Оглавление

ВВЕДЕНИЕ

Современное формальное определение алгоритма было дано в 1930–1950-х годах в произведениях А. Тьюринга, Э. Поста, А. Черча, Н. Винера, А.А. Маркова.

Алгоритм (процедура) — решение задач в виде точных последовательно выполняемых предписаний. Это интуитивное определение сопровождается описанием интуитивных **свойств (признаков)** алгоритмов: эффективность, определенность, конечность [1].

Эффективность — возможность исполнения предписаний за конечное время.

Например, алгоритм — процедура, состоящая из «конечного числа команд, каждая из которых выполняется механически за фиксированное время и с фиксированными затратами»[1].

Функция может быть эффективно вычислена с помощью алгоритмов, если существует механическая процедура, из которой для конкретных значений ее аргументов может быть найдено значение этой функции.

Определенность — возможность точного математического определения или формального описания содержания команд и последовательности их применения в этой процедуре.

Конечность — выполнение алгоритма для конкретных исходных данных за конечное количество шагов.

Для доказательства алгоритмов в теории используются алгоритмические преобразования слов и фраз формального языка.

В формальных описаниях алгоритм конструктивно связан с концепцией машины, предназначенной для автоматизированных преобразований символьной информации.

Для автоматических расчетов разрабатываются модели алгоритмов распознавания языков и машина, которая работает с этими моделями. Таким образом, они сочетают математическое и формальное определение алгоритма с конструктивным, что позволяет создавать модели на компьютере.

Алгоритмические вычисления используются во всех областях науки и техники. Например, в монографии Д. Кнута [1] рассматривается множество проблемно ориентированных алгоритмических решений.

Интуитивно понятный графический метод построения алгоритмов в виде диаграмм и схем по-прежнему актуален и поддерживается стандартами языкового редактирования.

Объектно ориентированное обобщение алгоритмических языков позволяет включить алгоритмическое мышление и организовать крупномасштабное программирование с огромной производительностью и ресурсами компьютерной памяти, что постоянно увеличивает стандартные библиотеки.

Теория алгоритмических языков и компиляторов действительно связана с алгоритмами, но это независимая область знания и применения алгоритмов.

Общая теория алгоритмов обращается к проблеме эффективной вычислимости. Было разработано несколько формальных определений алгоритма, в которых эффективность и окончательность вычислений можно количественно оценить с помощью количества элементарных шагов и требуемого объема памяти.

Аналогичными моделями алгоритмических преобразований символьной информации являются:

– конечные автоматы;

– машина Тьюринга;

– машина Поста;

– ассоциативное исчисление или нормальные алгоритмы Маркова;

– рекурсивные функции.

Некоторые из этих моделей составляют основу методов программирования и используются в алгоритмических языках.

В современной **программной инженерии** алгоритмы как методы решения задач занимают видное место по сравнению с традиционной математикой. И неважно, есть ли в абстрактных алгоритмических моделях чистое алгоритмическое решение. Если необходимо решение проблемы, широко используются эвристики, и «доказательством» работоспособности алгоритма является его успешное тестирование [2].

[1] Ахо А. Теория синтаксического анализа, перевода и компиляции : в 2 т. Т. 1 / А. Ахо, Дж. Ульман ; пер. с англ. В.Н. Агафонова ; под ред. В.М. Курочкина. — Москва : Мир, 1978. — 612 с.

1. ОСНОВНЫЕ ПОНЯТИЯ И МОДЕЛИ АЛГОРИТМОВ

Для формального описания алгоритма необходимо формальное описание решаемой проблемы. В большинстве случаев описание проблемы носит неформальный (вербальный) характер, следовательно, переход к алгоритму неформальный и требует проверки и тестирования, а также нескольких итераций для приближенного решения.

Верификация (от лат. verus — *истинный* и facere — *делать*) — способ подтверждения любых теоретических положений, алгоритмов, программ и процедур путем сравнения их с экспериментальными данными (справочными или эмпирическими), алгоритмами и программами.

Тестирование используется для определения соответствия объекта испытаний указанным спецификациям.

Теория алгоритмов не может предоставить универсального и формального способа описания проблемы и ее алгоритмического решения. Однако ценность теории состоит в том, что она дает примеры таких описаний и определения алгоритмически неразрешимых задач.

