# Содержание

**Предисловие** **14**

Как пользоваться книгой 15
Требования к уровню подготовки 15
Упражнения 16
Поддержка в World Wide Web 16
Благодарности 16

**ГЛАВА 1. Автоматы: методы и понятия** **17**

1.1. Зачем изучается теория автоматов? 18
1.1.1. Введение в теорию конечных автоматов 18
1.1.2. Структурные представления 20
1.1.3. Автоматы и сложность 21
1.2. Введение в теорию формальных доказательств 21
1.2.1. Дедуктивные доказательства 22
1.2.2. Сведение к определениям 25
1.2.3. Другие формы теорем 27
1.2.4. Теоремы без гипотезы 30
1.3. Дополнительные схемы доказательств 30
1.3.1. Доказательства эквивалентностей, связанных с множествами 30
1.3.2. Контрапозиция 32
1.3.3. Доказательство методом “от противного” 34
1.3.4. Контрпримеры 34
1.4. Индуктивные доказательства 36
1.4.1. Индукция по целым числам 36
1.4.2. Более общие формы целочисленных индуктивных доказательств 39
1.4.3. Структурная индукция 40
1.4.4. Совместная индукция 43
1.5. Основные понятия теории автоматов 45
1.5.1. Алфавиты 46
1.5.2. Цепочки 46
1.5.3. Языки 47
1.5.4. Проблемы 48
Резюме 50
Литература 52

**ГЛАВА 2. Конечные автоматы** **53**

2.1. Неформальное знакомство с конечными автоматами 54
2.1.1. Основные правила 54
2.1.2. Протокол 55
2.1.3. Возможность игнорирования автоматом некоторых действий 57

2.1.4. Система в целом как автомат 59
2.1.5. Проверка протокола с помощью автомата-произведения 61
2.2. Детерминированные конечные автоматы 61
2.2.1. Определение детерминированного конечного автомата 62
2.2.2. Как ДКА обрабатывает цепочки 62
2.2.3. Более простые представления ДКА 64
2.2.4. Расширение функции переходов на цепочки 65
2.2.5. Язык ДКА 68
2.2.6. Упражнения к разделу 2.2 69
2.3. Недетерминированные конечные автоматы 71
2.3.1. Неформальное описание недетерминированного конечного автомата 72
2.3.2. Определение недетерминированного конечного автомата 73
2.3.3. Расширенная функция переходов 74
2.3.4. Язык НКА 75
2.3.5. Эквивалентность детерминированных и недетерминированных конечных автоматов 77
2.3.6. Плохой случай для конструкции подмножеств 81
2.3.7. Упражнения к разделу 2.3 83
2.4. Приложение: поиск в тексте 85
2.4.1. Поиск цепочек в тексте 85
2.4.2. Недетерминированные конечные автоматы для поиска в тексте 86
2.4.3. ДКА, распознающий множество ключевых слов 87
2.4.4. Упражнения к разделу 2.4 89
2.5. Конечные автоматы с эпсилон-переходами 89
2.5.1. Использование ε-переходов 89
2.5.2. Формальная запись ε-НКА 91
2.5.3. Что такое ε-замыкание 91
2.5.4. Расширенные переходы и языки ε-НКА 92
2.5.5. Устранение ε-переходов 94
2.5.6. Упражнения к разделу 2.5 97
Резюме 97
Литература 98

**ГЛАВА 3. Регулярные выражения и языки 101**

3.1. Регулярные выражения 101
3.1.1. Операторы регулярных выражений 102
3.1.2. Построение регулярных выражений 104
3.1.3. Приоритеты регулярных операторов 106
3.1.4. Упражнения к разделу 3.1 108
3.2. Конечные автоматы и регулярные выражения 108
3.2.1. От ДКА к регулярным выражениям 109
3.2.2. Преобразование ДКА в регулярное выражение методом исключения состояний 114
3.2.3. Преобразование регулярного выражения в автомат 120
3.2.4. Упражнения к разделу 3.2 124
3.3. Применение регулярных выражений 126

3.3.1. Регулярные выражения в UNIX 126
3.3.2. Лексический анализ 128
3.3.3. Поиск образцов в тексте 130
3.3.4. Упражнения к разделу 3.3 132
3.4. Алгебраические законы для регулярных выражений 132
3.4.1. Ассоциативность и коммутативность 133
3.4.2. Единичные и нулевые элементы 134
3.4.3. Дистрибутивные законы 134
3.4.4. Закон идемпотентности 135
3.4.5. Законы, связанные с оператором итерации 136
3.4.6. Установление законов для регулярных выражений 136
3.4.7. Проверка истинности алгебраических законов для регулярных выражений 139
3.4.8. Упражнения к разделу 3.4 140
Резюме 141
Литература 142

**ГЛАВА 4. Свойства регулярных языков 143**

4.1. Доказательство нерегулярности языков 143
4.1.1. Лемма о накачке для регулярных языков 144
4.1.2. Применение леммы о накачке 145
4.1.3. Упражнения к разделу 4.1 147
4.2. Свойства замкнутости регулярных языков 148
4.2.1. Замкнутость регулярных языков относительно булевых операций 149
4.2.2. Обращение 154
4.2.3. Гомоморфизмы 156
4.2.4. Обратный гомоморфизм 157
4.2.5. Упражнения к разделу 4.2 163
4.3. Свойства разрешимости регулярных языков 166
4.3.1. Преобразования различных представлений языков 167
4.3.2. Проверка пустоты регулярных языков 169
4.3.3. Проверка принадлежности регулярному языку 170
4.3.4. Упражнения к разделу 4.3 171
4.4. Эквивалентность и минимизация автоматов 171
4.4.1. Проверка эквивалентности состояний 172
4.4.2. Проверка эквивалентности регулярных языков 175
4.4.3. Минимизация ДКА 177
4.4.4. Почему минимизированный ДКА невозможно улучшить 180
4.4.5. Упражнения к разделу 4.4 182
Резюме 183
Литература 183

**ГЛАВА 5. Контекстно-свободные грамматики и языки 185**

5.1. Контекстно-свободные грамматики 185
5.1.1. Неформальный пример 185
5.1.2. Определение контекстно-свободных грамматик 187
5.1.3. Порождения с использованием грамматики 189

5.1.4. Левые и правые порождения 191
5.1.5. Язык, задаваемый грамматикой 193
5.1.6. Выводимые цепочки 194
5.1.7. Упражнения к разделу 5.1 195
5.2. Деревья разбора 197
5.2.1. Построение деревьев разбора 197
5.2.2. Крона дерева разбора 199
5.2.3. Вывод, порождение и деревья разбора 200
5.2.4. От выводов к деревьям разбора 201
5.2.5. От деревьев к порождениям 202
5.2.6. От порождений к рекурсивным выводам 205
5.2.7. Упражнения к разделу 5.2 207
5.3. Приложения контекстно-свободных грамматик 207
5.3.1. Синтаксические анализаторы 208
5.3.2. Генератор синтаксических анализаторов YACC 210
5.3.3. Языки описания документов 211
5.3.4. XML и определения типа документа 213
5.3.5. Упражнения к разделу 5.3 219
5.4. Неоднозначность в грамматиках и языках 220
5.4.1. Неоднозначные грамматики 220
5.4.2. Исключение неоднозначности из грамматик 222
5.4.3. Левые порождения как способ выражения неоднозначности 225
5.4.4. Существенная неоднозначность 226
5.4.5. Упражнения к разделу 5.4 228
Резюме 229
Литература 230

**ГЛАВА 6. Автоматы с магазинной памятью 233**

6.1. Определение автоматов с магазинной памятью 233
6.1.1. Неформальное введение 233
6.1.2. Формальное определение автомата с магазинной памятью 235
6.1.3. Графическое представление МП-автоматов 237
6.1.4. Конфигурации МП-автомата 238
6.1.5. Упражнения к разделу 6.1 241
6.2. Языки МП-автоматов 242
6.2.1. Допустимость по заключительному состоянию 242
6.2.2. Допустимость по пустому магазину 244
6.2.3. От пустого магазина к заключительному состоянию 244
6.2.4. От заключительного состояния к пустому магазину 247
6.2.5. Упражнения к разделу 6.2 249
6.3. Эквивалентность МП-автоматов и КС-грамматик 251
6.3.1. От грамматик к МП-автоматам 251
6.3.2. От МП-автоматов к грамматикам 255
6.3.3. Упражнения к разделу 6.3 259
6.4. Детерминированные автоматы с магазинной памятью 260
6.4.1. Определение детерминированного МП-автомата 260
6.4.2. Регулярные языки и детерминированные МП-автоматы 261

6.4.3. Детерминированные МП-автоматы и КС-языки 262
6.4.4. Детерминированные МП-автоматы и неоднозначные грамматики 263
6.4.5. Упражнения к разделу 6.4 264
Резюме 265
Литература 266

**ГЛАВА 7. Свойства контекстно-свободных языков 269**

7.1. Нормальные формы контекстно-свободных грамматик 269
7.1.1. Удаление бесполезных символов 269
7.1.2. Вычисление порождающих и достижимых символов 271
7.1.3. Удаление ε-продукций 273
7.1.4. Удаление цепных продукций 276
7.1.5. Нормальная форма Хомского 280
7.1.6. Упражнения к разделу 7.1 284
7.2. Лемма о накачке для контекстно-свободных языков 287
7.2.1. Размер деревьев разбора 287
7.2.2. Утверждение леммы о накачке 288
7.2.3. Приложения леммы о накачке к КС-языкам 290
7.2.4. Упражнения к разделу 7.2 293
7.3. Свойства замкнутости контекстно-свободных языков 295
7.3.1. Подстановки 295
7.3.2. Приложения теоремы о подстановке 297
7.3.3. Обращение 298
7.3.4. Пересечение с регулярным языком 298
7.3.5. Обратный гомоморфизм 302
7.3.6. Упражнения к разделу 7.3 304
7.4. Свойства разрешимости КС-языков 306
7.4.1. Сложность взаимных преобразований КС-грамматик и МП-автоматов 306
7.4.2. Временн*ая* сложность преобразования к нормальной форме Хомского 308
7.4.3. Проверка пустоты КС-языков 309
7.4.4. Проверка принадлежности КС-языку 311
7.4.5. Обзор неразрешимых проблем КС-языков 314
7.4.6. Упражнения к разделу 7.4 315
Резюме 316
Литература 317

**ГЛАВА 8. Введение в теорию машин Тьюринга 319**

8.1. Задачи, не решаемые компьютерами 319
8.1.1. Программы печати “Hello, world” 320
8.1.2. Гипотетическая программа проверки приветствия мира 322
8.1.3. Сведение одной проблемы к другой 325
8.1.4. Упражнения к разделу 8.1 328
8.2. Машина Тьюринга 328
8.2.1. Поиски решения всех математических вопросов 329
8.2.2. Описание машин Тьюринга 330

8.2.3. Конфигурации машин Тьюринга 331
8.2.4. Диаграммы переходов для машин Тьюринга 334
8.2.5. Язык машины Тьюринга 337
8.2.6. Машины Тьюринга и останов 338
8.2.7. Упражнения к разделу 8.2 339
8.3. Техника программирования машин Тьюринга 340
8.3.1. Память в состоянии 340
8.3.2. Многодорожечные ленты 342
8.3.3. Подпрограммы 344
8.3.4. Упражнения к разделу 8.3 346
8.4. Расширения базовой машины Тьюринга 346
8.4.1. Многоленточные машины Тьюринга 347
8.4.2. Эквивалентность одноленточных и многоленточных машин Тьюринга 348
8.4.3. Время работы и конструкция “много лент к одной” 349
8.4.4. Недетерминированные машины Тьюринга 351
8.4.5. Упражнения к разделу 8.4 353
8.5. Машины Тьюринга с ограничениями 355
8.5.1. Машины Тьюринга с односторонними лентами 356
8.5.2. Мультистековые машины 358
8.5.3. Счетчиковые машины 361
8.5.4. Мощность счетчиковых машин 362
8.5.5. Упражнения к разделу 8.5 364
8.6. Машины Тьюринга и компьютеры 365
8.6.1. Имитация машины Тьюринга на компьютере 365
8.6.2. Имитация компьютера на машине Тьюринга 366
8.6.3. Сравнение времени работы компьютеров и машин Тьюринга 371
Резюме 374
Литература 376

**ГЛАВА 9. Неразрешимость 377**

9.1. Неперечислимый язык 378
9.1.1. Перечисление двоичных цепочек 378
9.1.2. Коды машин Тьюринга 379
9.1.3. Язык диагонализации 380
9.1.4. Доказательство неперечислимости $L_d$ 381
9.1.5. Упражнения к разделу 9.1 382
9.2. Неразрешимая РП-проблема 382
9.2.1. Рекурсивные языки 383
9.2.2. Дополнения рекурсивных и РП-языков 385
9.2.3. Универсальный язык 387
9.2.4. Неразрешимость универсального языка 389
9.2.5. Упражнения к разделу 9.2 390
9.3. Неразрешимые проблемы, связанные с машинами Тьюринга 392
9.3.1. Сведения 392
9.3.2. Машины Тьюринга, допускающие пустой язык 394
9.3.3. Теорема Райса и свойства РП-языков 397

9.3.4. Проблемы, связанные с описаниями языков в виде машин Тьюринга 399
9.3.5. Упражнения к разделу 9.3 400
9.4. Проблема соответствий Поста 401
9.4.1. Определение проблемы соответствий Поста 402
9.4.2. “Модифицированная” ПСП 404
9.4.3. Завершение доказательства неразрешимости ПСП 407
9.4.4. Упражнения к разделу 9.4 412
9.5. Другие неразрешимые проблемы 413
9.5.1. Проблемы, связанные с программами 413
9.5.2. Неразрешимость проблемы неоднозначности КС-грамматик 413
9.5.3. Дополнение языка списка 416
9.5.4. Упражнения к разделу 9.5 418
9.6. Резюме 420
9.7. Литература 421

**ГЛАВА 10. Труднорешаемые проблемы 423**

10.1. Классы $\mathcal{P}$ и $\mathcal{NP}$ 424
10.1.1. Проблемы, разрешимые за полиномиальное время 424
10.1.2. Пример: алгоритм Крускала 424
10.1.3. Недетерминированное полиномиальное время 429
10.1.4. Пример из $\mathcal{NP}$: проблема коммивояжера 429
10.1.5. Полиномиальные сведения 431
10.1.6. NP-полные проблемы 432
10.1.7. Упражнения к разделу 10.1 434
10.2. Первая NP-полная проблема 436
10.2.1. Проблема выполнимости 436
10.2.2. Представление экземпляров ВЫП 438
10.2.3. NP-полнота проблемы ВЫП 439
10.2.4. Упражнения к разделу 10.2 445
10.3. Ограниченная проблема выполнимости 445
10.3.1. Нормальные формы булевых выражений 446
10.3.2. Преобразование формул в КНФ 447
10.3.3. NP-полнота проблемы ВКНФ 450
10.3.4. NP-полнота проблемы 3-выполнимости 455
10.3.5. Упражнения к разделу 10.3 456
10.4. Еще несколько NP-полных проблем 457
10.4.1. Описание NP-полных проблем 458
10.4.2. Проблема независимого множества 458
10.4.3. Проблема узельного покрытия 462
10.4.4. Проблема ориентированного гамильтонова цикла 464
10.4.5. Неориентированные гамильтоновы циклы и ПКОМ 470
10.4.6. Вывод относительно NP-полных проблем 472
10.4.7. Упражнения к разделу 10.4 473
10.5. Резюме 477
10.6. Литература 478

**ГЛАВА 11. Дополнительные классы проблем** **481**
11.1. Дополнения языков из $\mathcal{NP}$ 482
11.1.1. Класс языков со-$\mathcal{NP}$ 482
11.1.2. NP-полные проблемы и со-$\mathcal{NP}$ 483
11.1.3. Упражнения к разделу 11.1 484
11.2. Проблемы, разрешимые в полиномиальном пространстве 485
11.2.1. Машины Тьюринга с полиномиальным пространством 485
11.2.2. Связь $\mathcal{PS}$ и $\mathcal{NPS}$ с определенными ранее классами 486
11.2.3. Детерминированное и недетерминированное полиномиальное пространство 487
11.3. Проблема, полная для $\mathcal{PS}$ 489
11.3.1. PS-полнота 490
11.3.2. Булевы формулы с кванторами 491
11.3.3. Вычисление булевых формул с кванторами 492
11.3.4. PS-полнота проблемы КБФ 494
11.3.5. Упражнения к разделу 11.3 498
11.4. Классы языков, основанные на рандомизации 499
11.4.1. Быстрая сортировка — пример рандомизированного алгоритма 499
11.4.2. Вариант машины Тьюринга с использованием рандомизации 500
11.4.3. Язык рандомизированной машины Тьюринга 502
11.4.4. Класс $\mathcal{RP}$ 504
11.4.5. Распознавание языков из $\mathcal{RP}$ 506
11.4.6. Класс $\mathcal{ZPP}$ 506
11.4.7. Соотношение между $\mathcal{RP}$ и $\mathcal{ZPP}$ 507
11.4.8. Соотношения с классами $\mathcal{P}$ и $\mathcal{NP}$ 509
11.5. Сложность проверки простоты 509
11.5.1. Важность проверки пустоты 510
11.5.2. Введение в модулярную арифметику 512
11.5.3. Сложность вычислений в модулярной арифметике 514
11.5.4. Рандомизированная полиномиальная проверка простоты 515
11.5.5. Недетерминированные проверки простоты 516
11.5.6. Упражнения к разделу 11.5 519
Резюме 520
Литература 521

**Предметный указатель** **523**

ГЛАВА 4

# Свойства регулярных языков

В этой главе рассматриваются свойства регулярных языков. В разделе 4.1 предлагается инструмент для доказательства нерегулярности некоторых языков — теорема, которая называется “леммой о накачке” (“pumping lemma”)[1].

