	—	—	3	5
3	4	3	—	6	—	1	—	—	—	6
4	—	6	—	—	—	1	5	—	3	9

Для обозначения новых состояний можно использовать некоторые из символов реализуемых состояний, в связи с чем можно рассмотреть преобразование

$$\{1, 5, 6, 9\} \to 1,\ \{2, 4\} \to 2,\ \{3, 4, 6\} \to 3,\ \{2, 9\} \to 4.$$

В этом случае группе Γ_u будет соответствовать следующая вновь образуемая строка:

$$|32341| \quad |215-3|1$$

Минимизация частичного автомата. Так мы будем называть минимизацию числа внутренних состояний частичного автомата, то есть нахождение автомата, реализующего заданный частичный автомат и обладающего при этом условии минимальным числом состояний.

Решение этой задачи сводится к нахождению некоторого преобразования множества внутренних состояний автомата, при котором некоторые группы совместимости заменяются новыми состояниями: производится совмещение состояний внутри этих групп. При этом допускается пересечение групп, то есть одно и то же состояние

может оказаться совмещенным с различными другими состояниями, в результате чего оно внесет свой вклад в образование нескольких новых состояний (здесь можно провести аналогию с процессом сжатия троичной матрицы: одна и та же строка ее может быть склеена с несколькими различными другими строками, и это может оказаться выгодным с точки зрения используемого критерия оптимальности результата).

Получаемый при таком преобразовании новый автомат будет реализовывать старый только в том случае, если задающая преобразование совокупность групп совместимости будет удовлетворять следующим условиям: во-первых, каждое из состояний исходного автомата должно входить по крайней мере в одну из групп, во-вторых, каждая производная любой из этих групп (также являющаяся некоторой группой) должна целиком входить в какую-либо из групп этой же совокупности. Удовлетворяющую этим условиям совокупность групп совместимости будем называть *правильной группировкой.*

Легко показать, что множество всех максимальных групп совместимости является правильной группировкой. Это означает, что задаваемое им преобразование множества состояний автомата приводит к автомату, реализующему исходный автомат.

Например, автомат, граф условий совместимости которого показан на рис. 4.2.2, может быть реализован автоматом, получаемым с помощью преобразования:

$$\{1, 2, 3, 4\} \to 1, \quad \{3, 4, 5, 6\} \to 2, \quad \{2, 3, 4, 5\} \to 3.$$

В этом случае мы получим автомат с тремя состояниями, представляемый парой матриц

$$\Psi = \begin{matrix} a\,b\,c \\ \left[\begin{matrix} 1\,2\,3 \\ 1\,2\,3 \\ 1\,2\,3 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \qquad \Phi = \begin{matrix} a\,b\,c \\ \left[\begin{matrix} 1\,1\,1 \\ 0\,0\,1 \\ 0\,1\,1 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}.$$

Заметим, что элементу ψ_1^b можно было присвоить значение 3.

Этот результат уже лучше, чем полученный ранее, поскольку в данном случае вместо четырех состояний мы получили три. Однако это не наилучшее решение. Дей-

ствительно, существует преобразование $\{1, 2, 3\} \to 1$, $\{4, 5, 6\} \to 2$, приводящее к автомату

$$\Psi = \begin{matrix} a\,b\,c \\ \left[\begin{matrix} 1\,2\,1 \\ 1\,2\,2 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 2 \end{matrix} \qquad \Phi = \begin{matrix} a\,b\,c \\ \left[\begin{matrix} 1\,1\,1 \\ 0\,0\,1 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 2 \end{matrix}$$

имеющему только два внутренних состояния и в то же время реализующему исходный автомат.

Как же находить наилучшее решение? Из рассмотренного примера видно, что приводящая к нему правильная группировка не обязательно должна состоять из максимальных групп совместимости — таких групп может и совсем там не оказаться. Можно, конечно, осуществить перебор различных совокупностей групп совместимости, выделить из них правильные группировки, а затем выбрать среди последних минимальную, то есть состоящую из минимального числа групп. При этом можно воспользоваться тем обстоятельством, что всевозможные группы совместимости представляются подмножествами максимальных групп совместимости, нахождение которых относительно несложно.

Однако такой перебор весьма громоздок и может быть реализован только при рассмотрении довольно простых автоматов, число состояний которых невелико.

Графическая формулировка задачи. Каждому частичному автомату можно поставить в соответствие некоторый граф условий совместимости. Справедливо и обратное утверждение: каждому графу условий совместимости можно поставить в соответствие некоторый частичный автомат. Это утверждение нетрудно доказать путем построения соответствующего автомата, которое можно провести, например, поставив в соответствие каждой дуге графа некоторый столбец матрицы Ψ, а каждой паре несовместимых состояний — столбец матрицы Φ. Отсюда следует, что задачу минимизации частичного автомата можно сформулировать над ориентированным графом произвольного вида.

Напомним, что правильная группировка должна удовлетворять двум условиям. Введем понятие *квазиправильной группировки*, называя так группировку, удовлетворяющую второму из этих условий, но не обязательно удовлетворяющую первому из них. Это значит, что

любая производная любой из групп данной группировки должна целиком входить в некоторую группу этой же группировки. Однако не требуется, чтобы каждое состояние автомата принадлежало какой-либо из групп — могут существовать состояния, не принадлежащие ни одной из них.

Допустим, что такие состояния имеются. Если мы образуем из них одноэлементные группы и дополним ими квазиправильную группировку, то получим правильную группировку.

Заметим далее, что если некоторая совокупность совместимых пар (будем называть ее *парной группировкой*) представляет собой квазиправильную группировку, то ей соответствует множество вершин такого подграфа графа условий совместимости, который получается из последнего путем последовательного удаления некоторых начальных вершин вместе с исходящими из них дугами. Начальными мы называем такие вершины ориентированного графа, в которые не заходят никакие дуги. Удовлетворяющий данному условию подграф ориентированного графа назовем *усеченным*. Очевидно, что в нем сохраняются все пути, исходящие из остающихся вершин.

Очевидно и обратное: каждый усеченный подграф графа условий совместимости соответствует некоторой квазиправильной парной группировке. Одна из таких группировок представляется, в частности, самим графом условий совместимости. Назовем эту парную группировку *полной*.

Каждая квазиправильная парная группировка содержит перечень таких пар состояний, которые можно совместить одновременно, не совмещая при этом никаких других состояний. С другой стороны, каждый такой перечень образует некоторую квазиправильную парную группировку. А ведь любое допустимое преобразование частичного автомата, приводящее его к некоторому другому автомату, реализующему исходный, соответствует перечню такого типа, то есть квазиправильной парной группировке. Отсюда следует, что поиск оптимального преобразования, приводящего к минимальному автомату, можно отобразить некоторым перебором усеченных подграфов графа условий совместимости.

Рассматривая какой-либо усеченный подграф, можно допустить, что все не отображенные в нем пары состояний являются несовместимыми, и искать решение при этом предположении. Поскольку усеченный подграф проще того графа, из которого он взят, такую задачу решить можно также проще. В частности, можно найти соответствующее указанному предположению множество всех максимальных групп совместимости и рассмотреть задаваемое им преобразование.

Например, рассмотрим следующие три усеченные подграфа (рис. 4.2.3) графа, изображенного на рис. 4.2.2. Этим усеченным подграфам будут соответствовать

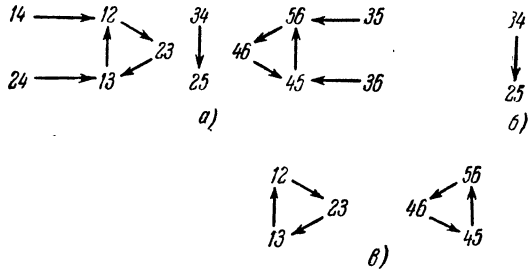


Рис. 4.2.3.

уже знакомые нам решения: преобразование $\{1, 2, 3, 4\} \to \to 1$, $\{3, 4, 5, 6\} \to 2$, $\{2, 3, 4, 5\} \to 3$ для рис. 4.2.3, *а*; преобразование $\{1\} \to 1$, $\{2, 5\} \to 2$, $\{3, 4\} \to 3$, $\{6\} \to 4$ для рис. 4.2.3, *б* и преобразование $\{1, 2, 3\} \to 1$, $\{4, 5, 6\} \to 2$ для рис. 4.2.3, *в*.

Перебор усеченных подграфов целесообразно упорядочить, с целью его сокращения. Рассмотрим один из способов такого упорядочения.

Нахождение точного решения. Нахождение минимального автомата можно начать с получения множества всех максимальных групп совместимости. Эту задачу можно рассматривать как задачу поиска всех максимальных полных подграфов в симметрическом графе, уже знакомую нам по предыдущей главе. В данном случае вершины графа будут соответствовать состояниям, а его ребра будут представлять пары совместимых состояний.

Итак, допустим, что множество всех максимальных групп совместимости уже найдено. Обозначим его через G и положим, что его мощность равна g. Если число состояний минимального автомата равно k, то должна существовать некоторая минимальная правильная группировка мощности k. Каждая группа правильной группировки представляет собой подмножество некоторого элемента множества G; будем говорить в этом случае, что группировка *вкладывается* в множество G. Если $k<g$, то это означает, что из множества G можно выбросить некоторый элемент (то есть некоторую максимальную группу совместимости), получив таким образом множество H, причем минимальная правильная группировка будет вкладываться в это множество H. Поскольку нас интересуют именно такие правильные группировки, мощность которых меньше g (решение, приводящее к автомату с g состояниями, у нас уже имеется: оно задается самим множеством G), то мы ищем только их. Если, например, получая множество H путем выбрасывания некоторого элемента из множества G, мы вынуждены будем затем каждый раз расширять это множество H, чтобы в него можно было вкладывать правильные группировки, причем каждый раз мощность множества H будет становиться не меньше, чем g, то это будет свидетельствовать о том, что решение, представляемое самим множеством G, является оптимальным.

Рассмотрим пример графа условий совместимости, представленный на рис. 4.2.4.

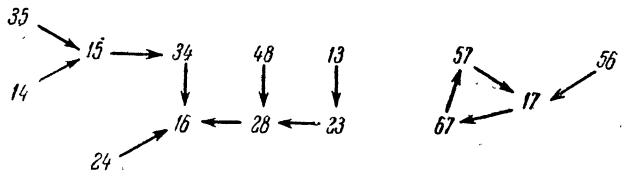


Рис. 4.2.4.

Соответствующее этому графу множество G всех максимальных групп совместимости может быть получено одним из алгоритмов, изложенных в параграфе 6 предыдущей главы. Оказывается, что оно состоит из

групп $\{1,5,6,7\}$, $\{1,3,4\}$, $\{2,3\}$, $\{2,4,8\}$ и $\{3,5\}$. Множество G удобно представить булевой матрицей $\boldsymbol{G}=\|G \ni C\|$, где через C обозначено множество внутренних состояний рассматриваемого частичного автомата.

$$\boldsymbol{G}=\begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix}$$

Будем искать минимальную правильную группировку L, учитывая то обстоятельство, что она должна вкладываться в множество G и должна покрывать все множество состояний C. Можно искать эту группировку в форме соответствующей булевой матрицы $\boldsymbol{L}=\|L \ni C\|$, получаемой из матрицы $\boldsymbol{G}$ путем удаления некоторых ее строк, замены значений некоторых ее элементов с 1 на 0, а также путем расщепления строк. Естественно, что, поскольку мощность множества G равна 5, нас будут интересовать лишь такие правильные группировки, мощность которых не более 4.

Обратим внимание на строку 1. Она оказывается *особенной*, то есть в матрице существуют столбцы, содержащие единицу лишь на пересечении с этой строкой (в данном случае столбцы 6 и 7). Очевидно, что какова бы ни была найденная правильная группировка, в ней должна иметься группа, являющаяся подмножеством максимальной группы совместимости, представленной строкой 1, и не являющаяся подмножеством какой-либо другой группы из G. Следовательно, мы не можем удалить из матрицы эту строку. Из нее, может быть, удастся выбросить лишь некоторые единицы (но только не в столбцах 6 или 7), но это не приведет непосредственно к уменьшению мощности рассматриваемого множества групп, поэтому не будем пока проверять наличие такой возможности. Аналогичное замечание можно сделать и в адрес строки 4 ($g_4^8=1$), также оказывающейся особенной.

Среди остальных строк особенно «удобной» оказывается строка 5. Соответствующая ей максимальная группа совместимости $\{3, 5\}$ является парой, которой на

графе условий совместимости соответствует начальная вершина. Поэтому удаление строки 5 происходит безболезненно: из графа удаляется лишь эта вершина.

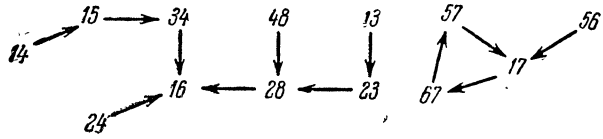


Рис. 4.2.5.

Итак, мы получаем граф, показанный на рис. 4.2.5, которому соответствует множество максимальных групп совместимости, представленное следующей булевой матрицей:

$$G = \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{matrix} 1\,2\,3\,4\,5\,6\,7\,8 \\ \left[\begin{matrix} 1\,0\,0\,0\,1\,1\,1\,0 \\ 1\,0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0\,0\,1 \end{matrix}\right] \end{matrix}$$

Этим самым мы находим уже лучшее решение, соответствующее автомату с четырьмя состояниями.

Рассмотрим теперь строку 3, которая представляет также пару состояний, в данном случае — состояния 2 и 3. Соответствующая вершина графа оказывается не начальной, и, удаляя ее, мы вынуждены одновременно выбросить и вершину 13. Посмотрим, к чему это приведет.

Текущие значения матрицы G и графа условий совместимости должны соответствовать друг другу. Поэтому мы должны построить новое множество максимальных групп, соответствующее новому виду графа. В данном случае мы должны учесть то обстоятельство, что пара $\{1, 3\}$ считается теперь несовместимой, так же как и пара $\{2, 3\}$. Пара $\{1, 3\}$ входит только в одну из максимальных групп совместимости, а именно, в группу $\{1, 3, 4\}$, представленную строкой 2 матрицы G. Удаление этой пары из множества совместимых пар приводит к расщеплению группы $\{1, 3, 4\}$ на две группы $\{1, 4\}$ и $\{3, 4\}$. Этот процесс отражается следующим изменением матрицы G: из нее выбрасываются строки 3 и 2, а вместо них вводятся новые строки, представляющие группы

{1, 4} и {3, 4}. Присвоив этим строкам те же номера 2 и 3, мы получим следующую матрицу:

$$G = \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{matrix} 1\,2\,3\,4\,5\,6\,7\,8 \\ \left[\begin{matrix} 1\,0\,0\,0\,1\,1\,1\,0 \\ 1\,0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0\,0\,1 \end{matrix}\right] \end{matrix}$$

соответствующую графу, представленному на рис. 4.2.6.

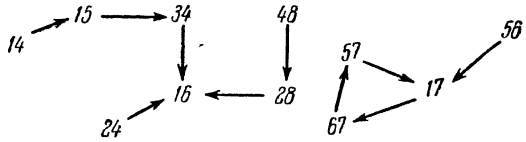


Рис. 4.2.6.

Теперь уже три строки матрицы G являются особенными, и можно рассчитывать на удаление лишь строки 2, представляющей пару {1, 4}. Действительно, соответствующая этой паре вершина графа оказывается начальной, в связи с чем она легко отсекается. Матрица G принимает следующее значение:

$$G = \begin{matrix} \\ 1 \\ 3 \\ 4 \end{matrix} \begin{matrix} 1\,2\,3\,4\,5\,6\,7\,8 \\ \left[\begin{matrix} 1\,0\,0\,0\,1\,1\,1\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0\,0\,1 \end{matrix}\right] \end{matrix}$$

Становится очевидным, что число строк в ней сократить дальше нельзя: все строки матрицы особенные. Однако, может быть, удастся уменьшить число единиц в матрице? Такого рода минимизация также представляет определенный интерес, поскольку она позволяет находить преобразования, при которых новые состояния получаются в результате совмещения меньшего количества старых состояний, и это приводит к повышению неопределенности поведения получаемого автомата, то есть к сужению области, на которой определяется его поведение. Повышение этой неопределенности, в свою очередь, представляет собой источник дополнительных возможностей упрощения структуры автомата.

Очевидно, что единицы, единственные в своих столбцах, выбрасывать нельзя. Поэтому стоит обратить

внимание лишь на два элемента матрицы G: элемент g_3^4 и элемент g_4^4. Замена значения элемента g_3^4 с 1 на 0 эквивалентна условному переводу пары $\{3,4\}$ из множества совместимых пар в множество несовместимых и должна поэтому сопровождаться удалением соответствующей вершины 34 из графа условий совместимости. Но, как это видно на графе, вместе с ней выбрасывается и вершина 15, соответствующая паре $\{1,5\}$, что приводит к расщеплению группы $\{1,5,6,7\}$ на две группы $\{1,6,7\}$ и $\{5,6,7\}$ и к увеличению числа строк в матрице G, причем все строки оказываются особенными. Поэтому значение элемента g_3^4 менять нельзя.

Удалению же другой единицы — замене значения элемента g_4^4 с 1 на 0 — соответствует отсечение в графе вершин 24 и 48. Обе вершины являются начальными, поэтому никаких затруднений при этом не возникает. В результате граф условий совместимости принимает следующий окончательный вид (рис. 4.2.7).

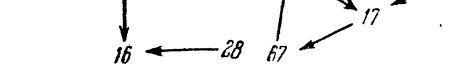


Рис. 4.2.7.

Полученное при этом значение G представляет искомую минимальную правильную группировку L:

$$L = \| L \ni C \| = \begin{matrix} 1\,2\,3\,4\,5\,6\,7\,8 \\ \left[\begin{matrix} 1\,0\,0\,0\,1\,1\,1\,0 \\ 0\,0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0\,1 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 3 \\ 4 \end{matrix}$$

Предостережение. Заметим, что граф условий совместимости содержит не всю информацию, необходимую при минимизации числа состояний частичного автомата (хотя для рассмотренного примера она оказалась достаточной). Действительно, он представляет собой графическую форму перечня непосредственных производных от пар совместимых состояний, но не отражает аналогичного отношения между группами совместимости, содержащими более чем два состояния. Эта

дополнительная информация содержится в матрице Ψ и может быть извлечена оттуда по мере необходимости.

Эта информация — излишняя, пока рассматриваются множества всех максимальных групп совместимости, соответствующие усеченным подграфам графа условий совместимости. Именно с такими ситуациями мы и

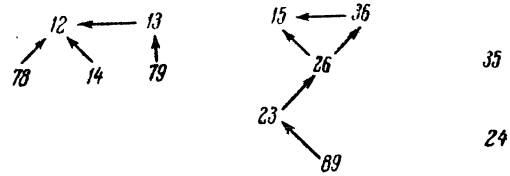


Рис. 4.2.8.

имели дело в разобранном примере. Данную информацию необходимо учитывать лишь при разложении некоторой группы совместимости на составные части (например, с целью поглощения их другими группами рассматриваемой группировки).

Например, автомат, заданный матрицами

$$\Psi = \left[\begin{array}{ccc|c} - & - & 2 & 1 \\ 2 & 3 & 1 & 2 \\ 6 & - & 1 & 3 \\ 4 & - & 1 & 4 \\ 6 & 8 & - & 5 \\ - & 6 & 5 & 6 \\ 1 & - & 3 & 7 \\ 2 & 3 & - & 8 \\ 3 & 2 & 1 & 9 \end{array}\right], \quad \Phi = \left[\begin{array}{ccc|c} - & - & 0 & 1 \\ - & - & 0 & 2 \\ - & - & 0 & 3 \\ 0 & 1 & - & 4 \\ 0 & 0 & - & 5 \\ 1 & 1 & - & 6 \\ 1 & 0 & 1 & 7 \\ 1 & 0 & 1 & 8 \\ 1 & 0 & 1 & 9 \end{array}\right]$$

(граф условий совместимости этого автомата показан на рис. 4.2.8), обладает множеством G всех максимальных групп совместимости, представленным следующей матрицей:

$$G = \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left[\begin{array}{ccccccccc|c} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 3 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 5 \end{array}\right] \end{array}$$

Строки 2, 3, 4 и 5 этой матрицы являются особенными, и выбросить их нельзя. Появляется искушение удалить строку 1, поскольку все продукты ее разложения на пары, а именно, группы {1, 2}, {1, 3} и {2, 3}, поглощаются остальными группами, то есть группой {1, 2, 4}, {1, 3, 5} или {2, 3, 6}. Однако делать этого нельзя, так как группа {1, 2, 3} является производной от группы {7, 8, 9}, что легко увидеть на матрице Ψ. Такая операция была бы возможной, если бы, например, элемент ψ_9^1 вместо значения 3 имел значение «—».

§ 3. Структурные модели автоматов

Элементы и базисы. В то время как функциональная модель автомата содержит информацию о том, как автомат работает, *структурная модель* должна показывать, как он устроен, то есть из каких элементов он состоит и как эти элементы связаны между собой.

Структурная модель дискретного автомата отражает схему реального релейного устройства, и элементы дискретного автомата ставятся в соответствие некоторым конструктивным единицам релейного устройства, называемым *реальными элементами*. Последние, как правило, представляют собой элементарные релейные устройства и уже не состоят из других, более простых релейных устройств. В основе их функционирования лежит непосредственное использование некоторого физического явления, поэтому анализ их работы возможен лишь с привлечением понятий соответствующей теории (радиотехники, электроники, физики магнитных явлений, оптики и т. д.), в связи с чем мы сочтем возможным не касаться его в данной книге. Рассматривая какой-либо реальный элемент, мы будем интересоваться не тем, на использовании какого именно физического явления он построен, а тем, какую функцию он реализует и при каких условиях его рассмотрение можно заменить рассмотрением некоторого абстрактного элемента, который мы будем называть *автоматным элементом*.

