Московский государственный технический университет имени Н.Э. Баумана

Факультет «Фундаментальные науки»
Кафедра «Математическое моделирование»

А.Н. Канатников

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Конспект лекций

Для студентов кафедры ИУ9

Москва
2009

9. НОРМАЛЬНЫЕ АЛГОРИФМЫ МАРКОВА

9.1. Определение

Алфавит и подстановки. Схемы. Нормальный алгорифм Маркова. Терминология. Примеры НА. Эквивалентные НА. Принцип нормализации.

Мы считаем, что задан некоторый конечный алфавит, и рассматриваем разные способы преобразования слов в этом алфавите. В качестве базовой операции рассмотрим так называемую ***контекстную замену***. При этой операции определенное подслово данного слова заменяется на другое слово. Иными словами, если исходное слово имеет вид $X = Y * W * Z$, то контекстная замена $W \to V$ приводит к слову $X' = Y * V * Z$.

Существует несколько вариантов контекстной замены, которые различаются по своему действию при нескольких вхождениях подслова и наличием специальных подстановочных знаков. Мы остановимся на простейшей операции, при которой заменяется только первое вхождение данного слова.

Нормальный алгорифм Маркова[*] определяется некоторым непустым конечным упорядоченным набором формул вида $W \to V$ в заданном алфавите A, где W и V — произвольные слова в A (одно из двух слов может быть пустым). Каждая такая формула относится к одной из двух категорий: первая категория — простые формулы, вторая — терминальные (заключительные). Терминальные формулы выделяют, ставя обычно точку после стрелки, например $abba \to\cdot\ ab$. Здесь предполагается, что символы $\to$ и $\cdot$ не входят в алфавит A. Указанный упорядоченный набор формул называют ***схемой нормального алгорифма*** в алфавите A.

Действие нормального алгорифма состоит в пошаговом преобразовании исходного слова с помощью контекстной замены. Формально это действие на одном шаге описывается следующим образом. Пусть X — текущее слово. Последовательно просматривается список формул в схеме алгоритма начиная с первой. Для очередной формулы вида $W \to * V$, где символ $*$ обозначает или пустое слово, или точку, ищется первое вхождение слова W в слове X. Если такое вхождение найдено, выполняется подстановка вместо этого вхождения слова V. При этом, если формула подстановки терминальная, работа алгоритма заканчивается. Если вхождений нет, переходим к следующей формуле схемы. Если все формулы схемы проанализированы, но вхождений не найдено, алгоритм заканчивает работу.

Введем некоторые обозначения. Алгоритмы будем обозначать готическими буквами: $\mathfrak{A}$, $\mathfrak{B}$ и т.д. Если слово X преобразуется в слово Y с помощью простой подстановки, будем писать $\mathfrak{A}$: $X \vdash Y$ и говорить, что алгоритм $\mathfrak{A}$ переводит X в Y. В аналогичной ситуации при терминальной подстановке пишем $\mathfrak{A}$: $X \vdash\cdot\ Y$ и говорим, что алгоритм $\mathfrak{A}$ терминально переводит X в Y. Если в слове X не может быть выполнена ни одна подстановка схемы, будем писать $\mathfrak{A}$: $X\dashv$ и говорить, что алгоритм $\mathfrak{A}$ не переводит слово X.

Предположим, что в результате работы алгоритма $\mathfrak{A}$ мы получили последовательность слов $X = X_0,\ X_1,\ \ldots,\ X_k = Y$, где $\mathfrak{A}$: $X_{j-1} \vdash X_j$, $j = \overline{0,\ k-1}$. Тогда будем обозначать $\mathfrak{A}$: $X \models Y$ и говорить, что алгоритм $\mathfrak{A}$ преобразует X в Y. Если на последнем шаге $\mathfrak{A}$: $X_{k-1} \vdash\cdot\ X_k$, т.е. последнее слово в последовательности получено применением терминальной подстановки, то будем писать $\mathfrak{A}$: $X \models\cdot\ Y$ и говорить, что алгоритм $\mathfrak{A}$ преобразует X в Y терминально. Короче: „переводит“ за один шаг, „преобразует“ за конечное число шагов, „терминально“ —

[*]Вообще говоря, в русском языке используют написания „алгоритм“ и „алгорифм“ как равноценные. Словосочетание „нормальный алгорифм Маркова“ используется как установившийся термин.

последняя подстановка (возможно, единственная) терминальная. По определению считаем, что для любого слова $\mathfrak{A}$: $X \vDash X$.

С каждым нормальным алгорифмом связывается словарная функция, которую мы будем обозначать так же, как и сам алгоритм. Для произвольного слова $X \in A^*$ применение нормального алгорифма приводит к одному из трех результатов:

1) на некотором k-м шаге мы получаем слово X_k, для которого $\mathfrak{A}$: $X_{k-1} \vdash\cdot X_k$;

2) на некотором k-м шаге мы получаем слово X_k, для которого $\mathfrak{A}$: $X_k\dashv$;

3) алгоритм продолжает работу неограниченное число шагов.

В первых двух случаях считаем слово X_k результатом работы алгоритма и записываем $X_k = \mathfrak{A}(X)$ (как значение словарной функции). При этом также говорим, что алгоритм $\mathfrak{A}$ применим к слову X и пишем $!\mathfrak{A}(X)$. В третьем случае говорим, что алгоритм $\mathfrak{A}$ неприменим к слову X и пишем $\neg!\mathfrak{A}(X)$.

Пример 9.1. Рассмотрим некоторый алфавит A и схему из одной подстановки $\to\cdot P$, где P — некоторое фиксированное слово. Соответствующий алгоритм $\mathfrak{A}_P$ применим к любому слову и за один шаг приписывает к любому слову слово P, т.е. $\mathfrak{A}_P(X) = P * X$.

Отметим, что подстановка с пустой левой частью применима всегда и первое вхождение пустой строки — перед первым символом слова. Если в схему алгоритма добавить какие-то другие подстановки, они никогда не будут использованы. Поэтому подстановки с пустой левой частью уместны в конце схемы.

Пример 9.2. Рассмотрим нормальный алгорифм со схемой из одной формулы $\to P$. Такой алгорифм неприменим ни к какому слову, поскольку на каждом шаге подстановка срабатывает и приводит к вставке в начало слова фрагмента P, т.е. мы получаем неограниченную последовательность X, PX, PPX, ...

Отметим, что отсутствие в схеме терминальных подстановок еще не означает, что алгоритм неприменим к любому слову. Может образоваться последовательность слов, завершающаяся непереводимым словом. Это тоже считается нормальным завершением работы алгоритма.

Как добавить фрагмент в конец слова? Это более сложная задача, чем добавление фрагмента в начало слова. Условимся о следующем. В формулах подстановки будем использовать специальные символы, не входящие в алфавит (для этих символов будем использовать греческий алфавит). На содержательном уровне эти символы играют ту же роль, что и символ „?“ в имени файла, т.е. в левой части такой символ обозначает произвольный символ алфавита (но не пустое слово), а в правой части — тот символ алфавита, на который указывает этот же символ в левой части. С формальной стороны использование такого символа, например ξ, означает, что для каждого символа алфавита в формуле подстановки ξ заменяется на этот символ, давая тем самым конкретную формулу подстановки. Иначе говоря, если в формуле есть специальный символ, то это не формула подстановки, а схема формул. Каждая схема формул дает конечный набор конкретных формул, который и подразумевается взамен схемы формул. Например, пусть дан алфавит $A = \{a, b, c\}$. Тогда запись $a\xi \to \xi a$ обозначает три формулы подстановки: $aa \to aa$, $ab \to ba$, $ac \to ca$.