Одними из важнейших свойств регулярных языков являются “свойства замкнутости”. Эти свойства позволяют создавать распознаватели для одних языков, построенных из других с помощью определенных операций. Например, пересечение двух регулярных языков также является регулярным. Таким образом, при наличии автоматов для двух различных регулярных языков можно (механически) построить автомат, который распознает их пересечение. Поскольку автомат для пересечения языков может содержать намного больше состояний, чем любой из двух данных автоматов, то “свойство замкнутости” может оказаться полезным инструментом для построения сложных автоматов. Конструкция для пересечения уже использовалась в разделе 2.1.

Еще одну важную группу свойств регулярных языков образуют “свойства разрешимости”. Изучение этих свойств позволяет ответить на важнейшие вопросы, связанные с автоматами. Так, можно выяснить, определяют ли два различных автомата один и тот же язык. Разрешимость этой задачи позволяет “минимизировать” автоматы, т.е. по данному автомату найти эквивалентный ему с наименьшим возможным количеством состояний. Задача минимизации уже в течение десятилетий имеет большое значение при проектировании переключательных схем, поскольку стоимость схемы (площади чипа, занимаемого схемой) снижается при уменьшении количества состояний автомата, реализованного схемой.

## 4.1. Доказательство нерегулярности языков

В предыдущих разделах было установлено, что класс языков, известных как регулярные, имеет не менее четырех различных способов описания. Это языки, допускаемые ДКА, НКА и $\varepsilon$-НКА; их можно также определять с помощью регулярных выражений.

Не каждый язык является регулярным. В этом разделе предлагается мощная техника доказательства нерегулярности некоторых языков, известная как “лемма о накачке”. Ни-

---

[1] В русскоязычной литературе в свое время был принят термин “лемма о разрастании”. Однако, на наш взгляд, “накачка” точнее отражает суть происходящего. — *Прим. ред.*

же приводится несколько примеров нерегулярных языков. В разделе 4.2 лемма о накачке используется вместе со свойствами замкнутости регулярных языков для доказательства нерегулярности других языков.

### 4.1.1. Лемма о накачке для регулярных языков

Рассмотрим язык $L_{01} = \{0^n1^n \mid n \geq 1\}$. Этот язык состоит из всех цепочек вида 01, 0011, 000111 и так далее, содержащих один или несколько нулей, за которыми следует такое же количество единиц. Утверждается, что язык $L_{01}$ нерегулярен. Неформально, если бы $L_{01}$ был регулярным языком, то допускался бы некоторым ДКА $A$, имеющим какое-то число состояний $k$. Предположим, что на вход автомата поступает $k$ нулей. Он находится в некотором состоянии после чтения каждого из $k + 1$ префиксов входной цепочки, т.е. $\varepsilon$, 0, 00, …, $0^k$. Поскольку есть только $k$ различных состояний, то согласно "принципу голубятни", прочитав два различных префикса, например, $0^i$ и $0^j$, автомат должен находится в одном и том же состоянии, скажем, $q$.

Допустим, что, прочитав $i$ или $j$ нулей, автомат $A$ получает на вход 1. По прочтении $i$ единиц он должен допустить вход, если ранее получил $i$ нулей, и отвергнуть его, получив $j$ нулей. Но в момент поступления 1 автомат $A$ находится в состоянии $q$ и не способен "вспомнить", какое число нулей, $i$ или $j$, было принято. Следовательно, его можно "обманывать" и заставлять работать неправильно, т.е. допускать, когда он не должен этого делать, или наоборот.

Приведенное неформальное доказательство можно сделать точным. Однако к заключению о нерегулярности языка $L_{01}$ приводит следующий общий результат.

**Теорема 4.1** (*лемма о накачке для регулярных языков*). Пусть $L$ — регулярный язык. Существует константа $n$ (зависящая от $L$), для которой каждую цепочку $w$ из языка $L$, удовлетворяющую неравенству $|w| \geq n$, можно разбить на три цепочки $w = xyz$ так, что выполняются следующие условия.

1. $y \neq \varepsilon$.
2. $|xy| \leq n$.
3. Для любого $k \geq 0$ цепочка $xy^kz$ также принадлежит $L$.

Это значит, что всегда можно найти такую цепочку $y$ недалеко от начала цепочки $w$, которую можно "накачать". Таким образом, если цепочку $y$ повторить любое число раз или удалить (при $k = 0$), то результирующая цепочка все равно будет принадлежать языку $L$.

**Доказательство.** Пусть $L$ — регулярный язык. Тогда $L = L(A)$ для некоторого ДКА $A$. Пусть $A$ имеет $n$ состояний. Рассмотрим произвольную цепочку $w$ длиной не менее $n$, скажем, $w = a_1a_2\ldots a_m$, где $m \geq n$ и каждый $a_i$ есть входной символ. Для $i = 0, 1, 2, \ldots, n$ определим состояние $p_i$ как $\hat{\delta}(q_0, a_1a_2\ldots a_i)$, где $\delta$— функция переходов автомата $A$, а $q_0$ — его начальное состояние. Заметим, что $p_0 = q_0$.

Рассмотрим $n + 1$ состояний $p_i$ при $i = 0, 1, 2, \ldots, n$. Поскольку автомат $A$ имеет $n$ различных состояний, то по "принципу голубятни" найдутся два разных целых числа $i$ и $j$ ($0 \le i < j \le n$), при которых $p_i = p_j$. Теперь разобьем цепочку $w$ на $xyz$.

1. $x = a_1 a_2 \ldots a_i$.
2. $y = a_{i+1} a_{i+2} \ldots a_j$.
3. $z = a_{j+1} a_{j+2} \ldots a_m$.

Таким образом, $x$ приводит в состояние $p_i$, $y$ — из $p_i$ обратно в $p_i$ (так как $p_i = p_j$), а $z$ — это остаток цепочки $w$. Взаимосвязи между цепочками и состояниями показаны на рис. 4.1. Заметим, что цепочка $x$ может быть пустой при $i = 0$, а $z$ — при $j = n = m$. Однако цепочка $y$ не может быть пустой, поскольку $i$ строго меньше $j$.

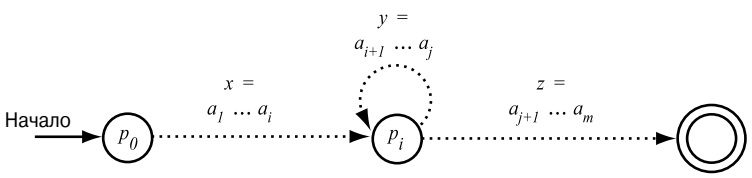


*Рис. 4.1. Каждая цепочка, длина которой больше числа состояний автомата, приводит к повторению некоторого состояния*

Теперь посмотрим, что происходит, когда на вход автомата $A$ поступает цепочка $xy^k z$ для любого $k \ge 0$. При $k = 0$ автомат переходит из начального состояния $q_0$ (которое есть также $p_0$) в $p_i$, прочитав $x$. Поскольку $p_i = p_j$, то $z$ переводит $A$ из $p_i$ в допускающее состояние (см. рис. 4.1).

Если $k > 0$, то по $x$ автомат $A$ переходит из $q_0$ в $p_i$, затем, читая $y^k$, $k$ раз циклически проходит через $p_i$, и, наконец, по $z$ переходит в допускающее состояние. Таким образом, для любого $k \ge 0$ цепочка $xy^k z$ также допускается автоматом $A$, т.е. принадлежит языку $L$. □

## 4.1.2. Применение леммы о накачке

Рассмотрим несколько примеров использования леммы о накачке. В каждом примере эта лемма применяется для доказательства нерегулярности некоторого предлагаемого языка.

### Лемма о накачке как игра двух противников

В разделе 1.2.3 говорилось о том, что любую теорему, утверждение которой содержит несколько чередующихся кванторов "для всех" ("для любого") и "существует", можно представить в виде игры двух противников. Лемма о накачке служит важным примером теорем такого типа, так как содержит четыре разных квантора: "**для любого** регулярного языка $L$ **существует** $n$, при котором **для всех** $w$ из $L$, удовлетворяющих неравенству $|w| \ge n$, **существует** цепочка $xyz$, равная $w$, удовлетворяющая …". Применение леммы о накачке можно представить в виде игры со следующими правилами.

1. Игрок 1 выбирает язык $L$, нерегулярность которого нужно доказать.
2. Игрок 2 выбирает $n$, но не открывает его игроку 1; первый игрок должен построить игру для всех возможных значений $n$.
3. Игрок 1 выбирает цепочку $w$, которая может зависеть от $n$, причем ее длина должна быть не меньше $n$.
4. Игрок 2 разбивает цепочку $w$ на $x$, $y$ и $z$, соблюдая условия леммы о накачке, т.е. $y \neq \varepsilon$ и $|xy| \leq n$. Опять-таки, он не обязан говорить первому игроку, чему равны $x$, $y$ и $z$, хотя они должны удовлетворять условиям леммы.
5. Первый игрок "выигрывает", если выбирает $k$, которое может быть функцией от $n$, $x$, $y$ и $z$ и для которого цепочка $xy^kz$ не принадлежит $L$.

**Пример 4.2.** Покажем, что язык $L_{eq}$, состоящий из всех цепочек с одинаковым числом нулей и единиц (расположенных в произвольном порядке), нерегулярен. В терминах игры, описанной во врезке "Лемма о накачке как игра двух противников", мы являемся игроком 1 и должны иметь дело с любыми допустимыми ходами игрока 2. Предположим, что $n$ — это та константа, которая согласно лемме о накачке должна существовать, если язык $L_{eq}$ регулярен, т.е. "игрок 2" выбирает $n$. Мы выбираем цепочку $w = 0^n1^n$, которая наверняка принадлежит $L_{eq}$.

Теперь "игрок 2" разбивает цепочку $w$ на $xyz$. Нам известно лишь, что $y \neq \varepsilon$ и $|xy| \leq n$. Но эта информация очень полезна, и мы "выигрываем" следующим образом. Поскольку $|xy| \leq n$, и цепочка $xy$ расположена в начале цепочки $w$, то она состоит только из нулей. Если язык $L_{eq}$ регулярен, то по лемме о накачке цепочка $xz$ принадлежит $L_{eq}$ (при $k = 0$ в лемме)[2]. Цепочка $xz$ содержит $n$ единиц, так как все единицы цепочки $w$ попадают в $z$. Но в $xz$ нулей меньше $n$, так как потеряны все нули из $y$. Поскольку $y \neq \varepsilon$, то вместе $x$ и $z$ содержат не более $n - 1$ нулей. Таким образом, предположив, что язык $L_{eq}$ регулярен, приходим к ошибочному выводу, что цепочка $xz$ принадлежит $L_{eq}$. Следовательно, методом от противного доказано, что язык $L_{eq}$ нерегулярен. □

**Пример 4.3.** Докажем нерегулярность языка $L_{pr}$, образованного всеми цепочками из единиц, длины которых — простые числа. Предположим, что язык $L_{pr}$ регулярен. Тогда должна существовать константа $n$, удовлетворяющая условиям леммы о накачке. Рассмотрим некоторое простое число $p \geq n + 2$. Такое $p$ должно существовать, поскольку множество простых чисел бесконечно. Пусть $w = 1^p$.

Согласно лемме о накачке можно разбить цепочку $w = xyz$ так, что $y \neq \varepsilon$ и $|xy| \leq n$. Пусть $|y| = m$. Тогда $|xz| = p - m$. Рассмотрим цепочку $xy^{p-m}z$, которая по лемме о накачке должна принадлежать языку $L_{pr}$, если он действительно регулярен. Однако

$$|xy^{p-m}z| = |xz| + (p - m)|y| = p - m + (p - m)m = (m + 1)(p - m).$$

[2] Заметим, что можно выиграть и при k = 2 или любом другом значении k, кроме k = 1.

Похоже, что число $|xy^{p-m}z|$ не простое, так как имеет два множителя $m + 1$ и $p - m$. Однако нужно еще убедиться, что ни один из этих множителей не равен 1, потому что тогда число $(m + 1)(p - m)$ будет простым. Из неравенства $y \neq \varepsilon$ следует $m \geq 1$ и $m + 1 > 1$. Кроме того, $m = |y| \leq |xy| \leq n$, а $p \geq n + 2$, поэтому $p - m \geq 2$.

Мы начали с предположения, что предлагаемый язык регулярен, и пришли к противоречию, доказав, что существует некоторая цепочка, которая не принадлежит этому языку, тогда как по лемме о накачке она должна ему принадлежать. Таким образом, язык $L_{pr}$ нерегулярен. □

### 4.1.3. Упражнения к разделу 4.1

**4.1.1.** Докажите нерегулярность следующих языков:

а) $\{0^n1^n \mid n \geq 1\}$. Это язык $L_{01}$, который рассматривался в начале раздела. Ему принадлежат все цепочки, состоящие из нулей, за которыми следует такое же количество единиц. Здесь для доказательства примените лемму о накачке;

б) множество цепочек, состоящих из "сбалансированных" скобок "(" и ")", которые встречаются в правильно построенном арифметическом выражении;

в) (*) $\{0^n10^n \mid n \geq 1\}$;

г) $\{0^n1^m2^n \mid n$ и $m$ — произвольные целые числа$\}$;

д) $\{0^n1^m \mid n \leq m\}$;

е) $\{0^n1^{2n} \mid n \geq 1\}$.

**4.1.2.** (!) Докажите нерегулярность следующих языков:

а) (*) $\{0^n \mid n$ — полный квадрат$\}$;

б) $\{0^n \mid n$ — полный куб$\}$;

в) $\{0^n \mid n$ — степень числа 2$\}$;

г) множество цепочек из нулей и единиц, длины которых — полные квадраты;

д) множество цепочек из нулей и единиц вида $ww$, где некоторая цепочка $w$ повторяется дважды;

е) множество цепочек из нулей и единиц вида $ww^R$, где за цепочкой следует цепочка, обратная к ней (формальное определение цепочки, обратной к данной, см. в разделе 4.2.2);

ж) множество цепочек из нулей и единиц вида $w\overline{w}$, где цепочка $\overline{w}$ образована из $w$ путем замены всех нулей единицами и наоборот. Например, $\overline{011} = 100$, так что цепочка 011100 принадлежит данному языку;

з) множество цепочек из нулей и единиц вида $w1^n$, где $w$ — цепочка из нулей и единиц длиной $n$.

**4.1.3.** (**!!**) Докажите нерегулярность следующих языков:

а) множество всех цепочек из нулей и единиц, которые начинаются с единицы и удовлетворяют следующему условию: если интерпретировать такую цепочку как целое число, то это число простое;[3]

б) множество цепочек вида $0^i1^j$, для которых наибольший общий делитель чисел $i$ и $j$ равен 1.

**4.1.4.** (**!**) При попытке применения леммы о накачке к регулярным языкам "противник выигрывает", и закончить доказательство не удается. Определите, что именно происходит не так, как нужно, если в качестве $L$ выбрать следующий язык:

а) (*****) пустое множество;

б) (*****) {00, 11};

в) (*****) $(\mathbf{00} + \mathbf{11})^*$;

г) $\mathbf{01}^*\mathbf{0}^*\mathbf{1}$.

## 4.2. Свойства замкнутости регулярных языков

В этом разделе доказано несколько теорем вида "если определенные языки регулярны, а язык $L$ построен из них с помощью определенных операций (например, $L$ есть объединение двух регулярных языков), то язык $L$ также регулярен". Эти теоремы часто называют *свойствами замкнутости* регулярных языков, так как в них утверждается, что класс регулярных языков замкнут относительно определенных операций. Свойства замкнутости выражают идею того, что если один или несколько языков регулярны, то языки, определенным образом связанные с ним (с ними), также регулярны. Кроме того, данные свойства служат интересной иллюстрацией того, как эквивалентные представления регулярных языков (автоматы и регулярные выражения) подкрепляют друг друга в нашем понимании этого класса языков, так как часто один способ представления намного лучше других подходит для доказательства некоторого свойства замкнутости.

Основные свойства замкнутости регулярных языков выражаются в том, что эти языки замкнуты относительно следующих операций.

1. Объединение.
2. Пересечение.
3. Дополнение.
4. Разность.
5. Обращение.

---

[3] Иными словами, это двоичные записи простых чисел. — *Прим. ред.*

6. Итерация (звездочка).
7. Конкатенация.
8. Гомоморфизм (подстановка цепочек вместо символов языка).
9. Обратный гомоморфизм.

### 4.2.1. Замкнутость регулярных языков относительно булевых операций

Сначала рассмотрим замкнутость для трех булевых операций: объединение, пересечение и дополнение.

1. Пусть $L$ и $M$ — языки в алфавите $\Sigma$. Тогда язык $L \cup M$ содержит все цепочки, которые принадлежат хотя бы одному из языков $L$ или $M$.
2. Пусть $L$ и $M$ — языки в алфавите $\Sigma$. Тогда язык $L \cap M$ содержит все цепочки, принадлежащие обоим языкам $L$ и $M$.
3. Пусть $L$ — некоторый язык в алфавите $\Sigma$. Тогда язык $\overline{L}$, *дополнение* $L$, — это множество тех цепочек в алфавите $\Sigma^*$, которые не принадлежат $L$.

**Что делать, если языки имеют разные алфавиты?**

При объединении или пересечении двух языков $L$ и $M$ может оказаться, что они определены в разных алфавитах. Например, возможен случай, когда $L_1 \subseteq \{a, b\}$, а $L_2 \subseteq \{b, c, d\}$. Однако, если язык $L$ состоит из цепочек символов алфавита $\Sigma$, то $L$ можно также рассматривать как язык в любом конечном алфавите, включающем $\Sigma$ (надмножестве $\Sigma$). Например, можно представить указанные выше языки $L_1$ и $L_2$ как языки в алфавите $\{a, b, c, d\}$. То, что ни одна цепочка языка $L_1$ не содержит символов $c$ или $d$, несущественно, как и то, что ни одна цепочка языка $L_2$ не содержит $a$.