Основное содержание теории дискретных автоматов составляет исследование отношений между функциональными и структурными моделями. Важнейшую роль

здесь играют задача анализа и, в особенности, задача синтеза. *Анализ* дискретного автомата заключается в получении функциональной модели автомата по заданной структурной модели. *Синтез* представляет собой обратную задачу: нахождение структурной модели автомата по заданной функциональной. Эта задача оказывается значительно сложнее в связи с тем, что, в отличие от задачи анализа, ее решение не единственно и среди многих возможных решений требуется выбрать оптимальное в каком-то смысле.

Исходная для синтеза информация задается следующим образом. Во-первых, описывается функциональная модель автомата. Во-вторых, указывается, из каких элементов его можно синтезировать, то есть задается *элементный базис*. В-третьих, определяется *синтаксис структур*, то есть формулируются правила взаимных соединений элементов, выделяющие из всевозможных структур класс допустимых, или *правильных*. Синтаксис играет компенсирующую роль: заменяя реальные элементы абстрактными, мы допускаем некоторую идеализацию, которая тем не менее оказывается допустимой, пока синтезированные из данных элементов структуры являются правильными, то есть удовлетворяют синтаксису. Однако, как только мы переходим к рассмотрению неправильной структуры, может проявиться нежелательный эффект указанной идеализации: поведение реального релейного устройства, обладающего такой структурой, может существенно отклониться от поведения его абстрактного двойника.

Будем считать, что элементный базис в совокупности с правилами соединения элементов, то есть с синтаксисом структур, образует *базис синтеза* или просто *базис*.

В элементный базис могут входить самые разнообразные элементы, которые сами являются простейшими дискретными автоматами. Выбор их диктуется как уровнем развития технологии производства реальных релейных элементов, так и требованиями, предъявляемыми к базису в целом со стороны методов синтеза. Основные требования, которым должен удовлетворять базис, — это полнота и эффективность.

Базис называется *полным* относительно некоторого класса автоматов, если в нем может быть синтезирован

любой автомат этого класса. Вопрос о том, удовлетворяет ли некоторый базис этому принципиально важному требованию, решается на довольно абстрактном уровне.

Требование *эффективности* базиса не столь определенно и может быть сформулировано примерно так: более эффективным является тот базис, синтезируемые в котором структуры оказываются лучшими в каком-то смысле, то есть проще, дешевле, надежнее и т. д. При определении эффективности базиса необходимо учитывать, с одной стороны, ряд свойств реальных элементов,

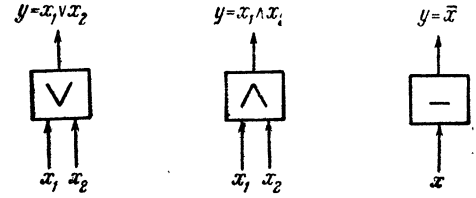


Рис. 4.3.1.

для чего может потребоваться некоторое расширение модели дискретного автомата, отражающее эти свойства. С другой стороны, эффективность базиса оказывается в зависимости от применяемых методов синтеза, поскольку для одних базисов эти методы могут быть развиты сильнее, для других — слабее. Таким образом, эффективность базиса может повышаться как путем совершенствования технологии производства реальных элементов, так и путем улучшения методов синтеза в данном базисе.

Какие же все-таки элементы могут входить в базис? Мы рассмотрим два типа автоматных элементов: логические элементы и элементы памяти.

Логическими элементами называются элементарные комбинационные автоматы, функциональные свойства которых представляются достаточно простыми булевыми функциями. Например, широко известны логические элементы, реализующие булевы функции $x_1 \vee x_2$, $x_1 \wedge x_2$ и $\bar{x}$ (графические представления этих элементов приведены на рис. 4.3.1). Получают распространение аналогичные многовходовые элементы, называемые *дизъюнкторами* и *конъюнкторами* и реализующие функ-

ции $x_1 \vee x_2 \vee \ldots \vee x_k$ и $x_1 \wedge x_2 \wedge \ldots \wedge x_k$ (рис. 4.3.2, *а*), а также инверсии этих функций $\overline{x_1 \vee x_2 \vee \ldots \vee x_k}$ и $\overline{x_1 \wedge x_2 \wedge \ldots \wedge x_k}$ (рис. 4.3.2, *б*).

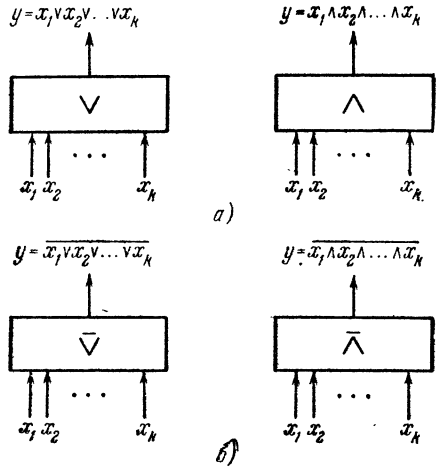


Рис. 4.3.2.

Элементами памяти служат некоторые элементарные последовательностные автоматы. Мы ограничимся рассмотрением лишь двух из них: элемента задержки и триггера.

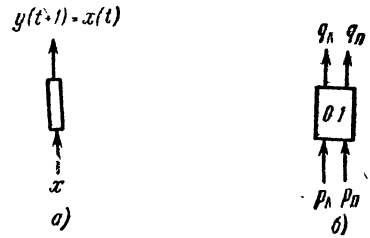


Рис. 4.3.3.

Элемент задержки можно рассматривать как элементарный синхронный автомат, функции которого сводятся к задержке на один такт значения одной логической переменной: значение входной переменной в момент t всегда равно значению выходной переменной в момент $t+1$. Графическое изображение элемента задержки дано на рис. 4.3.3, *а*.

Триггером мы назовем асинхронный автомат с двумя внутренними состояниями, которые могут фиксироваться и в каждое из которых при определенных условиях

автомат можно перевести. Среди различных возможных вариантов автомата, удовлетворяющих этим условиям, остановимся на следующем (рис. 4.3.3, *б*). Будем считать, что при комбинации 01 значений входных переменных $p_л$, $p_п$ триггер переходит в состояние 1, в котором выходные переменные $q_л$ и $q_п$ принимают значения 0 и 1, соответственно, при комбинации 10 триггер переходит в состояние 0, в котором $q_л=1$ и $q_п=0$, наконец, при комбинации 00 состояние триггера не изменяется. При комбинации 11 поведение триггера не определяется, в связи с чем рекомендуется избегать подачи этой комбинации на вход триггера.

Сети. Возьмем некоторую совокупность автоматных элементов, в которой могут присутствовать как одинаковые, так и различные элементы. Выделим из множества P всех входных полюсов этих элементов некоторое подмножество X, а из множества Q всех выходных полюсов — некоторое подмножество Y. Отобразим $P \setminus X$ в Q и будем считать, что это отображение задает множество *связей* между элементами множества P, с одной стороны, и множества Q, с другой стороны. Полученную таким образом структуру будем называть *сетью*, элементы множества X — *входными полюсами сети*, а элементы множества Y — ее *выходными полюсами*.

При небольшом числе элементов в сети удобно пользоваться графическим представлением, пример которого приведен на рис. 4.3.4 (входные и выходные полюсы сети обозначены через x_1, x_2, x_3 и y_1, y_2, соответственно). В других случаях целесообразнее представлять сеть в форме некоторого списка, содержащего перечень элементов и связей между ними.

Очевидно, что структура любого дискретного автомата может быть представлена некоторой сетью. Естественно возникает вопрос о справедливости обратного утверждения: можно ли рассматривать любую сеть как структуру некоторого автомата?

Оказывается, что нельзя. Необходимым условием такого рассмотрения является возможность проведения анализа, в результате которого однозначно определяются функциональные свойства автомата. Между тем уже на примере довольно простой сети, изображенной

на рис. 4.3.5, *а*, можно убедиться, что такой анализ не всегда возможен. Действительно, при значении 0 входной переменной x невозможно определить, какое же значение примет выходная переменная y: возникает противоречение типа «если $y=0$, то $y=1$, а если $y=1$, то

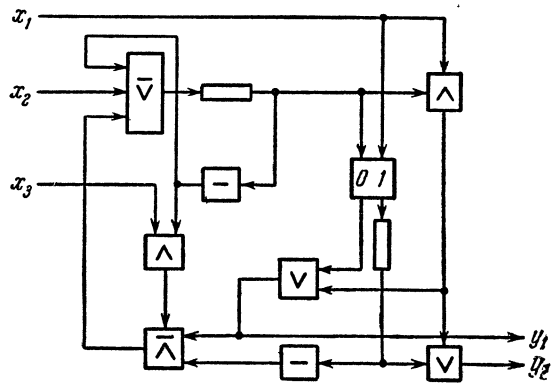


Рис. 4.3.4

$y=0$». Это противоречие можно было бы разрешить, например, добавив в сеть элемент задержки так, как это показано на рис. 4.3.5, *б*. В этом случае сеть можно рассматривать как синхронный автомат, в котором при

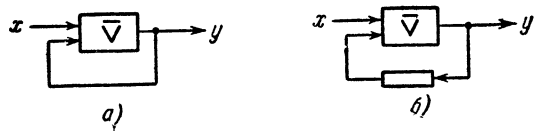


Рис. 4.3.5.

$x=0$ переменная y приняла бы последовательность значений 1, 0, 1, 0, 1, 0, ..., соответствующих следующим друг за другом моментам дискретного времени.

В связи со сказанным мы будем в дальнейшем рассматривать лишь *правильные сети*, то есть такие, которые можно рассматривать как структуры дискретных автоматов. Более того, мы ограничимся рассмотрением некоторых более узких классов сетей, соответствующих определенным классам автоматов.

Особое внимание в дальнейшем изложении будет уделено линейно-упорядоченным и k-ярусным сетям.

Линейно-упорядоченной мы назовем сеть, не имеющую контуров. Это значит, что элементы такой сети можно пронумеровать таким образом, что любая межэлементная связь будет соединять выходной полюс элемента с меньшим номером с входным полюсом элемента с бóльшим номером. Будем считать также, что в сети такого типа не может быть связей, соединяющих выходной полюс некоторого элемента с входным полюсом этого же элемента.

k-ярусной называется линейно-упорядоченная сеть, элементы которой могут быть разбиты на классы (ярусы) с номерами 1, 2, ..., k так, чтобы любая межэлементная связь связывала выходной полюс некоторого элемента i-го яруса с входным полюсом элемента $i+1$-го яруса ($i=1, 2, \ldots, k-1$). Кроме того, входные полюсы такой сети могут непосредственно связываться лишь с входными полюсами элементов первого яруса, а выходными полюсами сети могут служить лишь выходные полюсы элементов последнего, k-го яруса.

Комбинационные сети. Структуру комбинационного автомата удобно представлять линейно-упорядоченной сетью, составленной исключительно из логических элементов. Будем называть такую сеть *комбинационной*.

Если соответствующее какой-либо комбинационной сети отображение $P \setminus X$ в Q оказывается взаимно однозначным отображением $P \setminus X$ на некоторое подмножество из Q, то такая сеть называется *сходящейся*. Если же комбинационная сеть не удовлетворяет этому условию, то она называется *расходящейся*. Другими словами, связи расходящейся комбинационной сети могут ветвиться, соединяя выходной полюс некоторого элемента сети с несколькими входными полюсами других элементов, в то время как для сходящейся сети это недопустимо.

Если комбинационная сеть обладает лишь одним выходным полюсом (то есть реализует одну булеву функцию) и является к тому же сходящейся, то ее можно представить в виде одной формулы, задающей суперпозицию булевых функций, реализуемых элементами сети.

Для подобного представления расходящейся сети требуется уже не одна формула, а некоторая система формул.

Например, изображенной на рис. 4.3.6, *а* сходящейся комбинационной сети соответствует формула

$$y = ((b \wedge c) \wedge a) \vee (a \wedge \bar{d}).$$

Адекватное формульное описание расходящейся комбинационной сети может быть получено путем введения дополнительных переменных для обозначения тех связей,

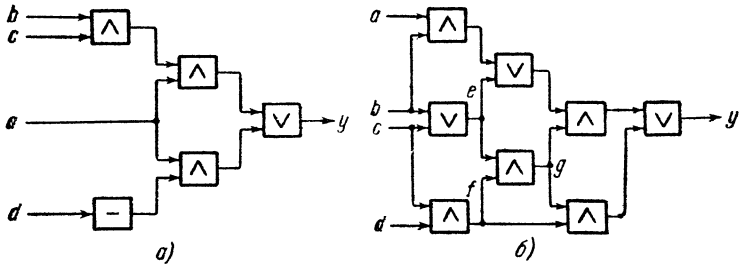


Рис. 4.3.6.

удаление которых превращает сеть в сходящуюся. В этом случае мы получаем некоторую систему формул. Например, сеть, представленная на рис. 4.3.6, *б*, может быть описана следующей системой формул:

$$e = b \vee c,$$
$$f = c \wedge d,$$
$$g = e \wedge f,$$
$$y = (((a \wedge b) \vee e) \wedge g) \vee (g \wedge f).$$

Очевидно, что система формул потребуется и для описания комбинационной сети с более чем одним выходным полюсом.

Нетрудно прийти к выводу, что исследование комбинационных автоматов сводится в основном к исследованию отношений между булевыми функциями и их представлениями в виде суперпозиций некоторых элементарных функций, реализуемых отдельными элементами. При анализе по заданной суперпозиции находится

реализуемая ею булева функция (или система булевых функций) и представляется в некоторой более удобной форме. При синтезе решается обратная задача: заданная булева функция (или система) представляется в форме некоторой суперпозиции, задающей структуру комбинационного автомата.

Например, анализируя сеть, представленную на рис. 4.3.6, *б*, нетрудно выяснить, что она реализует весьма простую булеву функцию $y=cd$, которую, очевидно, можно реализовать одним элементом. Задача синтеза, как всегда, решается труднее, в связи с чем мы посвятим ей специальный параграф.

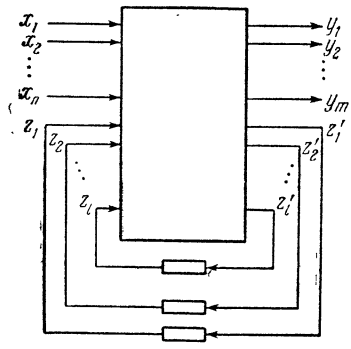


Рис. 4.3.7.

Заметим, что все булевы функции, реализуемые рассмотренными в этом параграфе элементами, являются *симметрическими*, то есть не зависят от перестановок значений аргументов, а зависят лишь от того, сколько аргументов принимает значение 1. Для таких элементов изображение сети упрощается: каждая связь может быть задана как упорядоченная пара связываемых элементов, без указания номера входного полюса.

Синхронные сети. Перейдем к рассмотрению сетей, содержащих логические элементы и элементы задержки. Допустим, что в этих сетях могут встречаться контуры, но только такие, в которых будет по крайней мере по одному элементу задержки. Назовем такие сети *синхронными* и покажем, что их можно рассматривать как структуры синхронных автоматов.

Действительно, какова бы ни была сеть описанного типа, ее можно представить так, как это показано на рис. 4.3.7, где элементы задержки вынесены в отдельную группу, а остальная часть сети является, очевидно, комбинационной (поскольку в ней не остается ни контуров, ни элементов задержки). Те ее полюсы, которые соединены с входными и выходными полюсами элементов за-

держки, мы обозначили через $z'_1, z'_2, \ldots, z'_l$ и $z_1, z_2, \ldots, z_l$, соответственно. Через $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_m$ обозначены, как и ранее, входные и выходные полюсы сети в целом.

Известно, что комбинационная сеть реализует некоторую систему булевых функций. Представим ее в векторной форме:

$$y = \varphi(x, z),$$
$$z' = \psi(x, z).$$

Так как по определению элемента задержки $z'_i(t) = z_i(t+1)$ и $z'(t) = z(t+1)$, то это значит, что сеть в целом реализует следующее функциональное отношение между входом и выходом:

$$y(t) = \varphi(x(t), z(t)),$$
$$z(t+1) = \psi(x(t), z(t)).$$

Здесь мы перешли к рассмотрению поведения сети в дискретном времени, интересуясь состоянием ее полюсов лишь в моменты времени $t = 1, 2, 3, \ldots$ Становится очевидным, что эту сеть можно рассматривать как синхронный автомат, в котором входные и выходные состояния заданы в структурном алфавите, значениями булевых векторов x и y, а не в абстрактном алфавите, когда каждое состояние представляется некоторым одним символом. Аналогично задана и структура внутренних состояний: значениями булева вектора z.

В некоторых случаях оказывается полезным использование следующей модификации понятия синхронной сети. Считается, что эффект задержки присущ самим логическим элементам, в связи с чем отпадает надобность во введении отдельно элементов задержки. Синхронная сеть в таких случаях полагается состоящей только из логических элементов, причем наличие контуров в такой сети не может привести к каким-либо противоречиям при определении ее функциональных свойств.

Асинхронные сети. Анализируя поведение какой-либо сети, составленной из логических элементов (не обладающих эффектом задержки) и содержащей, в отличие от комбинационных сетей, контуры, мы можем

прийти к одному из следующих выводов. Или при любой комбинации значений входных переменных сеть будет переходить в некоторое устойчивое состояние и оставаться в нем, пока данная комбинация не изменится, или же это условие при некоторых обстоятельствах может нарушаться (как это было в случае сети, изображенной на рис. 4.3.5, *а*).

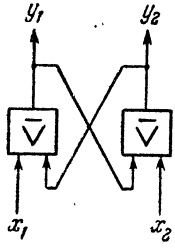


Рис. 4.3.8.

Сеть, удовлетворяющую сформулированному условию перехода в устойчивое состояние, назовем *асинхронной*. Ее можно рассматривать как структуру асинхронного автомата. Простейший пример такой сети представлен на рис. 4.3.8. Нетрудно убедиться, что эта сеть оказывается триггером.

Рассматривая триггеры как элементы, мы ограничимся в дальнейшем исследованием таких асинхронных сетей, каждая из которых состоит из одной комбинационной сети и некоторой совокупности триггеров, концентрирующих в себе «память» автомата (рис. 4.3.9). Входные и выходные состояния автомата представляются, как обычно, значениями булевых векторов $\boldsymbol{x}$ и $\boldsymbol{y}$, а внутренние состояния — значениями вектора $\boldsymbol{q}_п = (q_{1п}, q_{2п}, \ldots, q_{lп})$, где через $q_{iп}$ обозначен правый выходной полюс i-го триггера. Напомним, что мы считаем, что триггер находится в состоянии 1, если $q_п = 1$. Переменная $q_л$ принимает всегда значение, инверсное значению переменной $q_п$, в связи с чем можно считать, что булевы функции, реализуемые комбинационной сетью, зависят только от переменных $x_1, x_2, \ldots, x_n, q_{1п}, q_{2п}, \ldots, q_{lп}$.

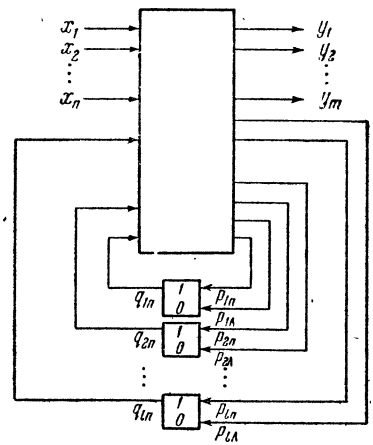


Рис. 4.3.9.

Эти функции делятся на два типа: во-первых, это выходные функции

$$y_i = \varphi_i(x_1, x_2, \ldots, x_n, q_{1\text{п}}, q_{2\text{п}}, \ldots, q_{l\text{п}}),$$
$$i = 1, 2, \ldots, m,$$

во-вторых, это *функции возбуждения* триггеров

$$p_{i\text{л}} = \psi'(x_1, x_2, \ldots, x_n, q_{1\text{п}}, q_{2\text{п}}, \ldots, q_{l\text{п}}),$$
$$i = 1, 2, \ldots, l$$

(функции левого входа), и

$$p_{i\text{п}} = \psi''(x_1, x_2, \ldots, x_n, q_{1\text{п}}, q_{2\text{п}}, \ldots, q_{l\text{п}}),$$
$$i = 1, 2, \ldots, l$$

(функции правого входа). В векторной форме данная система принимает следующий компактный вид:

$$\boldsymbol{y} = \varphi(\boldsymbol{x}, \boldsymbol{q}_{\text{п}}),$$
$$\boldsymbol{p}_{\text{л}} = \psi'(\boldsymbol{x}, \boldsymbol{q}_{\text{п}}),$$
$$\boldsymbol{p}_{\text{п}} = \psi''(\boldsymbol{x}, \boldsymbol{q}_{\text{п}}).$$

При решении задачи синтеза асинхронного автомата в рамках данной модели важное место занимает нахождение функций возбуждения триггеров. Определение этих функций допускает некоторую произвольность, обусловленную следующими соображениями.

Если при реализации некоторого перехода между внутренними состояниями i-й триггер должен сменить состояние 0 на состояние 1 (представим символически изменение такого типа как $0 \to 1$), то на его вход должна быть подана вполне определенная комбинация 01 значений переменных $p_{i\text{л}}$ и $p_{i\text{п}}$. Но если при этом переходе данный триггер должен остаться в состоянии 0 (то есть если его состояние должно измениться по формуле $0 \to 0$), то допустима как входная комбинация 00, не изменяющая состояния триггера, так и комбинация 10, всегда приводящая его в состояние 0, в каком бы состоянии он ни находился до этого. Совокупность допустимых в данном случае входных комбинаций может быть представлена троичным вектором — 0.