Чтобы фиксировать порядок формул подстановки в схеме алгоритма при замене схем формул условимся, что алфавит считается упорядоченным. Тогда в каждой схеме формул выбираем самый первый (левый) специальный символ и, последовательно его заменяя в схеме символами алфавита от первого до последнего, получим набор формул, который и заменяет исходную схему формул. Если в новых формулах остаются специальные символы, процедуру повторяем.

Пример 9.3. По-видимому, можно строго показать, что ни один нормальный алгорифм в алфавите A не реализует вставку в конец любого слова фрагмента P. В чем проблема? На последнем шаге должна произойти замена какого-то подслова в конце слова фрагментом P. Но не факт, что это подслово не имеет других вхождений. Возможная стратегия такова. Нужна

какая-то комбинация символов, которая гарантированно не встретится при работе алгоритма. Пусть это будет слово S. Тогда можно составить такую схему:

$$(S\xi \to \xi S,\ S \to\cdot\ P,\ \ \to S).$$

Предполагая, что в исходном слове подслово S не встретится, заключаем, что на первом шаге применяется 3-я подстановка, которая в начало слова вставляет фрагмент S. Далее работает первая схема подстановок, которая любую букву справа от S переставляет в позицию перед S. Первая схема будет пропущена, когда S окажется в конце слова. Тогда сработает вторая подстановка, которая заменит S на P и остановит работу.

Еще одно решение — расширение алфавита A добавлением к нему еще одной буквы $\overline{a} \notin A$. Тогда схема

$$(\overline{a}\xi \to \xi\overline{a},\ \overline{a} \to\cdot\ P,\ \ \to \overline{a})$$

даст нормальный алгорифм, который к любом слову в алфавите A добавит в конце фрагмент P.

В связи с последним примером введем некоторые понятия. Если алфавит B как часть содержит алфавит A, то говорим, что B есть расширение A. Запись этого обычная: $A \subset B$. Формально считаем, что каждый алфавит есть расширение самого себя (т.е. подразумевается нестрогое включение). Мы говорим, что $\mathfrak{A}$ — алгоритм над алфавитом A, если $\mathfrak{A}$ — алгоритм в некотором расширении B алфавита A. Формально алгоритм в алфавите A является алфавитом над A.

Как связаны алгоритм и алфавит? В схеме алгоритма есть некоторый набор символов. Если алфавит содержит все эти символы (но может содержать и еще какие-то символы), то алгоритм в этом алфавите. Если часть символов схемы не входит в алфавит (такие символы можно назвать служебными), то это алгоритм над алфавитом. Как видим жесткой связи между двумя понятиями нет. В принципе мы можем взять любую схему и рассматривать ее как алгоритм над любым алфавитом. Однако если эта парочка несогласована, не следует думать, что алгоритм будет работать так, как мы задумали.

Пусть A — некоторый алфавит. Назовем алгоритмы $\mathfrak{A}$, $\mathfrak{B}$ над алфавитом A ***эквивалентными относительно*** A, если для любого слова $X \in A^*$ выполнены условия:

- если $!\mathfrak{A}(X)$ и $\mathfrak{A}(X) \in A^*$, то $!\mathfrak{B}(X)$ и $\mathfrak{B}(X) = \mathfrak{A}(X)$;
- если $!\mathfrak{B}(X)$ и $\mathfrak{B}(X) \in A^*$, то $!\mathfrak{A}(X)$ и $\mathfrak{A}(X) = \mathfrak{B}(X)$.

Приведенное определение равносильно тому, что два алфавита $\mathfrak{A}$ и $\mathfrak{B}$ либо оба неприменимы к слову $X \in A^*$, либо оба применимы, причем два результата либо равны, либо выходят за пределы алфавита A.

Алгоритмы $\mathfrak{A}$, $\mathfrak{B}$ над алфавитом A назовем ***вполне эквивалентными относительно*** A, если для любого слова $X \in A^*$ либо оба неприменимы к X, либо оба дают одинаковый результат, т.е. либо $\neg!\mathfrak{A}(X) \wedge \neg!\mathfrak{B}(X)$, либо $!\mathfrak{A}(X) \wedge !\mathfrak{B}(X) \wedge (\mathfrak{A}(X) = \mathfrak{B}(X))$.

В дальнейшем для любых алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ и любого слова X будем использовать запись $\mathfrak{A}(X) \simeq \mathfrak{B}(X)$, если либо оба алгоритма неприменимы к слову X, либо оба они применимы и дают одинаковый результат. Выражение $\mathfrak{A}(X) \simeq \mathfrak{B}(X)$ будем называть ***условным равенством***. С помощью этого понятия можно сказать, что алгоритмы $\mathfrak{A}$ и $\mathfrak{B}$ вполне эквивалентны относительно алфавита A, если $\mathfrak{A}(X) \simeq \mathfrak{B}(X)$ для любого $X \in A^*$. В этом случае часто пишут $\mathfrak{A} \simeq \mathfrak{B}$.

Принцип нормализации. Всякий алгоритм в алфавите A вполне эквивалентен относительно A некоторому нормальному алгорифму над алфавитом A.

Этот принцип не является строгим математическим утверждением, поскольку здесь есть не определенное с математической точки зрения понятие: алгоритм в алфавите A. Здесь имеется ввиду, некая механическая процедура, которая для любого слова в алфавите A дает результат — слово в том же алфавите или оказывается неприменимым к этому слову. Но и такое пояснение не есть строгое математическое определение.

9.2. Теорема о переводе и теорема приведения

Постановка задачи о переводе. Лемма о взаимно однозначном соответствии. Теорема о переводе. Теорема о приведении (к двухбуквенному алфавиту).

Введенное понятие эквивалентности алгоритмов по существу означает, что эти алгоритмы дают одинаковые результаты. Предполагается, что эти алгоритмы действуют в одном алфавите. Однако ясно, что один и тот же алгоритм (в содержательном смысле) можно реализовывать в разных алфавитах (подобно тому, что одни и те же знания можно описать разными языками). Как осуществлять перенос алгоритма в другой алфавит? Отметим, что, каков бы ни был по существу перевод алгоритма из одного алфавита в другой, этот перевод — тоже алгоритм.

Пусть задан алфавит A и его некоторое расширение $C = A \cup \{t_1, t_2, \ldots, t_n\}$. Выберем еще две буквы p и q, не входящие в C, и введем еще одно расширение алфавита A — алфавит $B = A \cup \{p, q\}$. Поставим задачу о переносе алгоритмов в алфавите C на алфавит B. Такой перенос может строиться как перенос соответствующих словарных функций с одного алфавита на другой. А для этого необходимо установить соответствие между словами в двух алфавитах.