Один из примеров — на обычном языке, и задача формулируется как разработка алгоритма распознавания принадлежности любого предложения к определенному регулярному языку. Доказано, что регулярный язык можно формально преобразовать в **модель алгоритма** для решения этой проблемы за конечное число шагов. Эта модель представляет собой **конечный автомат**.

Регулярные языковые расширения используются: при описании словаря формальных алгоритмических языков и алгоритмов, при исследовании шаблонов редактирования, в современном программировании (Java, Phyton, C#, PHP, JS), в компиляторах и системах управления обработкой данных, в качестве интерактивных языков в операционных системах. Следовательно, конечный автомат может использоваться для алгоритмического решения проблем в этих областях.

Алфавит языка обозначается как конечное множество символов. Например: $\sum = \{a, b, c, d\}$, $\sum = \{0, 1\}$.

Символ и цепочка символов образуют слово — *a*, *b*, 0, *abbcd*, 0111000. Пустое слово **(*e*)** не содержит символов.

Множество слов $S = \{a, ab, aaa, bc\}$ в алфавите Σ называют языком $L(\Sigma)$.

Язык $S = L(\Sigma)$ может содержать неограниченное количество слов.

Для их определения используются различные формальные правила. В простейшем случае это **алгебраическая формула**, которая содержит операции по формированию слов из символов алфавита и ранее полученных слов.

Рассмотрите следующие шаги, чтобы сформировать новые наборы из существующих наборов слов.

1. Символы алфавита могут быть связаны путем **конкатенации** (присоединения) к строкам словесных символов, связанных с новыми словами.

Конкатенация двух слов *x*|*y* обозначает, что к слову *x* справа приписано слово *y* или *x*|*y* = *xy*, причем $xy \neq yx$.

Произведение $S_1|S_2 = S_1S_2$ множеств слов S_1 и S_2 — это множество всех различимых слов, построенных конкатенацией соответствующих слов из S_1 и S_2.

Если $S_1 = \{a, aa, ba\}$, $S_2 = \{e, bb, ab\}$, то $S_1S_2 = \{a, aa, ba, abb, aabb, baab, ...\}$. Для конкатенации выполняется ассоциативность, но коммутативность и идемпотентность не выполняются: $S_1S_2 \neq S_2S_1$; $SS \neq S$.

2. **Объединение** $(S_1 \cup S_2)$ или $(S_1 + S_2)$ множеств:

$S_1 = \{a, aa, ba\}$, $S_2 = \{e, bb, ab\}$, $S_1 \cup S_2 = \{a, aa, ba, e, bb, ab\}$. Для операции объединения выполняются следующие законы:

коммутативность объединения: $S_1 \cup S_2 = S_2 \cup S_1$;

идемпотентность объединения: $S \cup S = S$;

ассоциативность объединения: $S_1 \cup (S_2 \cup S_3) = (S_1 \cup S_2) \cup S_3$;

дистрибутивность конкатенации (умножения) и объединения: $S_1(S_2 \cup S_3) = S_1S_2 \cup S_1S_3$.

3. **Итерация множества** $\{S\}^*$ состоит из пустого слова и всех слов вида $S^\wedge$: $S^1 = S$, $S^2 = SS$, $S^3 = SSS$.

Формулы, содержащие эти операции с множествами слов, называют **регулярными выражениями**.

Ассоциативность итерации: $S_1{*}(S_2{*}S_3) = (S_1{*}S_2){*}S_3$.

Дистрибутивность объединения с итерацией: $S_1{*}(S_2 \cup S_3) = S_1{*}S_2 \cup S_1{*}S_3$. Если a, b — любые регулярные выражения, то $(a \cup b)^* = (a^* \cup b^*)^* = (a^*b^*)^* = (a^*b)^*a^*$; $a^* = a^*a^* = (a^*)^* = (a \cup a^2 \cup \ldots \cup a^k)^*$; $(a^*b)^* = (a \cup b)^*b$.

Следовательно, формулы могут содержать круглые скобки и могут быть преобразованы с использованием этих законов.

Регулярные выражения допускают формальные алгебраические преобразования.