Таблица 4.3.1

Тип изменения состояния триггера	Требуемая входная комбинация
$0 \to 0$	— 0
$0 \to 1$	0 1
$1 \to 0$	1 0
$1 \to 1$	0 —

Аналогичный анализ других типов изменения состояния триггера приводит нас к получению таблицы 4.3.1.

§ 4. Кодирование внутренних состояний

Кодирование состояний синхронного автомата. Пусть входные и выходные состояния автомата заданы в структурном алфавите, внутренние — в абстрактном. Поскольку эти состояния должны быть представлены состояниями элементов памяти, требуется выбрать способ представления, или, как говорят, код внутренних состояний. В принципе можно сделать это любым образом, лишь бы различные состояния оказались представленными по-разному.

Однако различные способы кодирования состояний оказываются далеко не равноценными с точки зрения простоты структуры автомата, а также с точки зрения некоторых других критериев, таких, например, как быстродействие, надежность и т. д.

Наиболее существенным из этих критериев для синхронного автомата оказывается простота структуры автомата. В качестве промежуточной цели можно наметить простоту системы булевых функций, реализуемых комбинационной частью автомата. Действительно, выбирая различные способы кодирования состояний, мы получим различные системы булевых функций. Можно минимизировать эти системы в классе дизъюнктивных нормальных форм, а затем подсчитывать, например, число букв в полученных выражениях. Есть основания надеяться, что выбирая такой способ кодирования, который приводит к выражениям с минимальным числом букв, мы получим в конечном счете более простую структуру автомата.

Состязания между элементами памяти асинхронного автомата. Так называется явление, обусловленное инерционностью и нестандартностью элементов памяти. Дело в том, что каждый из реальных элементов памяти инерционен, причем время, затрачиваемое на изменение его состояния, не является постоянным. Оно зависит от ряда обстоятельств, не учитываемых в абстрактной модели автомата. Вследствие этого при переходе автомата из одного внутреннего состояния в другое может реализоваться некоторая последовательность *элементарных переходов* (соответствующих изменениям состояний отдельных элементов памяти), при которой автомат проходит через некоторое множество промежуточных состояний и которая оказывается в общем случае непредсказуемой. Например, при реализации перехода $010001 \rightarrow 110101$ должны изменить свои состояния элементы памяти с номерами 1 и 4. Если элемент 1 «сработает» раньше, чем элемент 4, то автомат перейдет в промежуточное состояние 110001, если же более быстрым окажется элемент 4, то автомат перейдет в состояние 010101. Очевидно, что последующие действия автомата будут определяться уже значениями функции переходов на достигнутых промежуточных состояниях.

Таким образом, дальнейшее поведение автомата может оказаться в зависимости от того, какой из элементов памяти быстрее среагирует на прикладываемое к нему воздействие. Элементы как бы состязаются в быстроте реакции, чем и обусловлено название рассматриваемого явления. Если в конце концов автомат закономерно приходит в намеченное матрицей переходов состояние, то состязания можно считать *неопасными*, в противном случае их следует рассматривать как *опасные*.

Очевидно, что для того, чтобы поведение асинхронного автомата не отклонялось от заданного матрицей переходов, требуется каким-то образом устранить все опасные состояния между элементами памяти. Один из возможных способов решения этой задачи заключается в следующем.

Реализацию заданного в автомате перехода $i \rightarrow j$ можно обеспечить, придав функции переходов значение

j на всех возможных промежуточных состояниях, в которые автомат может попасть из состояния i (при заданном фиксированном входном состоянии). В этом случае элементы памяти, которые должны изменить свои состояния, будут подвергаться надлежащему постоянному воздействию на всем протяжении рассматриваемого перехода, независимо от того, в каком порядке они «сработают». Благодаря этому состязания между элементами памяти становятся неопасными и автомат неизбежно достигнет состояния j, причем весь переход $i \to j$, называемый в данном случае *прямым*, будет реализован с максимальным быстродействием.

Критерий отсутствия опасных состязаний. Одним из эффективных средств предотвращения опасных состязаний между элементами памяти является рациональный выбор способа кодирования внутренних состояний. Предлагаемые в этом направлении методы можно разбить на два основных класса. В одном из них ищутся такие способы кодирования, при которых все заданные переходы становятся элементарными, и, следовательно, состязания вообще устраняются. Если же оказывается, что для исходной матрицы переходов сделать это невозможно, то матрица надлежащим образом преобразуется, причем множество рассматриваемых внутренних состояний, как правило, расширяется. К другому классу относятся методы, устраняющие лишь опасные состязания и не связанные с расширением множества внутренних состояний путем добавления соответствующих строк в исходную матрицу переходов. Эти методы основаны на использовании прямых переходов. Именно такого типа метод и будет рассмотрен ниже.

Прежде всего, сформулируем условия отсутствия опасных состязаний между элементами памяти асинхронных автоматов, заданных матрицами переходов.

Обозначим через $U(i, j)$ множество всех возможных промежуточных состояний (включая исходное и конечное), в которые автомат может попасть при реализации перехода $i \to j$, если элементы памяти, которые должны изменить свои состояния, делают это в произвольном порядке. По определению функция переходов является однозначной функцией полного состояния, а при фиксированном входном состоянии — однозначной функцией

внутреннего состояния. Отсюда следует, что описанный выше способ организации прямых переходов может быть реализован в том и только в том случае, когда выполняется следующее условие:

> для каждой пары переходов $i \to j$ и $k \to l$, соответствующих одному и тому же входному состоянию, множества $U(i, j)$ и $U(k, l)$ не должны пересекаться (то есть должно выполняться $U(i, j) \cap U(k, l) = \varnothing$), если $j \neq l$.

Множество $U(i, j)$ можно найти, зная коды состояний i и j, обозначаемые через $z(i)$ и $z(j)$, соответственно. Нетрудно сообразить, что это множество будет образовано из всех таких состояний, коды которых совпадают с кодами $z(i)$ и $z(j)$ в тех компонентах, в которых совпадают между собой коды $z(i)$ и $z(j)$. Данное множество может быть представлено формально посредством троичного вектора $t(i, j)$, компоненты которого принимают значения одноименных компонент векторов $z(i)$ и $z(j)$, если те совпадают между собой, и принимают значение «—» в противном случае. Коды всех элементов представленного таким образом множества состояний могут быть получены путем произвольной подстановки единиц и нулей на места, занимаемые символами «—».

Например, если $z(i) = 100111$ и $z(j) = 101011$, то $t(i, j) = 10--11$, и это означает, что множество $U(i, j)$ возможных промежуточных состояний, соответствующих переходу $i \to j$, состоит из четырех элементов, коды которых суть 100011, 100111, 101011 и 101111.

Заметим, что множество, составленное из 2^n булевых векторов — всевозможных n-компонентных кодов внутренних состояний, может рассматриваться как булево пространство над множеством внутренних переменных $z_1, z_2, \ldots, z_n$. В этом случае множество $U(i, j)$, представленное значением вектора $t(i, j)$, является интервалом данного пространства.

Необходимым и достаточным условием непересечения множеств $U(i, j)$ и $U(k, l)$ является наличие такой одноименной компоненты в кодах $t(i, j)$, и $t(k, l)$, которая принимает значение 1 в одном из этих кодов и значение 0 в другом из них. Доказательство этого утверж-

дения тривиально: если данное условие выполняется, то любой элемент множества $U(i, j)$ будет отличен от любого элемента множества $U(k, l)$, в противном случае в этих множествах нетрудно найти общий элемент.

Кодирующая матрица и требования к ней. Назовем *кодирующей матрицей* такую булеву матрицу $\boldsymbol{Q}$, столбцами которой служат коды внутренних состояний автомата, а строки поставлены во взаимно однозначное соответствие с внутренними переменными.

Например, если

$$\boldsymbol{Q} = \begin{matrix} 1\,2\,3\,4\,5\,6 \\ \left[\begin{matrix} 1\,1\,0\,0\,0\,0 \\ 1\,0\,1\,0\,0\,1 \\ 1\,0\,0\,0\,1\,0 \\ 0\,1\,0\,0\,0\,1 \end{matrix}\right] \end{matrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

то это значит, что кодом состояния 1 служит булев вектор 1 1 1 0, то есть если автомат находится в состоянии 1, то внутренние переменные z_1, z_2, z_3 и z_4 принимают значения 1, 1, 1 и 0, соответственно, кодом состояния 2 является 1 0 0 1, и т. д.

Получение подобной матрицы для автомата, заданного матрицей переходов, будем называть *кодированием внутренних состояний.*

Сформулированное выше достаточное и необходимое условие, позволяющее устранить опасные состязания между элементами памяти асинхронного автомата, может быть перефразировано следующим образом:

> для каждой пары переходов $i \to j$ и $k \to l$, соответствующих одному и тому же входному состоянию и таких, что $j \neq l$, в кодирующей матрице $\boldsymbol{Q}$ должна найтись строка, в которой i-я и j-я компоненты будут иметь одно из значений 1 или 0, а k-я и l-я компоненты — другое.

Матричная формулировка задачи. Условие, предъявляемое к матрице $\boldsymbol{Q}$ при рассмотрении некоторой конкретной пары переходов $i \to j$ и $k \to l$, соответствующих одному и тому же входному состоянию и таких, что $j \neq l$, назовем *элементарным.* Оно может быть представлено формально троичным вектором, у кото-

рого i-я и j-я компоненты имеют одно из значений 1 или 0, k-я и l-я компоненты имеют другое из этих значений, а все остальные компоненты имеют значение «—».

Совокупность получаемых таким образом троичных векторов образует троичную матрицу S, которую мы назовем *матрицей условий*.

Задача нахождения матрицы Q, удовлетворяющей совокупности условий, задаваемых матрицей S, становится в таком виде уже знакомой нам задачей нахождения кратчайшей имплицирующей формы для S.

Действительно, можно считать, что каждая из строк матрицы S задает некоторое частичное двублочное разбиение на множестве столбцов матрицы Q, выделяя из этого множества два блока: к одному из них относятся столбцы, отмеченные единицами (в соответствующих компонентах рассматриваемой строки матрицы условий), к другому относятся столбцы, отмеченные нулями. Значение «—» здесь, как и раньше, является символом неопределенности: считается, что отмеченные этим символом столбцы могут принадлежать к любому из блоков заданного разбиения.

Продолжая эти рассуждения, мы убеждаемся, что решаемая нами задача сводится к нахождению такой кодирующей матрицы Q, которая имплицировала бы матрицу условий S. Именно в это требование трансформируется критерий отсутствия опасных состязаний между элементами памяти автомата. Дополнительным требованием, значительно усложняющим процесс решения, является условие получения такой из матриц, имплицирующих матрицу S, которая содержала бы минимально возможное число строк, соответствуя таким образом минимально возможному числу элементов памяти.

Получение и упрощение матрицы условий. Будем считать, что, по тем или иным соображениям, все состояния автомата целесообразно рассматривать как существенно различные, в связи с чем их надо кодировать различными кодами. В частности, такие соображения могут быть обоснованы предварительной минимизацией числа состояний автомата, после которой уже нет оснований беспокоиться о совмещении еще каких-либо состояний.

Данное требование легко учесть, добавив в матрицу переходов столбец устойчивых состояний, элементы которого совпадают своими значениями с номерами строк, в которых они расположены.

Например, матрица переходов частичного асинхронного автомата

$$\Psi = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \left[\begin{array}{cccc|c} 1 & 1 & 1 & 5 & 1 \\ 1 & 2 & 2 & 2 & 2 \\ 3 & 2 & 4 & 5 & 3 \\ 3 & 2 & 4 & 5 & 4 \\ -5 & 4 & 5 & & 5 \end{array}\right] \end{array}$$

принимает следующий вид:

$$\Psi = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \left[\begin{array}{ccccc|c} 1 & 1 & 1 & 5 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 \\ 3 & 2 & 4 & 5 & 3 & 3 \\ 3 & 2 & 4 & 5 & 4 & 4 \\ -5 & 4 & 5 & 5 & & 5 \end{array}\right] \end{array}$$

Анализируя эту матрицу, нетрудно найти, что в матрицу условий следует включить, в частности, следующие строки, соответствующие входному состоянию 1 (справа показаны рассматриваемые при этом пары переходов):

1 2 3 4 5	
1 — 0 — —	$1 \to 1,\ 3 \to 3$
1 1 0 — —	$2 \to 1,\ 3 \to 3$
1 — 0 0 —	$1 \to 1,\ 4 \to 3$
1 1 0 0 —	$2 \to 1,\ 4 \to 3$

Среди этих строк содержится некоторая избыточность, обязанная своим происхождением транзитивности отношения импликации. Действительно, если некоторые строки s_i и s_j матрицы S находятся в отношении $s_i \Rightarrow s_j$, то удаление из матрицы S строки s_j не приведет к какому-либо изменению в постановке решаемой задачи: если будет найдена матрица Q, имплицирующая строку s_i, то тем самым будет имплицирована и строка s_j. В данном случае строка 1 1 0 0 имплицирует каждую из трех остальных, которые, следовательно, можно выбросить из матрицы.

Рассматривая совокупность всех элементарных условий, порождаемых матрицей переходов, и выбрасывая те из них, которые имплицируются остающимися, мы получим матрицу R, называемую *минимальной матрицей условий*. В данном случае она будет иметь следующий вид:

$$R = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \left[\begin{array}{ccccc|cc} 1 & 1 & 0 & 0 & - & 1 & 1 \\ 1 & 0 & 0 & - & - & 2 & \\ 1 & 0 & - & 0 & - & 3 & 2 \\ - & 1 & 1 & - & 0 & 4 & \\ - & 1 & - & 1 & 0 & 5 & \\ 1 & - & - & 0 & 0 & 6 & \\ - & 1 & - & 0 & 0 & 7 & 3 \\ 1 & 0 & - & - & 1 & 8 & \\ - & 1 & 0 & - & 0 & 9 & 4 \\ - & - & 1 & 0 & - & 10 & 5 \end{array}\right] \end{array}$$

Для удобства обозрения матрица разбита на пять секций, соответствующих различным столбцам матрицы переходов.

Из способа построения матрицы R следует ее эквивалентность матрице S ($R \Leftrightarrow S$, то есть $R \Rightarrow S$ и $S \Rightarrow R$). Это означает, что задача поиска матрицы Q, имплицирующей матрицу S и составленной из минимального числа строк, эквивалентна аналогичной задаче, сформулированной для матрицы R, которая содержит лишь некоторые из элементарных условий, представляемые теперь троичными векторами r_i.

Нахождение кодирующей матрицы. Для нахождения минимальной кодирующей матрицы может быть использован описанный в параграфе 2.9 алгоритм получения точного решения или алгоритм матрамин — для получения некоторого приближения. Для рассматриваемого примера применение первого из этих алгоритмов приводит нас к матрице

$$Q = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}$$

Приближенный алгоритм матрамин дает в данном

случае не худшее решение:

$$Q = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \left[\begin{matrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ - & - & 1 & 0 & - \end{matrix}\right] \end{array}$$

Три элемента кодирующей матрицы оказываются неопределенными, что означает, что любое доопределение их значений будет удовлетворять условиям задачи.

Рассмотрение K-множеств. Время, затрачиваемое на поиск минимальной кодирующей матрицы, сильно зависит от размеров матрицы условий. В связи с этим представляет интерес метод образования матрицы условий, основанный не на рассмотрении переходов, а на рассмотрении *K-множеств*. Так называются совокупности внутренних состояний, переходы из которых при некотором фиксированном входном состоянии ведут в одно и то же состояние (также принадлежащее данной совокупности). В остальном процесс построения матрицы условий остается без изменений, только вместо матрицы S мы получаем некоторую матрицу S^*, а затем, выбрасывая из нее строки, имплицируемые остающимися, приходим к матрице R^*, эквивалентной матрице S^*.

Каждой из строк матрицы R, полученной при рассмотрении некоторой пары переходов $i \to j$ и $k \to l$, будет соответствовать некоторая строка матрицы R^*, полученная при рассмотрении двух K-множеств, одному из которых принадлежат, в частности, состояния i и j, а другому — состояния k и l, причем строка матрицы R будет имплицироваться соответствующей строкой матрицы R^*. Отсюда следует, что $R^* \Rightarrow R$, а значит, кодирующая матрица Q, которая имплицирует матрицу R^*, будет имплицировать и матрицу R.

Однако матрица R^* в общем случае неэквивалентна матрице R ($R^* \Rightarrow R$, но нельзя утверждать, что $R \Rightarrow R^*$). Поэтому решение, оптимальное для матрицы R^*, может оказаться неоптимальным для матрицы R. Выражаясь точнее, минимально необходимое число строк в матрице Q, имплицирующей матрицу R^*, может оказаться больше, чем это будет необходимо при рассмотрении матрицы R.

Вернемся к рассмотрению все того же автомата, скорректированная матрица переходов которого

$$\Psi = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{5} & \mathbf{1} \\ 1 & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} \\ \mathbf{3} & 2 & 4 & 5 & \mathbf{3} \\ 3 & 2 & \mathbf{4} & 5 & \mathbf{4} \\ \mathbf{-} & \mathbf{5} & 4 & \mathbf{5} & \mathbf{5} \end{array}\right] \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}$$

Совокупность переходов, представленных в столбце 1, разбивает множество внутренних состояний на K-множества $\{1, 2\}$ и $\{3, 4\}$, в столбце 2 — на K-множества $\{1\}$, $\{2, 3, 4\}$ и $\{5\}$, в столбце 3 — на K-множества $\{1\}$, $\{2\}$ и $\{3, 4, 5\}$, в столбце 4 — на K-множества $\{1, 3, 4, 5\}$ и $\{2\}$, наконец, в столбце 5 — на K-множества $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$ и $\{5\}$.

Такому разбиению будет соответствовать следующая матрица R^*:

$$R^* = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & - \\ 1 & 0 & 0 & 0 & - \\ - & 1 & 1 & 1 & 0 \\ 1 & - & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ - & - & 1 & 0 & - \end{array}\right] \end{array}$$

Находя для нее минимальную имплицирующую матрицу, мы получим:

$$Q = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 0 \\ - & - & 1 & 0 & - \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Как видно, это решение несколько хуже полученного ранее, что иллюстрирует неоптимальность процесса решения, основанного на рассмотрении K-множеств.

О неоптимальности метода прямых переходов. Мы рассмотрели метод такого кодирования внутренних состояний асинхронного автомата, при котором устраняются лишь опасные состязания между элементами памяти, но могут оставаться так называемые неопасные состязания. Этот метод достаточно практичен, хорошо согласуясь с требованиями, предъявляемыми к синтезируемому автомату: простоте его струк-

туры и быстродействию. Однако он не гарантирует получение автомата с минимальным числом элементов памяти.

Рассмотрим конкретный пример асинхронного автомата, заданного следующей матрицей переходов:

$$\Psi = \begin{array}{c} \begin{matrix} a & b & c & d & e \end{matrix} \\ \left[\begin{array}{ccccc} 1 & 5 & 1 & 5 & 2 \\ 2 & 6 & 2 & - & 2 \\ 1 & - & 4 & 3 & - \\ 2 & 5 & 4 & - & 6 \\ 2 & 5 & 1 & 5 & - \\ 7 & 6 & 2 & 3 & 6 \\ 7 & 5 & - & 3 & 6 \end{array}\right] \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \end{array}$$

Матрицу выходов мы рассматривать не будем, допустим лишь, что все семь внутренних состояний автомата, представленные в матрице переходов, несовместимы.

Реализуя рассмотренный выше метод прямых переходов, мы получим сначала минимальную матрицу условий

$$R = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 0 & 1 & 0 & - & - & - \\ 1 & 0 & 1 & - & 0 & - & - \\ 1 & - & 1 & - & - & 0 & 0 \\ 1 & 1 & - & 0 & - & 0 & - \\ 1 & 1 & - & - & - & 0 & 0 \\ 1 & 0 & - & - & 1 & 0 & - \\ 1 & - & 0 & 0 & 1 & - & - \\ 1 & - & 0 & - & 1 & 0 & - \\ 1 & - & 0 & - & 1 & - & 0 \\ - & 1 & 0 & 0 & - & 1 & - \\ - & 1 & - & 0 & 0 & 1 & - \\ - & 1 & - & - & 0 & 1 & 0 \\ - & 1 & - & 1 & - & 0 & 0 \\ - & 1 & - & - & 1 & 0 & 0 \end{array}\right] \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \end{matrix} \end{array}$$

Чтобы найти минимально возможное при прямых переходах число элементов памяти, следует перейти к рассмотрению множества максимальных совместимых совокупностей строк матрицы R и представить это множество булевой матрицей T, а затем найти одно из ее кратчайших покрытий.

В данном случае матрица T будет такова:

$$T = \begin{array}{c} \begin{array}{cccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \end{array} \\ \left[\begin{array}{cccccccccccccc} 1 & 1 & 1 & & & & & & & & & & & \\ 1 & 1 & & & & & & & & & & 1 & 1 & \\ 1 & & 1 & & 1 & & & & & & & & & \\ 1 & & & & 1 & & & & & & 1 & & & \\ & 1 & 1 & & & & & 1 & & & & & & \\ & & 1 & 1 & 1 & & & & & & & & 1 & \\ & & 1 & & 1 & & & & & & & 1 & 1 & \\ & & 1 & & & & & & 1 & 1 & & & & \\ & & & 1 & 1 & & 1 & 1 & 1 & & & 1 & & \\ & & & & 1 & & & & & & & 1 & & \\ & & & & & 1 & 1 & 1 & & & & & & \\ & & & & & 1 & & & & & 1 & & & \\ & & & & & 1 & & & & & & 1 & & \\ & & & & & & 1 & & & 1 & 1 & & & \\ & & & & & & 1 & & & & & 1 & 1 & \\ & & & & & & & 1 & & & & 1 & 1 & \end{array}\right] \end{array} \begin{array}{l} \alpha \\ \beta \\ \\ \\ \\ \\ \\ \gamma \\ \\ \\ \delta \\ \\ \\ \\ \\ \\ \end{array}$$

Чтобы не загромождать матрицу, мы показали только единичные значения ее элементов.