Аналогично, рассматривая дополнение языка $L$, который является подмножеством множества $\Sigma_1^*$ для некоторого алфавита $\Sigma_1$, можно взять дополнение *относительно* некоторого алфавита $\Sigma_2$, включающего $\Sigma_1$ (надмножества $\Sigma_1$). В этом случае дополнением $L$ будет $\Sigma_2^* - L$, т.е. дополнение языка $L$ относительно алфавита $\Sigma_2$ включает (среди прочих) все цепочки из $\Sigma_2^*$, которые содержат хотя бы один символ алфавита $\Sigma_2$, не принадлежащий $\Sigma_1$. Если взять дополнение $L$ относительно $\Sigma_1$, то ни одна цепочка, содержащая символы из $\Sigma_2 - \Sigma_1$, не попадет в $\overline{L}$. Таким образом, чтобы избежать неточностей, нужно указывать алфавит, относительно которого берется дополнение. Часто, однако, бывает очевидно, какой алфавит подразумевается в конкретном случае. Например, если язык $L$ определен некоторым автоматом, то в описании этого автомата указывается и алфавит. Итак, во многих ситуациях можно говорить о “дополнении”, не указывая алфавит.

Оказывается, что класс регулярных языков замкнут относительно всех трех булевых операций, хотя, как будет видно, в доказательствах используются совершенно разные подходы.

**Замкнутость относительно объединения**

**Теорема 4.4.** Если $L$ и $M$ — регулярные языки, то $L \cup M$ также регулярен.

**Доказательство.** Поскольку языки $L$ и $M$ регулярны, им соответствуют некоторые регулярные выражения. Пусть $L = L(R)$ и $M = L(S)$. Тогда $L \cup M = L(R + S)$ согласно определению операции + для регулярных выражений. □

### Замкнутость относительно регулярных операций

Доказательство замкнутости регулярных выражений относительно объединения было исключительно легким, поскольку объединение является одной из трех операций, определяющих регулярные выражения. Идея доказательства теоремы 4.4 применима также к конкатенации и итерации. Таким образом,

- если $L$ и $M$ — регулярные языки, то язык $LM$ регулярен;
- если $L$ — регулярный язык, то $L^*$ также регулярен.

**Замкнутость относительно дополнения**

Теорема для объединения языков легко доказывается с помощью регулярных выражений. Теперь рассмотрим дополнение. Знаете ли вы, как преобразовать регулярное выражение, чтобы оно определяло дополнение языка? Мы тоже не знаем. Однако это выполнимо, так как согласно теореме 4.5 можно начать с ДКА и построить ДКА, допускающий дополнение. Таким образом, начав с регулярного выражения для языка, можно найти выражение для его дополнения следующим образом.

1. Преобразовать регулярное выражение в $\varepsilon$-НКА.
2. Преобразовать $\varepsilon$-НКА в ДКА с помощью конструкции подмножеств.
3. Дополнить допускающие состояния этого ДКА.
4. Преобразовать полученный ДКА для дополнения обратно в регулярное выражение, используя методы из разделов 3.2.1 или 3.2.2.

**Теорема 4.5.** Если $L$ — регулярный язык в алфавите $\Sigma$, то язык $\overline{L} = \Sigma^* - L$ также регулярен.

**Доказательство**. Пусть $L = L(A)$ для некоторого ДКА $A = (Q, \Sigma, \delta, q_0, F)$. Тогда $\overline{L} = L(B)$, где $B$ — это ДКА $(Q, \Sigma, \delta, q_0, Q - F)$, т.е. автоматы $A$ и $B$ одинаковы, за исключением того, что допускающие состояния автомата $A$ стали не допускающими в $B$, и наоборот. Тогда $w$ принадлежит $L(B)$, если, и только если, $\hat{\delta}(q_0, w)$ принадлежит $Q - F$, т.е. $w$ не принадлежит $L(A)$. □

Заметим, что для приведенного выше доказательства важно, чтобы $\hat{\delta}(q_0, w)$ всегда было некоторым состоянием. Это значит, что в автомате $A$ все переходы определены. Если бы некоторые переходы отсутствовали, то определенные цепочки не вели бы ни в допускающее, ни в недопускающее состояния автомата $A$, т.е. отсутствовали бы как в $L(A)$, так и $L(B)$. К счастью, ДКА определен так, что в любом состоянии у него есть переход по каждому символу алфавита Σ, так что каждая цепочка приводит в состояние либо из $F$, либо из $Q - F$.

**Пример 4.6.** Пусть $A$ — автомат, изображенный на рис. 2.14. Напомним, что этот ДКА допускает те и только те цепочки символов 0 и 1, которые заканчиваются на 01. В терминах регулярных выражений $L(A) = (\mathbf{0} + \mathbf{1})^*\mathbf{01}$. Таким образом, дополнение к $L(A)$ содержит все те цепочки из нулей и единиц, которые *не* заканчиваются на 01. На рис. 4.2 представлен автомат для $\{0, 1\}^* - L(A)$. Он совпадает с автоматом на рис. 2.14, за исключением того, что допускающие состояния стали недопускающими, а два недопускающих состояния стали допускающими. □

**Пример 4.7.** Используем теорему 4.5 для доказательства нерегулярности определенного языка. В примере 4.2 была доказана нерегулярность языка $L_{eq}$, состоящего из цепочек с равными количествами символов 0 и 1. Это доказательство было непосредственным применением леммы о накачке. Теперь рассмотрим язык $M$, состоящий из цепочек с неравными количествами нулей и единиц.

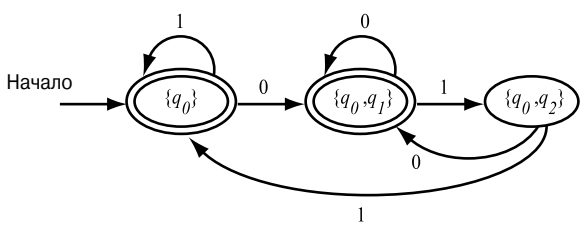


*Рис. 4.2. ДКА, допускающий дополнение языка* $(\mathbf{0} + \mathbf{1})^*\mathbf{01}$

В данном случае для доказательства нерегулярности языка $M$ лемму о накачке применить трудно. Интуиция подсказывает, что если начать с некоторой цепочки $w$ из $M$, разбить ее на $w = xyz$ и "накачать" $y$, то может оказаться, что $y$ — это цепочка вроде 01, в которой поровну символов 0 и 1. В этом случае не существует такого $k$, для которого цепочка $xy^kz$ имела бы поровну нулей и единиц, поскольку $xyz$ содержит неравные количества нулей и единиц, а при "накачивании" $y$ эти количества изменяются одинаково. Следовательно, мы не можем использовать лемму о накачке для того, чтобы прийти к противоречию с предположением, что язык $M$ регулярен.

Но все-таки язык $M$ нерегулярен. Объясняется это тем, что $M = \overline{L}_{eq}$. Поскольку дополнение к дополнению некоторого множества равно этому же множеству, то $L_{eq} = \overline{M}$.

Если $M$ регулярен, то по теореме 4.5 язык $L_{eq}$ также регулярен. Но мы знаем, что $L_{eq}$ *не* регулярен, и полученное противоречие доказывает нерегулярность языка $M$. □

**Замкнутость относительно пересечения**

Рассмотрим пересечение двух регулярных языков. Здесь почти нечего делать, поскольку операции объединения, дополнения и пересечения не являются независимыми. Пересечение языков $L$ и $M$ выражается через объединение и дополнение следующим тождеством.

$$L \cap M = \overline{\overline{L} \cup \overline{M}} \tag{4.1}$$

Вообще, пересечение двух множеств — это множество элементов, не принадлежащих дополнениям каждого из них. Это замечание, записанное в виде равенства (4.1), представляет один из *законов Де Моргана*. Другой закон имеет аналогичный вид, только объединение и пересечение меняются местами, т.е. $L \cup M = \overline{\overline{L} \cap \overline{M}}$ .

Вместе с тем, для пересечения двух регулярных языков можно построить ДКА непосредственно. Такая конструкция, в которой, по существу, параллельно работают два ДКА, весьма полезна сама по себе. Например, она использовалась для построения автомата (см. рис. 2.3), представляющего "произведение" действий двух участников — банка и магазина. *Конструкция произведения* формально представлена в следующей теореме.

**Теорема 4.8.** Если $L$ и $M$ — регулярные языки, то язык $L \bigcap M$ также регулярен.

**Доказательство**. Пусть $L$ и $M$ — языки автоматов $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ и $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$. Заметим, что алфавиты автоматов считаются одинаковыми, т.е. $\Sigma$ есть объединение алфавитов языков $L$ и $M$, если эти алфавиты различаются. В действительности конструкция произведения работает для НКА точно так же, как и для ДКА, но для максимального упрощения предположим, что $A_L$ и $A_M$ — ДКА.

Для $L \bigcap M$ построим автомат $A$, моделирующий автоматы $A_L$ и $A_M$ одновременно. Состояниями автомата $A$ будут пары состояний, первое из которых принадлежит $A_L$, а второе — $A_M$. Чтобы построить переходы автомата $A$, предположим, что он находится в состоянии $(p, q)$, где $p$ — состояние автомата $A_L$, а $q$ — состояние $A_M$. Нам известно, как ведет себя автомат $A_L$, получая на входе символ $a$. Пусть он переходит в состояние $s$. Также допустим, что автомат $A_M$ по входному символу $a$ совершает переход в состояние $t$. Тогда следующим состоянием автомата $A$ будет $(s, t)$. Таким образом, автомат $A$ моделирует работу автоматов $A_L$ и $A_M$. Эта идея в общих чертах представлена на рис. 4.3.

Остальные детали доказательства очень просты. Начальным состоянием автомата $A$ будет пара начальных состояний автоматов $A_L$ и $A_M$. Поскольку автомат $A$ допускает тогда и только тогда, когда допускают оба автомата $A_L$ и $A_M$, в качестве допускающих состояний автомата $A$ выбираем все пары $(p, q)$, где $p$ — допускающее состояние автомата $A_L$, а $q$ —$A_M$. Формально автомат $A$ определяется как

$$A = (Q_L \times Q_M, \Sigma, \delta, (q_L, q_M), (F_L \times F_M),$$

где $\delta((p, q), a) = (\delta_L(p, a), \delta_M(q, a))$.

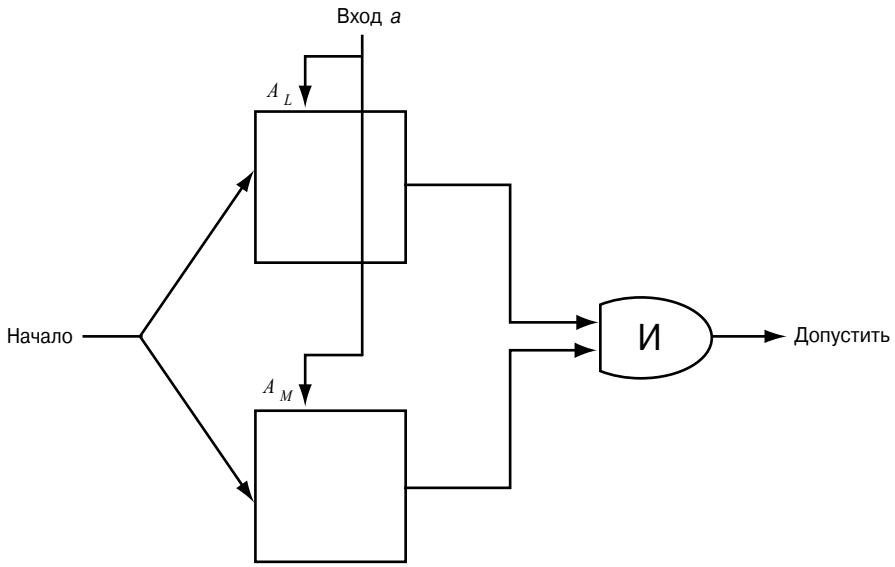


*Рис. 4.3. Автомат, имитирующий два других автомата и допускающий тогда и только тогда, когда допускают оба автомата*

Чтобы увидеть, почему $L(A) = L(A_L) \bigcap L(A_M)$, сначала заметим, что индукцией по $|w|$ легко доказать равенство $\hat{\delta}((q_L, q_M), w) = (\hat{\delta}_L(q_L, w), \hat{\delta}_M(q_M, w))$. Но $A$ допускает $w$ тогда и только тогда, когда $\hat{\delta}((q_L, q_M), w)$ является парой допускающих состояний, т.е. $\hat{\delta}_L(q_L, w)$ должно принадлежать $F_L$, а $\hat{\delta}_M(q_M, w)$ — $F_M$. Иными словами, цепочка $w$ допускается автоматом $A$ тогда и только тогда, когда ее допускают оба автомата $A_L$ и $A_M$. Итак, $A$ допускает пересечение языков $L$ и $M$. □

**Пример 4.9.** На рис. 4.4 представлены два ДКА. Автомат на рис. 4.4, *а* допускает все цепочки, имеющие 0, а автомат на рис. 4.4, *б* — все цепочки, имеющие 1. На рис. 4.4, *в* представлен автомат — произведение двух данных автоматов. Его состояния помечены как пары состояний исходных автоматов.

Легко доказать, что этот автомат допускает пересечение первых двух языков, т.е. те цепочки, которые содержат как 0, так и 1. Состояние *pr* представляет начальное условие, когда на вход автомата пока не поступили ни 0, ни 1. Состояние *qr* означает, что поступили только нули, а состояние *ps* — только единицы. Допускающее состояние *qs* представляет условие того, что на вход автомата поступили и нули, и единицы. □

**Замкнутость относительно разности**

Существует еще одна, четвертая, операция, часто применяемая к множествам и связанная с булевыми операциями, а именно, разность множеств. В терминах языков *разностью* $L - M$ языков $L$ и $M$ называют множество цепочек, которые принадлежат $L$ и не принадлежат $M$. Регулярные языки замкнуты относительно этой операции. Доказательство замкнутости относительно разности следует из доказанных выше теорем.

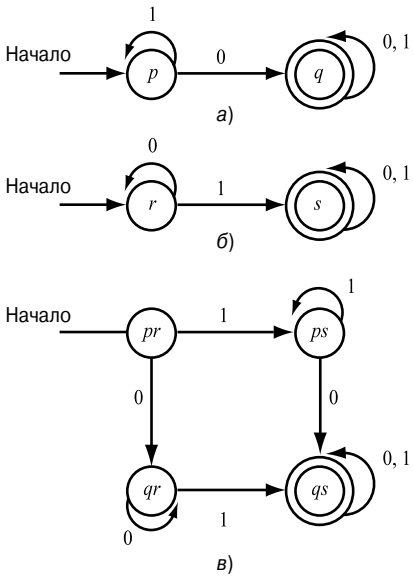


*Рис. 4.4. Конструкция произведения*

**Теорема 4.10.** Если $L$ и $M$ — регулярные языки, то язык $L - M$ также регулярен.

**Доказательство**. Заметим, что $L - M = L \cap \overline{M}$ . По теореме 4.5 регулярен язык $\overline{M}$ , а по теореме 4.8 — $L \cap \overline{M}$ . Следовательно, язык $L - M$ регулярен. □

## 4.2.2. Обращение

*Обращением* цепочки $a_1a_2\ldots a_n$ называется цепочка, записанная в обратном порядке, т.е. $a_na_{n-1}\ldots a_1$. Обращение $w$ обозначается $w^R$. Таким образом, $0010^R$ есть 0100, а $\varepsilon^R = \varepsilon$.

*Обращение языка* $L$, обозначаемое $L^R$, состоит из всех цепочек, обратных цепочкам языка $L$. Например, если $L = \{001, 10, 111\}$, то $L^R = \{100, 01, 111\}$.

Обращение является еще одной операцией, сохраняющей регулярность языков, т.е. если язык $L$ регулярен, то $L^R$ также регулярен. Это легко доказать двумя способами, первый из которых основан на автоматах, а второй — на регулярных выражениях. Доказательство, основанное на автоматах, приводится неформально, так что читатель при желании может восполнить детали. Затем приводится формальное доказательство, использующее регулярные выражения.

Если задан язык $L$, который есть $L(A)$ для некоторого конечного автомата $A$, вероятно, с недетерминизмом и $\varepsilon$-переходами, то можно построить автомат для $L^R$ следующим образом.

1. Обратить все дуги на диаграмме переходов автомата $A$.
2. Сделать начальное состояние автомата $A$ единственным допускающим состоянием нового автомата.

3. Создать новое начальное состояние $p_0$ с $\varepsilon$-переходами во все допускающие состояния автомата $A$.

В результате получим автомат, имитирующий автомат $A$ "в обратном порядке" и, следовательно, допускающий цепочку $w$ тогда и только тогда, кода $A$ допускает $w^R$.

Теперь докажем теорему для обращения формально.

**Теорема 4.11.** Если язык $L$ регулярен, то язык $L^R$ также регулярен.

**Доказательство**. Предположим, что язык $L$ определяется регулярным выражением $E$. Доказательство проводится структурной индукцией по длине выражения $E$. Покажем, что существует еще одно регулярное выражение $E^R$, для которого $L(E^R) = (L(E))^R$, т.е. язык выражения $E^R$ является обращением языка выражения $E$.

**Базис.** Если $E$ равно $\varepsilon$, $\varnothing$ или $a$, где $a$ — некоторый символ, то $E^R$ совпадает с $E$, т.е. $\{\varepsilon\}^R = \{\varepsilon\}$, $\varnothing^R = \varnothing$ и $\{a\}^R = \{a\}$.