Если буква $x \in C$ есть буква алфавита A, то ее переводом в алфавит B будет она сама, т.е. $x^\tau = x$, если $x \in A$. Если $x = t_i$, то полагаем $x^\tau = pq^ip$ (степень — в смысле операции конкатенации). Наш перевод должен сохранять операцию конкатенации. Значит, $(XY)^\tau = X^\tau Y^\tau$, в частности, если $X = x_1x_2\ldots x_k$, то $X^\tau = x_1^\tau x_2^\tau \ldots x_k^\tau$. Последняя формула — это определение отображения $\tau\colon C^* \to A^*$. Довольно очевидно, что это отображение сохраняет операцию, т.е. является гомоморфизмом одного моноида в другой. Также очевидны следующие свойства этого отображения.

Теорема 9.1. Пусть A — произвольный алфавит, B и C — его расширения, введенные выше. Тогда отображение τ обладает следующими свойствами:

1) отображение τ — мономорфизм, т.е. из равенства $X^\tau = Y^\tau$ следует равенство $X = Y$;
2) равенство $X^\tau = X$ верно тогда и только тогда, когда $X \in A^*$.

◄ Второе свойство очевидно. Действительно, если $X \in A^*$, то его переводом, согласно определению является оно само, поскольку перевод осуществляется побуквенно. Если $X \notin A^*$, то в X есть одна из букв $t_1, \ldots, t_n$. Значит в слове X^τ появляются буквы p и q, т.е. $X^\tau \notin A^*$.

Рассмотрим первое свойство. Пусть $X \neq Y$. Обозначим $X = x_1x_2\ldots x_m$, $Y = y_1y_2\ldots y_l$. Найдется такой индекс i, что $x_1 = y_1, \ \ldots, x_{i-1} = y_{i-1}$, но $x_i \neq y_i$. Тогда X^τ имеет вид $x_1^\tau \ldots x_{i-1}^\tau x_i^\tau \ldots$, а Y^τ — вид $y_1^\tau \ldots y_{i-1}^\tau y_i^\tau \ldots$. В данном случае $x_1^\tau = y_1^\tau, \ldots, x_{i-1}^\tau = y_{i-1}^\tau$. Однако $x_i^\tau \neq y_i^\tau$. Поэтому $X^\tau \neq Y^\tau$. ►

Мономорфизм τ, согласно доказанной теореме, устанавливает взаимно однозначное соответствие между множеством C^* всех слов в алфавите C и некоторым множеством слов в алфавите B. Это соответствие позволяет реализовать любую словарную функцию в алфавите C как словарную функцию в алфавите B.

Для нормального алгорифма $\mathfrak{A}$ в алфавите C со схемой $(P_1 \to* Q_1,\ P_2 \to* Q_2,\ P_k \to* Q_k)$ рассмотрим нормальный алгорифм $\mathfrak{A}^\tau$ в алфавите B со схемой $(P_1^\tau \to* Q_1^\tau,\ P_2^\tau \to* Q_2^\tau,\ P_k^\tau \to* Q_k^\tau)$. Этот алгоритм будем называть переводом нормального алгорифма $\mathfrak{A}$ из алфавита C в алфавит B.

Теорема 9.2 (о переводе алгоритма). Пусть A — произвольный алфавит, B и C — построенные выше его расширения, $\mathfrak{A}$ — нормальный алгорифм в алфавите C. Тогда для любого слова $X \in C^*$ выполнено равенство $\mathfrak{A}^\tau(X^\tau) \simeq \mathfrak{A}(X)^\tau$. При этом $\mathfrak{A}^\tau(X) \simeq \mathfrak{A}(X)^\tau$, если $X \in A^*$, и $\mathfrak{A}^\tau(X) \simeq \mathfrak{A}(X)$, если $X \in A^*$ и $\mathfrak{A}(X) \in A^*$.

◄ Следует проверить пошаговую работу алгоритма: если утверждение теоремы верно для любого алгоритма, то оно верно и для алгоритма, в котором все подстановки имеют статус терминальных. Такой алгоритм выполняет в точности то же, что и заданный, но останавливается после первого шага. Для этого достаточно убедиться, что произвольная подстановка

$P \to Q$ применима к слову $X \in C^*$ тогда и только тогда, когда подстановка применима к слову X^τ, причем результат второй есть перевод результата первой.

Если $X = V_1PV_2$, то по свойствам операции перевода $X^\tau = V_1^\tau Q^\tau V_2^\tau$, и мы видим, что подстановка $P^\tau \to Q^\tau$ применима к слову X^τ, причем

$$\mathfrak{B}^\tau(X^\tau) = V_1^\tau Q^\tau V_2^\tau = (V_1QV_2)^\tau = \mathfrak{B}(X)^\tau$$

(здесь $\mathfrak{B}$ — алгоритм со схемой из единственной подстановки $P \to\cdot\, Q$).

Покажем, что каждому вхождению фрагмента P^τ в слово X^τ соответствует вхождение фрагмента P в слово X. Достаточно рассмотреть случай, когда P — это одна буква. Пусть $Y = X^\tau = y_1y_2\ldots y_m$ и вхождение фрагмента P^τ начинается с буквы y_j. Если $P \in A$, то $P^\tau = P$, это единственная буква $y_j \in A$. Эта буква не входит в слова $t_i^\gamma = pq^ip$. Значит, в слове X букве y_j соответствует та же буква. Если $P = t_i$, то $y_j = p$ и не является буквой алфавита A. При этом $y_{j+1} = q$, а буквенная пара pq в слове Y обозначает начало перевода x_l^τ некоторой буквы x_l слова X. Аналогично слово P^τ заканчивается парой qp, что соответствует концу перевода в слове X^τ. Наконец, фрагмент Y^τ содержит всего лишь две буквы p, а значит в слове X^τ может соответствовать переводу только одной буквы. Тем самым показано, что если существует вхождение слова P^τ в слово X^τ, то буква P входит в слово X. Отсюда легко сделать вывод, что для произвольного слова P из существования вхождения P^τ в X^τ следует существование вхождения P в X. ►

Теорема 9.3 (о приведении к двухбуквенному алфавиту). Пусть A — произвольный алфавит, p и q — две разные буквы, не входящие в A. Тогда любой нормальный алгорифм над A эквивалентен относительно A некоторому нормальному алгорифму в алфавите $A \cup \{p,\, q\}$.

◄ Пусть $\mathfrak{A}$ — нормальный алгорифм над алфавитом A и $t_1,\, t_2,\, \ldots,\, t_n$ — множество всех букв, использованных в схеме $\mathfrak{A}$, но не входящих в алфавит $B = A \cup \{p,\, q\}$. Тогда $\mathfrak{A}$ — нормальный алгорифм в алфавите $C = A \cup \{t_1,\, t_2,\, \ldots,\, t_n\}$. Построим перевод τ из алфавита C в алфавит B. Согласно теореме 9.2, имеем $\mathfrak{A}^\tau(X^\tau) \simeq \mathfrak{A}(X)^\tau$, причем, если $X \in A$, $\mathfrak{A}(X) \in A$, то $\mathfrak{A}^\tau(X) \simeq \mathfrak{A}(X)$. Это означает, что алгоритмы $\mathfrak{A}$ и $\mathfrak{A}^\tau$ эквивалентны. При этом $\mathfrak{A}^\tau$ — алгоритм в алфавите B. ►

9.3. Операции над нормальными алгорифмами

Расширение НА. Замыкание. Композиция. Соединение. Разветвление. Повторение.