Кратчайшее покрытие матрицы можно найти, например, с помощью Л-программы окрапок 4. Одно из кратчайших покрытий представляется четырьмя строками, отмеченными буквами. Это значит, что для кодирования внутренних состояний по методу прямых переходов требуется как минимум четыре внутренние переменные, то есть в структуру автомата требуется ввести по крайней мере четыре элемента памяти.

Кодирующая матрица легко получается путем совмещения соответствующих строк матрицы условий R. Например, строка δ матрицы T соответствует совокупности строк 6, 10, 11 и 12 матрицы условий R. Совмещая эти строки (вспомним о возможности инверсирования при этом некоторых из них), мы получим строку 1011101, которая будет представлять собой состояния одного из элементов памяти, соответствующие внутренним состояниям 1, 2, 3, 4, 5, 6 и 7 рассматриваемого автомата. Аналогично определяются состояния остальных трех элементов памяти. Заметим, что при определении состояний элемента памяти, соответствующего строке α матрицы T, можно считать, что они полностью задаются одной лишь строкой 3 матрицы условий,

поскольку условия, задаваемые строками 1 и 2, будут удовлетворены элементом памяти, соответствующим строке β матрицы T: эта строка также содержит единицы в столбцах 1 и 2.

Получаемая таким образом кодирующая матрица Q приобретает следующее значение:

$$Q = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \left[\begin{matrix} 1 & - & 1 & - & - & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{matrix}\right] \end{array}$$

Соседнее кодирование состояний. Между тем, минимального числа элементов памяти можно достигнуть, отказавшись от рассмотрения только прямых переходов, заданных исходной матрицей переходов асинхронного автомата. Каждый из них можно заменить некоторой последовательностью элементарных переходов, то есть таких, каждый из которых изменяет состояние лишь одного элемента памяти. Естественно, что при таких переходах состязания между элементами памяти возникнуть не могут.

Например, состояния уже знакомого нам автомата могут быть закодированы следующей матрицей:

$$Q = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \left[\begin{matrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{matrix}\right] \end{array}$$

(к семи состояниям, отраженным в матрице переходов, мы добавили восьмое), а сама матрица переходов преобразуется при этом следующим образом:

$$\Psi = \begin{array}{c} \begin{matrix} a & b & c & d & e \end{matrix} \\ \left[\begin{matrix} \mathbf{1} & 5 & \mathbf{1} & 5 & 8 \\ \mathbf{2} & 6 & \mathbf{2} & - & \mathbf{2} \\ 1 & 7 & 4 & \mathbf{3} & - \\ 8 & 3 & \mathbf{4} & - & 6 \\ 2 & \mathbf{5} & 1 & \mathbf{5} & - \\ 7 & \mathbf{6} & 2 & 7 & \mathbf{6} \\ 7 & 5 & - & 3 & 6 \\ 2 & - & - & - & 2 \end{matrix}\right] \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} \end{array}$$

Изменению подверглись элементы ψ_1^e, ψ_3^b, ψ_4^a, ψ_4^b и ψ_6^d матрицы переходов; эти изменения отражают замену соответствующих прямых переходов на эквивалентные им серии элементарных переходов между соседними состояниями. Добавлена также дополнительная строка, соответствующая восьмому состоянию.

Преобразованную таким образом матрицу переходов Ψ будем называть *матрицей элементарных переходов*, а сам процесс получения этой матрицы совместно с матрицей кодирования Q, обеспечивающей соседство соответствующих состояний, — *соседним кодированием* внутренних состояний.

В рассмотренном примере соседнее кодирование позволило обойтись всего тремя элементами памяти и вместе с тем полностью устранить все состязания между ними.

Графические представления. Булево пространство n-компонентных булевых векторов можно представить симметрическим графом, вершины которого будут соответствовать элементам булева пространства, а ребра — связывать вершины, соответствующие соседним элементам и также называющиеся соседними. Назовем такой граф *полным булевым графом*.

Например, при $n=3$ полный булев граф будет иметь вид, показанный на рис. 4.4.1.

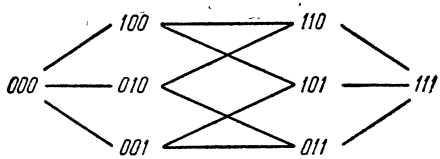


Рис. 4.4.1.

Кодирование внутренних состояний автомата можно представить как процесс размещения номеров этих состояний на вершинах полного булева графа. В дальнейшем будем условно говорить о *размещении состояний* на булевом графе. Каким же условиям оно должно удовлетворять?

Очевидно, что пары состояний, связанных элементарными переходами, должны разместиться на соседних

вершинах. Однако это условие мало что говорит о том, как же размещать исходные состояния, так как неизвестно, какие из прямых переходов следует разбить на серии элементарных.

Бо́льшую ясность вносит следующее условие. Внутренние состояния должны быть размещены на булевом графе так, чтобы задаваемое каждым входным состоянием разбиение множества C на K-множества могло быть отражено некоторым разбиением полного булева графа на связные подграфы, содержащие вершины соответствующих им K-множеств и не содержащие общих вершин.

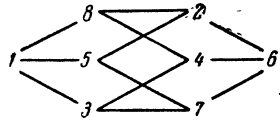


Рис. 4.4.2.

Например, рассмотренному выше решению соответствует следующее размещение состояний (рис. 4.4.2). Соответствующие различным входным состояниям разбиения этого графа на связные подграфы показаны на

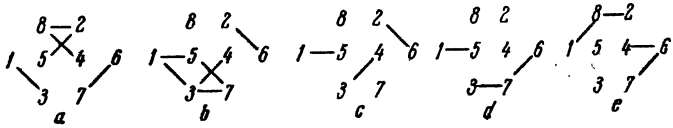


Рис. 4.4.3.

рис. 4.4.3. Они получаются путем выбрасывания некоторых ребер из полного булева графа.

В каждом K-множестве устойчивым является лишь одно состояние, и именно к нему должны идти переходы от всех других состояний, принадлежащих этому же K-множеству. Совокупность этих переходов нетрудно отобразить, удалив в соответствующем подграфе некоторые ребра и заменив оставшиеся дугами так, чтобы в результате был получен ориентированный граф, в котором из любой вершины существовал бы путь в вершину, соответствующую упомянутому устойчивому состоянию. Например, описанное преобразование на рис. 4.4.3 приводит к рис. 4.4.4.

Поиск решения. Очевидно, что для кодирования l состояний требуется не менее $\log_2 l$ двоичных переменных. Поэтому поиск решения естественно начать с попыток размещения состояний на полном булевом

графе с 2^S вершинами, где S — ближнее сверху целое число для $\log_2 l$.

Рассмотрим метод такого размещения, опирающийся на анализ соотношений между K-множествами, иллюстрируя его на примере того же автомата, заданного первоначально матрицей переходов

$$\Psi = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \left[\begin{array}{ccccc|c} 1 & 5 & 1 & 5 & 2 & 1 \\ 2 & 6 & 2 & - & 2 & 2 \\ 1 & - & 4 & 3 & - & 3 \\ 2 & 5 & 4 & - & 6 & 4 \\ 2 & 5 & 1 & 5 & - & 5 \\ 7 & 6 & 2 & 3 & 6 & 6 \\ 7 & 5 & - & 3 & 6 & 7 \end{array}\right] \end{array}$$

Итак, попробуем разместить состояния этого автомата на трехмерном полном булевом графе, изображенном на рис. 4.4.1. Конечно, при этом мы не должны учитывать уже известное нам решение.

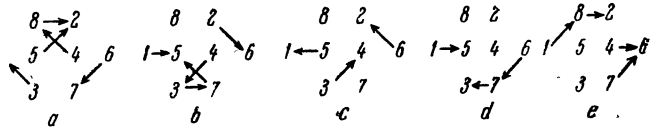


Рис. 4.4.4.

Каждое входное состояние автомата разбивает множество C внутренних состояний на K-множества, совокупность которых удобно представить следующей секционированной булевой матрицей:

$$\begin{array}{c} 1\,2\,3\,4\,5\,6\,7 \\ \left[\begin{array}{c|c} 1010000 & \\ 0101100 & a \\ \underline{0000011} & \\ 1001101 & \\ 0100010 & b \\ \underline{1000100} & \\ 0100010 & c \\ \underline{0011000} & \\ 0010011 & d \\ \underline{1000100} & \\ 1100000 & e \\ 0001011 & \end{array}\right] \end{array}$$

Для каждой пары состояний подсчитаем *коэффициент связи*, выражаемый числом K-множеств, в которые входит данная пара. Например, пара состояний $\{1, 2\}$ входит лишь в одно K-множество, соответствующее входному состоянию e, следовательно, коэффициент связи этой пары равен 1.

Полученные результаты выразим следующей *матрицей связи*:

2	3	4	5	6	7	
1	1	1	3	0	1	1
	0	1	1	2	0	2
		1	0	1	1	3
			2	1	2	4
				0	1	5
					3	6

Из общих соображений следует, что состояния, связанные сильнее, целесообразно размещать поближе друг к другу (расстояние между вершинами на графе измеряется длиной кратчайшей цепи, связывающей их). Естественно начать с рассмотрения состояний, обладающих максимальной суммой коэффициентов связи, и в соседстве с ними разместить именно те состояния, с которыми они связаны сильнее всего.

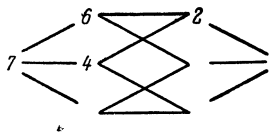


Рис. 4.4.5.

Поэтому мы выбираем сначала состояние 7 и размещаем его в вершине 000 (очевидно, что выбор вершины в начале процесса размещения может делаться совершенно произвольно), а в соседних ей вершинах 100 и 010 размещаем состояния 6 и 4, соответственно. К состояниям 6 и 4 тяготеет состояние 2, поэтому мы разместим его в вершине 110, соседней как вершине 100, так и вершине 010.

Полученное частичное размещение состояний представлено на рис. 4.4.5.

Поскольку пары состояний $\{6, 7\}$, $\{4, 7\}$, $\{2, 6\}$ и $\{2, 4\}$ оказались расположенными в соседних вершинах, о соответствующих переходах можно дальше не беспокоиться. Можно считать, что реализация перехода $4 \to 6$ также обеспечена, поскольку пара состояний $\{4, 6\}$ вхо-

дит в K-множество $\{4, 6, 7\}$, а этому K-множеству соответствует связный подграф, содержащий лишь те вершины, в которых и размещены данные состояния. Поэтому соответствующим элементам матрицы связи можно условно придать значение нуль:

$$\begin{array}{c|c} 2\,3\,4\,5\,6\,7 & \\ \hline 1\,1\,1\,3\,0\,1 & 1 \\ 0\,0\,1\,0\,0 & 2 \\ 1\,0\,1\,1 & 3 \\ 2\,0\,0 & 4 \\ 0\,1 & 5 \\ 0 & 6 \end{array}$$

после чего остается побеспокоиться о тех парах состояний, которым соответствуют элементы этой матрицы, значения которых остались отличными от нуля.

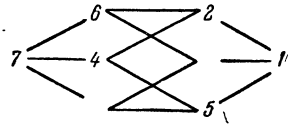


Рис. 4.4.6.

Из оставшихся неразмещенными состояний наибольшие требования предъявляются к состоянию 5. Его целесообразно разместить рядом с состоянием 4, поскольку коэффициент связи между этими состояниями равен 2. Соответствующее свободное место на графе единственно, и состояние 5 размещается в вершине 011. Затем, по аналогичным соображениям, рассматривается состояние 1 и находится, что лучше всего разместить это состояние в вершине 111, после чего граф частичного размещения состояний принимает следующий вид (рис. 4.4.6).

Наконец, осталось разместить состояние 3. Остались лишь две свободные вершины, и выбор падает на вершину 101, поскольку она лучше удовлетворяет требованиям, задаваемым коэффициентами связи. В оставшуюся вершину 001 мы помещаем дополнительное состояние 8.

Посмотрим теперь, как разместились на графе K-множества. Оказывается, что все они, кроме одного, разместились плотно, то есть на таких подмножествах, которые порождают связные подграфы, не содержащие дополнительных вершин. Такое размещение является

идеальным с нашей точки зрения, обеспечивая решение поставленной задачи соседнего кодирования.

Исключение составляет лишь одно K-множество, а именно, пара состояний $\{3, 4\}$. Эти состояния оказываются размещенными на вершинах, не являющихся соседними. Поэтому связывающий эти состояния подграф может быть построен только при условии добавления в него каких-либо дополнительных вершин. Резервом здесь может служить та совокупность вершин, размещенные в которых состояния не входят в разбиение,

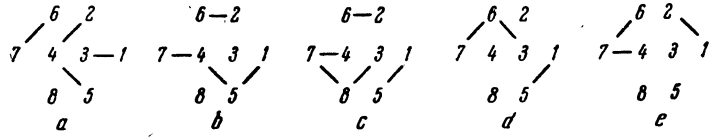


Рис. 4.4.7.

соответствующее K-множеству $\{3, 4\}$. Это разбиение задается входным состоянием c и содержит, кроме рассматриваемого K-множества $\{3, 4\}$, также K-множества $\{1, 5\}$ и $\{2, 6\}$. В резерв входят, как мы видим, вершины 000 и 001, в которых размещены состояния 7 и 8. Этот резерв достаточен для построения связного подграфа, содержащего интересующую нас пару вершин. Следовательно, найденное решение полностью удовлетворяет всем требованиям соседнего кодирования состояний.

Полученный нами результат отличается от приведенного выше решения и представляется следующим разбиением полного булева графа на подграфы, соответствующие K-множествам (рис. 4.4.7). Отсюда легко находится матрица элементарных переходов данного автомата:

a	b	c	d	e	
1	5	1	5	2	1
2	6	2	–	**2**	2
1	–	8	3	–	3
2	5	**4**	–	7	4
4	**5**	1	5	–	5
7	**6**	2	3	**6**	6
7	4	4	6	6	7
–	–	7	–	–	8

Кодирующая матрица Q имеет при этом следующее значение:

$$Q=\begin{matrix}\\1\\1\\1\end{matrix}\begin{matrix}1&2&3&4&5&6&7&8\\\end{matrix}\begin{bmatrix}1&1&1&0&0&1&0&0\\1&1&0&1&1&0&0&0\\1&0&1&0&1&0&0&1\end{bmatrix}$$

и поведение автомата можно представить следующим графом переходов (рис. 4.4.8).

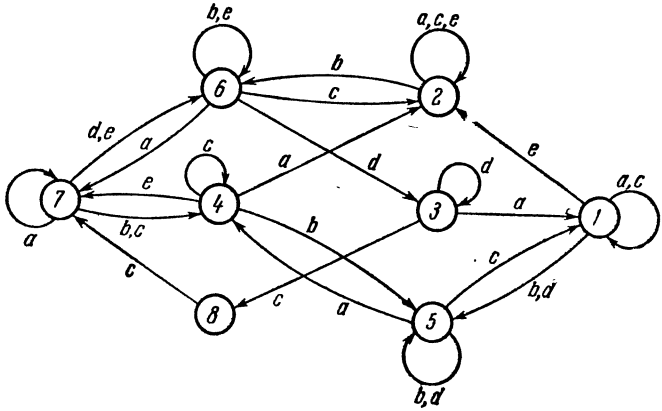


Рис. 4.4.8.

В рассмотренном примере нам удалось обойтись минимальным числом компонент в кодах состояний, разместив последние на вершинах S-мерного куба. В противном случае следовало бы увеличить размерность куба на единицу и начать процесс размещения сначала, если же и в этом случае решение не будет найдено, размерность получит следующее приращение, и т. д.

§ 5. Синтез комбинационных автоматов

Общие положения. Как уже говорилось, при синтезе комбинационного автомата решается следующая задача: задается базис синтеза и задается булева функция или система булевых функций $y=\varphi(x)$, которую

надо реализовать в этом базисе, то есть выразить в виде суперпозиции элементарных булевых функций, реализуемых элементами базиса. При этом получаемая суперпозиция должна удовлетворять определенным условиям, составляющим в совокупности синтаксис базиса.

Естественно попытаться найти какой-либо универсальный метод синтеза, которым можно было бы пользоваться при любом базисе. Простейшим из таких методов является метод сплошного перебора логических сетей, составленных сначала из одного элемента, затем из двух, из трех и т. д., с проверкой функциональных свойств перебираемых сетей. Как только окажется, что рассматриваемая сеть реализует заданную функцию (или систему), она принимается в качестве решения, причем очевидно, что это решение будет минимальным по числу элементов. Очевидно также, что это решение будет в конце концов получено, то есть можно говорить о «принципиальном» решении задачи синтеза в общем случае.

Однако эффективность этого метода настолько низка, что не приходится и говорить о его практическом использовании. Достаточно, например, заметить, что существует 10^{20} способов взаимного соединения 10 элементов, каждый из которых обладает двумя входными полюсами и одним выходным!

Существенно более перспективными оказываются методы функционального разложения, в которых задача реализации рассматриваемой булевой функции может сводиться, например, к задаче реализации нескольких других функций, более простых в некотором смысле. В ряде случаев эти методы позволяют получать достаточно удовлетворительные решения за приемлемое время.

Однако наиболее эффективными в настоящее время являются методы синтеза, развитые для конкретных базисов. С некоторыми из них мы и познакомимся в данном параграфе.

Базис произвольных днф. Так мы назовем базис, в котором синтезируются структуры, непосредственно соответствующие дизъюнктивным нормальным формам булевых функций. Это означает, что элементами базиса являются конъюнкторы и дизъюнкторы

с произвольным числом входов, реализующие конъюнкции и дизъюнкции любого числа переменных. Эти элементы могут соединяться так, чтобы образовались двухъярусные сети, в которых элементами первого яруса служат конъюнкторы, а элементами второго — дизъюнкторы. При этом выходные полюсы элементов первого яруса могут соединяться с входными полюсами элементов второго яруса, а выходные полюсы элементов второго яруса служат выходными полюсами комбинационной сети в целом. Входные же полюсы сети соединяются непосредственно с входными полюсами элементов первого яруса. Эти правила взаимного соединения элементов составляют синтаксис базиса синтеза.

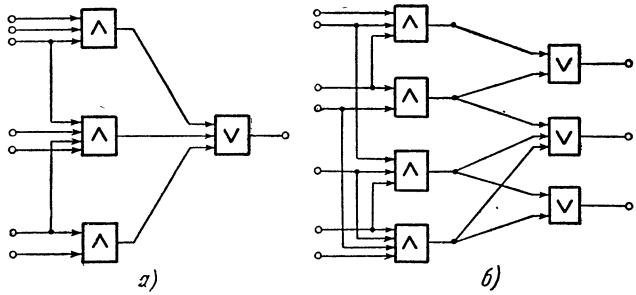


Рис. 4.5.1.

Типичные примеры получаемых в этом базисе комбинационных сетей показаны на рис. 4.5.1 (*а* — с одним выходным полюсом и *б* — с несколькими).

Обратим внимание на то обстоятельство, что для реализации функционального отношения $y=\varphi(x)$ недостаточно получить структуру комбинационного автомата в том виде, как это показано на данном рисунке. Требуется еще «подать» аргументы реализуемой системы функций на входные полюсы сети и «снять» значения функций с выходных полюсов. Другими словами, надо отметить полюсы сети символами переменных, связываемых реализуемой системой функций, после чего однозначно определяются дизъюнктивные нормальные формы последних.

Изображение структур рассматриваемых комбинационных автоматов может принять, например, следующий

вид (рис. 4.5.2), соответствующий дизъюнктивным нормальным формам:

а) $y = abc \vee \bar{a}b\bar{c}d \vee \bar{a}e;$

б) $y_1 = x_1x_2x_3 \vee x_3x_4,$

$y_2 = x_3x_4 \vee x_2\bar{x}_3x_5 \vee \bar{x}_1\bar{x}_3x_4x_5,$

$y_3 = x_2\bar{x}_3x_5 \vee \bar{x}_1\bar{x}_3x_4x_5.$

Остановимся еще на одном существенном обстоятельстве. Отмечая некоторые из входных полюсов сети символами инверсий аргументов, мы предполагаем, что эти инверсии получаются где-то вне синтезируемой сети

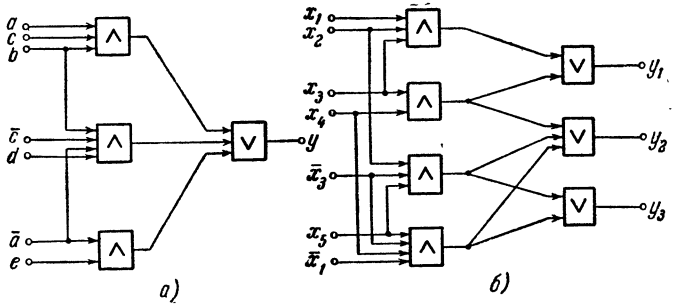


Рис. 4.5.2.

и доступны нам наравне с прямыми значениями аргументов. Именно это обстоятельство обеспечивает полноту данного базиса, позволяя реализовывать в нем любые булевы функции.

Реализация одной булевой функции в базисе произвольных днф. Очевидно, что при решении этой задачи нужно построить в данном базисе сеть с одним выходным полюсом, состояние которого и будет представлять значение реализуемой булевой функции. При этом сеть должна быть оптимальной в каком-то смысле. Например, можно потребовать, чтобы сеть обладала минимально возможным числом элементов. В этом случае синтез сводится в основном к нахождению кратчайшей днф заданной булевой функции.

Эта задача уже хорошо знакома нам по предыдущим главам, и здесь мы ограничимся рассмотрением небольшого примера, также знакомого нам по параграфу 3.3.