**Индукция.** В зависимости от вида выражения $E$ возможны три варианта.

1. $E = E_1 + E_2$. Тогда $E^R = E_1^R + E_2^R$. Доказательство состоит в том, что обращение объединения двух языков получается, если сначала вычислить, а затем объединить обращения этих языков.
2. $E = E_1E_2$. Тогда $E^R = E_2^R E_1^R$. Заметим, что необходимо обратить не только сами языки, но и их порядок. Например, если $L(E_1) = \{01, 111\}$, а $L(E_2) = \{00, 10\}$, то $L(E_1E_2) = \{0100, 0110, 11100, 11110\}$. Обращение этого языка есть $\{0010, 0110, 00111, 01111\}$.

   Если соединить обращения языков $L(E_2)$ и $L(E_1)$ в таком порядке, как они здесь записаны, то получим язык

   $$\{00, 01\}\{10, 111\} = \{0010, 00111, 0110, 01111\},$$

   который равен языку $(L(E_1E_2))^R$. В общем случае, если цепочка $w$ из $L(E)$ является конкатенацией цепочек $w_1$ из $L(E_1)$ и $w_2$ из $L(E_2)$ , то $w^R = w_2^R w_1^R$.
3. $E = E_1^*$. Тогда $E^R = (E_1^R)^*$. Доказательство состоит в том, что любая цепочка $w$ из $L(E)$ может быть записана как $w_1w_2\ldots w_n$, где каждая $w_i$ принадлежит $L(E)$. Но

   $$w^R = w_n^R w_{n-1}^R \ldots w_1^R.$$

   Каждая $w_i^R$ принадлежит $L(E^R)$, т.е. $w^R$ принадлежит $(E_1^R)^*$. И наоборот, любая цепочка из $L((E_1^R)^*)$ имеет вид $w_1w_2\ldots w_n$, где каждая цепочка $w_i$ является обращением некоторой цепочки из $L(E_1)$. Следовательно, обращение данной цепочки $w_n^R w_{n-1}^R \ldots w_1^R$ принадлежит языку $L(E_1^*)$, который равен $L(E)$. Таким образом, доказано, что цепочка принадлежит $L(E)$ тогда и только тогда, когда ее обращение принадлежит $L((E_1^R)^*)$.

□

**Пример 4.12.** Пусть язык $L$ определяется регулярным выражением $(\mathbf{0} + \mathbf{1})\mathbf{0}^*$. Тогда согласно правилу для конкатенации $L^R$ — это язык выражения $(\mathbf{0}^*)^R(\mathbf{0} + \mathbf{1})^R$. Если приме-

нить правила для итерации и объединения к двум частям этого выражения, а потом использовать базисное правило, которое говорит, что обратными к **0** и **1** будут эти же выражения, то получим, что язык $L^R$ определяется регулярным выражением $\mathbf{0}^*(\mathbf{0} + \mathbf{1})$. □

### 4.2.3. Гомоморфизмы

*Гомоморфизм цепочек* — это такая функция на множестве цепочек, которая подставляет определенную цепочку вместо каждого символа данной цепочки.

**Пример 4.13.** Функция $h$, определенная как $h(0) = ab$ и $h(1) = \varepsilon$, является гомоморфизмом. В любой цепочке из символов 0 и 1 $h$ заменяет все нули цепочкой $ab$, а все единицы — пустой цепочкой. Например, применяя $h$ к цепочке 0011, получим $abab$. □

Формально, если $h$ есть некоторый гомоморфизм на алфавите Σ, а $w = a_1a_2\ldots a_n$ — цепочка символов в Σ, то $h(w) = h(a_1)h(a_2)\ldots h(a_n)$. Таким образом, сначала $h$ применяется к каждому символу цепочки $w$, а потом полученные цепочки символов соединяются в соответствующем порядке. Например, рассмотрим гомоморфизм $h$ из примера 4.13 и цепочку $w = 0011$: $h(w) = h(0)h(0)h(1)h(1) = (ab)(ab)(\varepsilon)(\varepsilon) = abab$, что и утверждается в этом примере.

Гомоморфизм языка определяется с помощью его применения к каждой цепочке языка, т.е. если $L$ — язык в алфавите Σ, а $h$ — гомоморфизм на Σ, то $h(L) = \{h(w) \mid w$ принадлежит $L\}$. Рассмотрим язык $L$ регулярного выражения $\mathbf{10}^*\mathbf{1}$, т.е. все цепочки, которые начинаются и заканчиваются единицей, а между ними содержат произвольное число нулей. Пусть $h$ — гомоморфизм из примера 4.13. Тогда $h(L)$ — это язык выражения $(\mathbf{ab})^*$. Объясняется это тем, что $h$ исключает все единицы, заменяя их $\varepsilon$, а вместо каждого нуля подставляет цепочку $ab$. Идея применения гомоморфизма непосредственно к регулярному выражению используется для доказательства замкнутости регулярных языков относительно гомоморфизма.

**Теорема 4.14.** Если $L$ — регулярный язык в алфавите Σ, а $h$ — гомоморфизм на Σ, то язык $h(L)$ также регулярен.

**Доказательство**. Пусть $L = L(R)$ для некоторого регулярного выражения $R$. Вообще, если $E$ есть регулярное выражение с символами из алфавита Σ, то пусть $h(E)$ — выражение, полученное в результате замены каждого символа $a$ в выражении $E$ цепочкой $h(a)$. Утверждается, что выражение $h(R)$ определяет язык $h(L)$.

Это легко доказать с помощью структурной индукции. Если применить гомоморфизм $h$ к любому подвыражению $E$ выражения $R$, то язык выражения $h(E)$ совпадет с языком, полученным в результате применения этого гомоморфизма к языку $L(E)$. Формально, $L(h(E)) = h(L(E))$.

**Базис.** Если $E$ есть $\varepsilon$ или ∅, то $h(E)$ совпадает с $E$, поскольку $h$ не влияет на цепочку $\varepsilon$ или язык ∅. Следовательно, $L(h(E)) = L(E)$. В то же время, если $E$ равно ∅ или $\varepsilon$, то $L(E)$ либо не содержит ни одной цепочки, либо состоит из цепочки без символов. Таким образом, в обоих случаях $h(L(E)) = L(E)$. Из этого следует, что $L(h(E)) = L(E) = h(L(E))$.

Возможен еще один базисный вариант, когда $E = \mathbf{a}$ для некоторого символа $a$ из $\Sigma$. В этом случае $L(E) = \{a\}$, и $h(L(E)) = \{h(a)\}$. Выражение $h(E)$ представляет собой цепочку символов $h(a)$. Таким образом, язык $L(h(E))$ также совпадает с $\{h(a)\}$, и, следовательно, $L(h(E)) = h(L(E))$.

**Индукция.** В зависимости от операции в регулярном выражении возможны три ситуации. Все они просты, поэтому обоснуем индукцию только для объединения, $E = F + G$. Способ применения гомоморфизмов к регулярным выражениям гарантирует, что $h(E) = h(F + G) = h(F) + h(G)$. Нам также известно, что $L(E) = L(F) \cup L(G)$ и

$$L(h(E)) = L(h(F) + h(G)) = L(h(F)) \cup L(h(G)) \tag{4.2}$$

по определению операции + для регулярных выражений. Наконец,

$$h(L(E)) = h(L(F) \cup L(G)) = h(L(F)) \cup h(L(G)), \tag{4.3}$$

поскольку $h$ применяется к языку путем применения его к каждой цепочке этого языка по отдельности. По индуктивной гипотезе $L(h(F)) = h(L(F))$ и $L(h(G)) = h(L(G))$. Таким образом, правые части выражений (4.2) и (4.3) эквивалентны, и, следовательно, $L(h(E)) = h(L(E))$.

Для случаев, когда выражение $E$ является конкатенацией или итерацией, доказательства не приводятся, поскольку они аналогичны доказательству, представленному выше. Итак, можно сделать вывод, что $L(h(R))$ действительно равняется $h(L(R))$, т.е. применение гомоморфизма к регулярному выражению языка $L$ дает регулярное выражение, определяющее язык $h(L)$. □

## 4.2.4. Обратный гомоморфизм

Гомоморфизм можно применять “назад”, и это также сохраняет регулярность языков. Предположим, что $h$ — гомоморфизм из алфавита $\Sigma$ в цепочки, заданные в другом (возможно, том же) алфавите $T$[4]. Пусть $L$ — язык в алфавите $T$. Тогда $h^{-1}(L)$, читаемое как “обратное $h$ от $L$”, — это множество цепочек $w$ из $\Sigma^*$, для которых $h(w)$ принадлежит $L$. На рис. 4.5, *а* представлено применение гомоморфизма к языку $L$, а на рис. 4.5, *б* — использование обратного гомоморфизма.

**Пример 4.15.** Пусть $L$ — язык регулярного выражения $(\mathbf{00} + \mathbf{1})^*$, т.е. все цепочки из символов 0 и 1, в которых нули встречаются парами. Таким образом, цепочки 0010011 и 10000111 принадлежат $L$, а 000 и 10100 — нет.

Пусть $h$ — такой гомоморфизм: $h(a) = 01$, $h(b) = 10$. Утверждается, что $h^{-1}(L)$ — это язык регулярного выражения $(\mathbf{ba})^*$, т.е. все цепочки, в которых повторяются пары $ba$. Докажем, что $h(w)$ принадлежит $L$ тогда и только тогда, когда цепочка $w$ имеет вид $baba\ldots ba$.

[4] Под “T” подразумевается прописная буква греческого алфавита “тау”, следующая за буквой “сигма”.

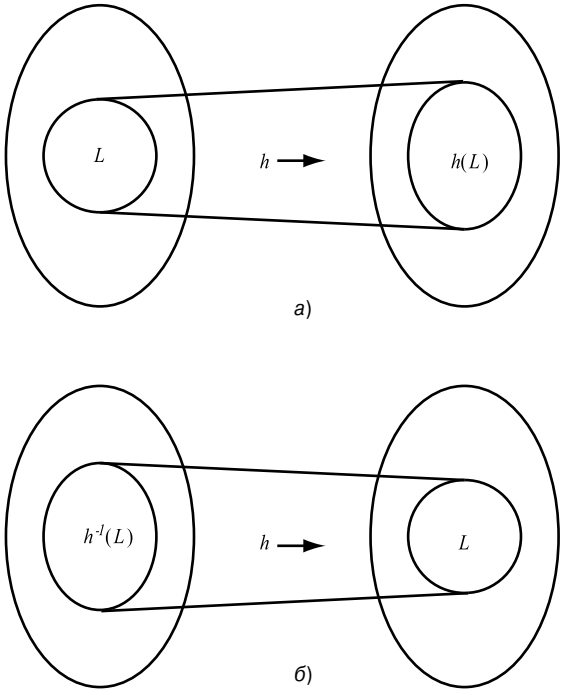


*Рис. 4.5. Гомоморфизм, применяемый в прямом и обратном направлении*

*Достаточность*. Предположим, что цепочка $w$ состоит из $n$ повторений $ba$ для некоторого $n \geq 0$. Заметим, что $h(ba) = 1001$, т.е. $h(w)$ — это $n$ повторений цепочки 1001. Поскольку цепочка 1001 построена из двух единиц и пары нулей, то она принадлежит языку $L$. Следовательно, цепочка, состоящая из любого числа повторений 1001, также образована единицами и парами нулей и принадлежит $L$. Таким образом, $h(w)$ принадлежит $L$.

*Необходимость*. Теперь предположим, что $h(w)$ принадлежит $L$, и покажем, что цепочка $w$ имеет вид $baba…ba$. Существует четыре условия, при которых цепочка имеет *другой* вид. Покажем, что при выполнении любого из них $h(w)$ не принадлежит $L$, т.е. докажем утверждение, противоположное тому, что нам нужно доказать.

1. Если $w$ начинается символом $a$, то $h(w)$ начинается с 01. Следовательно, она содержит отдельный 0 и поэтому не принадлежит $L$.
2. Если $w$ заканчивается символом $b$, то в конце $h(w)$ стоит 10, и опять-таки в цепочке $h(w)$ есть изолированный 0.
3. Если в цепочке $w$ дважды подряд встречается $a$, то $h(w)$ содержит подцепочку 0101. Снова в $w$ есть изолированный нуль.
4. Аналогично, если в $w$ есть два символа $b$ подряд, то $h(w)$ содержит подцепочку 1010 с изолированным 0.

Таким образом, при выполнении хотя бы одного из вышеперечисленных условий цепочка $h(w)$ не принадлежит $L$. Но если ни одно из условий 1–4 не выполняется, то цепочка $w$ имеет вид $baba…ba$. Чтобы понять, почему это происходит, предположим, что ни одно из этих условий не выполняется. Тогда невыполнение условия 1 означает, что $w$ должна начинаться символом $b$, а невыполнение 2 — что она должна заканчиваться символом $a$. Невыполнение условий 3 и 4 говорит, что символы $a$ и $b$ должны чередоваться. Следовательно, логическое **ИЛИ** условий 1–4 эквивалентно утверждению "цепочка $w$ имеет вид, отличный от $baba…ba$". Но выше было доказано, что из логического **ИЛИ** условий 1–4 следует, что $h(w)$ не принадлежит $L$. Это утверждение противоположно тому, что нужно доказать, а именно, что "если $h(w)$ принадлежит $L$, то цепочка $w$ имеет вид $baba…ba$". □

Далее докажем, что обратный гомоморфизм регулярного языка также регулярен, и покажем, как эту теорему можно использовать.

**Теорема 4.16.** Если $h$ — гомоморфизм из алфавита $\Sigma$ в алфавит $T$, $L$ — регулярный язык над $T$, то язык $h^{-1}(L)$ также регулярен.

**Доказательство**. Начнем с ДКА $A$ для языка $L$. По $A$ и $h$ строится ДКА для $h^{-1}(L)$ с помощью схемы, представленной на рис. 4.6. Этот ДКА использует состояния автомата $A$, но переводит входной символ в соответствии с $h$ перед тем, как решить, в какое состояние перейти.

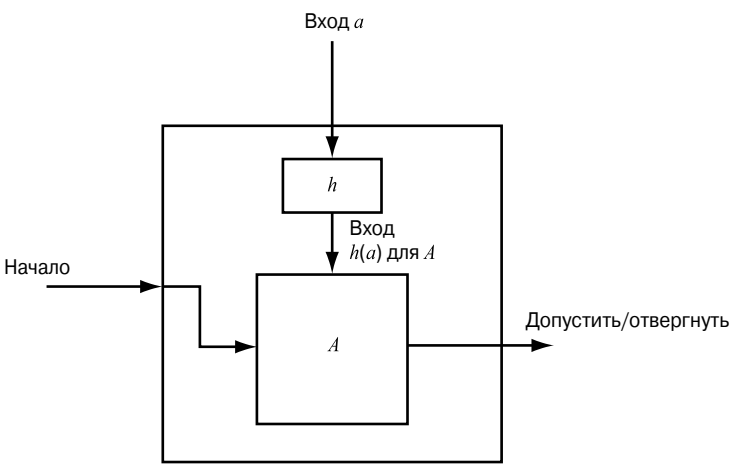


*Рис. 4.6. ДКА для* $h^{-1}(L)$ *применяет гомоморфизм h ко входным символам, а потом имитирует ДКА для L*

Формально, пусть $L$ — это $L(A)$, где ДКА $A = (Q, T, \delta, q_0, F)$. Определим ДКА

$$B = (Q, \Sigma, \gamma, q_0, F),$$

функция переходов $\gamma$ которого строится по правилу $\gamma(q, a) = \hat{\delta}(q, h(a))$. Таким образом, переход автомата $B$ по входному символу $a$ является результатом последовательности переходов, совершаемых автоматом $A$ при получении цепочки символов $h(a)$. Напомним,

что, хотя $h(a)$ может равняться $\varepsilon$, состоять из одного или нескольких символов, функция $\hat{\delta}$ определена так, чтобы справиться со всеми этими случаями.

С помощью индукции по $|w|$ легко показать, что $\hat{\gamma}(q_0, w) = \hat{\delta}(q_0, h(w))$. Поскольку допускающие состояния автоматов $A$ и $B$ совпадают, то $B$ допускает цепочку $w$ тогда и только тогда, когда $A$ допускает цепочку $h(w)$. Иными словами, $B$ допускает только те цепочки, которые принадлежат языку $h^{-1}(L)$. □

**Пример 4.17.** В этом примере обратный гомоморфизм и некоторые другие свойства замкнутости регулярных множеств используются для доказательства одного необычного свойства конечных автоматов. Предположим, что, допуская входную цепочку, некоторый автомат должен побывать в каждом состоянии хотя бы по одному разу. Точнее, допустим, что $A = (Q, \Sigma, \delta, q_0, F)$ — ДКА, и нас интересует язык $L$, состоящий из всех цепочек $w$ в алфавите $\Sigma^*$, для которых $\hat{\delta}(q_0, w)$ принадлежит $F$, и для каждого состояния $q$ из $Q$ существует некоторый префикс $x_q$ цепочки $w$, для которого $\hat{\delta}(q_0, x_q) = q$. Будет ли язык $L$ регулярным? Докажем, что такой язык регулярен, хотя доказательство довольно сложное.

Начнем с языка $M = L(A)$, т.е. множества цепочек, допускаемых автоматом $A$ обычным путем, независимо от того, в какие состояния он переходит, обрабатывая входную цепочку. Заметим, что $L \subseteq M$, так как определение языка $L$ накладывает дополнительные ограничения на цепочки из $L(A)$. Доказательство регулярности языка $L$ начинается с использования обратного гомоморфизма для вставки состояний автомата $A$ во входные символы. Точнее, определим новый алфавит $T$ как состоящий из символов, которые можно представить в виде троек $[paq]$, где $p$ и $q$ — состояния из $Q$, $a$ — символ из $\Sigma$, и $\delta(p, a) = q$.