Над алфавитами можно выполнять различные операции. Комбинируя определенным образом алгоритмы, можно получать новые алгоритмы. В некоторых ситуациях такое комбинирование не затрагивает сути комбинируемых алгоритмов. В таких ситуациях возникает операция над алгоритмами. Поскольку для нас важна не формальная запись алгоритма, а выполняемое им действие, операции над алгоритмами могут выполняться с точностью до эквивалентности, т.е. эквивалентные алгоритмы мы не различаем.

Расширение алфавита. Как было отмечено, нормальный алгорифм слабо связан с алфавитом, над которым он действует. Одна и та же схема порождает алгоритмы над разными алфавитами. Различие между такими алгоритмами — лишь в области применимости.

Рассмотрим два варианта расширения нормального алгорифма. Пусть $\mathfrak{A}$ — нормальный алгорифм над алфавитом A, заданный схемой $T = \{P_1 \to* Q_1,\, \ldots,\, P_n \to* Q_n\}$. Алгоритм $\mathfrak{A}^e$, который над алфавитом $B \supset A$ задан в точности той же схемой T, называется ***естественным расширением*** алгоритма $\mathfrak{A}$. Этот алгоритм вполне эквивалентен исходному алгоритму, поскольку расширение алфавита никак не отражается на его работе, если исходное слово берется в алфавите A. Однако фактическая область применимости при этом расширяется, при

этом действие алфавита может зависеть от того, входят ли новые буквы (т.е. из $B \setminus A$) в схему или нет.

При втором варианте расширения алфавита мы также сохраняем эквивалентность алгоритма относительно A, но сохраняем также и область применимости. Пусть $\mathfrak{A}$ — нормальный алгорифм над алфавитом A, заданный схемой $T = \{P_1 \to * Q_1, \ldots, P_n \to * Q_n\}$. Алгоритм $\mathfrak{A}^f$ над $B \supset A$ со схемой

$$\begin{cases} \xi \to \xi, \quad \xi \in B \setminus A, \\ P_1 \to * Q_1, \\ P_2 \to * Q_2, \\ \ldots\ldots\ldots\ldots \\ P_n \to * Q_n \end{cases}$$

называется ***формальным расширением*** алгоритма $\mathfrak{A}$. Если в схему не входят новые буквы (из множества $B \setminus A$), в частности, если $\mathfrak{A}$ — алгоритм в алфавите A, то добавление новых подстановок не отразится на работе алгоритма, начинающейся со слова в алфавите A, т.е. новый алгоритм вполне эквивалентен исходному. Однако алгоритм $\mathfrak{A}^f$ неприменим к словам, содержащим хотя бы одну новую букву. Другими словами, формальное расширение алгоритма сохраняет область применимости алгоритма.

Пример 9.4. Рассмотрим алгоритм $\mathfrak{A}$ над алфавитом A со схемой

$$\begin{cases} *\xi \to \xi *, \\ * \to \cdot\, P, \\ \Lambda \to *, \end{cases}$$

где символ $*$ не входит в A. Этот алгоритм добавляет в конец слова из алфавита A слово P (которое можем считать словом в алфавите A). Рассмотрим алфавит $B = A \cup \{*\}$ и формальное расширение $\mathfrak{A}^f$ алгоритма $\mathfrak{A}$, расширив его схему:

$$\begin{cases} * \to *, \\ *\xi \to \xi *, \\ * \to \cdot\, P, \\ \Lambda \to *. \end{cases}$$

Нетрудно увидеть, что этот алгоритм неприменим ни к одному слову в алфавите A: на первом шаге в начало слова вставляется символ $*$, а далее работает только первая подстановка, приводящая к зацикливанию.

Мы видим, что наличие в схеме одной из новых букв может привести к алгоритму, не эквивалентному исходному.

Замыкание. В общем случае есть два сценария прекращения работы нормального алгорифма: по терминальной подстановке и по отсутствию подходящей подстановки. Можно модифицировать алгоритм так, что второй сценарий никогда не будет случаться. Такая модификация называется ***замыканием нормального алгорифма***. Выполняется замыкание добавлением в конец схемы подстановки $\Lambda \to \cdot\, \Lambda$. Наличие подобной подстановки в конце схемы означает, что при любом варианте подстановка произойдет, т.е. прекращение работы алгоритма по отсутствию подходящей подстановки никогда не происходит.

Пример 9.5. Рассмотрим алгоритм $\mathfrak{A}$ над алфавитом A со схемой

$$\begin{cases} *\xi \to \xi *, \\ * \to \cdot\, P, \\ \Lambda \to *, \end{cases}$$

где символ $*$ не входит в A. Его замыканием, согласно определению, является алгоритм со схемой

$$\begin{cases} *\xi \to \xi *, \\ * \to\cdot\ P, \\ \Lambda \to *, \qquad \Lambda \to\cdot\ \Lambda. \end{cases}$$

Отметим, что добавление в конец схемы еще одной подстановки по сути ничего не меняет: ни при каких условиях добавленная подстановка не будет применяться из-за предшествующей подстановки. Последнее надо трактовать так: исходный алгоритм уже является замкнутым.

Как показывает пример, наличие подстановки с пустой левой частью — достаточное условие замкнутости. С другой стороны, если в схеме алгоритма отсутствует подстановка с пустой левой частью, то, по крайней мере, есть одно слово в рассматриваемом алфавите, к которому неприменима ни одна подстановка схемы: пустое слово. Значит, наличие подстановки с пустой левой частью — и необходимое условие замкнутости.

Теорема 9.4. Пусть A — произвольный алфавит. Замыкание $\overline{\mathfrak{A}}$ алгоритма $\mathfrak{A}$ вполне эквивалентно исходному алгоритму $\mathfrak{A}$.

◀ Утверждение почти тривиально. Пусть X — исходное слово. Если в алгоритме gA есть подстановка τ, применимая к X, то эта же подстановка есть и в алгоритме $\overline{\mathfrak{A}}$, причем множество подстановок в схеме $\mathfrak{A}$, предшествующих τ, совпадает с множеством подстановок схемы $\overline{\mathfrak{A}}$, предшествующих τ. Значит, если к слову X в алгоритме $\mathfrak{A}$ будет применена подстановка τ, то и в алгоритме $\overline{\mathfrak{A}}$ будет применена она же. В этом случае результаты работы двух алгоритмов будут одинаковы, т.е. они дают одно и то же преобразование слова X и имеют одно и то же условие прекращения работы (это условие — терминальность τ).

Если в алгоритме $\mathfrak{A}$ нет ни одной подстановки, применимой к X, то этот алгоритм прекращает работу и итогом этой работы будет слово X. Для алгоритма $\overline{\mathfrak{A}}$ это означает, что к слову X может быть применена только последняя подстановка $\Lambda \to\cdot\ \Lambda$. В результате его применения слово X не изменится, а алгоритм завершит работу. Мы видим, что и в этом случае алгоритмы работают одинаково.