Языки, определяемые регулярными выражениями, называются регулярными языками, а набор слов — регулярными множествами.

Пример 1.1

Регулярные выражения регулярного языка в алфавите Σ = {0,1}:

(0 ∪ (1(0)*)) = 0 ∪ 10*;

(0 ∪ 1)* = (0* ∪ 1*)*;

(0 ∪ 1)*011 — все слова из 0 и 1, заканчивающиеся на 011;

(a ∪ b)(a ∪ + b)* = (a ∪ b)(a* ∪ b*)* — слова, начинающиеся с a или b;

(00 ∪ 11)*((01 ∪ 10)(00 ∪ 11)*(01 ∪ 10)(00 ∪ 11)*)* — все слова, содержащие четное число 0 и 1.

Пример 1.2

Входной алфавит: Σ = {i, +, –},

где i — идентификатор;

(+, –) — знаки арифметических операций.

Примеры правильных арифметических выражений:

i, – i, i + i, i – i, – i – i, i + i – i, ...

Обозначим знаки арифметических действий буквами p = (+), m = (–).

Тогда соответствующие правильные (регулярные) арифметические выражения имеют вид i, mi, ipi, imi, $mimi$, $ipimi$, ... Регулярное выражение, определяющее регулярный язык:

$L(M) = (mi + i)((p + m)i)^*$.

Утверждение. Для каждого регулярного множества (языка $L(\Sigma)$) можно построить по крайней мере одно регулярное выражение, но для каждого регулярного выражения существует только одно регулярное множество.

Основные модели алгоритмов

Алгоритм распознавания предложений регулярного языка называют конечным автоматом **(КА)**.

Конечный автомат определяется символами: $M = (Q, \sum, \delta, q_0, F)$,

где $Q = \{q_0, q_1, \ldots, q_n\}$ — конечное множество состояний;

$\sum = \{a, b, c, \ldots\}$ — входной алфавит (конечное множество);

δ: $Q^* \sum \rightarrow \{P_j\}$ — функция переходов, P_j — подмножество Q.

Конечное множество значений для этого функционального отношения может быть определено перечислением в **таблице переходов** (табл. 1.1).

Таблица 1.1

Q	$\sum$	P_j
q_i	a	q_j

q_0 — начальное состояние;

F — множество заключительных состояний.

Конечный автомат называется **недетерминированным** (НДКА), если P_j содержит более одного состояния.

КА называется **детерминированным** (ДКА), если P_j содержит не более одного состояния.

КА **полностью определенный**, если P_j в детерминированном автомате не пустое. Если есть пустые элементы множества P_j, то автомат **частично определен**.

Работа КА или выполнение обычного алгоритма распознавания слов на языке может быть представлена последовательностью шагов, которые определяются текущим состоянием Q, входным символом $\sum$ и последующим состоянием P_j.

Используется конструктивное описание принципа работы КА, например машины M, со следующей организацией (рис. 1.1).

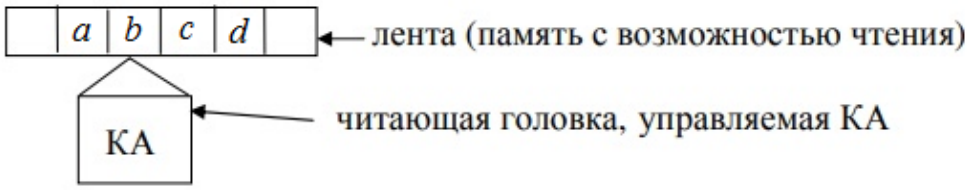

Рис. 1.1. Машина, исполняющая конечный автомат

КА считывает символ для входа в текущее состояние $q_i \in Q$, переходит в следующее состояние $q_j \in Q$ и меняет считывающую головку на следующий символ.

Автомат принимает входное слово, если оно достигает конечного состояния F, последовательно считывая символы из памяти и переходя в следующие состояния согласно таблице переходов. В этом случае входное слово заканчивается и автомат останавливается.

Конфигурация КА: $k = (q, \omega)$, где q — текущее состояние КА, ω — непрочитанная цепочка символов слова на ленте, включая символ под читающей головкой;

$k = (q, \omega)$ — текущая конфигурация;

$k_0 = (q_0, \omega_0)$ — начальная конфигурация;

$k_f = (q, e)$ — заключительная конфигурация, $q \in F$, (e) — символ, обозначающий конец строки.