Таким образом, символы алфавита $T$ представляют переходы автомата $A$. Важно понимать, что запись $[paq]$ представляет собой единый символ, а не конкатенацию трех символов. Можно обозначить этот символ одной буквой, но при этом трудно описать его связь с $p$, $q$ и $a$.

Теперь определим гомоморфизм $h([paq]) = a$ для всех $p$, $a$ и $q$. Это значит, что гомоморфизм $h$ удаляет из каждого символа алфавита $T$ компоненты, представляющие состояния, и оставляет только символ из $\Sigma$. Первый шаг доказательства регулярности языка $L$ состоит в построении языка $L_1 = h^{-1}(M)$. Поскольку язык $M$ регулярен, то согласно теореме 4.16 язык $L_1$ также регулярен. Цепочками языка $L_1$ будут цепочки из $M$, к каждому символу которых присоединяется пара состояний, представляющая некоторый переход автомата.

В качестве простого примера рассмотрим автомат с двумя состояниями, представленный на рис. 4.4, *а*. Алфавит $\Sigma = \{0, 1\}$, а алфавит $T$ состоит из четырех символов $[p0q]$, $[q0q]$, $[p1p]$ и $[q1q]$. Например, поскольку по символу 0 есть переход из $p$ в $q$, то $[p0q]$ — один из символов алфавита $T$. Так как цепочка 101 допускается этим автоматом, применив к ней обратный гомоморфизм $h^{-1}$, получим $2^3 = 8$ цепочек, две из которых, например, равны $[p1p][p0q][q1q]$ и $[q1q][q0q][p1p]$.

Теперь по языку $L_1$ построим язык $L$ с помощью ряда операций, сохраняющих регулярность языков. Наша первая цель — исключить все те цепочки языка $L_1$, в которых состояния указаны неправильно. Поскольку каждый символ вида $[paq]$ означает, что автомат был в состоянии $p$, прочитал $a$ и затем перешел в состояние $q$, последовательность таких символов, представляющая допускающее вычисление в автомате $A$, должна удовлетворять следующим трем условиям.

1. Первым состоянием в первом символе должно быть $q_0$ — начальное состояние $A$.
2. Каждый переход автомата должен начинаться там, где закончился предыдущий, т.е. первое состояние в символе должно равняться второму состоянию в предыдущем символе.
3. Второе состояние в последнем символе должно принадлежать $F$. Если выполняются первые два условия, то и это условие будет выполнено, поскольку каждая цепочка языка $L_1$ образована из цепочки, допускаемой автоматом $A$.

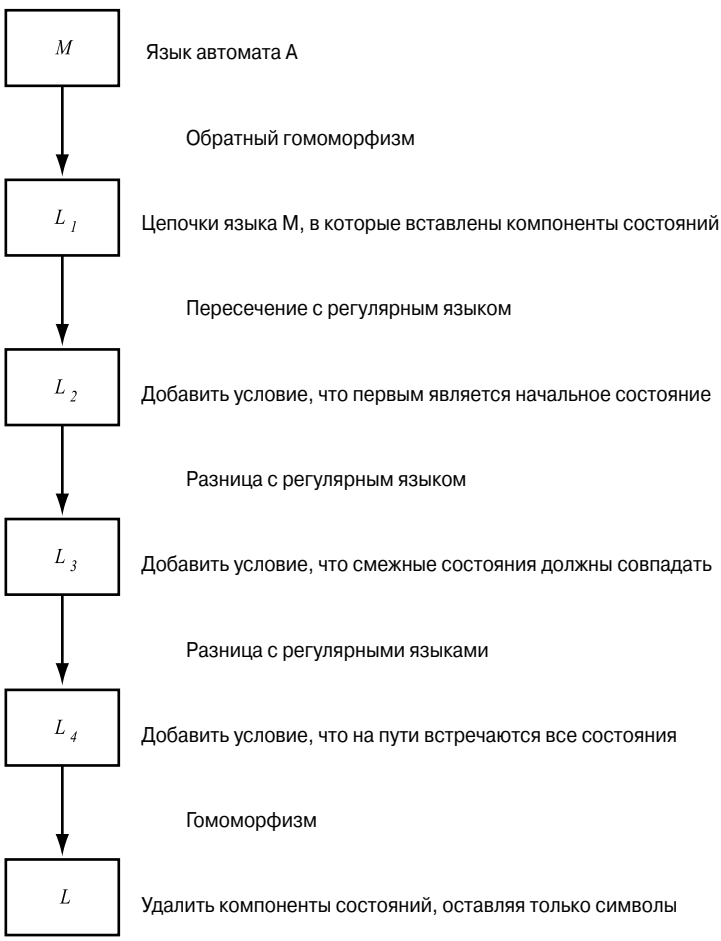


*Рис. 4.7. Построение языка* L *по языку* M *с помощью операций, сохраняющих регулярность языков*

План построения языка $L$ представлен на рис. 4.7.

Условие 1 обеспечивается пересечением языка $L_1$ со множеством цепочек, которые начинаются символом вида $[q_0aq]$ для некоторого символа $a$ и состояния $q$. Пусть $E_1$ — выражение $[q_0a_1q_1] + [q_0a_2q_2] + \ldots$, где $a_iq_i$ — все пары из $\Sigma \times Q$, для которых $\delta(q_0, a_i) = q_i$. Пусть $L_2 = L_1 \cap L(E_1T^*)$. Регулярное выражение $E_1T^*$ обозначает все цепочки из $T^*$, которые начинаются стартовым состоянием (здесь $T$ можно рассматривать как сумму всех его символов). Поэтому язык $L_2$ состоит из всех цепочек, полученных в результате применения обратного гомоморфизма $h^{-1}$ к языку $M$, у которых первым компонентом в первом символе является начальное состояние, т.е. язык $L_2$ удовлетворяет условию 1.

Чтобы обеспечить выполнение условия 2, проще всего вычесть из $L_2$ (используя операцию разности множеств) все цепочки, нарушающие это условие. Пусть $E_2$ — регулярное выражение, состоящее из суммы (объединения) конкатенаций всех пар символов, которые друг другу не подходят. Это все пары вида $[paq][rbs]$, где $q \neq r$. Тогда регулярное выражение $T^*E_2T^*$ обозначает все цепочки, не удовлетворяющие условию 2.

Теперь можно определить $L_3 = L_2 - L(T^*E_2T^*)$. Цепочки языка $L_3$ удовлетворяют условию 1, поскольку цепочки языка $L_2$ начинаются стартовым состоянием. Они также удовлетворяют условию 2, так как в результате вычитания $L(T^*E_2T^*)$ будут удалены все цепочки, для которых это условие не выполняется. Наконец, они удовлетворяют условию 3 (последнее состояние является допускающим), поскольку доказательство было начато с цепочек языка $M$, допускаемых автоматом $A$. В результате $L_3$ состоит из цепочек языка $M$ с состояниями допускающего вычисления такой цепочки, вставленными в каждый символ. Заметим, что язык $L_3$ регулярен, так как он построен из регулярного языка $M$ с помощью операций обратного гомоморфизма, пересечения и разности множеств, сохраняющих регулярность.

Напомним, что наша цель состоит в том, чтобы допустить только те цепочки из $M$, при обработке которых автомат проходит через каждое состояние. Выполнение этого условия можно обеспечить с помощью операции разности множеств. Пусть для каждого состояния $q$ регулярное выражение $E_q$ представляет собой сумму всех символов алфавита $T$, в которые не входит состояние $q$ ($q$ не стоит ни на первой, ни на последней позиции). В результате вычитания языка $L(E_q^*)$ из $L_3$ получим цепочки, представляющие допускающее вычисление автомата $A$ и проходящие через состояние $q$, по крайней мере, один раз. Если вычесть из $L_3$ языки $L(E_q^*)$ для всех $q$ из $Q$, то получим допускающие вычисления автомата $A$, проходящие через все состояния. Обозначим этот язык $L_4$. По теореме 4.10 язык $L_4$ также регулярен.

Последний шаг состоит в построении языка $L$ из $L_4$ с помощью исключения компонентов состояний, т.е. $L = h(L_4)$. Теперь $L$ является множеством цепочек в алфавите $\Sigma^*$, допускаемых автоматом $A$, причем при их обработке автомат проходит через каждое состояние, по крайней мере, один раз. Поскольку регулярные языки замкнуты относительно гомоморфизмов, делаем вывод, что язык $L$ регулярен. □

### 4.2.5. Упражнения к разделу 4.2

**4.2.1.** Пусть $h$ — гомоморфизм из алфавита $\{0, 1, 2\}$ в алфавит $\{a, b\}$, определенный как $h(0) = a$, $h(1) = ab$ и $h(2) = ba$:

а) (*) найдите $h(0120)$;

б) найдите $h(21120)$;

в) (*) найдите $h(L)$ для $L = L(\mathbf{01}^*\mathbf{2})$;

г) найдите $h(L)$ для $L = L(\mathbf{0} + \mathbf{12})$;

д) (*) найдите $h^{-1}(L)$ для $L = \{ababa\}$, т.е. языка, состоящего из одной-единственной цепочки *ababa*;

е) (!) найдите $h^{-1}(L)$ для $L = L(\mathbf{a}(\mathbf{ba})^*)$.

**4.2.2.** (*!) Если $L$ — язык, $a$ — символ, то $L/a$, *частное* $L$ и $a$, — это множество цепочек $w$, для которых $wa$ принадлежит $L$. Например, если $L = \{a, aab, baa\}$, то $L/a = \{\varepsilon, ba\}$. Докажите, что из регулярности $L$ следует регулярность $L/a$. *Указание*. Начните с ДКА для $L$ и рассмотрите множество допускающих состояний.

**4.2.3.** (!) Если $L$ — язык, $a$ — символ, то $a\backslash L$ — это множество цепочек $w$, для которых $aw$ принадлежит $L$. Например, если $L = \{a, aab, baa\}$, то $a\backslash L = \{\varepsilon, ab\}$. Докажите, что из регулярности $L$ следует регулярность $a\backslash L$. *Указание.* Вспомните, что регулярные языки замкнуты относительно обращения и операции деления из упражнения 4.2.2.

**4.2.4.** (!) Какие из следующих тождеств истинны?

а) $(L/a)a = L$ (в левой части представлена конкатенация языков $L/a$ и $\{a\}$).

б) $a(a\backslash L) = L$ (снова представлена конкатенация с языком $\{a\}$, но на этот раз слева).

в) $(La)/a = L$.

г) $a\backslash(aL) = L$.

**4.2.5.** Операция из упражнения 4.2.3 иногда рассматривается как "производная", а выражение $a\backslash L$ записывается как $\frac{dL}{da}$. Эти производные применяются к регулярным выражениям аналогично тому, как обычные производные применяются к арифметическим выражениям. Таким образом, если $R$ — регулярное выражение, то $\frac{dR}{da}$ обозначает то же, что и $\frac{dL}{da}$, если $L = L(R)$:

а) докажите, что $\frac{d(R+S)}{da} = \frac{dR}{da} + \frac{dS}{da}$;

б) (*!) напишите правило для "производной" от $RS$. *Указание*. Нужно рассмотреть два случая: язык $L(R)$ включает или не включает цепочку $\varepsilon$. Это правило

не совпадает с "правилом произведения" для обычных производных, но похоже на него;

в) (**!**) найдите "производную" от итерации, т.е. $\frac{d(R^*)}{da}$;

г) используя формулы (а)–(в), найдите производную регулярного выражения $(\mathbf{0} + \mathbf{1})^*\mathbf{011}$;

д) (*****) опишите такие языки $L$, для которых $\frac{dL}{d0} = \varnothing$;

е) (***!**) опишите такие языки $L$, для которых $\frac{dL}{d0} = L$.

**4.2.6.** (**!**) Докажите замкнутость регулярных языков относительно следующих операций:

а) $min(L) = \{w \mid w$ принадлежит $L$, но ни один собственный префикс цепочки $w$ не принадлежит $L\}$;

б) $max(L) = \{w \mid w$ принадлежит $L$, но для любого $x \neq \varepsilon$ цепочка $wx$ не принадлежит $L\}$;

в) $init(L) = \{w \mid$ для некоторого $x$ цепочка $wx$ принадлежит $L\}$.

*Указание.* Как и в упражнении 4.2.2, проще всего начать с ДКА для $L$ и преобразовать его для получения нужного языка.

**4.2.7.** (**!**) Если $w = a_1a_2\ldots a_n$ и $x = b_1b_2\ldots b_n$ — цепочки одинаковой длины, то определим $alt(w, x)$ как цепочку, в которой символы цепочек $w$ и $x$ чередуются, начиная с $w$, т.е. $a_1b_1a_2b_2\ldots a_nb_n$. Если $L$ и $M$ — языки, определим $alt(L, M)$ как множество цепочек вида $alt(w, x)$, где $w$ — произвольная цепочка из $L$, а $x$ — любая цепочка из $M$ такой же длины. Докажите, что из регулярности языков $L$ и $M$ следует регулярность языка $alt(L, M)$.

**4.2.8.** (***!!**) Пусть $L$ — язык. Определим $half(L)$ как множество первых половин цепочек языка $L$, т.е. множество $\{w \mid$ существует $x$, для которой $wx$ принадлежит $L$, причем $|x| = |w|\}$. Например, если $L = \{\varepsilon, 0010, 011, 010110\}$, то $half(L) = \{\varepsilon, 00, 010\}$. Заметим, что цепочки с нечетной длиной не влияют на $half(L)$. Докажите, что если язык $L$ регулярен, то $half(L)$ также регулярен.

**4.2.9.** (**!!**) Упражнение 4.2.8 можно распространить на многие функции, определяющие длину части цепочки. Если $f$ — функция, определенная на множестве целых чисел, то обозначим через $f(L)$ множество цепочек $\{w \mid$ существует цепочка $x$, для которой $|x| = f(|w|)$ и $wx$ принадлежит $L\}$. Например, операция $half$ соответствует тождественной функции $f(n) = n$, так как для цепочек из языка $half(L)$ выполняется $|x| = |w|$. Покажите, что если язык $L$ регулярен, то язык $f(L)$ также регулярен, где $f$ — одна из следующих функций:

а) $f(n) = 2n$ (т.е. используются первые трети цепочек);

б) $f(n) = n^2$ (длина выбираемой части цепочки равна квадратному корню длины оставшейся части цепочки);

в) $f(n) = 2^n$ (длина выбираемой части цепочки равна логарифму длины ее остатка).

**4.2.10.** (**!!**) Пусть $L$ — язык, не обязательно регулярный, в алфавите $\{0\}$, т.е. цепочки языка $L$ состоят из одних нулей. Докажите, что язык $L^*$ регулярен. *Указание*. На первый взгляд эта теорема кажется абсурдной. Чтобы проиллюстрировать истинность утверждения теоремы, приведем один небольшой пример. Рассмотрим язык $L = \{0^i \mid i$ — простое число$\}$, который, как известно, нерегулярен (см. пример 4.3). Нетрудно доказать, что если $j \geq 2$, то $0^j$ принадлежит $L^*$. Поскольку числа 2 и 3 — простые, цепочки 00 и 000 принадлежат $L$. Если $j$ — четное число, то $0^j$ можно получить, повторив $j/2$ раз цепочку 00, а если $j$ — нечетное, можно взять одну цепочку 000 и $(j - 3)/2$ цепочек 00. Следовательно, $L^* = \varepsilon + \mathbf{000}^*$.

**4.2.11.** (**!!**) Докажите замкнутость регулярных языков относительно следующей операции: $cycle(L) = \{w \mid$ цепочку $w$ можно представить в виде $w = xy$, где $yx$ принадлежит $L\}$. Например, если $L = \{01, 011\}$, то $cycle(L) = \{01, 10, 011, 110, 101\}$. *Указание.* Начните с ДКА для языка $L$ и постройте $\varepsilon$-НКА для $cycle(L)$.

**4.2.12.** (**!!**) Пусть $w_1 = a_0a_0a_1$, а $w_i = w_{i-1}w_{i-1}a_i$ для всех $i > 0$. Например, $w_3 = a_0a_0a_1a_0a_0a_1a_2a_0a_0a_1a_0a_0a_1a_2a_3$. Кратчайшим регулярным выражением для языка $L_n = \{w_n\}$, т.е. языка, состоящего из цепочки $w_n$, будет сама $w_n$, причем длина этого выражения равна $2^{n+1} - 1$. Однако, если применить операцию пересечения, то для языка $L_n$ можно записать выражение длиной $O(n^2)$. Найдите такое выражение. *Указание.* Найдите $n$ языков с регулярными выражениями длины $O(n)$, пересечение которых равно $L_n$.

**4.2.13.** Свойства замкнутости можно использовать для доказательства нерегулярности некоторых языков. Докажите, что язык

$L_{0n1n} = \{0^n1^n \mid n \geq 0\}$

нерегулярен. Докажите нерегулярность следующих языков, преобразовав их с помощью операций, сохраняющих регулярность, в язык $L_{0n1n}$:

а) (*****) $\{0^i0^j \mid i \neq j\}$;

б) $\{0^n1^m2^{n-m} \mid n \geq m \geq 0\}$.

**4.2.14.** В теореме 4.8 представлена "конструкция произведения", в которой по двум данным ДКА построен ДКА, допускающий пересечение языков данных автоматов:

а) покажите, как построить конструкцию произведения для НКА (без $\varepsilon$-переходов);

б) (**!**) продемонстрируйте, как построить конструкцию произведения для $\varepsilon$-НКА;

в) (*****) покажите, как изменить конструкцию для произведения так, чтобы результирующий ДКА допускал разность языков двух данных ДКА;

г) измените конструкцию для произведения так, чтобы результирующий ДКА допускал объединение языков двух данных ДКА.