Наши рассуждения показывают, что пошаговая последовательность работы двух алгоритмов совпадает. Следовательно, они вполне эквивалентны. ▶

Композиция нормальных алгорифмов. Алгоритм $\mathfrak{A}$ (в нашем понимании нормальный алгорифм) перерабатывает исходное слово X в некоторое слово Y, т.е. задает некоторое отображение (словарную функцию) $\mathfrak{A}$: $D \to A^*$, $D \subset A^*$, т.е. частичную словарную функцию. Для таких функций есть привычная и естественная операция — композиция. Отображение $\mathfrak{B} \circ \mathfrak{A}$ — это функция $\mathfrak{B}(\mathfrak{A}(X))$, областью определения которой является множество $\mathfrak{A}^{-1}(\mathrm{dom}(\mathfrak{B}))$, где $\mathrm{dom}(\mathfrak{B})$ — область определения функции $\mathfrak{B}$, а $\mathfrak{A}^{-1}(Y)$ — стандартное обозначение полного прообраза множества Y.

Два алгоритма $\mathfrak{A}$ и $\mathfrak{B}$ задают словарную функцию $\mathfrak{B}\circ\mathfrak{A}$, которая описывает последовательное применение двух алгоритмов (рис. 9.1). Ясно, что эта функция эффективно вычислимая (налицо конечная последовательность правил, описывающих получение значения этой функции). Однако возникает вопрос: можно ли эту функцию задать нормальным алгорифмом? Исходя из принципа нормализации мы должны дать положительный ответ. Но принцип — всего лишь надежда, возможно уверенность, но никак не строгое математическое утверждение. Мы постоянно должны подтверждать этот принцип, проверяя его в конкретных ситуациях, как и в ситуации с композицией.

Рис. 9.1

Теорема 9.5. Пусть $\mathfrak{A}$ и $\mathfrak{B}$ — нормальные алгорифмы в алфавите A. Существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, такой, что для любого слова $X \in A^*$ выполняется соотношение $\mathfrak{C}(X) \simeq \mathfrak{B}(\mathfrak{A}(X))$.

◀ Здесь „существует“ — вполне в духе математической логики и оснований математики: речь идет о построении конкретной схемы, обеспечивающей нужное свойство.

Собственно, надо соединить две схемы в одну, но при этом обеспечить, чтобы до определенного момента подходили подстановки только первой схемы $\mathfrak{A}$, а затем только второй схемы $\mathfrak{B}$. Первое сделать нетрудно: достаточно расположить подстановки первой схемы перед подстановками второй. Чтобы не было проскакивания (когда ни одна подстановка алгоритма $\mathfrak{A}$ не подходит), алгоритм $\mathfrak{A}$ надо замкнуть. Чтобы после завершения работы $\mathfrak{A}$ можно было продолжить, надо схему $\mathfrak{A}$ перестроить, изменив терминальные подстановки. При этом каждая такая подстановка должна инициировать процесс, при котором слово изменяется так, что ни одна подстановка алгоритма $\mathfrak{A}$ не будет применена. После этого заработает алгоритм $\mathfrak{B}$, остановку которого надо переделать в заключительный процесс. Вот эти туманные соображения можно реализовать следующим образом.

Рассмотрим алфавит $\tilde{A}$, который содержит столько же букв, сколько и A, причем $A \cap \tilde{A} = \varnothing$. Тогда у каждой буквы $x \in A$ появляется двойник $\tilde{x}$. Выберем еще две разные буквы p и q, не входящие ни в A, ни в $\tilde{A}$. Новые буквы используем для построения схемы с нужными свойствами. Алгоритм $\mathfrak{A}$ замкнем (если он не замкнут) и в схеме замыкания алгоритма $\mathfrak{A}$ (применяемого первым) заменим каждую подстановку вида $P \to\cdot\ Q$ подстановкой $P \to pQ$ (слова P и Q могут быть пустыми). Полученную схему обозначим $\mathfrak{A}_p$.

Алгоритм $\mathfrak{B}$ замкнем, затем в его схеме заменим все буквы алфавита A на их двойники из алфавита $\tilde{A}$. Кроме того, заменим формулы вида $\Lambda \to P$ на $p \to pP$, а все терминальные формулы вида $P \to\cdot\ Q$ заменяем на $P \to qQ$. Полученную схему обозначим $\mathfrak{B}_{pq}$. Теперь строим следующую схему:

$$\begin{cases} \xi p \to p\xi & \text{(букву } p \text{ протаскиваем в начало)},\\ p\xi \to p\tilde{\xi} & \text{(первую букву меняем на двойника)},\\ \tilde{\xi}\eta \to \tilde{\xi}\tilde{\eta} & \text{(остальные буквы меняем двойниками)},\\ \tilde{\xi}q \to q\tilde{\xi} & \text{(букву } q \text{ протаскиваем в начало)},\\ q\tilde{\xi} \to q\xi & \text{(двойника первой буквы меняем на саму букву)},\\ \xi\tilde{\eta} \to \xi\eta & \text{(остальные двойники меняем на буквы)},\\ pq \to\cdot\ \Lambda & \text{(заканчиваем)},\\ \mathfrak{B}_{pq}, & \\ \mathfrak{A}_p. & \end{cases}$$

Здесь ξ, η — произвольная буква алфавита A, а $\tilde{\xi}$, $\tilde{\eta}$ — их двойники из алфавита $\tilde{A}$.

В сконструированной схеме первые три групповые подстановки перетаскивают спецсимвол p в начало слова и заменяют все буквы двойниками, подготавливая работу второго алгоритма. Запускаются эти подстановки, когда в слове появляется символ p, т.е. когда завершит работу группа $\mathfrak{A}_p$. Следующие три групповые подстановки делают обратную операцию, возвращая взамен двойников изначальные буквы. Седьмая строка завершает работу алгоритма.

Из построения видно, что в результате работы сконструированной схемы мы получим последовательную работу алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$. ▶

Рассмотрим случай композиции алгоритмов в разных алфавитах.

Теорема 9.6 (общая теорема композиции). Для любого нормального алгорифма $\mathfrak{A}$ в алфавите A и нормального алгорифма $\mathfrak{B}$ в алфавите B существует нормальный алгорифм $\mathfrak{C}$ над алфавитом $A \cup B$, для которого $\mathfrak{C}(X) \simeq \mathfrak{B}(\mathfrak{A}(X))$, $X \in A^*$.

◄ Построим формальные расширения $\mathfrak{A}_1$, $\mathfrak{B}_1$ алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ на алфавит $C = A \cup B$. Согласно теореме 9.5, существует нормальный алгорифм $\mathfrak{C}$, для которого $\mathfrak{B}_1(\mathfrak{A}_1(X)) \simeq \mathfrak{C}(X)$, $X \in C^*$. В силу эквивалентности $\mathfrak{A}$ и $\mathfrak{A}_1$, $\mathfrak{B}$ и $\mathfrak{B}_1$ получаем $\mathfrak{B}(\mathfrak{A}(X)) \simeq \mathfrak{C}(X)$, $X \in C^*$. ►

Замечание 9.1. Теорему 9.6 можно переформулировать, заменив понятие „алгоритм в алфавите“ понятием „алгоритм над алфавитом“. Действительно, если $\mathfrak{A}$ — алгоритм над алфавитом A, $\mathfrak{B}$ — алгоритм над алфавитом B, то $\mathfrak{A}$ — алгоритм в алфавите $\tilde{A} \supset A$, который получен добавлением к A спецсимволов алгоритма $\mathfrak{A}$. Аналогично $\mathfrak{B}$ — алгоритм в алфавите $\tilde{B}$. Значит, существует нормальный алгорифм $\mathfrak{B} \circ \mathfrak{A}$ над алфавитом $\tilde{C} = \tilde{A} \cup \tilde{B}$. Сузив алфавит $\tilde{C}$ до $C = A \cup B$, получим требуемое утверждение.