Шаг алгоритма — переход из одной конфигурации КА в другую:

$K_i \to K_j$ или $(q_i, \omega_i) \to (q_j, \omega_j)$.

Функция переходов, заданная в табличной форме, может быть представлена графом переходов $kG = (Q, R)$, где Q — вершины графа, R — бинарное отношение между парой вершин, которое представлено множеством дуг (q_i, q_j).

$(q_i, q_j) \in Q^*Q$, если существует символ $a \in \sum$ и $\delta(q_i, a) = q_j$. На дугах графа (q_i, q_j) отмечаются соответствующие символы алфавита.

Пример 1.3

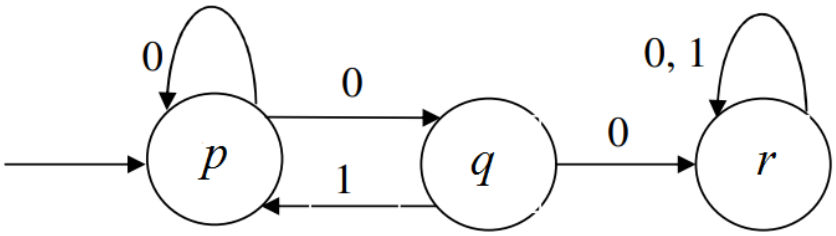

Рис. 1.2. Детерминированный читающий КА

Для ДКА, приведенного на рис. 1.2: состояния $Q = \{p, q, r\}$, входной алфавит $\sum = \{0, 1\}$, начальное состояние p, конечное — r, таблица переходов указана в табл. 1.2.

Таблица 1.2

Q	Σ	
	0	1
p	q	p
q	r	p
r	r	r

Исполнение алгоритма — это последовательность шагов, в которых изменяется конфигурация КА:

$(p, 01001) \rightarrow (q, 1001) \rightarrow (p, 001) \rightarrow (q, 01) \rightarrow (r, 1) \rightarrow (r, e)$,

где $(p, 01001)$ — начальная конфигурация;

(r, e) — конечная конфигурация.

В результате применения слова 01001 в начальном состоянии p автомат переходит в следующее состояние q и следующее значение цепочки символов на входе 1001.

Автомат *M* допускает слово ω_0, если существует $(q_0, \omega_0) \rightarrow {}^*(q_f, e)$, где $\rightarrow {}^*( )$ обозначает **транзитивное замыкание** и существует путь, соединяющий q_0 и q_f для входного слова ω_0.

Язык *L*(*M*), определяемый (распознаваемый, допускаемый) автоматом *M*, включает множество всех слов, допускаемых *M*.

Преобразование регулярных выражений в конечный автомат

Утверждение. Язык *L* является регулярным тогда и только тогда, когда он определяется КА. Для любого регулярного языка, представленного регулярным выражением, можно построить КА — распознаватель слов, допустимых языком.

Метод Ямады преобразования *L*(*M*) → *M* позволяет формально выполнить преобразование [3; 4]:

1) разметка состояний устанавливает позиции символов в регулярном выражении (пример 1.2):

$L(M) = (mi + i)\big((p + m)i\big)^*0123456$;

2) рассмотрим 2^6 + 1 подмножеств мест предположения состояния и сформируем регулярным выражением возможные переходы между состояниями. Явная избыточность состояний уже устранена, так как для некоторых состояний нет переходов.

Общие оценки числа состояний и переходов КА:

– число переходов (ребер графа) КА равно (*n* + 1), где *n* — число букв в регулярном выражении (в данном примере *n* = 7 состояний);

– в полностью определенном КА нижнюю границу числа состояний m определяем из условия $m^*s = n + 1$, где s — число символов входного алфавита (в данном случае $3m = 7$ и $m =]7/3[= 3$);

– если $m^*s > n + 1$, то автомат частично определенный и верхняя граница $m \leq n + 1$ — количество позиций в регулярном выражении (в данном примере m = 7).

Для рассматриваемого примера можно построить КА с использованием верхней оценки (рис. 1.3).