**4.2.15.** В доказательстве теоремы 4.8 утверждалось, что с помощью индукции по длине цепочки $w$ можно доказать следующее равенство:

$\hat{\delta}((q_L, q_M), w) = (\hat{\delta}_L(q_L, w), \hat{\delta}_M(q_M, w))$.

Приведите это доказательство.

**4.2.16.** Завершите доказательство теоремы 4.14, рассмотрев случаи, когда выражение $E$ является конкатенацией двух подвыражений или итерацией некоторого выражения.

**4.2.17.** В теореме 4.16 пропущено доказательство индукцией по длине цепочки $w$ того, что $\hat{\gamma}(q_0, w) = \hat{\delta}(q_0, h(w))$. Восполните этот пробел.

## 4.3. Свойства разрешимости регулярных языков

В этом разделе мы сформируем важные вопросы, связанные с регулярными языками. Сначала нужно разобраться, что значит задать вопрос о некотором языке. Типичный язык бесконечен, поэтому бессмысленно предъявлять кому-нибудь цепочки этого языка и задавать вопрос, требующий проверки бесконечного множества цепочек. Гораздо разумнее использовать одно из конечных представлений языка, а именно: ДКА, НКА, $\varepsilon$-НКА или регулярное выражение.

Очевидно, что представленные таким образом языки будут регулярными. В действительности не существует способа представления абсолютно произвольных языков. В следующих главах предлагаются конечные методы описания более широких классов, чем класс регулярных языков, и можно будет рассматривать вопросы о языках из них. Однако алгоритмы разрешения многих вопросов о языках существуют только для класса регулярных языков. Эти же вопросы становятся "неразрешимыми" (не существует алгоритмов ответов на эти вопросы), если они поставлены с помощью более "выразительных" обозначений (используемых для выражения более широкого множества языков), чем представления, разработанные для регулярных языков.

Начнем изучение алгоритмов для вопросов о регулярных языках, рассмотрев способы, которыми одно представление языка преобразуется в другое. В частности, рассмотрим временну*ю* сложность алгоритмов, выполняющих преобразования. Затем рассмотрим три основных вопроса о языках.

1. Является ли описываемый язык пустым множеством?
2. Принадлежит ли некоторая цепочка $w$ представленному языку?
3. Действительно ли два разных описания представляют один и тот же язык? (Этот вопрос часто называют "эквивалентностью" языков.)

### 4.3.1. Преобразования различных представлений языков

Из главы 3 известно, что каждое из четырех представлений регулярных языков можно преобразовать в любое из остальных трех. На рис. 3.1 представлены переходы от одного представления к другому. Хотя существуют алгоритмы для любого из этих преобразований, иногда нас интересует не только осуществимость некоторого преобразования, но и время, необходимое для его выполнения. В частности, важно отличать алгоритмы, которые занимают экспоненциальное время (время как функция от размера входных данных) и, следовательно, могут быть выполнены только для входных данных сравнительно небольших размеров, от тех алгоритмов, время выполнения которых является линейной, квадратичной или полиномиальной с малой степенью функцией от размера входных данных. Последние алгоритмы "реалистичны", так как их можно выполнить для гораздо более широкого класса экземпляров задачи. Рассмотрим временную сложность каждого из обсуждавшихся преобразований.

**Преобразование НКА в ДКА**

Время выполнения преобразования НКА или $\varepsilon$-НКА в ДКА может быть экспоненциальной функцией от количества состояний НКА. Начнем с того, что вычисление $\varepsilon$-замыкания $n$ состояний занимает время $O(n^3)$. Необходимо найти все дуги с меткой $\varepsilon$, ведущие от каждого из $n$ состояний. Если есть $n$ состояний, то может быть не более $n^2$ дуг. Разумное использование системных ресурсов и хорошо спроектированные структуры данных гарантируют, что время исследования каждого состояния не превысит $O(n^2)$. В действительности для однократного вычисления всего $\varepsilon$-замыкания можно использовать такой алгоритм транзитивного замыкания, как алгоритм Уоршалла (Warshall)[5].

После вычисления $\varepsilon$-замыкания можно перейти к синтезу ДКА с помощью конструкции подмножеств. Основное влияние на расход времени оказывает количество состояний ДКА, которое может равняться $2^n$. Для каждого состояния можно вычислить переходы за время $O(n^3)$, используя $\varepsilon$-замыкание и таблицу переходов НКА для каждого входного символа. Предположим, нужно вычислить $\delta(\{q_1, q_2, \ldots, q_k\}, a)$ для ДКА. Из каждого состояния $q_i$ можно достичь не более $n$ состояний вдоль путей с меткой $\varepsilon$, и каждое из этих состояний может иметь не более, чем $n$ дуг с меткой $a$. Создав массив, проиндексированный состояниями, можно вычислить объединение не более $n$ множеств, состоящих из не более, чем $n$ состояний, за время, пропорциональное $n^2$.

Таким способом для каждого состояния $q_i$ можно вычислить множество состояний, достижимых из $q_i$ вдоль пути с меткой $a$ (возможно, включая дуги, отмеченные $\varepsilon$). Поскольку $k \le n$, то существует не более $n$ таких состояний $q_i$, и для каждого из них вычис-

---

[5] Обсуждение алгоритмов транзитивного замыкания можно найти в книге A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1984. (А. Ахо, Дж. Хопкрофт, Дж. Ульман. *Структуры данных и алгоритмы*, М.: Издательский дом "Вильямс", 2000.)

ление достижимых состояний занимает время $O(n^2)$. Таким образом, общее время вычисления достижимых состояний равно $O(n^3)$. Для объединения множеств достижимых состояний потребуется только $O(n^2)$ дополнительного времени, следовательно, вычисление одного перехода ДКА занимает время $O(n^3)$.

Заметим, что количество входных символов считается постоянным и не зависит от $n$. Таким образом, как в этой, так и в других оценках времени работы количество входных символов не рассматривается. Размер входного алфавита влияет только на постоянный коэффициент, скрытый в обозначении "О большого".

Итак, время преобразования НКА в ДКА, включая ситуацию, когда НКА содержит $\varepsilon$-переходы, равно $O(n^3 2^n)$. Конечно, на практике обычно число состояний, которые строятся, намного меньше $2^n$. Иногда их всего лишь $n$. Поэтому можно установить оценку времени работы равной $O(n^3 s)$, где $s$ — это число состояний, которые в действительности есть у ДКА.

**Преобразование ДКА в НКА**

Это простое преобразование, занимающее время $O(n)$ для ДКА с $n$ состояниями. Все, что необходимо сделать, — изменить таблицу переходов для ДКА, заключив в скобки {} состояния, а также добавить столбец для $\varepsilon$, если нужно получить $\varepsilon$-НКА. Поскольку число входных символов (т.е. ширина таблицы переходов) считается постоянным, копирование и обработка таблицы занимает время $O(n)$.

**Преобразование автомата в регулярное выражение**

Рассмотрев конструкцию из раздела 3.2.1, заметим, что на каждом из $n$ этапов (где $n$ — число состояний ДКА) размер конструируемого регулярного выражения может увеличиться в четыре раза, так как каждое выражение строится из четырех выражений предыдущего цикла. Таким образом, простая запись $n^3$ выражений может занять время $O(n^3 4^n)$. Улучшенная конструкция из раздела 3.2.2 уменьшает постоянный коэффициент, но не влияет на экспоненциальность этой задачи (в наихудшем случае).

Аналогичная конструкция требует такого же времени работы, если преобразуется НКА, или даже $\varepsilon$-НКА, но это здесь не доказывается. Однако использование конструкции для НКА имеет большое значение. Если сначала преобразовать НКА в ДКА, а затем ДКА — в регулярное выражение, то на это потребуется время $O(n^3 4^{n3^{2n}})$, которое является дважды экспоненциальным.

**Преобразование регулярного выражения в автомат**

Для преобразования регулярного выражения в $\varepsilon$-НКА потребуется линейное время. Необходимо эффективно проанализировать регулярное выражение, используя метод, занимающий время $O(n)$ для регулярного выражения длины $n$[6]. В результате получим де-

[6] Методы анализа, с помощью которых можно выполнить эту задачу за время O(n), обсуждаются в книге A. V. Aho, R. Sethi, and J. D. Ullman, *Compiler Design: Principles, Tools, and Techniques,* Addison-Wesley, 1986. (А. Ахо, Р. Сети, Дж. Ульман. *Компиляторы: принципы, инструменты и технологии*, М.: Издательский дом "Вильямс", 2001.)

рево с одним узлом для каждого символа регулярного выражения (хотя скобки в этом дереве не встречаются, поскольку они только управляют разбором выражения).

Полученное дерево заданного регулярного выражения можно обработать, конструируя $\varepsilon$-НКА для каждого узла. Правила преобразования регулярного выражения, представленные в разделе 3.2.3, никогда не добавляют более двух состояний и четырех дуг для каждого узла дерева выражения. Следовательно, как число состояний, так и число дуг результирующего $\varepsilon$-НКА равны $O(n)$. Кроме того, работа по созданию этих элементов в каждом узле дерева анализа является постоянной при условии, что функция, обрабатывающая каждое поддерево, возвращает указатели в начальное и допускающие состояния этого автомата.

Приходим к выводу, что построение $\varepsilon$-НКА по регулярному выражению занимает время, линейно зависящее от размера выражения. Можно исключить $\varepsilon$-переходы из $\varepsilon$-НКА с $n$ состояниями, преобразовав его в обычный НКА за время $O(n^3)$ и не увеличив числа состояний. Однако преобразование в ДКА может занять экспоненциальное время.

### 4.3.2. Проверка пустоты регулярных языков

На первый взгляд ответ на вопрос “является ли регулярный язык $L$ пустым?” кажется очевидным: язык $\varnothing$ пуст, а все остальные регулярные языки — нет. Однако, как говорилось в начале раздела 4.3, при постановке задачи явный перечень цепочек языка $L$ не приводится. Обычно задается некоторое представление языка $L$, и нужно решить, обозначает ли оно язык $\varnothing$, или нет.

Если язык задан с помощью конечного автомата любого вида, то вопрос пустоты состоит в том, есть ли какие-нибудь пути из начального состояния в допускающие. Если есть, то язык непуст, а если все допускающие состояния изолированы от начального, то язык пуст. Ответ на вопрос, можно ли перейти в допускающее состояние из начального, является простым примером достижимости в графах, подобным вычислению $\varepsilon$-замыкания, рассмотренному в разделе 2.5.3. Искомый алгоритм можно сформулировать следующим рекурсивным образом.

**Базис.** Начальное состояние всегда достижимо из начального состояния.

**Индукция.** Если состояние $q$ достижимо из начального, и есть дуга из $q$ в состояние $p$ с любой меткой (входным символом или $\varepsilon$, если рассматривается $\varepsilon$-НКА), то $p$ также достижимо.

Таким способом можно вычислить все множество достижимых состояний. Если среди них есть допускающее состояние, то ответом на поставленный вопрос будет “нет” (язык данного автомата *не* пуст), в противном случае ответом будет “да” (язык пуст). Заметим, что если автомат имеет $n$ состояний, то вычисление множества достижимых состояний занимает время не более $O(n^2)$ (практически это время пропорционально числу дуг на диаграмме переходов автомата, которое может быть и меньше $n^2$).

Если язык $L$ представлен регулярным выражением, а не автоматом, то можно преобразовать это выражение в $\varepsilon$-НКА, а далее продолжить так, как описано выше. Поскольку автомат, полученный в результате преобразования регулярного выражения длины $n$, содержит не более $O(n)$ состояний и переходов, для выполнения алгоритма потребуется время $O(n)$.

Можно проверить само выражение — пустое оно, или нет. Сначала заметим, что если в данном выражении ни разу не встречается $\varnothing$, то его язык гарантированно не пуст. Если же в выражении встречается $\varnothing$, то язык такого выражения не обязательно пустой. Используя следующие рекурсивные правила, можно определить, представляет ли заданное регулярное выражение пустой язык.

**Базис.** $\varnothing$ обозначает пустой язык, но $\varepsilon$ и **a** для любого входного символа $a$ обозначают не пустой язык.

**Индукция.** Пусть $R$ — регулярное выражение. Нужно рассмотреть четыре варианта, соответствующие возможным способам построения этого выражения.

1. $R = R_1 + R_2$. $L(R)$ пуст тогда и только тогда, когда оба языка $L(R_1)$ и $L(R_2)$ пусты.
2. $R = R_1R_2$. $L(R)$ пуст тогда и только тогда, когда хотя бы один из языков $L(R_1)$ и $L(R_2)$ пуст.
3. $R = R_1^*$. $L(R)$ не пуст: он содержит цепочку $\varepsilon$.
4. $R = (R_1)$. $L(R)$ пуст тогда и только тогда, кода $L(R_1)$ пуст, так как эти языки равны.

### 4.3.3. Проверка принадлежности регулярному языку

Следующий важный вопрос состоит в том, принадлежит ли данная цепочка $w$ данному регулярному языку $L$. В то время, как цепочка $w$ задается явно, язык $L$ представляется с помощью автомата или регулярного выражения.

Если язык $L$ задан с помощью ДКА, то алгоритм решения данной задачи очень прост. Имитируем ДКА, обрабатывающий цепочку входных символов $w$, начиная со стартового состояния. Если ДКА заканчивает в допускающем состоянии, то цепочка $w$ принадлежит этому языку, в противном случае — нет. Этот алгоритм является предельно быстрым. Если $|w| = n$ и ДКА представлен с помощью подходящей структуры данных, например, двухмерного массива (таблицы переходов), то каждый переход требует постоянного времени, а вся проверка занимает время $O(n)$.

Если $L$ представлен способом, отличным от ДКА, то преобразуем это представление в ДКА и применим описанную выше проверку. Такой подход может занять время, экспоненциально зависящее от размера данного представления и линейное относительно $|w|$. Однако, если язык задан с помощью НКА или $\varepsilon$-НКА, то намного проще и эффективнее непосредственно проимитировать этот НКА. Символы цепочки $w$ обрабатываются по одному, и запоминается множество состояний, в которые НКА может попасть после

прохождения любого пути, помеченного префиксом цепочки $w$. Идея такой имитации была представлена на рис. 2.10.

Если длина цепочки $w$ равна $n$, а количество состояний НКА равно $s$, то время работы этого алгоритма равно $O(ns^2)$. Чтобы обработать очередной входной символ, необходимо взять предыдущее множество состояний, число которых не больше $s$, и для каждого из них найти следующее состояние. Затем объединяем не более $s$ множеств, состоящих из не более, чем $s$ состояний, для чего нужно время $O(s^2)$.

Если заданный НКА содержит $\varepsilon$-переходы, то перед тем, как начать имитацию, необходимо вычислить $\varepsilon$-замыкание. Такая обработка очередного входного символа $a$ состоит из двух стадий, каждая из которых занимает время $O(s^2)$. Сначала для предыдущего множества состояний находим последующие состояния при входе $a$. Далее вычисляем $\varepsilon$-замыкание полученного множества состояний. Начальным множеством состояний для такого моделирования будет $\varepsilon$-замыкание начального состояния НКА.

И наконец, если язык $L$ представлен регулярным выражением, длина которого $s$, то за время $O(s)$ можно преобразовать это выражение в $\varepsilon$-НКА с числом состояний не больше $2s$. Выполняем описанную выше имитацию, что требует $O(ns^2)$ времени для входной цепочки $w$ длиной $n$.

### 4.3.4. Упражнения к разделу 4.3

**4.3.1.** (*) Приведите алгоритм определения, является ли регулярный язык $L$ бесконечным. *Указание.* Используйте лемму о накачке для доказательства того, что если язык содержит какую-нибудь цепочку, длина которой превышает определенную нижнюю границу, то этот язык должен быть бесконечным.

**4.3.2.** Приведите алгоритм определения, содержит ли регулярный язык $L$ по меньшей мере 100 цепочек.

**4.3.3.** Пусть $L$ — регулярный язык в алфавите $\Sigma$. Приведите алгоритм проверки равенства $L = \Sigma^*$, т.е. содержит ли язык $L$ все цепочки в алфавите $\Sigma$.

**4.3.4.** Приведите алгоритм определения, содержат ли регулярные языки $L_1$ и $L_2$ хотя бы одну общую цепочку.

**4.3.5.** Пусть $L_1$ и $L_2$ — два регулярных языка с одним и тем же алфавитом $\Sigma$. Приведите алгоритм определения, существует ли цепочка из $\Sigma^*$, которая не принадлежит ни $L_1$, ни $L_2$.

## 4.4. Эквивалентность и минимизация автоматов

В отличие от предыдущих вопросов — пустоты и принадлежности, алгоритмы решения которых были достаточно простыми, вопрос о том, определяют ли два представления двух регулярных языков один и тот же язык, требует значительно бо́льших интеллектуальных

усилий. В этом разделе мы обсудим, как проверить, являются ли два описания регулярных языков *эквивалентными* в том смысле, что они задают один и тот же язык. Важным следствием этой проверки является возможность минимизации ДКА, т.е. для любого ДКА можно найти эквивалентный ему ДКА с минимальным количеством состояний. По существу, такой ДКА один: если даны два эквивалентных ДКА с минимальным числом состояний, то всегда можно переименовать состояния так, что эти ДКА станут одинаковыми.

### 4.4.1. Проверка эквивалентности состояний

Начнем с вопроса об эквивалентности состояний одного ДКА. Наша цель — понять, когда два различных состояния $p$ и $q$ можно заменить одним, работающим одновременно как $p$ и $q$. Будем говорить, что состояния $p$ и $q$ *эквивалентны*, если

- для всех входных цепочек $w$ состояние $\hat{\delta}(p, w)$ является допускающим тогда и только тогда, когда состояние $\hat{\delta}(q, w)$ — допускающее.