Композицию нормальных алгорифмов $\mathfrak{A}$ и $\mathfrak{B}$ будем обозначать как композицию функций: $\mathfrak{B} \circ \mathfrak{A}$ (справа налево).

Соединение нормальных алгорифмов. Соединением алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ в алфавите A называется алгоритм $\mathfrak{C} = \mathfrak{A}\&\mathfrak{B}$, который каждому слову $X \in A^*$ ставит в соответствие слово $\mathfrak{A}(X)\mathfrak{B}(X)$ (рис. 9.2). Как и в случае композиции в данном случае необходимо показать, что объединение нормальных алгорифмов может быть реализовано как нормальный алгорифм.

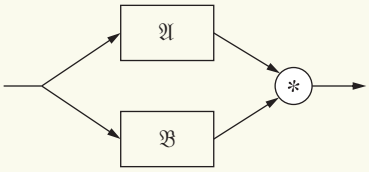

Рис. 9.2

Теорема 9.7. Для любых нормальных алгорифмов $\mathfrak{A}$ и $\mathfrak{B}$ в алфавите A существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, для которого $\mathfrak{C}(X) \simeq \mathfrak{A}(X)\mathfrak{B}(X)$, $X \in A^*$.

◄ Процесс перехода от X к $\mathfrak{A}(X)\mathfrak{B}(X)$ можно представить как последовательность преобразований:

$$X \to X\,\overline{X} \to \overline{\mathfrak{A}}(\overline{X})X = \overline{\mathfrak{A}(X)}X \to \overline{\mathfrak{A}(X)}\mathfrak{B}(X) \to \mathfrak{A}(X)\mathfrak{B}(X),$$

где черта сверху обозначает слово-двойник или алгоритм в алфавите-двойнике. Общее преобразование представляет собой композицию нескольких алгоритмов. Построим эти алгоритмы.

Первое преобразование осуществляется последовательным применением двух алгоритмов $\mathfrak{C}_1$ со схемой $\big(p\xi \to \xi\overline{\xi}p,\ p \to\cdot\ \Lambda,\ \Lambda \to p\big)$ и $\mathfrak{C}_2$ со схемой $\overline{\xi}\eta \to \eta\overline{\xi}$.

Второе преобразование осуществляется переводом $\overline{\mathfrak{A}}$ алгоритма $\mathfrak{A}$ на алфавит $\overline{A}$. Третье преобразование — с помощью алгоритма $\mathfrak{B}$. Наконец, необходимо вместо букв-двойников подставить оригиналы. Это осуществляется алгоритмом со схемой $\overline{\xi} \to \xi$. ►

Доказанную теорему расширяем на более общий случай алгоритмов в разных алфавитах.

Теорема 9.8. Для любого нормального алгорифма $\mathfrak{A}$ в алфавите A и нормального алгорифма $\mathfrak{B}$ в алфавите B существует нормальный алгорифм $\mathfrak{C}$ в алфавите $C = A \cup B$, для которого $\mathfrak{C}(X) \simeq \mathfrak{A}(X)\mathfrak{B}(X)$, $X \in C^*$.

◄ Как и в случае композиции, строим формальные расширения алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ на алфавит C, а затем применяем теорему 9.7. ►

Следствие 9.1. Для любого нормального алгорифма $\mathfrak{A}$ в алфавите A, нормального алгорифма $\mathfrak{B}$ в алфавите B и буквы p существует нормальный алгорифм $\mathfrak{C}$ в алфавите $C = A \cup B \cup \{p\}$, для которого $\mathfrak{C}(X) \simeq \mathfrak{A}(X)p\mathfrak{B}(X)$, $X \in (A \cup B)^*$.

◄ Алгоритм $\mathfrak{C}$ есть объединение трех алфавитов: $\mathfrak{A}$, алфавита $\mathfrak{P}$, который любое слово в алфавите $A \cup B$ заменяет на букву p, и алфавита $\mathfrak{B}$. Отметим, что объединение алфавитов ассоциативно: это просто поточечная конкатенация словарных функций, а теоремы лишь утверждают, что любая такая конкатенация может быть реализована как нормальный алгорифм. ►

Разветвление нормальных алгорифмов. На практике бывает, что при реализации того или иного метода в данный момент надо выбрать одно из возможных действий в зависимости от полученного результата. В языках программирования это реализуется условными конструкциями, например if ... else ... end. С точки зрения алгоритмов здесь участвуют три алгоритма $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$: первый $\mathfrak{L}$ формирует условие, при выполнении которого реализуется второй алгоритм $\mathfrak{A}$, а при нарушении — третий $\mathfrak{B}$ (рис. 9.3). Конкретизировать это общее описание можно с помощью словарных функции следующим образом:

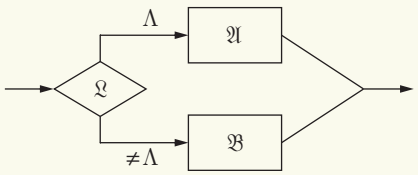

Рис. 9.3

$$\mathfrak{C}(X) = \begin{cases} \mathfrak{A}(X), & \mathfrak{L}(X) = \Lambda; \\ \mathfrak{B}(X), & \mathfrak{L}(X) \neq \Lambda. \end{cases}$$

Соответствующий алгоритм $\mathfrak{C}$ однозначно описывается следующими условиями:

1) $X \in A^* \wedge !\mathfrak{L}(X) \wedge (\mathfrak{L}(X) = \Lambda) \to \mathfrak{C}(X) \simeq \mathfrak{A}(X)$;
2) $X \in A^* \wedge !\mathfrak{L}(X) \wedge (\mathfrak{L}(X) \neq \Lambda) \to \mathfrak{C}(X) \simeq \mathfrak{B}(X)$;
3) $X \in A^* \to (\neg !\mathfrak{L}(X) \to \neg !\mathfrak{C}(X))$.

Такой алгоритм будем называть ***разветвлением на алгоритмы $\mathfrak{A}$ и $\mathfrak{B}$ под управлением $\mathfrak{L}$*** и обозначать $S(\mathfrak{L}, \mathfrak{A}, \mathfrak{B})$ (от английского select).

Ясно, что если все словарные функции $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$, эффективно вычислимы, то построенная функция $\mathfrak{C}$ эффективно вычислима. Как реализовать эту конструкцию в виде нормального алгорифма?

Теорема 9.9. Пусть $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$ — нормальные алгорифмы в алфавите A. Тогда существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, осуществляющий разветвление на $\mathfrak{A}$ и $\mathfrak{B}$ под управлением $\mathfrak{L}$.