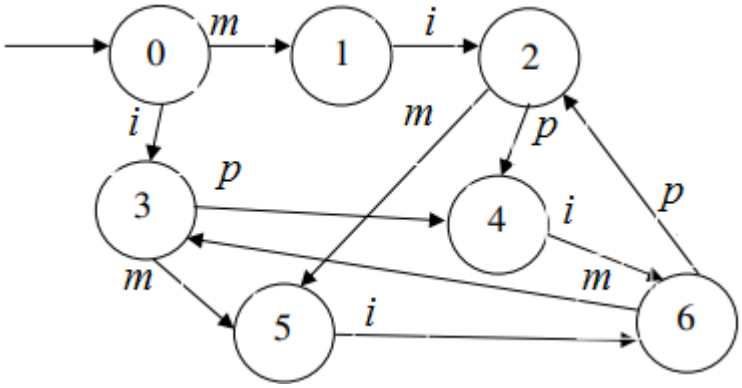

Рис. 1.3. Преобразование регулярного выражения в КА

Утверждение. Для каждого регулярного множества существует определяющий его КА с минимальным числом состояний.

Слово *различает* состояния q_i и q_j, если $(q_i, \omega) \to {}^*(q_m, e)(q_j, \omega) \to {}^*(q_n, e)$ и одно из состояний q_m или q_n принадлежит F.

Состояния k *не различимы*, если они не различимы цепочкой длины k.

Состояния не различимы (*эквивалентны* $q_i = q_j$), если они не различимы при $k \to \infty$.

В автомате на рис. 1.3 каждую пару эквивалентных состояний $\{2, 3\}$, $\{4, 5\}$ заменяем одним состоянием $\{2, 3\} \to 2$, $\{4, 5\} \to 3$.

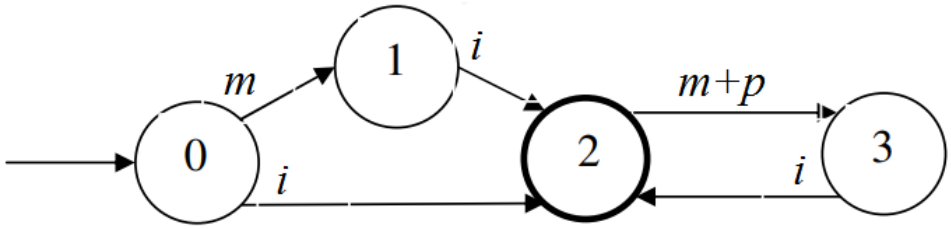

Рис. 1.4. ДКА распознавания $L(M)$

Рассмотренную технику преобразования можно упростить, если учесть очевидные алгебраические свойства операций с регулярными выражениями:

– строки в круглых скобках с операцией (+) имеют общее начальное состояние перед открывающей круглой скобкой и конечное состояние после закрывающей круглой скобки;

– итерация обозначает цикл с произвольным количеством повторений, с пустым количеством циклов сохраняется состояние, с которым мы вошли в цикл;

– при конкатенации нескольких символов различимые состояния заменяют конкатенацию между парой символов:

$L(M) = (mi + i)\big((p + m)i\big)^*01232.$

Состояния размещаем в найденные для них позиции, исключая повторения:

$L(M) = (mi + i)\big((p + m)i\big)^* = 0(m1i + i)2\big((p + m)3i2\big)^*01232.$

Заменяем линейную помеченную строку графом КА (рис. 1.4). Начальное состояние — 0, конечное состояние — 2.

В общем случае регулярное выражение может быть преобразовано в недетерминированный КА (НДКА).

Пример 1.4

Рассмотрим НДКА распознавания $L(M)$ (рис. 1.5).

$L(M) = aa*bd* + ad* = (aa*b + a)d* = (0a_1(a_1)*b + 0a)2(d_2)*012.$

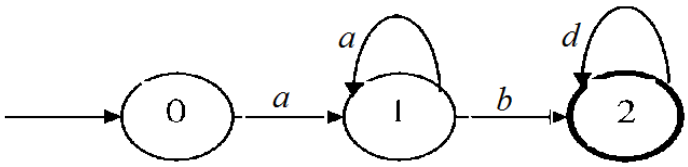

Рис. 1.5. НДКА распознавания $L(M)$

При выполнении алгоритма (НДКА) по методу поиска по образцу сохраняются номера позиций, в которых можно выбрать несколько переходов. При неудачном поиске осуществляется возврат на ближайшую позицию (назад). Шаги по разным ветвям выполняются параллельно:

(0, *add*) → (1, *dd*)

→ (2, *dd*) → (2, *d*) → (2, *e*).