Менее формально, эквивалентные состояния $p$ и $q$ невозможно различить, если просто проверить, допускает ли автомат данную входную цепочку, начиная работу в одном (неизвестно, каком именно) из этих состояний. Заметим, что состояния $\hat{\delta}(p, w)$ и $\hat{\delta}(q, w)$ могут и не совпадать — лишь бы оба они были либо допускающими, либо недопускающими.

Если два состояния $p$ и $q$ не эквивалентны друг другу, то будем говорить, что они *различимы*, т.е. существует хотя бы одна цепочка $w$, для которой одно из состояний $\hat{\delta}(p, w)$ и $\hat{\delta}(q, w)$ является допускающим, а другое — нет.

**Пример 4.18.** Рассмотрим ДКА на рис. 4.8. Функцию переходов этого автомата обозначим через $\delta$. Очевидно, что некоторые пары состояний не эквивалентны, например $C$ и $G$, потому что первое из них допускающее, а второе — нет. Пустая цепочка различает эти состояния, так как $\hat{\delta}(C, \varepsilon)$ — допускающее состояние, а $\hat{\delta}(G, \varepsilon)$ — нет.

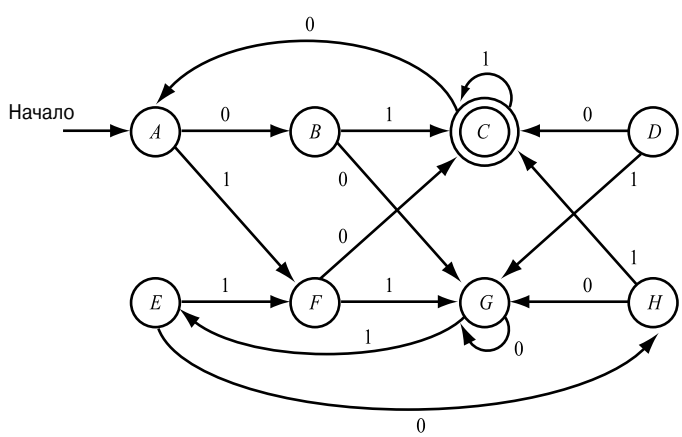


*Рис. 4.8. Автомат с эквивалентными состояниями*

Рассмотрим состояния $A$ и $G$. Различить их с помощью цепочки $\varepsilon$ невозможно, так как оба они недопускающие. 0 также не различает их, поскольку по входу 0 автомат переходит в состояния $B$ и $G$, соответственно, а оба эти состояния недопускающие. Однако цепочка 01 различает $A$ и $G$, так как $\hat{\delta}(A, 01) = C$, $\hat{\delta}(G, 01) = E$, состояние $C$ — допускающее, а $E$ — нет. Для доказательства неэквивалентности $A$ и $G$ достаточно любой входной цепочки, переводящей автомат из состояний $A$ и $G$ в состояния, одно из которых является допускающим, а второе — нет.

Рассмотрим состояния $A$ и $E$. Ни одно из них не является допускающим, так что цепочка $\varepsilon$ не различает их. По входу 1 автомат переходит и из $A$, и из $E$ в состояние $F$. Таким образом, ни одна входная цепочка, начинающаяся с 1, не может различить их, поскольку $\hat{\delta}(A, 1x) = \hat{\delta}(E, 1x)$ для любой цепочки $x$.

Рассмотрим поведение в состояниях $A$ и $E$ на входах, которые начинаются с 0. Из состояний $A$ и $E$ автомат переходит в $B$ и $H$, соответственно. Так как оба эти состояния недопускающие, сама по себе цепочка 0 не отличает $A$ от $E$. Однако состояния $B$ и $H$ не помогут: по входу 1 оба эти состояния переходят в $C$, а по входу 0 — в $G$. Значит, ни одна входная цепочка, начинающаяся с 0, не может различить состояния $A$ и $E$. Следовательно, ни одна входная цепочка не различает состояния $A$ и $E$, т.е. они эквивалентны. □

Для того чтобы найти эквивалентные состояния, нужно выявить все пары различимых состояний. Как ни странно, но если найти все пары состояний, различимых в соответствии с представленным ниже алгоритмом, то те пары состояний, которые найти не удастся, будут эквивалентными. Алгоритм, который называется *алгоритмом заполнения таблицы*, состоит в рекурсивном обнаружении пар различимых состояний ДКА $A = (Q, \Sigma, \delta, q_0, F)$.

**Базис**. Если состояние $p$ — допускающее, а $q$ — не допускающее, то пара состояний $\{p, q\}$ различима.

**Индукция**. Пусть $p$ и $q$ — состояния, для которых существует входной символ $a$, приводящий их в различимые состояния $r = \delta(p, a)$ и $s = \delta(q, a)$. Тогда $\{p, q\}$ — пара различимых состояний. Это правило очевидно, потому что должна существовать цепочка $w$, отличающая $r$ от $s$, т.е. только одно из состояний $\hat{\delta}(r, w)$ и $\hat{\delta}(s, w)$ является допускающим. Тогда цепочка $aw$ отличает $p$ от $q$, так как $\hat{\delta}(p, aw)$ и $\hat{\delta}(q, aw)$ — это та же пара состояний, что и $\hat{\delta}(r, w)$ и $\hat{\delta}(s, w)$.

**Пример 4.19.** Выполним алгоритм заполнения таблицы для ДКА, представленного на рис. 4.8. Окончательный вариант таблицы изображен на рис. 4.9, где $x$ обозначает пары различимых состояний, а пустые ячейки указывают пары эквивалентных состояний. Сначала в таблице нет ни одного $x$.

В базисном случае, поскольку $C$ — единственное допускающее состояние, записываем $x$ в каждую пару состояний, в которую входит $C$. Зная некоторые пары различимых состояний, можно найти другие. Например, поскольку пара $\{C, H\}$ различима, а состоя-

ния $E$ и $F$ по входу 0 переходят в $H$ и $C$, соответственно, то пара $\{E, F\}$ также различима. Фактически, все $x$ на рис. 4.9, за исключением пары $\{A, G\}$, получаются очень просто: посмотрев на переходы из каждой пары состояний по символам 0 или 1, обнаружим (для одного из этих символов), что одно состояние переходит в $C$, а другое — нет. Различимость пары состояний $\{A, G\}$ видна в следующем цикле, поскольку по символу 1 они переходят в $F$ и $E$, соответственно, а различимость состояний $\{E, F\}$ уже установлена.

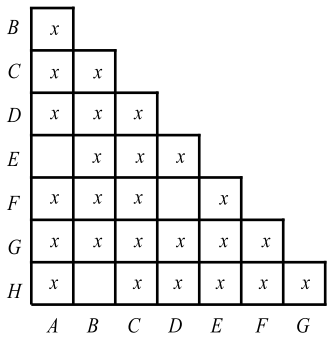


*Рис. 4.9. Таблица неэквивалентности состояний*

Однако обнаружить другие пары различимых состояний невозможно. Следовательно, оставшиеся три пары состояний $\{A, E\}$, $\{B, H\}$ и $\{D, F\}$ эквивалентны. Выясним, почему нельзя утверждать, что пара состояний $\{A, E\}$ различима. По входному символу 0 состояния $A$ и $E$ переходят в $B$ и $H$, соответственно, а про эту пару пока неизвестно, различима она, или нет. По символу 1 оба состояния $A$ и $E$ переходят в $F$, так что нет никакой надежды различить их этим способом. Остальные две пары, $\{B, H\}$ и $\{D, F\}$, различить нельзя, поскольку у них одинаковые переходы как по символу 0, так и по 1. Таким образом, алгоритм заполнения таблицы останавливается на таблице, представленной на рис. 4.9, и корректно определяет эквивалентные и различимые состояния. □

**Теорема 4.20.** Если два состояния не различаются с помощью алгоритма заполнения таблицы, то они эквивалентны.

**Доказательство**. Снова рассмотрим ДКА $A = (Q, \Sigma, \delta, q_0, F)$. Предположим, что утверждение теоремы неверно, т.е. существует хотя бы одна пара состояний $\{p, q\}$, для которой выполняются следующие условия.

1. Состояния $p$ и $q$ различимы, т.е. существует некоторая цепочка $w$, для которой только одно из состояний $\hat{\delta}(p,w)$ и $\hat{\delta}(q,w)$ является допускающим.
2. Алгоритм заполнения таблицы не может обнаружить, что состояния $p$ и $q$ различимы.

Назовем такую пару состояний *плохой парой*.

Если существуют плохие пары, то среди них должны быть такие, которые различимы с помощью кратчайших из всех цепочек, различающих плохие пары. Пусть пара

$\{p, q\}$ — плохая, а $w = a_1a_2\ldots a_n$ — кратчайшая из всех цепочек, различающих $p$ и $q$. Тогда только одно из состояний $\hat{\delta}(p, w)$ и $\hat{\delta}(q, w)$ является допускающим.

Заметим, что цепочка $w$ не может быть $\varepsilon$, так как, если некоторая пара состояний различается с помощью $\varepsilon$, то ее можно обнаружить, выполнив базисную часть алгоритма заполнения таблицы. Следовательно, $n \geq 1$.

Рассмотрим состояния $r = \delta(p, a_1)$ и $s = \delta(q, a_1)$. Эти состояния можно различить с помощью цепочки $a_2a_3\ldots a_n$, поскольку она переводит $r$ и $s$ в состояния $\hat{\delta}(p, w)$ и $\hat{\delta}(q, w)$. Однако цепочка, отличающая $r$ от $s$, короче любой цепочки, различающей плохую пару. Следовательно, $\{r, s\}$ не может быть плохой парой, и алгоритм заполнения таблицы должен был обнаружить, что эти состояния различимы.

Но индуктивная часть алгоритма заполнения таблицы не остановится, пока не придет к выводу, что состояния $p$ и $q$ также различимы, поскольку уже обнаружено, что состояние $\delta(p, a_1) = r$ отличается от $\delta(q, a_1) = s$. Получено противоречие с предположением о том, что существуют плохие пары состояний. Но если плохих пар нет, то любую пару различимых состояний можно обнаружить с помощью алгоритма заполнения таблицы, и теорема доказана. □

## 4.4.2. Проверка эквивалентности регулярных языков

Эквивалентность регулярных языков легко проверяется с помощью алгоритма заполнения таблицы. Предположим, что языки $L$ и $M$ представлены, например, один — регулярным выражением, а второй — некоторым НКА. Преобразуем каждое из этих представлений в ДКА. Теперь представим себе ДКА, состояния которого равны объединению состояний автоматов для языков $L$ и $M$. Технически этот ДКА содержит два начальных состояния, но фактически при проверке эквивалентности начальное состояние не играет никакой роли, поэтому любое из этих двух состояний можно принять за единственное начальное.

Далее проверяем эквивалентность начальных состояний двух заданных ДКА, используя алгоритм заполнения таблицы. Если они эквивалентны, то $L = M$, а если нет, то $L \neq M$.

**Пример 4.21.** Рассмотрим два ДКА (рис. 4.10). Каждый ДКА допускает пустую цепочку и все цепочки, которые заканчиваются символом 0, т.е. язык регулярного выражения $\varepsilon + (\mathbf{0} + \mathbf{1})^*\mathbf{0}$. Можно представить, что на рис. 4.10 изображен один ДКА, содержащий пять состояний от $A$ до $E$. Если применить алгоритм заполнения таблицы к этому автомату, то в результате получим таблицу, представленную на рис. 4.11.

Чтобы заполнить эту таблицу, начнем с размещения $x$ в ячейках, соответствующих тем парам состояний, из которых только одно является допускающим. Оказывается, что больше делать ничего не нужно. Остальные четыре пары $\{A, C\}$, $\{A, D\}$, $\{C, D\}$ и $\{B, E\}$ являются парами эквивалентных состояний. Необходимо убедиться, что в индуктивной части алгоритма заполнения таблицы различимые состояния не обнаружены. Например, с помощью такой таблицы (см. рис. 4.11) нельзя различить пару $\{A, D\}$, так как по символу 0 эти состояния переходят сами в себя, а по 1 — в пару состояний $\{B, E\}$, которая

осталась неразличимой. Поскольку в результате проверки установлено, что состояния $A$ и $C$ эквиваленты и являются начальными у двух заданных автоматов, делаем вывод, что эти ДКА действительно допускают один и тот же язык. □

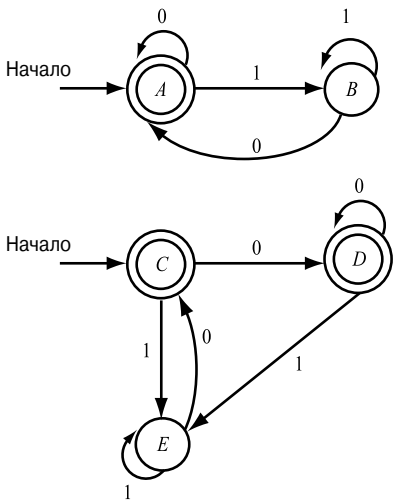


*Рис. 4.10. Два эквивалентных ДКА*

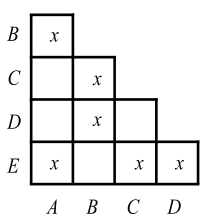


*Рис. 4.11. Таблица различимости для автоматов, представленных на рис. 4.10*

Время заполнения таблицы, а значит и время проверки эквивалентности двух состояний, полиномиально относительно числа состояний. Если число состояний равно $n$, то количество пар состояний равно $\binom{n}{2}$, или $n(n-1)/2$. За один цикл рассматриваются все пары состояний, чтобы определить, является ли одна из пар состояний-преемников различимой. Значит, один цикл занимает время не больше $O(n^2)$. Кроме того, если в некотором цикле не обнаружены новые пары различимых состояний, то алгоритм заканчивается. Следовательно, количество циклов не превышает $O(n^2)$, а верхняя граница времени заполнения таблицы равна $O(n^4)$.

Однако с помощью более аккуратно построенного алгоритма можно заполнить таблицу за время $O(n^2)$. С этой целью для каждой пары состояний $\{r, s\}$ необходимо составить список пар состояний $\{p, q\}$, "зависящих" от $\{r, s\}$, т.е., если пара $\{r, s\}$ различима, то $\{p, q\}$ также различима. Вначале такие списки создаются путем рассмотрения каждой

пары состояний $\{p, q\}$, и для каждого входного символа $a$ (а их число фиксировано) пара $\{p, q\}$ вносится в список для пары состояний-преемников $\{\delta(p, a), \delta(q, a)\}$.

Если обнаруживается, что пара $\{r, s\}$ различима, то в списке этой пары каждая пока неразличимая пара отмечается как различимая и помещается в очередь пар, списки которых нужно проверить аналогичным образом.

Общее время работы этого алгоритма пропорционально сумме длин списков, так как каждый раз либо что-то добавляется в списки (инициализация), либо в первый и последний раз проверяется наличие некоторой пары в списке (когда проходим по списку пары, признанной различимой). Так как размер входного алфавита считается постоянным, то каждая пара состояний попадает в $O(1)$ списков. Поскольку всего пар $O(n^2)$, суммарное время также $O(n^2)$.

### 4.4.3. Минимизация ДКА

Еще одним важным следствием проверки эквивалентности состояний является возможность "минимизации" ДКА. Это значит, что для каждого ДКА можно найти эквивалентный ему ДКА с наименьшим числом состояний. Более того, для данного языка существует единственный минимальный ДКА (с точностью до выбираемого нами обозначения состояний).

Основная идея минимизации ДКА состоит в том, что понятие эквивалентности состояний позволяет объединять состояния в блоки следующим образом.

1. Все состояния в блоке эквиваленты.
2. Любые два состояния, выбранные из разных блоков, неэквивалентны.

**Пример 4.22.** Рассмотрим рис. 4.9, на котором представлены эквивалентность и различимость для состояний, изображенных на рис. 4.8. Эти состояния разбиваются на эквивалентные блоки следующим образом: $(\{A, E\}, \{B, H\}, \{C\}, \{D, F\}, \{G\})$. Заметим, что каждая пара эквивалентных состояний помещена в отдельный блок, а состояния, отличимые от всех остальных, образуют отдельные блоки.

Для автомата, представленного на рис. 4.10, разбиение на блоки имеет вид $(\{A, C, D\}, \{B, E\})$. Этот пример показывает, что в блоке может быть более двух состояний. Может показаться случайностью, что состояния $A$, $C$ и $D$ помещены в один блок потому, что каждые два из них эквивалентны и ни одно из этих состояний не эквивалентно еще какому-нибудь состоянию, кроме этих. Однако следующая теорема утверждает, что такая ситуация следует из определения эквивалентности состояний. □

**Теорема 4.23.** Эквивалентность состояний транзитивна, т.е., если для некоторого ДКА $A = (Q, \Sigma, \delta, q_0, F)$ состояние $p$ эквивалентно $q$, а $q$ — $r$, то состояния $p$ и $r$ также эквивалентны.

**Доказательство**. Естественно ожидать, что любое отношение, называемое "эквивалентностью", обладает свойством транзитивности. Однако, просто назвав какое-то от-

ношение “эквивалентностью”, нельзя гарантировать, что оно транзитивно — это нужно доказать.

Предположим, что $\{p, q\}$ и $\{q, r\}$ — пары эквивалентных состояний, а пара $\{p, r\}$ — различима. Тогда должна существовать такая цепочка $w$, для которой только одно из состояний $\hat{\delta}(p, w)$ и $\hat{\delta}(r, w)$ является допускающим. Используя симметрию, предположим, что $\hat{\delta}(p, w)$ — допускающее.