◄ Один из вариантов построения требуемого алгоритма такой. На основе алгоритма $\mathfrak{L}$ с помощью некоторой буквы $p \notin A$ строим алгоритм $\mathfrak{L}_1$, обладающий свойствами: $!\mathfrak{L}(x) \Leftrightarrow !\mathfrak{L}_1(X)$; $\mathfrak{L}_1(X) = pX$, если $\mathfrak{L}(X) = \Lambda$; $\mathfrak{L}_1(X) = X$, если $\mathfrak{L}(X) \neq \Lambda$. Затем устраиваем композицию $\mathfrak{L}_1$, $\mathfrak{A}$ и $\mathfrak{B}$, но при этом скорректируем алгоритмы $\mathfrak{A}$ и $\mathfrak{B}$ так, чтобы отрабатывал только один из них. Для этого строим алгоритм $\mathfrak{B}_1$ следующим образом: берем замыкание алгоритма $\mathfrak{B}$, в начало его схемы добавляем подстановку $p \to\cdot \Lambda$, а в самой схеме каждую терминальную подстановку вида $P_i \to\cdot Q_i$ заменяем подстановкой $P_i \to\cdot qQ_i$, где $q \notin A$. Строим алгоритм $\mathfrak{A}_1$, добавляя в начало схемы алгоритма $\mathfrak{A}$ подстановку $q \to\cdot \Lambda$. После этого строим композицию $\mathfrak{C} = \mathfrak{A}_1 \circ \mathfrak{B}_1 \circ \mathfrak{L}_1$.

Работа построенного алгоритма протекает следующим образом. Для слова X алгоритм $\mathfrak{L}_1$ либо неприменим, либо дает pX, либо не изменяет слово X. Если $\mathfrak{L}_1$ (а значит, и $\mathfrak{L}$) неприменим к X, то и построенный алгоритм неприменим к X. Если алгоритм $\mathfrak{L}_1$ дает слово pX, то следующий алгоритм $\mathfrak{B}$ просто удаляет p и останавливается. Далее алгоритм $\mathfrak{A}$ работает со словом X. Если же алгоритм $\mathfrak{L}_1$ дает слово X, то далее алгоритм $\mathfrak{B}_1$ работает как $\mathfrak{B}$, давая в качестве результата слово $Y = \mathfrak{B}(X)$, в которое вставлена буква q. Для этого слова действие алгоритма $\mathfrak{A}_1$ состоит в удалении буквы q и остановке. Мы на выходе получаем $\mathfrak{B}(X)$.

Видно, что алгоритм $\mathfrak{C}$ удовлетворяет необходимым условиям. Остается построить алгоритм $\mathfrak{L}_1$. Сперва выбираем композицию $\mathfrak{L}_a \circ \mathfrak{L}$, где $\mathfrak{L}_a$ — нормальный алгорифм над A со схемой $(\xi\eta \to \xi,\ \xi \to\cdot \Lambda,\ \Lambda \to\cdot p)$, где $p \notin A$. В результате получаем алгорритм, который выдает два значения: Λ, если $\mathfrak{L}(X) \neq \Lambda$, и p, если $\mathfrak{L}(X) = \Lambda$. Теперь берем соединение построенного алгоритма с алгоритмом $\mathfrak{L}_b$ со схемой из одной подстановки $\Lambda \to\cdot \Lambda$ (он любое слово оставляет неизменным). ►

Как и в случае объединения алгоритмов, доказанная теорема обобщается на случай неодинаковых алфавитов.

Теорема 9.10 (общая о разветвлении). Пусть $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$ — нормальные алгорифмы в алфавитах L, A и B соответственно. Существует нормальный алгорифм $\mathfrak{C}$ над алфавитом $C = A \cup B \cup L$, осуществляющий разветвление на $\mathfrak{A}$ и $\mathfrak{B}$ под управлением $\mathfrak{L}$.

◀ Достаточно устроить формальное расширение алфавитов на алфавит C и применить теорему 9.9. ▶

Повторение алгоритма. Разветвление алгоритма — первый пример работы под управлением алгоритма: алгоритм $\mathfrak{L}$ играет управляющую роль, которая в этом случае состоит в выборе одного из двух алгоритмов. Еще один пример работы под управлением — повторение алгоритма. Суть процедуры в том, что после применения некоторого алгоритма A результат тестируется с помощью еще одного алгоритма $\mathfrak{L}$. По результатам теста принимается решение, повторить алгоритм $\mathfrak{A}$ или остановить работу. Схематично эта процедура показана на рис. 9.4.

Будем говорить о том, что алгоритм $\mathfrak{C}$ реализует повторение алгоритма $\mathfrak{A}$ под управлением алгоритма $\mathfrak{L}$, если алгоритм $\mathfrak{C}$ применим к слову X тогда и только тогда, когда существуют слова X_0, X_1, …, $X_n = Y$, удовлетворяющие условиям $X_i = \mathfrak{A}(X_{i-1})$, $i = \overline{1, n}$, $\mathfrak{L}(X_i) \neq \Lambda$ при $i = \overline{1, n-1}$ и $\mathfrak{L}(X_n) = \Lambda$. При этом слово Y есть результат работы алгоритма $\mathfrak{C}(X)$.

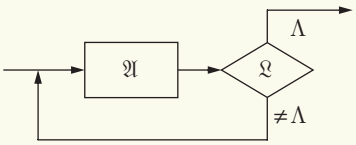

Рис. 9.4

Теорема 9.11. Для любых алгоритмов $\mathfrak{A}$ и $\mathfrak{L}$ в алфавите A существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, реализующий повторение $\mathfrak{A}$ под управлением $\mathfrak{L}$.

◀ Идея доказательства такова. По алгоритму $\mathfrak{L}$ строим алгоритм $\mathfrak{L}_1$, для которого $!\mathfrak{L}(X) \Leftrightarrow$ $\Leftrightarrow !\mathfrak{L}_1(X)$ и который добавляет к слову X символ p, если $\mathfrak{L}(X) = \Lambda$. Построение этого алгоритма описано в доказательстве теоремы 9.9. Рассмотрим алгоритм $\mathfrak{A}_1 = \mathfrak{L}_1 \circ \mathfrak{A}$. Он выполняет работу алгоритма $\mathfrak{A}$, а потом, если нужно, добавляет символ p слева, что сигнализирует о прекращении работы. Далее делаем следующее. Замыкаем $\mathfrak{A}_1$ и заменяем терминальные подстановки вида $P \to\cdot\, Q$ на подстановки $P \to qQ$ (буква q сигнализирует о завершении цикла). Получим алгоритм $\mathfrak{A}_q$. После этого строим схему

$$\begin{cases} \xi q \to q\xi, \\ pq \to\cdot\, q, \\ q \to \Lambda, \\ \mathfrak{A}_q. \end{cases}$$

Этот алгоритм решает поставленную задачу. ▶

Как и в других предшествующих ситуациях, обобщаем последнюю теорему.

Теорема 9.12 (общая о повторении). Для любого алгоритма $\mathfrak{A}$ в алфавите A и алгоритма $\mathfrak{L}$ в алфавите L существует алгоритм $\mathfrak{C}$ над алфавитом $C = A \cup L$, реализующий повторение $\mathfrak{A}$ под управлением $\mathfrak{L}$.