Пример 1.5

Всякая подцепочка из трех символов содержит два нуля:

$\big((001)*(100)*(010)*\big)*$.

Число 0 делится на 2, а число 1 — на 3:

$\big((00)*(111)*\big)*$ = (00 + 111).

Множество цепочек, в которых каждая пара 00 находится перед парой:

$11\big((00)(11)*01*\big)*$.

Множество цепочек, в которых количество единиц четно:

(0* + 11 + 101)*.

Утверждение. Два автомата M и M' эквивалентны, если допускают один и тот же язык $L(M) = L(M')$.

Утверждение. Для НДКА M можно построить эквивалентный ДКА M'.

Преобразование представим следующей процедурой:

– на i-м шаге множество состояний ДКА обозначим Q_i;

– Q_0 обозначает множество состояний в НДКА;

– находим различимые подмножества следующих состояний для всех условий, применяемых к Q_i, включаем их в Q_{i+1}. Итерация повторяется, пока $Q \neq Q_{i+1}$.

Лемма. Число состояний в ДКА не превышает $2^n - 1$, где n — число состояний в НДКА.

Число $2^n - 1$ — количество собственных подмножеств для множества, состоящего из n различимых элементов.

Следовательно, процедура конечна.

Пример 1.6

Эквивалентный детерминированный автомат (рис. 1.6) содержит четыре состояния и определен тем же регулярным выражением $L(M') = L(M)$.

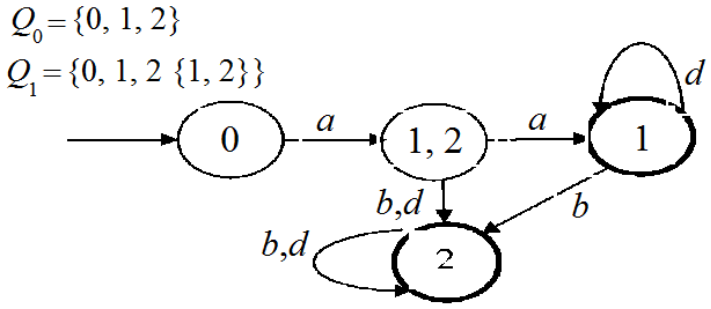

Рис. 1.6. ДКА получен из НДКА

Пример 1.7

НДКА задан таблицей переходов (табл. 1.3).

Таблица 1.3

Переходы	0	1
P	q, s	q
Q	r	q, r
R	s	p
S	q	p

Преобразование в ДКА:

$Q_0 = \{p, q, r, s\}$;

$Q_1 = \{p, q, r, s, \{q, s\}, \{r, q\}\}$;

$Q_2 = \{p, q, r, s, \{q, s\}, \{r, q\}, \{r, s\}\}$;

$Q_3 = \{p, q, r, s, \{q, s\}, \{r, q\}, \{q, r, p\}, \{r, s\}\}$;

$Q_4 = \{p, q, r, s, \{q, s\}, \{r, q\}, \{q, r, p\}, \{r, s\}, \{q, r, s\}\}$;

$Q_5 = Q_4$;

$Q_4 = \{p, q, r, s, \{q, s\}, \{r, q\}, \{q, r, p\}, \{r, s\}, \{q, r, s\}\} = \{P, Q, R, S, X, Y, Z, M, F\}$.

Таблица переходов КА представлена в табл. 1.4.

Таблица 1.4

0		1
P	X	Q
Q	R	Y
R	S	P
S	Q	P
X	Y	Z
Y	M	Z
Z	F	Z
M	X	P
F	F	Z

ДКА содержит девять состояний. Если S — конечное состояние в НДКА, то в ДКА выделены конечные состояния $\{S, X, M, F\}$, содержащие S.

Преобразование конечного автомата в регулярное выражение

Утверждение. Для любого конечного читающего автомата M можно построить регулярное выражение $L(M)$.