Теперь посмотрим, будет ли состояние $\hat{\delta}(q, w)$ допускающим. Если оно допускающее, то пара $\{q, r\}$ различима, так как состояние $\hat{\delta}(q, w)$ —допускающее, а $\hat{\delta}(r, w)$ — нет. Если $\hat{\delta}(q, w)$ не допускающее, то по аналогичным причинам пара $\{p, q\}$ различима. Полученное противоречие доказывает неразличимость пары $\{p, r\}$, т.е. состояния $p$ и $r$ эквивалентны. □

Теорему 4.23 можно использовать для обоснования очевидного алгоритма разбиения состояний. Для каждого состояния $q$ строится блок, состоящий из $q$ и всех эквивалентных ему состояний. Необходимо доказать, что результирующие блоки образуют разбиение множества состояний, т.е. ни одно состояние не принадлежит двум разным блокам.

Сначала заметим, что состояния внутри каждого блока взаимно эквивалентны, т.е., если $p$ и $r$ принадлежат блоку состояний, эквивалентных $q$, то согласно теореме 4.23 они эквивалентны.

Предположим, что существуют два пересекающихся, но не совпадающих блока, т.е. существует блок $B$, содержащий состояния $p$ и $q$, и блок $C$, который содержит $p$, но не $q$. Поскольку состояния $p$ и $q$ принадлежат одному блоку, то они эквивалентны. Рассмотрим возможные варианты построения блока $C$. Если этот блок образован состоянием $p$, то $q$ было бы в блоке $C$, так как эти состояния эквивалентны. Следовательно, существует некоторое третье состояние $s$, порождающее блок $C$, т.е. $C$ — это множество состояний, эквивалентных $s$.

Состояния $p$ и $s$ эквивалентны, так как оба принадлежат $C$. Также $p$ эквивалентно $q$, потому что оба эти состояния принадлежат $B$. Согласно свойству транзитивности, доказанному в теореме 4.23, состояние $q$ эквивалентно $s$. Но тогда $q$ принадлежит блоку $C$, что противоречит предположению о существовании пересекающихся, но не совпадающих блоков. Итак, эквивалентность состояний задает их разбиение, т.е. любые два состояния имеют или совпадающие, или не пересекающиеся множества эквивалентных им состояний. Следующая теорема подытоживает результаты проведенного анализа.

**Теорема 4.24.** Если для каждого состояния $q$ некоторого ДКА создать *блок*, состоящий из $q$ и эквивалентных ему состояний, то различные блоки состояний образуют *разбиение* множества состояний.[7] Это значит, что каждое состояние может принадлежать

[7] Нужно помнить, что один и тот же блок может формироваться несколько раз, начиная с разных состояний. Однако разбиение состоит из *различных* блоков, так что каждый блок встречается в разбиении только один раз.

только одному блоку. Состояния из одного блока эквивалентны, а любые состояния, выбранные из разных блоков, не эквивалентны. □

Теперь можно кратко сформулировать алгоритм минимизации ДКА $A = (Q, \Sigma, \delta, q_0, F)$.

1. Для выявления всех пар эквивалентных состояний применяем алгоритм заполнения таблицы.
2. Разбиваем множество состояний $Q$ на блоки взаимно эквивалентных состояний с помощью описанного выше метода.
3. Строим ДКА $B$ с минимальным числом состояний, используя в качестве его состояний полученные блоки. Пусть $\gamma$— функция переходов автомата $B$. Предположим, что $S$ — множество эквивалентных состояний автомата $A$, $a$ — входной символ. Тогда должен существовать один блок состояний $T$, содержащий $\gamma(q, a)$ для всех состояний $q$ из $S$. Если это не так, то входной символ $a$ переводит два состояния $p$ и $q$ из $S$ в состояния, принадлежащие разным блокам согласно теореме 4.24. Из этого можно сделать вывод, что состояния $p$ и $q$ не были эквивалентными и не могли вместе принадлежать $S$. В результате определяется функция переходов $\gamma(S, a) = T$. Кроме того:
   а) начальным состоянием ДКА $B$ является блок, содержащий начальное состояние автомата $A$;
   б) множеством допускающих состояний автомата $B$ является множество блоков, содержащих допускающие состояния ДКА $A$. Заметим, что если одно состояние в блоке является допускающим, то все остальные состояния этого блока также должны быть допускающими. Причина в том, что любое допускающее состояние отличимо от любого недопускающего, поэтому допускающее и недопускающее состояния не могут принадлежать одному блоку эквивалентных состояний.

**Пример 4.25.** Минимизируем ДКА, представленный на рис. 4.8. В примере 4.22 установлены блоки разбиения состояний. На рис. 4.12 изображен ДКА с минимальным числом состояний. Пять состояний этого автомата соответствуют пяти блокам эквивалентных состояний автомата на рис. 4.8.

Начальным состоянием минимизированного автомата является $\{A, E\}$, так как $A$ было начальным на рис. 4.8. Единственным допускающим — $\{C\}$, поскольку $C$ — это единственное допускающее состояние на рис. 4.8. Заметим, что переходы на рис. 4.12 правильно отражают переходы на рис. 4.8. Например, на рис. 4.12 есть переход из $\{A, E\}$ в $\{B, H\}$ по символу 0. Это очевидно, так как $A$ на рис. 4.8 переходит в $B$ при чтении 0, а $E$ — в $H$. Аналогично, при чтении 1 $\{A, E\}$ переходит в $\{D, F\}$. По рис. 4.8 легко увидеть, что оба состояния $A$ и $E$ переходят в $F$ по 1, так что для $\{A, E\}$ состояние-преемник по 1 также выбрано правильно. Тот факт, что ни $A$, ни $E$ не переходят в $D$ по 1, неважен. Читатель может проверить, что и все остальные переходы изображены правильно. □

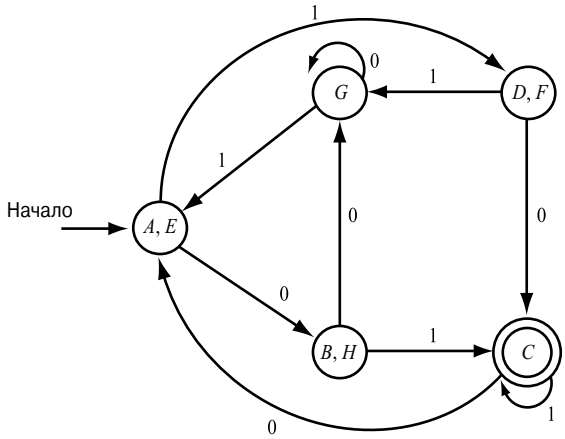


*Рис. 4.12. ДКА с минимальным числом состояний, эквивалентный автомату, изображенному на рис. 4.8*

### 4.4.4. Почему минимизированный ДКА невозможно улучшить

Предположим, что задан ДКА $A$, и мы минимизируем его до ДКА $M$ с помощью метода разбиения из теоремы 4.24. Эта теорема показывает, что невозможно получить эквивалентный ДКА, группируя состояния автомата $A$ в еще меньшее число групп. Но все же, может ли существовать другой ДКА $N$, не связанный с $A$, который допускал бы тот же язык, что и автоматы $A$ и $M$, но имел бы состояний меньше, чем автомат $M$? Методом от противного докажем, что такого автомата не существует.

Сначала применим алгоритм различимости состояний из раздела 4.4.1 к состояниям автоматов $M$ и $N$ так, как если бы это был один автомат. Можно предположить, что общих обозначений состояний у $M$ и $N$ нет, так что функция переходов комбинированного автомата будет объединением функций переходов автоматов $M$ и $N$, которые между собой не пересекаются. Состояния комбинированного ДКА будут допускающими тогда и только тогда, когда они являются допускающими в соответствующих им автоматах.

Начальные состояния автоматов $M$ и $N$ неразличимы, так как $L(M) = L(N)$. Далее, если состояния $\{p, q\}$ неразличимы, то для любого входного символа их преемники также неразличимы. Если бы преемников можно было различить, то $p$ и $q$ также можно было различить.

**Минимизация состояний НКА**

Может показаться, что метод разбиения состояний, минимизирующий ДКА, применим и для построения НКА с минимальным числом состояний, эквивалентного данному НКА или ДКА. Хотя методом полного перебора можно найти НКА с наименьшим количеством состояний, допускающий данный язык, просто сгруппировать состояния некоторого заданного НКА для этого языка нельзя.

Пример приведен на рис. 4.13. Никакие из трех состояний не являются эквивалентными. Очевидно, что допускающее состояние $B$ отличается от недопускающих состояний $A$ и $C$. Состояния $A$ и $C$ различаются входным символом 0. $C$ переходит в $\{A\}$ (недопускающее), тогда как $A$ переходит в $\{A, B\}$, которое включает допускающее состояние. Таким образом, группирование состояний не уменьшает количества состояний на рис. 4.13.

Однако можно построить меньший НКА для этого же языка, просто удалив состояние $C$. Заметим, что $A$ и $B$ (без $C$) допускают все цепочки, которые заканчиваются нулем, а добавление $C$ не позволяет допускать другие цепочки.

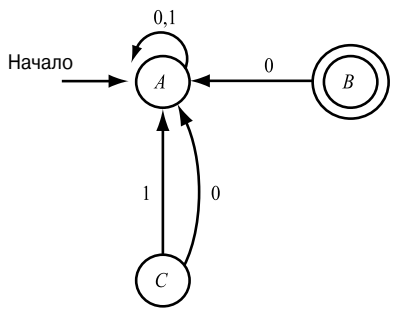


*Рис. 4.13. НКА, который невозможно минимизировать с помощью эквивалентности состояний*

Ни $M$, ни $N$ не могут иметь недостижимых состояний, иначе, исключив это состояние, можно было бы получить еще меньший ДКА для того же языка. Следовательно, каждое состояние автомата $M$ неотличимо хотя бы от одного состояния автомата $N$. Выясним, почему это так. Пусть $p$ — состояние автомата $M$. Тогда существует цепочка $a_1a_2\ldots a_k$, переводящая $M$ из начального состояния в $p$. Эта цепочка также переводит $N$ из начального в некоторое состояние $q$. Из того, что начальные состояния этих автоматов неразличимы, следует, что состояния-преемники, соответствующие входному символу $a_1$, также неразличимы. Состояния, следующие за этими состояниями при чтении $a_2$, также будут неразличимыми, и так далее, пока мы не придем к заключению, что состояния $p$ и $q$ неразличимы.

Поскольку автомат $N$ содержит меньше состояний, чем $M$, то должны существовать два состояния автомата $M$, которые неотличимы от одного и того же состояния автомата $N$. Значит, эти состояния неразличимы. Но автомат $M$ построен таким образом, что все его состояния отличимы друг от друга. Противоречие. Следовательно, предположение о существовании $N$ неверно, и $M$ действительно является ДКА с наименьшим количеством состояний среди всех ДКА, эквивалентных $A$. Формально доказана следующая теорема.

**Теорема 4.26.** Если из некоторого ДКА $A$ с помощью алгоритма, описанного в теореме 4.24, построен ДКА $M$, то $M$ имеет наименьшее число состояний из всех ДКА, эквивалентных $A$. □

Можно сформулировать более сильное утверждение, чем теорема 4.26. Между состояниями любого минимального ДКА $N$ и состояниями ДКА $M$ должно существовать взаимно

однозначное соответствие. Как уже доказано, каждое состояние $M$ эквивалентно одному состоянию $N$, и ни одно состояние $M$ не может быть эквивалентным двум состояниям $N$. Аналогично можно доказать, что ни одно состояние $N$ не может быть эквивалентным двум состояниям $M$, хотя каждое состояние автомата $N$ должно быть эквивалентно одному из состояний ДКА $M$. Следовательно, существует только один ДКА с минимальным количеством состояний, эквивалентный $A$ (с точностью до обозначений состояний).

### 4.4.5. Упражнения к разделу 4.4

**4.4.1.** (*) На рис. 4.14 представлена таблица переходов некоторого ДКА:

а) составьте таблицу различимости для этого автомата;

б) постройте эквивалентный ДКА с минимальным числом состояний.

| | *0* | *1* |
|---|---|---|
| →*A* | *B* | *A* |
| *B* | *A* | *C* |
| *C* | *D* | *E* |
| **D* | *D* | *A* |
| *E* | *D* | *F* |
| *F* | *G* | *E* |
| *G* | *F* | *G* |
| *H* | *G* | *D* |

*Рис. 4.14. ДКА, который нужно минимизировать*

Выполните упражнение 4.4.1 для ДКА, представленного на рис. 4.15.

| | 0 | 1 |
|---|---|---|
| →A | B | E |
| B | C | F |
| *C | D | H |
| D | E | H |
| E | F | I |
| *F | G | B |
| G | H | B |
| H | I | C |
| *I | A | E |

*Рис. 4.15. Еще один ДКА, который нужно минимизировать*

**4.4.3.** (!!) Пусть $p$ и $q$ — пара различимых состояний заданного ДКА $A$ с $n$ состояниями. Какой может быть самая точная верхняя граница длины кратчайшей цепочки, различающей $p$ и $q$, как функция от $n$?

## Резюме

- *Лемма о накачке.* Если язык регулярен, то в каждой достаточно длинной цепочке этого языка есть непустая подцепочка, которую можно "накачать", т.е. повторить произвольное число раз; получаемые при этом цепочки будут принадлежать данному языку. Эта лемма используется для доказательства *нерегулярности* многих языков.
- *Операции, сохраняющие регулярность языков.* Существует много операций, результат применения которых к регулярным языкам также является регулярным языком. В их числе объединение, конкатенация, замыкание (итерация), пересечение, дополнение, разность, обращение, гомоморфизм (замена каждого символа соответствующей цепочкой) и обратный гомоморфизм.
- *Проверка пустоты регулярного языка.* Существует алгоритм, который по такому заданному представлению регулярного языка, как автомат или регулярное выражение, определяет, является ли представленный язык пустым множеством.
- *Проверка принадлежности регулярному языку.* Существует алгоритм, который по заданной цепочке и некоторому представлению регулярного языка определяет, принадлежит ли цепочка языку.
- *Проверка различимости состояний.* Два состояния некоторого ДКА различимы, если существует входная цепочка, которая переводит в допускающее только одно из этих состояний. Если начать с того, что все пары, состоящие из допускающего и недопускающего состояний, различимы, и найти дополнительные пары, которые по одному символу переходят в различимые состояния, можно обнаружить все пары различимых состояний.
- *Минимизация детерминированных конечных автоматов.* Состояния любого ДКА можно разбить на группы взаимно неразличимых состояний. Состояния из двух разных групп всегда различимы. Если заменить каждую группу одним состоянием, получим эквивалентный ДКА с наименьшим числом состояний.

## Литература

За исключением очевидных свойств замкнутости регулярных выражений (относительно объединения, конкатенации и итерации), которые были доказаны Клини [6], почти все результаты свойств замкнутости воспроизводят аналогичные результаты, полученные для контекстно-свободных языков (этому классу языков посвящены следующие главы). Таким образом, лемма о накачке для регулярных языков является упрощением соответствующего результата для контекстно-свободных языков (Бар-Хиллел, Перлес и Шамир [1]). Из результатов этой работы следуют некоторые другие свойства замкнутости, представленные в данной главе, а замкнутость относительно обратного гомоморфизма обоснована в [2].

Операция деления (см. упражнение 4.2.2) представлена в [3]. В этой работе обсуждается более общая операция, в которой вместо одиночных символов находятся регулярные языки. Ряд операций "частичного удаления", начиная с упражнения 4.2.8, в котором говорилось о первых половинах цепочек регулярного языка, был определен в [8]. Сейферас и Мак-Нотон [9] изучили общий случай, когда операция удаления сохраняет регулярность языков.

Алгоритмы разрешения, такие как проверка пустоты и конечности регулярных языков, а также проверка принадлежности к регулярному языку, берут свое начало в [7]. Алгоритмы минимизации числа состояний ДКА появились в [5]. В работе [4] предложен наиболее эффективный алгоритм нахождения минимального ДКА.


1. Y. Bar-Hillel, M. Perles, and E. Shamir, "On formal properties of simple phrase-structure grammars," *Z. Phonetik. Spachwiss. Kommunikations-forsch.* **14** (1961), pp. 143–172.
2. S. Ginsburg and G. Rose, "Operations which preserve definability in languages," *J. ACM* **10**:2 (1963), pp. 175–195. (Гинзбург С., Роуз Дж. Об инвариантности классов языков относительно некоторых преобразований. — Кибернетический сборник, Новая серия, вып. 5. — М.: Мир, 1968. — С. 138–166.)
3. S. Ginsburg and E. H. Spanier, "Quotients of context-free languages," *J. ACM* **10**:4 (1963), pp. 487–492.
4. J. E. Hopcroft, "An *n* log *n* algorithm for minimizing the states in a finite automaton," in Z. Kohavi (ed.) *The Theory of Machines and Computations*, Academic Press, New York, pp. 189–196. (Хопкрофт Дж. Алгоритм для минимизации конечного автомата. — Кибернетический сборник, Новая серия, вып. 11. — М.: Мир, 1974. — С. 177–184.)
5. D. A. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Inst.* **257**:3-4 (1954), pp. 161–190 and 275–303.
6. S. C. Kleene, "Representation of events in nerve nets and finite automata," in C. E. Shannon and J. McCarthy, *Automata Studies,* Princeton Univ. Press, 1956, pp. 3–42. (Клини С.К. Представление событий в нервных сетях. — сб. "*Автоматы*". — М.: ИЛ, 1956. — С. 15–67.)
7. E. F. Moore, "Gedanken experiments on sequential machines," in C. E. Shannon and J. McCarthy, *Automata Studies,* Princeton Univ. Press, 1956, pp. 129–153. (Мур Э.Ф. Умозрительные эксперименты с последовательностными машинами. — сб. "*Автоматы*". — М.: ИЛ, 1956. — С. 179–210.)
8. R. E. Stearns and J. Hartmanis, "Regularity-preserving modifications of regular expressions," *Information and Control* **6**:1 (1963), pp. 55–69.
9. J. I. Seiferas and R. McNaughton, "Regularity-preserving modifications," *Theoretical Computer Science* **2**:2 (1976), pp. 147–154.