Допустим, что требуется реализовать неполностью определенную булеву функцию, заданную парой матриц:

$$\| M^1 \ni X \| = \begin{matrix} a\,b\,c\,d\,e\,f \\ \left[\begin{matrix} 0\,0\,1\,1\,0\,0 \\ 0\,1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1\,0 \\ 1\,0\,1\,0\,0\,0 \\ 1\,1\,0\,0\,0\,1 \\ 1\,1\,0\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1 \\ 1\,1\,0\,1\,0\,0 \\ 0\,1\,1\,1\,1\,1 \end{matrix}\right] \end{matrix}, \quad \| M^0 \ni X \| = \begin{matrix} a\,b\,c\,d\,e\,f \\ \left[\begin{matrix} 1\,0\,0\,0\,0\,1 \\ 1\,0\,1\,1\,1\,1 \\ 0\,1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,1\,1\,0 \\ 0\,1\,1\,0\,0\,1 \\ 0\,1\,0\,1\,0\,1 \end{matrix}\right] \end{matrix}.$$

Используя какой-либо из методов минимизации булевых функций в классе д н ф по критерию минимума числа членов, мы находим кратчайшую д н ф, которая и представляет собой искомое решение. Например, применение алгоритма м и н с о б приводит нас к следующей форме:

$$y = ab \vee be \vee \bar{d}ef \vee \bar{b}\bar{e}\bar{f},$$

которая задает структуру соответствующего комбинационного автомата, представленную на рис. 4.5.3.

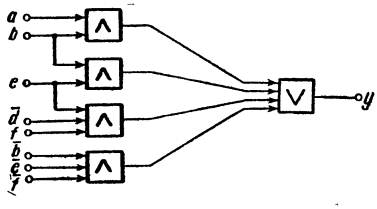


Рис. 4.5.3.

Иногда минимизация производится по несколько иным критериям. Например, можно минимизировать число входных полюсов комбинационной сети, отказываясь от дублирования прямых значений аргументов на входных полюсах их инверсиями. Можно стремиться к минимуму используемых инверсных значений аргументов. При некоторых обстоятельствах возникает задача «выравнивания нагрузки среди аргументов», когда минимизируется максимальное число конъюнкторов,

непосредственно связанных с одним и тем же входным полюсом сети. Многообразие этих критериев порождает множество различных методов синтеза, которые, однако, не будут здесь рассматриваться.

О реализации системы булевых функций. При синтезе комбинационного автомата, реализующего систему булевых функций, возникают новые проблемы. Очевидно, что решение этой задачи можно получить путем реализации функций по отдельности. Однако такое решение вряд ли будет лучшим в том или ином смысле.

Рассмотрим, например, требование минимума числа элементов в сети, синтезируемой в базисе произвольных днф. В этом случае число элементов второго яруса будет, как правило, равно числу функций. Исключением из этого правила будет случай, когда некоторая функция может быть представлена одной элементарной конъюнкцией: тогда значение данной функции может быть снято непосредственно с выходного полюса соответствующего элемента первого яруса. Другим исключением является случай, когда некоторые функции оказываются равными или могут стать такими при соответствующем их доопределении. Оба этих исключения довольно тривиальны, поэтому основные усилия должны быть направлены на минимизацию числа элементов первого яруса.

Исходная система булевых функций может задаваться по-разному, в зависимости от таких параметров системы, как число функций, число аргументов, степень определенности функций и т. д. Например, если общее число элементов булева пространства M, на которых задано значение по крайней мере одной из булевых функций рассматриваемой системы, невелико, то систему можно задать парой матриц, первая из которых — булева — представляет своими строками совокупность упомянутых элементов из M, а вторая — троичная — показывает значения заданных функций на этих элементах, причем наряду со значениями 0 и 1 могут использоваться символы неопределенности «—».

Рассмотрим конкретный пример системы из пяти слабо определенных булевых функций p, q, r, s, и t от шести переменных a, b, c, d, e и f. Эта система задается

следующей парой матриц:

a b c d e f		*p q r s t*
1 0 0 1 1 0		1 – 0 – –
0 1 1 0 1 1		1 – 1 0 –
0 0 1 0 1 0		0 – 0 – 1
0 1 0 1 0 1		– 1 – 1 0
1 0 0 1 1 1		1 0 – 1 –
1 1 0 1 0 0		– – 1 – 0
1 0 1 0 0 1		0 1 – 0 –
0 1 0 0 1 0		0 – 1 – 0
0 1 0 1 0 0		– 0 – 1 –
1 1 0 0 1 0		– 1 0 – 1
0 1 1 1 0 1	,	– – 1 0 –
0 0 1 0 0 0		1 – 1 0 –
1 0 1 1 1 0		– 0 – – 1
0 0 0 0 1 0		– – 0 – 1
1 1 0 1 0 1		– 0 1 1 –
1 0 0 0 0 1		– – – 1 0
0 0 1 1 1 0		1 0 – – 0
0 1 1 0 0 1		0 1 1 – 0
1 0 0 0 1 0		– – 1 1 0
0 1 1 1 1 0		– 0 – 0 1

Некоторые из элементов второй матрицы выделены жирным шрифтом. Зачем это сделано, будет сказано несколько ниже.

Минимизация числа конъюнкторов. Совокупность элементов первого яруса должна соответствовать некоторой системе элементарных конъюнкций, из которых можно построить днф для любой из заданных функций. Естественно попытаться найти такую систему, которая содержала бы минимальное число конъюнкций, и, как обычно в подобных ситуациях, сформулировать данную задачу как задачу нахождения кратчайшего покрытия некоторой булевой матрицы.

Однако, прежде чем строить такую матрицу, разумно учесть то обстоятельство, что элементы первого яруса синтезируемой сети требуются для реализации лишь таких конъюнкций, ранг которых больше единицы. Конъ-

юнкции же первого ранга являются «бесплатными» в том смысле, что их значения заданы уже на входных полюсах сети, которые, следовательно, могут быть непосредственно связаны с соответствующими элементами второго яруса. Поэтому целесообразно начать решение задачи с поиска таких конъюнкций первого ранга, которые могли бы входить в д н ф некоторых функций заданной системы, не будучи там избыточными.

Нетрудно реализовать перебор всех элементарных конъюнкций ранга 1, поскольку их число невелико, и проанализировать каждую из них. При этом анализе мы выясняем, например, что конъюнкция a не может входить в д н ф функции p, поскольку последняя принимает значение 0 на элементе 101001, когда конъюнкция a принимает значение 1. В д н ф функции p не может входить и конъюнкция $\bar{a}$, так как p принимает значение 0 на элементе 001010, когда $\bar{a}$ обращается в единицу.

В результате анализа мы находим всего пять элементарных конъюнкций ранга 1, а именно, конъюнкции $\bar{c}$, d, $\bar{d}$, $\bar{e}$ и f, которые могут входить в д н ф функций s, p, q, r и r, соответственно. Оказывается, что конъюнкция $\bar{c}$ полностью реализует функцию s, то есть принимает значение 1 на всех элементах булева пространства, на которых обязательно значение 1 для функции s. Остальные же функции реализуются не полностью, а лишь на некоторой части обязательных единичных значений. Именно такие значения были нами заблаговременно отмечены в приведенной выше матрице.

Побеспокоимся теперь о реализации остальных обязательных единичных значений, которых теперь осталось уже немного — 10. Здесь мы применим методику, несколько напоминающую правила нахождения интервально-поглощающих подмножеств из M^1, используемые при минимизации слабо определенной булевой функции (глава 3, § 2).

Пусть, например, функция x принимает значение 1, в частности, на элементе m_i, функция y — на m_j, ..., функция z — на m_k. Если ни одна из этих функций не имеет обязательных нулевых значений на минимальном покрывающем интервале для элементов m_i, m_j, ..., m_k,

то соответствующая этому интервалу элементарная конъюнкция может войти одновременно в днф каждой из перечисленных функций.

В данном случае такое совмещение оказывается возможным лишь на двух парах элементов: ($p1$, $t5$), ($t1$, $t4$). Заметим, что мы пользуемся отдельной для каждого столбца нумерацией невыделенных единичных элементов матрицы, сверху вниз.

Теперь легко строится булева матрица, к нахождению кратчайшего покрытия которой сводится задача минимизации числа конъюнкторов. Она имеет следующий вид:

$$\begin{array}{c} p\ q\ r\ \ t \\ 1\,2\ 1\ 1\,2\ 1\,2\,3\,4\,5 \\ \left[\begin{array}{l} 1\,0\ 0\ 0\,0\ 0\,0\,0\,0\,1 \\ 0\,1\ 0\ 0\,0\ 0\,0\,0\,0\,0 \\ 0\,0\ 1\ 0\,0\ 0\,0\,0\,0\,0 \\ 0\,0\ 0\ 1\,0\ 0\,0\,0\,0\,0 \\ 0\,0\ 0\ 0\,1\ 0\,0\,0\,0\,0 \\ 0\,0\ 0\ 0\,0\ 0\,1\,0\,0\,0 \\ 0\,0\ 0\ 0\,0\ 0\,0\,1\,0\,0 \\ 0\,0\ 0\ 0\,0\ 1\,0\,0\,1\,0 \end{array}\right] \end{array}$$

Кратчайшее покрытие находится чрезвычайно просто, поскольку оказывается, что в него входят все строки данной матрицы. Итак, минимальное число конъюнкторов равно восьми — числу строк. Осталось найти реализуемые ими конъюнкции.

Выбор элементарных конъюнкций. Рассмотрим первую строку построенной булевой матрицы. Единицы в ней соответствуют двум элементам пространства M: элементу 0 1 1 0 1 1, на котором принимает значение 1 функция p, и элементу 0 1 1 1 1 0, на котором принимает значение 1 функция t. Легко находится минимальный покрывающий интервал для этих элементов. Он представляется троичным вектором 0 1 1 — 1 —, и ему соответствует элементарная конъюнкция $\bar{a}bce$.

Любой другой интервал пространства M, содержащий оба указанных элемента, должен поглощать найденный минимальный покрывающий интервал, откуда следует, что любая элементарная конъюнкция,

обеспечивающая реализацию данных двух единичных значений функций p и t, может быть получена путем выбрасывания некоторых символов из конъюнкции $\bar{a}bce$. При этом желательно оставить в конъюнкции минимально возможное число символов, так как этим самым мы минимизируем число входных полюсов соответствующего конъюнктора.

Рассмотрим совокупность элементов пространства M, на которых принимает значение 0 по крайней мере одна из функций p или t:

$$\begin{array}{c} a\,b\,c\,d\,e\,f \\ \left[\begin{array}{c} 0\,0\,1\,0\,1\,0 \\ 0\,1\,0\,1\,0\,1 \\ 1\,1\,0\,1\,0\,0 \\ 1\,0\,1\,0\,0\,1 \\ 0\,1\,0\,0\,1\,0 \\ 1\,0\,0\,0\,0\,1 \\ 0\,0\,1\,1\,1\,0 \\ 0\,1\,1\,0\,0\,1 \\ 1\,0\,0\,0\,1\,0 \end{array}\right] \end{array}$$

Каждый из этих булевых векторов ортогонален полученному троичному вектору 0 1 1 — 1 — по той или иной компоненте, а то и по нескольким сразу. Нужно заменить значения некоторых компонент троичного вектора с 0 или 1 на «—», однако так, чтобы этот вектор оставался по-прежнему ортогонален всем перечисленным булевым векторам.

Нетрудно сообразить, что решение этой задачи сводится к нахождению кратчайшего столбцового покрытия булевой матрицы, представляющей отношение покомпонентной ортогональности между троичным вектором, с одной стороны, и заданной системой булевых векторов, с другой стороны. Эта матрица получается из предыдущей с помощью оператора $\oplus$, реализующего покомпонентную операцию «дизъюнкция с исключением», в которой вторым операндом служит троичный вектор 0 1 1 — 1 — (причем в столбцах, соответствующих компонентам этого вектора со значением

«—», проставляются нули):

a	*b*	*c*	*d*	*e*	*f*
0	1	0	0	0	0
0	0	1	0	1	0
1	0	0	0	1	0
1	1	0	0	1	0
0	0	1	0	0	0
1	1	1	0	1	0
0	1	0	0	0	0
0	0	0	0	1	0
1	1	1	0	0	0

Единственным кратчайшим столбцовым покрытием этой матрицы является совокупность столбцов (*b*, *c*, *e*). Это значит, что в троичном векторе можно всем компонентам, кроме этих, присвоить значение «—», получив таким образом вектор — 1 1 — 1 —, которому соответствует элементарная конъюнкция *bce*.

Подобным же образом мы находим остальные семь элементарных конъюнкций и представляем полученное решение в целом парой матриц, первая из которых — троичная — представляет перечень элементарных конъюнкций, образующих базу дизъюнктивного разложения булевых функций заданной системы, а вторая — булева — показывает, в д н ф, каких именно функций входят данные конъюнкции:

a	*b*	*c*	*d*	*e*	*f*	*p*	*q*	*r*	*s*	*t*
—	—	0	—	—	—	0	0	0	1	0
—	—	—	1	—	—	1	0	0	0	0
—	—	—	0	—	—	0	1	0	0	0
—	—	—	—	—	1	0	0	1	0	0
—	1	1	—	1	—	1	0	0	0	1
—	—	—	—	0	0	1	0	1	0	0
0	—	—	—	—	1	0	1	0	0	0
0	1	—	—	—	—	0	0	1	0	0
1	0	—	0	—	—	0	0	1	0	0
1	1	—	—	1	—	0	0	0	0	1
1	—	1	—	—	—	0	0	0	0	1
0	—	—	0	—	0	0	0	0	0	1

Заметим, что мы не включили в базу найденную ранее конъюнкцию $\bar{e}$, так как реализуемые ею единичные значения функции *r* полностью перекрываются другими конъюнкциями, входящими в д н ф этой функции.

Обсуждение результата. Полученный результат может быть представлен и в графической форме, что мы и попытались сделать на рис. 4.5.4. Однако такая форма вряд ли лучше используемой нами матричной формы даже для рассматриваемого примера, когда межъярусные связи «слабо перемешаны». С усложнением примеров графическое представление раньше теряет обозримость, чем матричное, поэтому не следует злоупотреблять им.

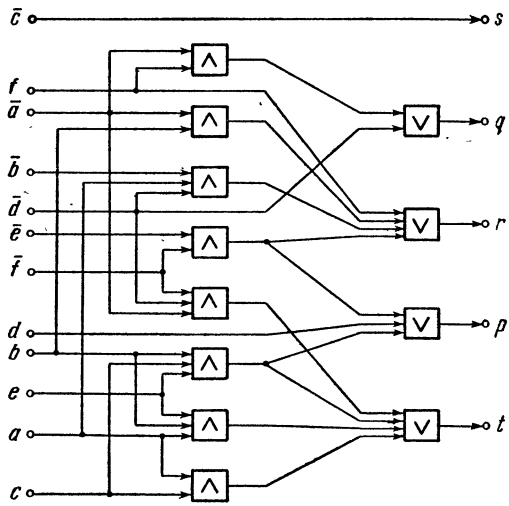


Рис. 4.5.4.

Непосредственно по полученным матрицам легко находить такие характеристики синтезированной сети, как максимальное число входных полюсов используемых элементов, максимальную нагрузку на входные полюсы и на конъюнкторы и т. д. Заметим, что изложенный метод синтеза позволяет в какой-то степени варьировать эти характеристики. Например, на этапе минимизации числа конъюнкторов можно выравнивать нагрузки конъюнкторов, а на этапе выбора элементарных конъюнкций — нагрузки входных полюсов сети.

Заметим также, что синтезированная сеть в данном случае оказалась линейно-упорядоченной, но уже не

ярусной, поскольку, например, некоторые входные полюсы сети оказались соединенными непосредственно с элементами второго яруса.

Инверсные конъюнкторы и инверсные дизъюнкторы. Так мы будем называть элементы, реализующие функции $\overline{x_1 \wedge x_2 \wedge \ldots \wedge x_n}$ и $\overline{x_1 \vee x_2 \vee \ldots \vee x_n}$, соответственно. Эти элементы замечательны тем, что каждый из них может обеспечить полноту базиса синтеза, позволяя реализовывать любые булевы функции при некотором синтексисе структур.

Рассмотрим пока класс двухъярусных сетей, элементами которых могут служить инверсные конъюнкторы с произвольным числом входных полюсов.

Используя тождество $\overline{\overline{ab} \wedge \overline{cd}} = ab \vee cd$, нетрудно показать, что синтез сети данного класса, реализующей заданную булеву функцию, может быть сведен к синтезу соответствующей двухъярусной сети в базисе произвольных днф с последующей заменой всех элементов полученной сети на инверсные конъюнкторы.

Например, рассмотренная в начале этого параграфа неполностью определенная булева функция, реализованная сетью, изображенной на рис. 4.5.3, реализуется также сетью, представленной на рис. 4.5.5 и составленной исключительно из инверсных конъюнкторов.

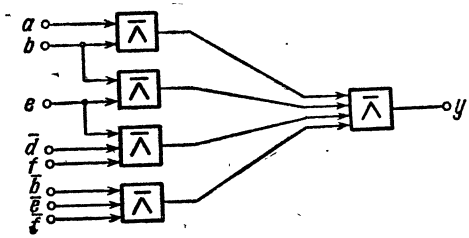


Рис. 4.5.5.

Таким образом, при синтезе сетей рассматриваемого класса мы в полной мере можем использовать методы минимизации булевых функций в классе днф. К этому же можно свести и синтез двухъярусных сетей из инверсных дизъюнкторов. Посмотрим, как это делается.

Если $f=\overline{\overline{(a \vee b)} \vee \overline{(c \vee d)}}$, то $\bar{f}=\bar{a}\bar{b} \vee \bar{c}\bar{d}$. Отсюда следует, что при реализации булевой функции $\varphi(x_1, x_2, \ldots, x_n)$ можно перейти к функции $\bar{\varphi}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ (такая функция называется *двойственной* по отношению к функции $\varphi(x_1, x_2, \ldots, x_n)$), реализовать эту функцию в базисе произвольных д н ф, а затем заменить все элементы полученной сети инверсными дизъюнкторами.

Например, рассматривая все ту же булеву функцию, мы легко находим двойственную ей функцию, для чего достаточно произвести обмен значениями между матрицами $\|M^1 \ni X\|$ и $\|M^0 \ni X\|$ и инверсировать все элементы этих матриц:

$$\|M^1 \ni X\|=\begin{array}{c} \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \left[\begin{array}{cccccc} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \qquad \|M^0 \ni X\|=\begin{array}{c} \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Далее можно получить кратчайшую д н ф этой функции. Довольно просто получается совокупность максимальных интервально-покрываемых подмножеств из M^1, представляемая следующей булевой матрицей:

$$\|\sup M^* \ni M^1\|=\begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \end{array}$$

и находится кратчайшее покрытие этой матрицы:

$$\left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right]$$

Интервалы, соответствующие элементам данного покрытия, расширяются на множестве $M \setminus M^0$, в результате чего мы получаем следующую д н ф:

$$bce \vee \bar{a}b\bar{d} \vee ac \vee a\bar{b}e.$$

Построив задаваемую этой формулой комбинационную сеть и заменив в ней все элементы на инверсные дизъюнкторы, мы получаем окончательный результат, пред-

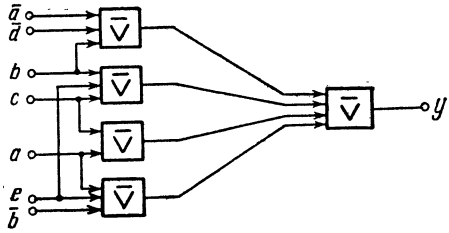


Рис. 4.5.6.

ставленный на рис. 4.5.6, — сеть интересующего нас типа, реализующую исходную неполностью определенную булеву функцию.

Синтез многоярусных сетей. При решении задач практического синтеза комбинационных автоматов приходится учитывать ряд ограничений, характеризующих возможности использования реальных элементов. Эти ограничения могут задаваться формально значениями некоторых параметров. Примерами таких параметров для инверсных дизъюнкторов и конъюнкторов могут служить число p входных полюсов элемента и коэффициент ветвления q его выходного полюса, показывающий максимальное число входных полюсов других элементов, с которыми может соединяться указанный выходной полюс. Последнее ограничение имеет энергетический характер, определяя максимально допустимую нагрузку выходного полюса реального элемента.

Очевидно, что при конечных значениях p и q уже нельзя будет для любой булевой функции найти реализующую ее двухъярусную сеть. Однако полнота базиса может быть восстановлена снятием ограничения на

число ярусов в сети, если только значение p не будет меньше 2. В связи с этим возникает задача синтеза многоярусных сетей.

Задача эта оказывается значительно сложнее рассмотренной выше задачи синтеза двухъярусных сетей. Чтобы убедиться в этом, достаточно рассмотреть, например, четырехъярусную сеть из инверсных конъюнкторов. Поскольку каждая пара ярусов реализует некоторую систему д н ф, то можно считать, что сеть в целом реализует систему д н ф *второго порядка*, то есть систему д н ф от промежуточных булевых переменных, каждая из которых представляется посредством некоторой д н ф от исходных переменных, поставленных в соответствие входным полюсам сети. Решающим шагом является здесь выбор системы указанных промежуточных переменных, после чего задача сводится к синтезу двух двухъярусных сетей. Однако этот шаг довольно труден.

С увеличением числа ярусов мы переходим к рассмотрению системы д н ф более высокого порядка, со многими уровнями промежуточных переменных. Задача становится еще более сложной и применяемые на практике алгоритмы ее решения все меньше претендуют на точность.