Для допускающего (финального) состояния исключим промежуточное состояние (кроме начального q_0) следующим образом.

Пусть q_i — предшествующее состояние, q_s — промежуточное и q_j — следующее, таким образом:

1) над каждой дугой (q_i, q_s) запишем регулярное выражение Q_{is};

2) на дуге (q_s, p_j) — выражение Q_{sj};

3) петлю в s обозначим регулярным выражением S^*;

4) дугам (q_i, q_j) после исключения промежуточного состояния q_s припишем регулярные выражения $Q_{is}S^*Q_{sj}$.

После удаления всех промежуточных вершин исходное и конечное состояния остаются. Конечные состояния объединяются путем добавления соответствующих регулярных выражений. В частном случае, если $q_0 \in F$, состояние q_0 останется, и дуге будет присвоено регулярное выражение.

Когда остаются два состояния, начальное и конечное, промежуточное регулярное выражение можно записать как $(R + SU^*T)^*SU^*$.

Полученный КА представлен на рис. 1.7.

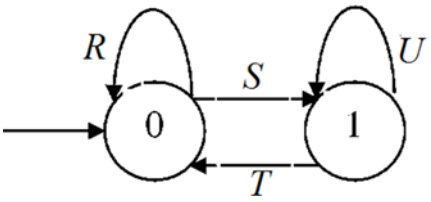

Рис. 1.7. Формирование регулярного выражения

Пример 1.8

Преобразуем ДКА в регулярное выражение (рис. 1.8).

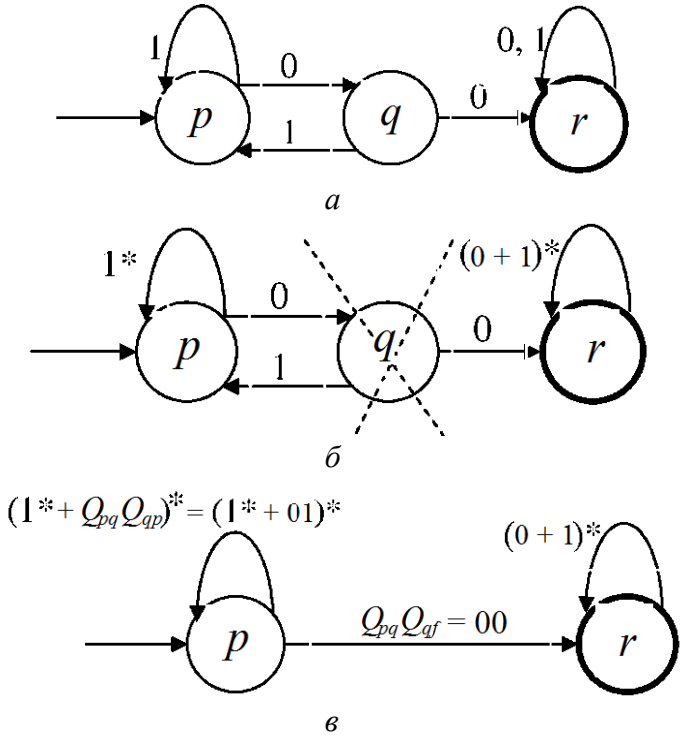

Рис. 1.8. Три стадии преобразования КА в регулярное выражение:
а — исходный КА; *б* — исключение состояния; *в* — свертка для двух состояний

Завершающий шаг — соединение (конкатенация) выражений для начального и финального состояний:

$R = (1^* + Q_{pq}Q_{qp})^* = (1^* + 01)$;

$U = (0 + 1)^*$;

$S = Q_{pq}Q_{qf} = 00$;

T отсутствует;

$(R + SU^*T)^*SU^* = (1^* + 01)^*00(0 + 1)^*$;

$L(M) = (1^* + 01)^*00(0 + 1)^*001023$.

Состояния q_i и q_j связаны дугой (q_i, q_j), если существует смежный символ в регулярном выражении. В этом случае дуга помечается соответствующим символом.

Получен НДКА:

$Q_0 = \{q_0, q_1, q_2, q_3\}$;

$Q_1 = \{q_0, q_1, q_2, q_3, \{q_{1,2}\}\}$.