§ 6. Третий уровень языка ЛЯПАС

Б о л ь ш и е п р о г р а м м ы и с в я з а н н ы е с н и м и п р о б л е м ы. Допустим, что программист хорошо представляет себе суть разрабатываемого им алгоритма и хочет выразить его средствами данного алгоритмического языка. Будем считать, что эти средства высокоэффективны, если они позволяют программисту получить желаемый результат достаточно быстро, скажем за один рабочий день. Очевидно, что эффективность алгоритмического языка зависит от сложности алгоритмов, которые можно представить в этом языке.

Первый уровень языка ЛЯПАС высокоэффективен при представлении таких алгоритмов решения логических задач, которым соответствуют машинные программы объемом до 200—300 команд. Конечно, средствами первого уровня можно обходиться и при разработке значительно более сложных алгоритмов, однако эффек-

тивность этих средств в этом случае резко понижается вследствие потери хорошей обозримости алгоритма и практической невозможности выполнить работу по выражению алгоритма «за один присест». Достаточно напомнить, что при многократном возвращении к этой работе программисту каждый раз приходится долго и мучительно восстанавливать в своей памяти подробности общего плана по выражению алгоритма, а также уточнять, в какой степени этот план уже реализован.

Второй уровень языка ЛЯПАС позволяет при наличии в эльтеке соответствующих подпрограмм достаточно легко выражать алгоритмы на порядок более сложные, то есть такие, которые трансформируются в машинные программы объемом до 2000—3000 команд. Казалось бы, ничто не мешает нам повысить этот потолок, обогатив эльтеку более сложными подпрограммами. Однако здесь мы наталкиваемся на новое препятствие, связанное с ограниченностью объема оперативной памяти универсальных вычислительных машин. Это препятствие имеет принципиальный характер, поскольку увеличение объема оперативной памяти, представляющее тенденцию современного развития вычислительной техники, не устраняет его, а лишь несколько отодвигает.

Может случиться, что объем оперативной памяти окажется недостаточным для представления перерабатываемой алгоритмом информации, в связи с чем возникнет необходимость использования дополнительной памяти наряду с оперативной и организации надлежащего информационного обмена между двумя типами памяти. Такой обмен может быть задан посредством операторов *зап* и *счит*, включаемых в рассматриваемый алгоритм. Конечно, задача программиста при этом усложняется, однако можно считать, что второй уровень ЛЯПАСа оказывается вполне достаточным для ее решения.

С более серьезными затруднениями мы сталкиваемся, когда емкость оперативной памяти оказывается недостаточной для помещения самой программы. В этом случае программу необходимо *сегментировать*, то есть разбить на некоторые куски (*сегменты*), последовательно вводимые в оперативную память для реализации. Реализация сегментов должна как-то чередоваться

с работой блоков программирующей системы (ПС), транслирующих сегменты на машинный язык и подготавливающих их к реализации. Требует своего решения также задача информационного согласования сегментов.

Далеко не все возможные варианты сегментирования большой программы оказываются равноценными. Реализация блоков ПС и обмен информацией с дополнительной памятью требуют значительных затрат времени, поэтому при сегментировании следует стремиться к минимизации этих потерь путем наиболее рационального выделения сегментов.

Решение этой задачи можно, в принципе, поручить машине, то есть полностью автоматизировать весь процесс сегментирования. Это позволило бы нам затем абстрагироваться от ограничений, присущих реальным вычислительным машинам, и вести программирование так, как если бы этих ограничений не было. Однако во многих случаях это привело бы к существенным дополнительным потерям времени при реализации программы, поскольку для выбора оптимального варианта сегментирования может потребоваться информация о содержании решаемой задачи. Такая информация известна программисту, но отсутствует в программе или по крайней мере ее очень трудно оттуда извлечь. Положение осложняется тем, что в ряде случаев необходимость сегментирования может оказать более глубокое влияние на структуру алгоритма и даже на выбор метода решения задачи. Эти зависимости пока трудно формализовать.

С другой стороны, задача сегментирования может быть полностью решена программистом в рамках второго уровня ЛЯПАСа. Действительно, информационный обмен между оперативной и дополнительной памятью может быть организован, как уже говорилось, с помощью операторов *зап* и *счит*. Операторы *локал* и *маш* позволяют связывать машинные программы, полученные в результате независимой трансляции различных участков большой Л-программы. Таким образом, сегментирование Л-программы может быть отражено некоторым ее расширением. Однако необходимость учета массы чисто технических деталей, в которые приходится вникать программисту, делает довольно громоздким как

само программное выражение расширения, так и весь процесс его получения.

В качестве выхода из создавшейся ситуации предлагается расширение языка ЛЯПАС, описываемое в данном параграфе. В нем содержатся средства, позволяющие повысить на порядок сложность рассматриваемых алгоритмов, то есть обеспечивающие достаточную легкость составления Л-программ, машинные эквиваленты которых обладают объемом до 20—30 тысяч команд. Назовем это расширение *третьим уровнем ЛЯПАСа*. С введением этого уровня программисту предоставлены, образно выражаясь, некоторые «кнопки» для управления процессом сегментирования. Вся техническая работа по реализации даваемых программистом указаний возложена на систему автоматического программирования.

Суперпрограммы. Большие программы, сегментирование которых неизбежно, будем называть *суперпрограммами*. Как правило, мы получаем их, используя при программировании Л-операторы, которым соответствуют уже достаточно большие подпрограммы. Зная объем этих подпрограмм, мы можем сами классифицировать составленную нами программу как суперпрограмму. Если же мы не сделаем этого и попытаемся реализовать программу без предварительного сегментирования, то ПС откажется от такой реализации и информирует нас, что перед нами суперпрограмма и ее следует сегментировать.

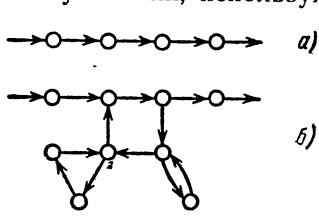


Рис. 4.6.1.

Структура сегментированной программы может быть *линейной*, когда каждый из сегментов реализуется однократно (рис. 4.6.1, *а*), либо *циклической* — когда возможна многократная реализация некоторых сегментов (рис. 4.6.1, *б*).

При реализации сегментированной суперпрограммы с линейной структурой может быть рекомендован *режим интерпретации*, то есть такая последовательная обработка сегментов, при которой каждый из них

трансформируется в машинную программу, а затем реализуется, после чего машинная программа сегмента уничтожается (сохраняются лишь полученные при ее реализации результаты) и аналогичной обработке подвергается следующий сегмент. Таким образом, реализация сегментов чередуется с работой ПС, транслирующей их на машинный язык, причем каждый сегмент транслируется лишь один раз.

В случае циклической структуры сегментированной программы положение меняется: если перед каждой реализацией какого-либо сегмента заново производить его трансляцию, то дополнительные затраты времени могут стать неоправданными. В этом случае может оказаться полезным *режим заготовки*, при котором сегмент транслируется однократно, а затем его машинная программа хранится в дополнительной памяти, откуда извлекается по мере надобности. Однако не следует злоупотреблять этим режимом, помня о расходе дополнительной памяти на хранение машинной программы.

При выборе режима обработки того или иного сегмента следует оценить «накладные расходы» на его трансляцию. Если, например, время трансляции сегмента существенно меньше времени его реализации, то целесообразно обрабатывать сегмент в режиме интерпретации, независимо от того, какое место занимает он в суперпрограмме. Основным критерием выбора режима обработки служит рост времени реализации суперпрограммы в целом при замене режима заготовки некоторого сегмента режимом интерпретации: если такой рост относительно невелик, рекомендуется использование режима интерпретации, при котором уменьшается загрузка дополнительной памяти.

Ограничимся рассмотрением здесь простейшего варианта режима заготовки. В этом варианте управление процессом сегментирования возлагается на программиста. Чтобы упростить его функции, положим, что сегментами могут служить лишь подпрограммы, причем любая подпрограмма будет реализовываться как сегмент, если только она будет отмечена соответствующим образом. Примем, что *меткой сегмента* будет служить дополнительный символ * перед именем подпрограммы. Заметим, что такое упрощение процесса сегментирова-

ния не приводит в то же время к существенному сокращению его возможностей.

Решение вопроса о целесообразности реализации той или иной подпрограммы как сегмента должно приниматься программистом на основе учета той роли, которая отведена подпрограмме в конкретной суперпрограмме. Впрочем, если некоторая подпрограмма требует достаточно большого времени реализации, ее всегда можно реализовывать как сегмент.

Важной особенностью сегмента является то, что в его распоряжение может быть предоставлена практически вся оперативная память, в связи с чем можно позволить ему потерять ту гибкость, которой обладают подпрограммы, выраженные в ЛЯПАСе (будем называть их *Л-блоками*), и которая облегчает процедуру компиляции, и заранее зафиксировать его расположение в оперативной памяти. Если к тому же конкретизировать набор внешних операндов сегмента, то настройку последнего можно считать законченной. Такой сегмент можно представить в форме машинной программы, и это представление, называемое *М-блоком*, в ряде случаев может оказаться весьма полезным, поскольку при обращении к сегменту в такой форме нет необходимости в предварительном его переводе на машинный язык.

Принципы реализации суперпрограмм. Реализация сегментированной программы характеризуется следующими тремя особенностями.

Во-первых, когда рассматриваемая программа подготавливается к реализации, то есть переводится на машинный язык, то эта подготовка не затрагивает подпрограмму-сегмент, которая будет подвергаться соответствующему преобразованию, лишь когда потребуется впервые ее реализовать. Не исключено, что при реализации программы такой потребности не возникнет, и в этом случае упоминаемая в программе подпрограмма-сегмент вообще не будет преобразовываться в машинную форму.

Во-вторых, когда доходит очередь до реализации подпрограммы-сегмента, то производится *сегментное* прерывание реализуемой программы, расположенной в оперативной памяти. Оно заключается во вводе в опера-

тивную память подпрограммы-сегмента и переходе к ее реализации. При первой встрече с данным сегментом он переводится на машинный язык и фиксируется в такой форме в дополнительной памяти, при повторных реализациях используется уже непосредственно эта форма. При каждом переходе к реализации нового сегмента старая машинная программа вытесняется из оперативной памяти и восстанавливается там лишь после полной реализации подпрограммы-сегмента. Восстановление сопровождается переходом к продолжению реализации программы с той точки, в которой ее выполнение было прервано.

В-третьих, сегментирование носит иерархический характер. Подпрограмма-сегмент может иметь свои подчиненные подпрограммы, некоторые из которых могут, в свою очередь, оказаться сегментами. Отношение непосредственного подчинения между различными сегментами, входящими, в конечном счете, в некоторую суперпрограмму, можно представить логическим деревом, подобно тому, как представляется аналогичное отношение между подпрограммами некоторой программы. Полезно ввести понятие *глубины сегментации*, аналогичное понятию глубины компиляции: будем считать, что если некоторый сегмент реализуется на глубине сегментации k, то непосредственно подчиненные ему сегменты будут реализовываться на глубине $k+1$. Особое место занимает *головной сегмент*, реализуемый на глубине 0. Этим сегментом служит, как правило, та программа, которую составляет программист — остальные сегменты лишь используются им, представляя собой результаты предыдущих разработок. Можно считать, что каких-либо ограничений на допустимую глубину сегментации практически не накладывается.

В остальном суперпрограмма реализуется как обычная программа.

Пример суперпрограммы. В качестве такого примера рассмотрим программу крадикоф минимизации булевых функций с большим числом переменных, основные блоки которой описаны в параграфе 3.2. Напомним, что эти блоки позволяют находить кратчайшую днф заданной булевой функции, если только мощность циклического остатка множества простых импликант не

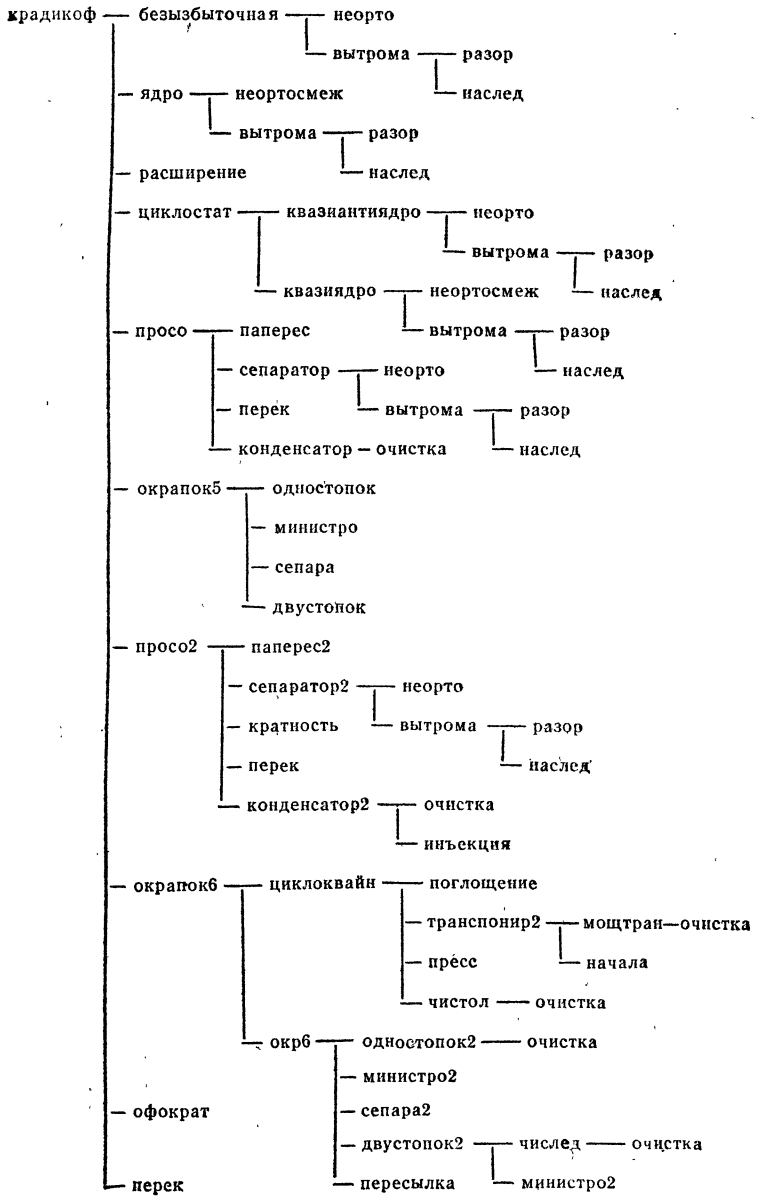
превысит 32. В противном случае подпрограмма просо нахождения простых совокупностей должна быть заменена подпрограммой просо2, решающей эту же задачу в более широком диапазоне. Основной особенностью последней подпрограммы является способ представления находимых простых совокупностей: они представляются уже не в форме булевых векторов, выделяющих эти совокупности из циклического остатка, а в виде списков. Заменяется также подпрограмма окрапок5 нахождения кратчайшего покрытия булевой матрицы. Эта задача решается теперь подпрограммой окрапок6, работающей с иной формой представления булевой матрицы: матрица представляется перечислением тех ее элементов, которые имеют значение 1. Данная подпрограмма позволяет находить кратчайшие покрытия булевых матриц со многими строками и многими столбцами, если общее число единиц в матрице не будет слишком велико.

Сложность программы крадикоф достаточно наглядно иллюстрируется деревом подчинения подпрограмм (которые уже все знакомы нам, за исключением программы офократ, которая занимается оформлением полученных результатов, приводя их к тому же виду, в котором задаются исходные данные), изображенным на стр. 484.

Программа оказывается довольно сложной, в связи с чем она сегментируется. За отдельные сегменты можно принять подпрограммы безызбыточная, ядро, расширение, циклостат, просо, окрапок5, просо2 и окрапок6. Можно заметить, что дополнительные затраты машинного времени на реализацию процесса сегментирования будут невелики, поскольку каждый из перечисленных сегментов реализуется не более одного раза.

Нахождение кратчайшей днф булевой функции — крадикоф

1. Троичной матрицей A задано множество элементарных конъюнкций исходной дизъюнктивной нормальной формы булевой функции. Требуется получить кратчайшую днф этой функции и представить ее аналогичным образом троичной матрицей A^*.


крадикоф
безызбыточная
неорто
вытрома
разор
наслед
ядро
неортосмеж
вытрома
разор
наслед
расширение
наслед
циклостат
квазиантиядро
неорто
вытрома
разор
наслед
квазиядро
неортосмеж
вытрома
разор
наслед
просо
паперес
вытрома
разор
наслед
сепаратор
неорто
перек
вытрома
разор
наслед
конденсатор
очистка
окрапок5
одностопок
министро
сепара
двустопок
просо2
паперес2
сепаратор2
неорто
кратность
вытрома
разор
наслед
перек
конденсатор2
очистка
инъекция
окрапок6
циклоквайн
поглощение
транспонир2
мощтран
очистка
пресс
начала
чистол
очистка
окр6
одностопок2
очистка
министро2
сепара2
двустопок2
числед
очистка
пересылка
министро2
офократ
перек

2. Внешние операнды:

$\alpha\beta_к^-$ :: $\boldsymbol{A}$ в коде (10, 01, 00),

$\alpha\beta_к^+$ :: $\boldsymbol{A}^*$ в коде (10, 01, 00),

$\gamma_к$ — комплекс памяти (включающий α и β),

$\delta_п$ :: $\Psi(B)$, где B — множество столбцов матриц $\boldsymbol{A}$ и $\boldsymbol{A}^*$,

$\tau^+ = 0$, если задача решена,

$\tau^+ = f_{10}$, если задача не решена из-за нехватки памяти.

Задаются: a_α, a_β, a_γ, b_α, b_β, b_γ, причем $[a_\alpha] < [a_\beta]$, $[b_\gamma] > 2[b_\alpha] + 128$.

Сегменты: безызбыточная, ядро, расширение, циклостат, просо, окрапок5, просо2, окрапок6.

Прочие подпрограммы: перек, офократ.

Внутренние операнды: $\boldsymbol{n}$, q, $\boldsymbol{R}$.

4.

* **001 100**

§ 0 $a_\gamma \Rightarrow c$

* перек αc // $b_\gamma - 200 > 1 \Rightarrow q + c$

$\Rightarrow c + q \Rightarrow a_\gamma$

* перек βc //

* * безызбыточная $\alpha\beta\gamma\delta q$ // $\longmapsto 3$

* * ядро $\alpha\beta n\delta q\gamma$ // $\longmapsto 3 b_\alpha \oplus n \circ\!\!\to 3n + 1 \oplus$

$\oplus b_\alpha \circ\!\!\to 3$

* * расширение $\alpha\beta q$ // $\longmapsto 3$

* * циклостат $\alpha\beta\gamma\delta n q$ // $\longmapsto 3$

$b_\alpha - n \Rightarrow p \circ\!\!\to 3 b_\alpha \times 2 \Rightarrow a + a_\alpha \Rightarrow c + a \Rightarrow a_{22}$

$q - a \times 2 \Rightarrow b_{22}$

* перек βc // $\circ \boldsymbol{n} \circ b_{23} p - 41 \longmapsto 1$

* * просо $\alpha\beta\gamma\delta \boldsymbol{T} n$ // $\longmapsto 3 a_\gamma + 40 \Rightarrow a_{30}$

* * окрапок5 $\boldsymbol{T}\gamma \boldsymbol{Z} \boldsymbol{n}$ // $\to 2$

§ 1 $b_{22} > 3 \Rightarrow b_{21} \times 7 \Rightarrow b_{22} + a_{22} \Rightarrow a_{21}$

* * просо2 $\alpha\beta\gamma\delta \boldsymbol{T} n \boldsymbol{S}$ // $\longmapsto 3 a_{22} + b_{22} \Rightarrow \boldsymbol{c}$

* перек $\boldsymbol{S} c$ // $a_{21} + b_{21} \Rightarrow a_{20}$

* * окрапок6 $\boldsymbol{T}\boldsymbol{S}\boldsymbol{R} p$ // $\longmapsto 3$

§ 2 * офократ $\alpha\beta n \boldsymbol{R} \boldsymbol{n}$ // 0

§ 3 .

О локальной эффективности алгоритмов. Реализация больших программ связана, как правило, с большими затратами времени, в связи с чем особое значение приобретает оптимизация этих программ по быстродействию. Одним из основных путей такой оптимизации является разбиение области решаемых задач на некоторые участки, для каждого из которых выбирается подходящий алгоритм, оказывающийся здесь лучше других. Будем говорить, что этот алгоритм *локально эффективен*.

Например, мы познакомились с рядом алгоритмов минимизации булевых функций в классе днф. Все они решают в конечном счете одну и ту же задачу, то есть являются *конкурирующими алгоритмами*. Однако один алгоритм оказывается наиболее эффективным при рассмотрении произвольных булевых функций с малым числом переменных, преимущества другого проявляются при минимизации слабо определенных булевых функций, третий рекомендуется применять в тех ситуациях, когда мы имеем дело с дизъюнктивной нормальной формой, содержащей небольшое число конъюнкций, хотя число переменных может исчисляться десятками.

Области предпочтительного использования этих алгоритмов не имеют четких границ, тем не менее они обладают некоторой спецификой, учитываемой соответствующими алгоритмами. Если же мы построим некий универсальный алгоритм того же порядка сложности, что и упомянутые специализированные алгоритмы, то он не сможет в той же мере учитывать особенности обрабатываемых булевых функций, в связи с чем его реализация будет сопряжена с некоторыми излишними потерями времени — следствием универсальности алгоритма.

Конечно, универсальный алгоритм можно построить как конгломерат специализированных, локально эффективных алгоритмов, однако такой алгоритм может оказаться слишком громоздким. Представляющая такой алгоритм программа потребует большого объема памяти, в то же время при решении каждой конкретной задачи в ней будет использована лишь некоторая, относительно небольшая часть.

Метапрограммы. Принятая методика реализации суперпрограмм позволяет обойти указанные затруднения. Действительно, трансляция сегментов в режиме интерпретации производится непосредственно перед их реализацией, что избавляет нас от необходимости загружать память полной машинной программой решения задачи. Здесь открываются возможности, наилучшим образом реализуемые в метапрограммах.

Метапрограммами мы будем называть суперпрограммы, в которых некоторые совокупности конкурирующих алгоритмов, оформленных в виде сегментов, сочетаются с правилами выбора из этих совокупностей.

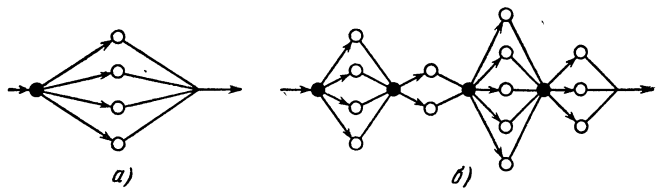


Рис. 4.6.2.

В простейшем случае входящие в метапрограмму конкурирующие алгоритмы предназначены для решения одной и той же задачи, и правила выбора, также оформленные в виде некоторой программы-сегмента, выбирают один из них, который и решает задачу целиком, причем в рассматриваемой конкретной ситуации делает это лучше других конкурирующих алгоритмов. Структура метапрограммы такого типа показана на рис. 4.6.2, *а*, где затемненным кружком представлена программа выбора. Более сложная структура метапрограммы получается при разбиении процесса решения на этапы, каждому из которых ставится в соответствие своя совокупность конкурирующих алгоритмов. Выбор эффективного алгоритма для реализации очередного этапа производится здесь на основании анализа результатов прохождения предшествующего этапа. Пример метапрограммы такого типа показан на рис. 4.6.2, *б*. Наконец, расчленение решающего процесса на этапы также может быть поставлено в зависимость от результатов, полученных на уже реализованных этапах.

Необходимой предпосылкой широкого применения принципов метапрограммирования является достаточно высокий уровень развития эльтеки подпрограмм, когда в ней накапливаются совокупности конкурирующих алгоритмов по многим задачам. Поскольку эти алгоритмы уже представлены в системе, то использование их в некоторой новой программе делает последнюю весьма компактной.

О методике построения метапрограмм. При построении метапрограмм приходится решать ряд специфичных задач. Некоторые из них мы сейчас рассмотрим.

Важнейшей задачей является построение рациональных правил выбора среди конкурирующих алгоритмов. Прежде всего, требуется найти области предпочтительного применения этих алгоритмов. Для этого необходимо получить численные оценки их эффективности как функции некоторых определяющих параметров, то есть величин, от которых существенно зависят такие характеристики алгоритмов, как быстродействие, требования к памяти, точность получаемых результатов (для приближенных алгоритмов). Например, при рассмотрении задачи минимизации булевых функций такими параметрами служат число переменных, число элементарных конъюнкций в днф и т. д. Выбор алгоритма может определяться не только этими параметрами, но также требованиями к качеству результата: в некоторых случаях необходимо точное решение задачи, в других — достаточно грубое приближение. Естественно, что эти требования подчиняются оптимизации в целом того решающего процесса, в котором выбираемый алгоритм реализует лишь некоторую часть. Впрочем, мы уже касались этого вопроса в параграфе 3.4, посвященном экспериментальному определению эффективности алгоритмов.

Практика работы с метапрограммами показывает, что не следует забывать и о решении другой задачи: организации информирования со стороны машины о ходе реализации метапрограммы. Приходится помнить о том, что надежность существующих вычислительных машин оставляет желать лучшего и что при реализации любого этапа возможны сбои. Иногда приходится учитывать также возможность ошибок в самом алгоритме,

которые могут остаться там незамеченными при отладке и проявиться в некоторой особой ситуации.

Поэтому полезно иметь возможность проследить за ходом решения задачи, в особенности если задача решается достаточно долго, а программа имеет сложную структуру. Особый интерес представляет траектория реализации метапрограммы (реализуемая последовательность выбираемых сегментов) и результаты реализации сегментов. Эта информация может систематически выводиться на печать с соответствующими комментариями. С конкретным примером такого информирования мы познакомимся в следующем параграфе.

§ 7. О программе синтеза асинхронных автоматов

Примером суперпрограммы может служить программа полного, или, как говорят, сквозного синтеза автоматов определенного класса, которая начинает свою работу с анализа исходных условий функционирования автомата и заканчивает ее построением таблицы соединений элементов заданного базиса. Такая программа состоит из серии сегментов, решающих задачи отдельных этапов процесса синтеза. В данном параграфе мы рассмотрим в общих чертах суперпрограмму такого характера, разработанную в Сибирском физико-техническом институте.

Постановка задачи. Рассматриваемые здесь автоматы являются асинхронными, в связи с чем их синтез должен производиться так, чтобы не возникали опасные состязания между элементами памяти. Число входных и выходных полюсов автомата может достигать нескольких десятков, а число внутренних состояний — нескольких сотен. Исходная информация, задающая требования к поведению автомата, представляется в виде матрицы переходов, уже знакомой нам по параграфу 4.1, а также в виде трех матриц, описывающих выходные полюсы автомата.

В общем случае при n входных полюсах число входных состояний автомата может достигать значения 2^n, что уже при $n=10$ делает весьма затруднительным составление матрицы переходов. Однако рассматриваемые нами автоматы обладают одной особенностью, сильно

облегчающей решаемую задачу: считается, что никакие два различных входных полюса не могут одновременно находиться в состоянии 1 (будем говорить, что данные автоматы обладают *ортогональными входными полюсами*). В этом случае допустимые входные состояния будут представляться уже не произвольными n-компонентными булевыми векторами, а только такими из них, в которых значение 1 принимает не более чем одна компонента. Очевидно, что их число равно $n+1$. Столбцы используемых в данном случае матриц переходов ставятся в соответствие именно этим допустимым входным состояниям, поэтому матрицы оказываются достаточно компактными.

Выходные полюсы автомата ради удобства разбиваются на два класса. К классу A относятся такие из них, состояние которых зависит как от внутреннего, так и от входного состояния, классу B принадлежат такие выходные полюсы, состояние которых зависит лишь от внутреннего состояния автомата.

Для представления полюсов типа A договоримся использовать две матрицы: одна из них будет задавать некоторый условный номер комбинации состояний этих полюсов как функцию входного и внутреннего состояний, другая будет показывать, какие именно комбинации задаются такими номерами. Состояния полюсов типа B удобно представлять непосредственно одной матрицей, определяющей их как функции внутреннего состояния. Примеры таких матриц будут приведены ниже.

Базис синтеза задается инверсными конъюнкторами, обладающими 8 входными полюсами (будем обозначать такие элементы через ш2) или 3 входными полюсами (ш3); выходные полюсы этих элементов могут соединяться не более чем с 4 входными полюсами других элементов.

Естественным требованием к результату синтеза является достижение минимума числа элементов в получаемой сети. Это требование будет удовлетворяться лишь в какой-то степени, поскольку точное решение данной задачи получить весьма трудно.

Определение макроструктуры программы. Будем придерживаться методики синтеза, ставшей

традиционной. Согласно ей сначала изыскиваются возможности сокращения числа внутренних состояний автомата, после чего производится их кодирование, при котором устраняются все опасные состязания между элементами памяти и находятся функции возбуждения последних, а также выходные функции. Эти функции подвергаются совместной минимизации в классе днф, а затем реализуются в заданном базисе. В результате получается таблица соединений элементов, и задача считается решенной.

Известно много методов решения каждой из перечисленных задач, однако лишь некоторые доведены до формы, в которой их можно непосредственно реализовать на вычислительной машине. В программе, ориентированной на синтез автоматов достаточно широкого класса, каждый из этапов целесообразно обеспечить серией конкурирующих алгоритмов, выбор среди которых будет производиться на основе анализа ситуации, возникающей при прохождении предыдущего этапа. Иными словами, программа должна быть метапрограммой.

В предыдущих параграфах мы познакомились уже с достаточно широким ассортиментом различных алгоритмов, используемых при синтезе дискретных автоматов, в связи с чем сочтем возможным не входить здесь в детали программ решения задач перечисленных этапов. Рассмотрим лишь одну из реализаций метапрограммы, соответствующую конкретной задаче. Эта реализация образуется серией алгоритмов (по одному на каждый из этапов), которые, как правило, несколько отличаются от уже знакомых нам по содержанию книги, но строятся подобно им.

Напомним, что при реализации суперпрограммы на машине следует обращать особое внимание на повышение надежности процесса вычислений. Одним из эффективных средств этого является организация соответствующего информирования со стороны машины о ходе реализации программы.

При реализации рассматриваемой суперпрограммы на печать выводится, прежде всего, исходная информация: словесная формулировка задачи, сопровождаемая таблицами, задающими поведение синтезируемого автомата. Затем кратко приводятся основные результаты,

получаемые при прохождении различных этапов: сообщается, какой именно алгоритм решает задачу этапа и что при этом получается. В заключение выдается окончательный результат: таблица соединений логических элементов.

Предусмотрена блокировка выдачи промежуточной информации.

Пример реализации программы. Приведем пример информации, выдаваемой на АЦПУ (алфавитно-цифровом печатающем устройстве) при реализации программы на машине М-220:

Синтез асинхронного автомата

Постановка задачи: синтезировать асинхронный автомат, свободный от опасных состязаний между элементами памяти и заданный следующими таблицами, в которых

S — множество входных состояний, реализуемых при возбуждении соответствующих входных полюсов автомата,

X — множество внутренних состояний,

A и B — множества выходных полюсов,

Y — множество выходных состояний для A-полюсов:

переходы

S \ X	1	2	3
1 :	2	1	1
2 :	2	3	2
3 :	2	3	4
4 :	2	5	4
5 :	2	5	6
6 :	2	1	6

выходные состояния из Y

S \ X	1	2	3
1 :	2	2	2
2 :	2	2	2
3 :	2	2	1
4 :	2	2	1
5 :	2	2	2
6 :	2	2	2

возбуждение A-полюсов

Y \ A	1
1	1
2	0

возбуждение B-полюсов

Y \ B	1	2	3	4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	1	0
5	0	0	0	1
6	0	0	0	1

Задача решается программой автомат, разработанной на базе системы ЛЯПАС в проблемной лаборатории счетно-решающих устройств Сибирского физико-технического института при Томском университете (1968 г.).

Процесс решения распадается на следующие этапы.

Минимизация числа внутренних состояний. Реализуется метод Ауфенкампа (программа Поттосина). Однако в данном случае число состояний не уменьшается.

Кодирование внутренних состояний. При кодировании устраняются опасные состязания между элементами памяти и минимизируется длина кодов, причем все переходы, перечисленные в исходной таблице, рассматриваются как прямые. Реализуется ускоренный циклический алгоритм с пошаговой оптимизацией, основанный на рассмотрении пар переходов (программа Янковской).

В результате получается следующая таблица кодирования:

S \ Z	1	2	3
1	0	0	0
2	1	0	0
3	1	0	1
4	1	1	1
5	1	1	0
6	0	1	0

Получение функций возбуждения триггеров. Находится система булевых функций, в общем случае неполностью определенных,

описывающих возбуждение триггеров с раздельными входами (программа Ротко).

Функции представляются в форме разложения по входным переменным:

		т1			2		3	
	Z 1 2 3	л	п	л	п	л	п	
X								
1 :	— 0 0	0	1	—	0	—	0	
:	1 0 —	0	—	—	0	1	0	
:	1 — —	0	—	1	0	1	0	
:	1 — 0	0	—	1	0	—	0	
:	— — 0	0	1	1	0	—	0	
2 :	1 0 —	0	—	—	0	0	1	
:	1 1 —	0	—	0	—	1	0	
:	0 — 0	—	0	1	0	—	0	
3 :	0 0 0	—	0	—	0	—	0	
:	1 0 0	0	—	—	0	—	0	
:	1 — 1	0	—	0	1	0	—	
:	— 1 0	1	0	0	—	—	0	

Минимизация системы функций возбуждения триггеров. Реализуется метод параллельного формирования обобщенных интервалов (программа Торопова).

В результате система принимает следующий вид:

		т1		2		3	
	Z 1 2 3	л	п	л	п	л	п
X							
1 :	— — —	0	1	1	0	1	0
2 :	1 0 —	0	0	0	0	0	1
:	— 1 —	0	0	0	0	1	0
:	0 — —	0	0	1	0	0	0
3 :	— — 1	0	0	0	1	0	0
:	— 1 0	1	0	0	0	0	0

Получение выходных функций автомата (программа Ротко).

Эти функции также представляются в форме разложения по входным переменным:

X	Z 1 2 3	A 1
1	— 0 0	0
	1 0 —	0
	1 — —	0
	1 — 0	0
	— — 0	0
2	1 0 —	0
	1 1 —	0
	0 — 0	0
3	0 0 0	0
	1 0 0	0
	1 — 1	1
	— 1 0	0

Z 1 2 3	B 1 2 3 4
0 0 0	1 0 0 0
1 0 0	0 1 0 0
1 0 1	0 0 1 0
1 1 1	0 0 1 0
1 1 0	0 0 0 1
0 1 0	0 0 0 1

Минимизация системы выходных функций автомата. Реализуется метод параллельного формирования обобщенных интервалов (программа Торопова).

В результате система принимает следующий вид:

X	Z 1 2 3	A 1
1		
2		
3	— — 1	1

Z 1 2 3	B 1 2 3 4
0 0 —	1 0 0 0
1 0 0	0 1 0 0
— — 1	0 0 1 0
— 1 0	0 0 0 1

Получение структуры автомата (программа Ротко). Минимизированная система булевых функций реализуется в базисе логических элементов ш 2 (то есть «НЕ И» на 8 входов) и ш 3 (то есть «НЕ И»

на 3 входа) с нагрузочным коэффициентом 4. Полученная структура описывается следующей таблицей межэлементных соединений:

Выходы	Элементы		Входные связи
	номер	тип	
	1	ш 3	x 1
	2	ш 3	15 17 x 2
	3	ш 3	20 x 2
	4	ш 3	14 x 2
	5	ш 3	21 x 3
	6	ш 3	16 20 x 3
	7	ш 3	21 x 3
	10	ш 3	14 15
	11	ш 3	15 16 17
	12	ш 3	21
	13	ш 3	16 20
	14	ш 3	6 17
	15	ш 3	1 4 20
	16	ш 3	1 3 21
	17	ш 3	1 14
	20	ш 3	5 15
	21	ш 3	2 16
A 1	22	ш 3	7
B 1	23	ш 3	10
B 2	24	ш 3	11
B 3	25	ш 3	12
B 4	26	ш 3	13

Интерпретация результатов. Приведем некоторые пояснения к данному примеру.

На этапе получения функций возбуждения триггеров и выходных функций используется свойство ортогональности входных полюсов автомата, позволяющее легко разложить эти функции по входным переменным и тем самым значительно облегчить решение последующей задачи минимизации булевых функций: минимизации будут подвергаться лишь коэффициенты данного разложения, то есть некоторые булевы функции, зависящие только от внутренних переменных.

Получаемое разложение наглядно представляется троичными матрицами, выводимыми на печать при реализации программы. В этих матрицах отражается и существующая неопределенность некоторых значений функций. В рассмотренном примере таким образом за-

дано шесть функций возбуждения триггеров, каждая из которых представлена соответствующим столбцом. Например, функция $p_{2л}$ возбуждения левого входного полюса триггера 2 определяется следующим образом: она должна имплицироваться конъюнкциями x_1z_1, $x_1z_1\bar{z}_3$, $x_1\bar{z}_3$ и $x_2z_1\bar{z}_3$ и она должна быть ортогональна конъюнкциям $x_2z_1z_2$, $x_3z_1z_3$ и $x_3z_2\bar{z}_3$.

Следует обратить внимание на то обстоятельство, что триггеры как таковые в явном виде отсутствуют и фигурируют на промежуточных этапах синтеза лишь условно. В получаемой структуре им соответствуют взаимно связанные друг с другом пары элементов заданного базиса, на которые возлагаются одновременно и другие обязанности, а именно — реализация функций возбуждения.

СПИСОК ПОДПРОГРАММ

Имя	Номер *)	Краткое описание	Стр.
анапер	402	Анализ интервала на пересечение с подмножеством из *M*.	311
анасовм	620	Анализ подмножества на внутреннюю совместимость.	111
ансиграс	015	Анализ симметрического графа на связность.	374
ансиграс два	016	Анализ симметрического графа на двусвязность.	377
ассимил	306	Расширение интервала и ассимиляция покрываемых элементов.	334
бедиэкс 1	201	Нахождение минимального безусловного теста по диагностической матрице.	194
бедиэкс 2	202	То же.	194
бедиэкс 3	353	То же.	194
безызбыточная	700	Приведение троичной матрицы к эквивалентной безызбыточной форме.	284
вклюмак	400	Включение в комплекс максимального элемента.	110, 129
выбор	307	Выбор пары интервал — элемент.	333
вытрома	365	Анализ троичной матрицы на вырожденность.	234
двупокр 2	124	Нахождение двустрочного покрытия булевой матрицы.	156
двустопок	175	Нахождение двустолбцового покрытия минора булевой матрицы.	185
диноф	411	Получение д н ф булевой функции по заданному покрытию.	315
квазиантиядро	711	Удаление элементов квазиантиядра.	294
квазиядро	722	Расширение квазиядра.	295
квасобу	404	Получение матрицы Квайна для слабо определенной булевой функции.	311

*) Номер каждой подпрограммы состоит из двух трехразрядных восьмеричных чисел, однако в списке приводится лишь второе из них, поскольку первое число оказывается общим для всех подпрограмм данного списка (и равно 001).

Продолжение

Имя	Номер	Краткое описание	Стр.
конденсатор	701	Построение матрицы $U(k+1)$ из $U(k)$.	303
крадикоф	100	Нахождение кратчайшей д н ф булевой функции.	483
крафсо	410	Получение кратчайшей д н ф слабо определенной булевой функции.	316
крацеп	114	Получение всех кратчайших цепей в подграфе.	380
локрацеп	115	Локализация кратчайших цепей.	378
максопод	621	Нахождение всех максимальных совместимых подмножеств.	113
макстро	003	Нахождение максимальной строки минора.	87, 128
макстрок	127	Нахождение максимальной среди отмеченных строк булевой матрицы.	147
манесок	372	Получение матрицы несовместимости классов.	209
матрамин	137	Минимизация матрицы разбиений.	266
министро	173	Нахождение минимальной строки строчного минора булевой матрицы.	184
минрасхэм	354	Нахождение пары ближних строк.	193
минсоб	303	Минимизация слабо определенной булевой функции.	329
минсто	002	Нахождение минимального столбца минора.	86, 89, 128
минстол	126	Нахождение минимального столбца булевой матрицы.	146
мипин	401	Получение минимального поглощающего интервала.	310
наслед	367	Нахождение следствий по правилам поиска ортогонального вектора.	236
неорто	703	Построение минора из неортогональных строк.	285
неортосмеж	705	Построение минора из неортогональных и смежных строк.	288
новички	305	Поиск новых элементов частичного решения.	331

Продолжение

Имя	Номер	Краткое описание	Стр.
овпод	373	Объединение в подмножествах.	210
одностопок	172	Нахождение одностолбцовых покрытий минора булевой матрицы.	185
окрапок4	130	Нахождение одного кратчайшего покрытия булевой матрицы.	157
окрапок5	176	Нахождение кратчайшего столбцового покрытия булевой матрицы.	186
оцесто	036	Вычисление оценки столбца секционированной матрицы.	192
очистка	061	Очистка комплекса.	331
паперес	702	Нахождение пар пересекающихся строк.	301
перебор	403	Перебор элементов выпуклого множества.	112, 130
перек	704	Перепись комплекса.	304
подграмакс	320	Нахождение максимальных полных подграфов симметрического графа.	386
поклас	370	Получение классов.	208
пораскоб	300	Решение системы логических уравнений путем последовательного раскрытия скобок в перемножаемых д н ф.	365
прикрапок	010	Нахождение приближения к кратчайшему покрытию булевой матрицы.	105, 125
прикрапок1	407	То же.	148
прикрафсо	415	Получение приближения к кратчайшей д н ф слабо определенной булевой функции.	317
проскал	051	Получение скалярного произведения двух комплексов.	97, 130
просо	713	Получение простых совокупностей.	300
профакт	105	Подсчет произведения факториалов.	126
прямор	210	Нахождение разбиения на совместимые подмножества методом прямого размещения.	172

Продолжение

Имя	Номер	Краткое описание	Стр.
радифс	374	Нахождение разбиения, удовлетворяющего дифференцированному отношению совместимости.	211
разбимат	032	Разбиение строчных миноров секционированной матрицы.	193
разор	366	Разделение строк троичной матрицы по отношению ортогональности к заданному вектору.	235
расопод	622	Нахождение минимального разбиения на совместимые подмножества.	116
расопод1	211	То же.	162
расширение	533	Расширение троичной матрицы.	290
сепара	174	Разделение (сепарация) строк минора булевой матрицы по заданному подмножеству столбцов.	184
сепаратор	712	Выделение элементов с заданным свойством.	302
симбинот	371	Симметрирование бинарного отношения.	209
слурин	122	Случайное расширение интервала.	315
сокрамат	125	Сокрашение булевой матрицы	147
сомакс	310	Нахождение максимальных множеств совместимости.	385
удамос	034	Удаление строчных миноров, состоящих из одинаковых строк.	191
устиз	364	Устранение избыточности в троичной матрице.	239
хорд	207	Поиск и реализация хороших шагов по правилу 2.	172
хоро	206	Поиск и реализация хороших шагов по правилу 1.	171
циклостат	773	Нахождение циклического остатка в множестве простых импликант.	293
экспансия	117	Нахождение множества достижимых вершин ориентированного графа.	378
элер	304	Формирование элементов полного решения.	332
ядро	706	Нахождение ядра безызбыточных форм.	287

ЛИТЕРАТУРА

Глава первая

1. АЛЬФА — система автоматизации программирования. Сб. статей под ред. А. П. Ершова, РИО СО АН СССР, Новосибирск, 1965.

2. Жоголев Е. А., Трифонов Н. П., Курс программирования. Изд-во «Наука», М., 1964.

3. Закревский А. Д., Алгоритмический язык ЛЯПАС и автоматизация синтеза дискретных автоматов. Изд-во Томского университета, Томск, 1966.

4. Карр Дж., Лекции по программированию. ИЛ, М., 1963.

5. Китов А. И., Программирование информационно-логических задач. Изд-во «Советское радио», М., 1967.

6. Ледли Р., Программирование и использование цифровых вычислительных машин. Изд-во «Мир», М., 1966.

7. Логический язык для представления алгоритмов синтеза релейных устройств. Изд-во «Наука», М., 1966.

8. Проектирование сверхбыстродействующих систем. Комплекс СТРЕТЧ, под ред. В. Бухгольца. Изд-во «Мир», М., 1965.

9. Современное программирование. Сб. статей. Изд-во «Советское радио», М., 1966.

10. Ляшенко В. Ф., Программирование для цифровых вычислительных машин М-20, БЭСМ-3М, БЭСМ-4, М-220. Изд-во «Советское радио», М., 1967.

11. Труды Сибирского физико-технического института, вып. 48. Автоматизация синтеза дискретных автоматов. Изд-во Томского университета, Томск, 1966.

12. Хомский Н., Три модели описания языка. Кибернетический сборник, № 2, 237—266, ИЛ, М., 1961.

13. Iverson K. E., A programming language. New York, Wiley, 1962.

14. Symbolic languages in data processing. Proceedings of the Symposium in Rome, March 26th—31st, 1962, New York — London, Gordon and Breach, 1962.

Глава вторая

15. Закревский А. Д., К минимизации дизъюнктивных нормальных форм булевых функций. Техническая кибернетика, № 4, 102—104, 1970.

16. Закревский А. Д., Островский В. И., Оптимизация поиска кратчайшего покрытия. Сб. «Вопросы синтеза цифровых автоматов», 84—95, Изд-во «Наука», М., 1966.

17. Карибский В. В., Пархоменко П. П., Согомонян Е. С., Техническая диагностика объектов контроля. Изд-во «Энергия», М., 1967.

18. Чегис И. А., Яблонский С. В., Логические способы контроля работы электронных схем. Труды Математического института им. Стеклова, том 51, 270—360, Изд-во АН СССР, М., 1958.

19. McCluskey E. J., Minimization of Boolean functions. Bell System Tech. J., vol. 35, № 6, 1417—1444, 1956.

20. Pyne J. B., McCluskey E. J., The reduction of redundancy in solving prime implicant tables. IRE Trans., vol. EC-11, № 4, 473—482, 1962.

21. Quine W. V., The problem of simplifying of truth functions. Am. Math. Monthly, vol. 59, № 8, 521—531, 1952.

22. Quine W. V., On cores and prime implicants of truth functions. Am. Math. Monthly, vol. 66, № 9, 755—760, 1959.

Глава третья

23. Берж К., Теория графов и ее применения. ИЛ, М., 1962.

24. Гильберт Д., Аккерман В., Основы теоретической логики. ИЛ, М., 1947.

25. Журавлев Ю. И., Теоретико-множественные методы в алгебре логики. Проблемы кибернетики, вып. 8, 5—44, 1962.

26. Закревский А. Д., Алгоритмы минимизации слабо определенных булевых функций. Кибернетика, № 2, 53—60, 1965.

27. Закревский А. Д., Новые алгоритмы минимизации слабо определенных булевых функций. Кибернетика, № 5, 21—28, 1969.

28. Закревский А. Д., О приближенных методах решения логических задач. Сб. «Вопросы синтеза цифровых автоматов», 5—13, Изд-во «Наука», М., 1966.

29. Закревский А. Д., К решению систем логических уравнений. Сб. «Принципы построения сетей и систем управления», 48—55, Изд-во «Наука», М., 1964.

30. Яблонский С. В., Функциональные построения в k-значной логике. Труды Математического института им. Стеклова, том 51, 5—142, Изд-во АН СССР, М., 1958.

31. Cherry C., Vaswani P. K. T., A new type of computer for problems in propositional logic, with greatly reduced scanning procedures. Information and Control, vol. 4, № 2—3, 155—168, 1961.

Глава четвертая

32. Айзерман М. А., Гусев Л. А., Розоноэр Л. И., Смирнова И. М., Таль А. А., Логика. Автоматы. Алгоритмы. Физматгиз, М., 1963.

33. Гаврилов М. А., Теория релейно-контактных схем. Изд-во АН СССР, М.—Л., 1950.

34. Гилл А., Введение в теорию конечных автоматов. Изд-во «Наука», М., 1966.

35. Глушков В. М., Синтез цифровых автоматов. Физматгиз, М., 1962.

36. Закревский А. Д., Янковская А. Е., Практические алгоритмы кодирования внутренних состояний асинхронных автоматов. Автоматика и вычислительная техника, № 3, 15—21, 1969.

37. Колдуэлл С., Логический синтез релейных устройств. ИЛ, М., 1962.

38. Лазарев В. Г., Пийль Е. И., Синтез асинхронных конечных автоматов. Изд-во «Наука», М., 1964.

39. Мур Э. Ф., Умозрительные эксперименты с последовательностными машинами. Сб. «Автоматы», ИЛ, М., 1956.

40. Поспелов Д. А., Логические методы анализа и синтеза схем. Изд-во «Энергия», М.—Л., 1964.

41. Поттосин Ю. В., Сравнительная оценка двух алгоритмов минимизации числа состояний дискретного автомата. Автоматика и вычислительная техника, № 4, 22—28, 1967.

42. Рогинский В. Н., Построение релейных схем управления. Изд-во «Энергия», М.—Л., 1964.

43. Якубайтис Э. А., Асинхронные логические автоматы. Изд-во «Зинатне», Рига, 1966.

44. Aufenkamp D. D., Hohn F. E., Analysis of sequential machines. IRE Trans., vol. EC-6, № 4, 276—285, 1957.

45. Aufenkamp D. D., Analysis of sequential machines. — II. IRE Trans., vol. EC-7, № 4, 299—306, 1958.

46. Ginsburg S., A technique for the reduction of a given machine to a minimal-state machine. IRE Trans., vol. EC-8, № 3, 346—355, 1959.

47. Huffman D. A., The synthesis of sequential switching circuits. J. Franklin Inst., vol. 257, № 3, 161—190, № 4, 275—303, 1954.

48. Liu C. N., A state variable assignment method for asynchronous sequential switching circuits. J. Assoc. Comput. Mach., vol. 10, № 2, 209—216, 1963.

49. Mealy G. H., A method for synthesizing sequential circuits. Bell System Tech. J., vol. 34, № 5, 1045—1079, 1955.

50. Paull M. C., Unger S. H., Minimizing the number of states in incompletely specified sequential switching functions. IRE Trans., vol. EC-8, № 3, 356—367, 1959.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Автомат автономный 401
— асинхронный 407
— дискретный 389
— комбинационный (без памяти) 391
— полный 403
— последовательностный (с памятью) 397
— синхронный 398
— частичный 403
Алгоритм конкурирующий 336, 486
— приближенный 83, 334
— точный 334
Анализ автомата 431
Антиядро безызбыточных форм троичной матрицы 256
— множества простых импликант 292

База разложения задачи 103
Базис производных д н ф 462
— синтеза 431
— — полный 431
— элементный 431
Близость булевых функций 278
Блок разбиения 256

Вектор булев 19
— троичный m-го ранга 224
Вершина графа 64
— — достижимая 377
— — смежная 103, 372
Вложение группировки 424
Время дискретное 398
Вход автомата 392
Выход автомата 392

Глубина реализации программы 59
— сегментирования 482

Граница множества верхняя 307
Граф булев полный 373, 455
— несовместимости 101
— ориентированный 64
— переходов 402
— поведения автомата 402
— полный 373
— связный 372
— симметрический 101
— условий совместимости 415
— — эквивалентности состояний 411
Группа (состояний) производная 418
— — непосредственная 418
— совместимости 415
— — максимальная 415
Группировка квазиправильная 421
— парная 422
— — полная 422
— правильная 420

Датчик 392
Дерево поиска 152
Диагноз 177
Диагностика 180
Дизъюнктор 432
— инверсный 473
Дуга графа 64

Импликанта булевой функции 278
— — — простая 279, 292
— матрицы разбиений главная 261
— строк троичной матрицы общая 260
Имя программы 85
Интервал булева пространства 216

Интервал булева пространства максимальный 221
— — — минимальный поглощающий 308
Итерация 152

Качество решения 335
Квазиантиядро кратчайшей д н ф 292
Квазиядро кратчайшей д н ф 292
— — формы троичной матрицы 256
Класс эквивалентности 409
K-множество 450
Код Грея 48
Кодирование внутренних состояний 446
— — — соседнее 455
Коды символов языка ЛЯПАС 141
Компилятор 92
Компиляция 92
Контур графа 64
Конъюнктор 432
— инверсный 473
Конъюнкция элементарная 272
— — полная 273
— элементарных событий 393
Корень системы логических уравнений 358
Коэффициент размерности булева вектора 76
— связи состояний 458

Л-блок 481

Матрица булева 21
— — α-каноническая 199
— — условно β-каноническая 204
— выходов 400
— диагностическая 177
— Квайна 311
— — простая 299
— кодирующая 446
— переходов 400
— поглощений простая 240
— разбиений 259
— различий 180
— связи 458
— секционированная 191
— смежности 103
Матрица троичная 204
— — безызбыточная 221
— — вырожденная 230
— — кратчайшая 219
— условий 447
— — минимальная 449
— — элементарных переходов 455
Машина абстрактная 66
М-блок 481
Метапрограмма 487
Метка 33
— конца программы 38
— начала подпрограммы 90
— сегмента 480
Механизм исполнительный 393
Минимизация числа состояний 412
Множество выпуклое по отношению включения 374
— опорное 197
— простых импликант, циклическая часть 292
Модель автомата Мили 398
— — Мура 399
— — синхронного аналитическая функциональная 398
— — структурная 430

Ограничение мощности покрытия 153
Операнды 33
— подпрограммы внешние 88
— — внутренние 88
— языка ЛЯПАС 33
— — —, индексы 34
— — —, — дополнительного набора 93
— — —, комплекс 33
— — —, — внешний 71
— — —, — оперативный (K) 68
— — —, — основной 33
— — —, — секционированный 69
— — —, — специальный индексов (H) 68
— — —, — — мощностей (B) 38, 68
— — —, — — начал (A) 68
— — —, — — переменных (G) 68
— — —, константы 34
— — —, — натуральные 34

Операнды языка ЛЯПАС, константы 124
— — —, — стандартные 35
— — —, переменные 33
— — —, — дополнительного набора 93
— — —, случайная переменная (*я*) 61
— — —, собственная переменная (τ) 39
Оператор 33
— бинарный 276
— языка ЛЯПАС, л-оператор (элементарный) 85
— — —, Л-оператор 85
Операции первого уровня языка ЛЯПАС вычислительные (над булевыми векторами) 38
— — — — — — логические, дизъюнкция (∨) 24
— — — — — — —, дизъюнкция с исключением (⊕) 25
— — — — — — —, инверсия (¬) 23
— — — — — — —, конъюнкция (∧) 24
— — — — — — модульные арифметические, вычитание (—) 27
— — — — — — — —, деление (:) 29, 39
— — — — — — — —, перемножение (×) 28
— — — — — — — —, — с округлением (⊼) 28
— — — — — — — —, сложение (+) 26
— — — — — — сдвига, неограниченный сдвиг (≷) 134
— — — — — — —, нормализация (←) 30
— — — — — — —, сдвиг влево (<) 30
— — — — — — —, — вправо (>) 30
— — — — — — характеристические, взвешивание (∇) 29
— — — — — — —, нахождение номера левой единицы (⊢) 29
— — — — — передачи информации, ввод (*ввод*) 73
Операции первого уровня языка ЛЯПАС передачи информации, ввод без сдвига (*ввод′*) 136
— — — — — — —, вывод на АЦПУ (*ацпу*) 138
— — — — — — —, десятичная печать ($печ_{10}$) 136
— — — — — — —, перфорация (*перф*) 74
— — — — — — —, — без сдвига (*перф′*) 136
— — — — — — —, печать (*печ*) 74
— — — — — — —, — без сдвига (*печ′*) 136
— — — — — — —, посылка во внешний комплекс (*зап*) 72
— — — — — — —, — из внешнего комплекса (*счит*) 72
— — — — — перехода, безусловный переход (→) 52
— — — — — —, переход к машинному языку (*маш*) 139
— — — — — —, условный переход по единице (⊢→) 54
— — — — — —, — — по нулю (o→) 52
— — — — — —, уход с возвращением (⇉,!) 58
— — — — присвоения значения, обмен (⇔) 43
— — — — — — —, отрицательное элементарное приращение (▽) 43
— — — — — — —, положительное элементарное приращение (△) 43
— — — — — — —, присвоение (⇒) 41
— — — — — — —, — максимального значения (⊽) 44
— — — — — — —, — нулевого значения (O) 43
— — — — прочие, контрольное суммирование (*сумма*) 139
— — — — — —, локализация (*локал*) 139
— — — — — —, перебор единиц (Ẋ) 59
— — — — — —, случайный перебор единиц (Ẍ) 61

Операции теоретико-множественные, дополнение 23
— —, объединение 24
— —, пересечение 24
Оптимизация процесса решения 335
Отношение бинарное 20, 277
— больше, меньше между булевыми векторами 215
— импликации между булевыми функциями 276
— — между троичными матрицами 259
— — между условиями 258
— — между явлениями 176
— ортогональности между вектором и матрицей 230
— — между входными полюсами 490
— — между троичными векторами 213
— пересечения между троичными векторами 214
— поглощения между троичными векторами 214
— равенства между булевыми функциями 276
— реализации между автоматами 414, 416
— — между последовательностями 413
— — между разбиениями 258
— — между физической и логической переменными 390
— смежности между троичными векторами 214
— совместимости между последовательностями (входными) 414
— — между состояниями автомата 414
— — между троичными векторами 260
— соответствия между элементарными конъюнкциями и интервалами 273
— соседства между троичными векторами 214
— эквивалентности между автоматами 411
— — между булевыми матрицами 199
Отношение эквивалентности между булевыми матрицами, α-эквивалентность 198
— — — — —, β-эквивалентность 199
— — между разбиениями 259
— — между состояниями автомата 409
— — между троичными векторами 257
— — — — матрицами (α-эквивалентность) 216
Отображение последовательностей 401
Оценка эффективности алгоритма количественная 336

Память 66
— дополнительная 66
— оперативная 66
Пара состояний производная 409
Параметр задачи определяющий 336
— размерности булева вектора 75
Паспорт подпрограммы 124
Перебор комбинаций лексикографический 366
Переменная автомата внутренняя 388
— — входная 388
— — выходная 388
— булева 19
— логическая 390
— физическая 388
Переход между состояниями автомата прямой 444
— — — — элементарный 443
Перечень внешних операндов подпрограммы 86, 98
Подграф полный максимальный 382
— усеченный 492
Подкомплекс 69
Подмножество выпуклое 130
— из M интервально-поглощаемое 307
— совместимое 102
— — максимальное 104
Подпрограмма 82, 85
Покрытие булевой матрицы 82
— — — кратчайшее 82
— — — столбцовое 180

Полюс автомата входной 390
— — выходной 390
— подпрограммы входной 95
— — выходной 95
— — дополнительный 95
— — основной 95
— сети входной 434
— — выходной 434
Последовательность входная 400
— — для абстрактной машины 73
— — для дискретного автомата 400
— выходная 401
— — частичная 404
Предложение программы 51, 63
Представление бинарного отношения булевой матрицей 21
— подмножества булевым вектором 20, 269
— числа булевым вектором 22
Преобразование равносильное троичной матрицы 217
Преобразователь элементарный 392
Прерывание сегментное 481
Признак 177
Причина 177
Программа 33, 38
— линейная 38
—, Л-программа 38
— циклическая 51
Программирование иерархическое 104
Пространство булево (M) 197
Путь в графе 64

Разбиение 258
— множества 102
— — минимальное 102
— — — на совместимые подмножества 102, 119
— полное двублочное 256
— частичное двублочное 256
Различение явлений 178
Разложение булевой матрицы по минимальному столбцу 153
— задачи первичное 104
— троичного вектора по компоненте 218
Размерность булева вектора 19
Размерность стандартная 33
Размещение состояний (на булевом графе) 455
Рандомизация приближенного алгоритма 354
Расширение матрицы 223
Реализация логических переменных 390
— — — однородная 390
— программы 58, 70
Режим (реализации сегмента) заготовки 480
— интерпретации 479
Рефлексивность отношения эквивалентности 217, 409

Связность графа (k-связность) 372, 374
Сегмент 477
— головной 482
Сегментирование 477
Секция элементарная 202
Сепарация строк 183
Сеть 434
— асинхронная 440
— комбинационная 436
— — расходящаяся 436
— — сходящаяся 436
— k-ярусная 436
— линейно-упорядоченная 436
— правильная 435
— синхронная 438
Сжатие троичной матрицы максимальное 219
— — — прямое 243
Симметричность отношения эквивалентности 217, 409
Синтаксис структур автоматов 431
— языка ЛЯПАС 62
Синтез автомата 431
Система булевых функций полная 271
— счисления двоичная позиционная 22
Ситуация критическая 164
Склеивание троичных векторов 217
— — — обобщенное 218
Следствие 177
Событие элементарное 393
Совмещение состояний 415

Совокупность простых импликант простая 298
— строк троичной матрицы
— — — — простая 241
— — — — совместимая 260
— — — — — максимальная 260
— шагов полная 174
Сокращение булевой матрицы 152
Состояние автомата внутреннее 397
— — входное 392
— — — нереализуемое 392
— — выходное 392
— — граничное 403
— — начальное 399
— — полное 399
— — — устойчивое 407
Состязания между элементами памяти 443
— — — — неопасные 443
— — — — опасные 443
Строка матрицы максимальная 110
— — особенная 425
Структура правильная 431
Суперпрограмма 479
— с линейной структурой 479
— с циклической структурой 479

Таблица реализации программы 40
Тело подпрограммы 124
Тест 180
— безусловный диагностический 178
— — — минимальный 180
— условный диагностический 180
Точка ветвления 152
— особая 108
— фазовая 402
Транзитивность отношения импликации 278
— — эквивалентности 217, 409
Триггер 433

Уменьшение числа ярусов 155
Уровень языка ЛЯПАС второй 82
— — — первый 81
— — — третий 479
Условие 257
Условие достаточное 278
— кодирования элементарное 446
— необходимое 278
Устройство ввода информации 73
— вывода информации 74
— задержки 399
— оперативное 66
— релейное 389

Форма булевой функции
— — — алгебраическая 272
— — — векторная 271
— — — дизъюнктивная нормальная (д н ф) 273
— — — — — безызбыточная 276
— — — — — второго порядка 476
— — — — — кратчайшая 276
— — — — — минимальная 276
— — — — — совершенная 274
— — — — — сокращенная 276
— — — конъюнктивная нормальная (к н ф) 273
— — — элементарная 271
— матрицы разбиений имплицирующая 259
— — — — кратчайшая 260
— представления информации внешняя 136
— — — —, перфоформа 72
— — — —, печатная 72
— — — внутренняя 136
— секционированной матрицы циклическая 202
— упоминания Л-оператора косвенная 131
— — — прямая 131
Функция булева 270
— — двойственная 474
— — не полностью определенная 270, 306
— — полностью определенная 270, 306
— — простейшая, дизъюнкция 271
— — —, — с исключением (сумма по модулю два) 271
— — —, импликация 276
— — —, инверсия 271
— — —, конъюнкция 271
— — —, равенство 276

Функция булева симметрическая 438
— — слабо определенная 307
— возбуждения триггера 441
— выходов 398
— несовместимости 101
— переходов 398

Цена решения 335
Цепочка (α-цепочка) 62
— правильная 63
Цепь графа кратчайшая 378

Шаг особый 108
— плохой 164
— сомнительный 164
— хороший 164

Элемент автоматный 430
— задержки 433
— логический 432
Элемент памяти 67, 433
— перечня операндов подпрограммы простой 98
— — — — составной 98
— реальный 430
— ячейки памяти 67
Эльтека 120
— машинная 121
— описаний 121
Эффективность алгоритма 336
— — локальная 486
— базиса синтеза 432

Явление 177
Ядро безызбыточных д н ф 287
— — форм тупиковой матрицы 247
— множества простых импликант 292
Ячейка памяти 67

Аркадий Дмитриевич Закревский

Алгоритмы синтеза дискретных автоматов

(Серия «Теоретические основы технической кибернетики»)

М., 1971 г., 512 стр. с илл.

Редактор *В. В. Воржева*

Техн. редактор *В. Н. Кондакова*

Корректоры *З. В. Автонеева, Н. Б. Румянцева*

Сдано в набор 25/V 1971 г.

Подписано к печати 30/XI 1971 г.

Бумага 84×108$^{1}/_{32}$. Физ. печ. л. 16. Условн. печ. л. 26,88. Уч.-изд. л. 27,43. Тираж 8800 экз. Т-20014.

Цена книги 1 р. 89 к. Заказ № 1121.

Издательство «Наука»
Главная редакция
физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15.

Ордена Трудового Красного Знамени
Ленинградская типография № 2
имени Евгении Соколовой Главполиграфпрома
Комитета по печати при Совете Министров СССР.
Измайловский проспект, 29.