◀ Как и выше, достаточно построить формальное расширение алгоритмов на объединенный алфавит и затем применить теорему 9.11. ▶

Отметим, что реализованная схема повторения $\mathfrak{A}$ под управлением $\mathfrak{L}$ приводит к тому, что алгоритм $\mathfrak{A}$ выполняется по крайней мере один раз. Однако известно, что циклы можно

организовывать так, что тело цикла вообще может не выполняться ни разу. Предыдущий вариант алгоритма с повторением будем называть ***алгоритмом с постусловием*** и обозначать $RA(\mathfrak{L}, \mathfrak{A})$, а вариант алгоритма, когда цикл может ни разу не выполняться — ***алгоритмом с предусловием*** и обозначать $RB(\mathfrak{L}, \mathfrak{A})$. Схема такого алгоритма показана на рис. 9.5.

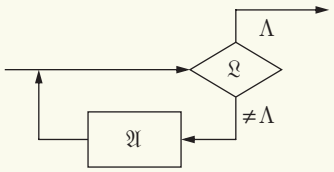

Рис. 9.5

Теорема 9.13. Для любого алгоритма $\mathfrak{A}$ в алфавите A и алгоритма $\mathfrak{L}$ в алфавите L существует алгоритм $\mathfrak{C}$ над алфавитом $C = A \cup L$, реализующий повторение $\mathfrak{A}$ под управлением $\mathfrak{L}$ следующим образом. Алгоритм $\mathfrak{C}$ применим к слову X тогда и только тогда, когда существуют слова $X_0 = X$, X_1, …, X_n, связанные соотношениями $X_i = \mathfrak{A}(X_{i-1})$, $i = \overline{1, n}$; $\mathfrak{L}(X_i) \neq \Lambda$, $i = \overline{0, n-1}$; $\mathfrak{L}(X_n) = \Lambda$.

◀ Построим алгоритм $\mathfrak{L}_1$, как выше, который к слову X добавляет символ p, если $\mathfrak{L}(X) = \Lambda$. Затем видоизменим алгоритм $\mathfrak{A}$, заменив разветвлением $(\mathfrak{A} \curlyvee \mathfrak{L}_1|\mathfrak{L})$ (т.е. разветвление на $\mathfrak{A}$ и $\mathfrak{L}_1$ под управлением $\mathfrak{L}$). Если $\mathfrak{L}(X) \neq \Lambda$, такой алгоритм работает как обычное повторение $\mathfrak{A}$, поскольку разветвление на $\mathfrak{A}$ и $\mathfrak{L}_1$ проходит после отрицательной проверки на $\mathfrak{L}$. Если же $\mathfrak{L}(X) = \Lambda$, то первое же разветвление даст $\mathfrak{L}_1$ с последующим завершением. что и требовалось. ▶

9.4. Теорема об универсальном нормальном алгорифме

Каждый алгоритм — заранее фиксированная последовательность действий. Его массовость обеспечивается произвольным характером исходного слова в данном алфавите. В таком контексте алгоритм можно уподобить программе для компьютера. Однако сам компьютер — тоже алгоритм: это автомат, который работает и с текстом программы, и с данными. Было бы печально, если бы требовалось при переходе к новой задаче менять компьютер.

Но если компьютер — тоже алгоритм, то его можно записать как нормальный алгорифм, для которого данными являются запись заданного нормального алгорифма (программа) и исходное слово, к которому нужно применить заданный алгоритм (данные). Такой нормальный алгорифм называют ***универсальным нормальным алгорифмом***.

Перейдем к точным формулировкам. Условимся, что задан и зафиксирован алфавит A, и мы рассматриваем нормальные алгорифмы в этом алфавите.

Сперва запись нормального алгорифма, а точнее, его схемы. Вводим новую букву $\gamma \notin A$ и рассматриваем слова в алфавите $A_\gamma = A \cup \gamma$ вида γP, где $P \in A_\gamma^*$ и заканчивается буквой γ. Каждое такое слово назовем ***системой слов*** в алфавите A (точнее γ-системой). Систему слов можно представить в виде

$$R = \gamma X_1 \gamma X_2 \ldots \gamma X_n \gamma,$$

где $X_i \in A^*$, $i = \overline{1, n}$, — ***элементы системы слов***. Отметим, что буква γ, рассматриваемая как слово, представляет собой ***пустую систему слов***.

Понятие системы слов позволяет представить в виде слова в некотором алфавите схему нормального алгорифма. Каждая подстановка есть упорядоченная пара слов в алфавите A и дополнительный атрибут (простая, терминальная). Вводим еще две буквы α и β, не входящие в A. Подстановку вида $P \to Q$ будем записывать в виде слова $P\alpha Q$, а заключительную подстановку $P \to\cdot\, Q$ — в виде слова $P\beta Q$. Систему таких слов назовем ***изображением нормального алгорифма***. Изображение нормального алгорифма $\mathfrak{A}$ будем обозначать $\mathfrak{A}^{\text{и}}$.

Пример 9.6. Для нормального алгорифма со схемой $(00 \to 00,\ 101 \to 0,\ 10 \to\cdot\ 1,\ 01 \to 01)$ изображение выглядит следующим образом:

$$\gamma 00\alpha 00\gamma 101\alpha 0\gamma 10\beta 1\gamma 01\alpha 01\gamma.$$

Введем четвертый спецсимвол δ и будем рассматривать слова вида $\mathfrak{A}^{\text{и}}\delta Q$, где $\mathfrak{A}^{\text{и}}$ — изображение нормального алгорифма $\mathfrak{A}$, а Q — слово в алфавите A. Нормальный алгорифм $\mathfrak{R}$ назовем ***универсальным нормальным алгорифмом***, если для любого нормального алгорифма $\mathfrak{A}$ в алфавите A и любого слова $Q \in A^*$ выполняется соотношение

$$\mathfrak{R}(\mathfrak{A}^{\text{и}}\delta Q) \simeq \mathfrak{A}(Q).$$

В дальнейшем в слове $\mathfrak{A}^{\text{и}}\delta Q$ левую часть $\mathfrak{A}^{\text{и}}$ (изображение нормального алгорифма) иногда будем называть программой, а правую часть Q — данными. Расширенный алфавит $A\alpha\beta\gamma\delta$ будем обозначать $\tilde{A}$.

Поскольку всевозможные пары „программа — данные“ представлены в виде слов в расширенном алфавите, согласно принципу нормализации универсальный алгорифм должен существовать.

Теорема 9.14. Универсальный нормальный алгорифм существует.

◀ Построение универсального алгорифма — сложная задача. Поэтому мы не станем строить конкретный алгорифм, а докажем его существование, опираясь на сочетания более простых алгорифмов.

В самом общем виде искомый алгоритм можно представить как повторение алгоритма $\mathfrak{P}_1$ под управлением алгоритма $\mathfrak{L}_2$ с последующим выполнением алгоритма $\mathfrak{P}_3$ (рис. 9.6). Алгоритм $\mathfrak{P}_1$ выполняет один шаг работы нормального алгорифма, т.е. либо находит подстановку и выполняет ее, либо ничего не делает, если подходящей подстановки нет. Алгоритм $\mathfrak{L}_2$ проверяет условие останова, т.е. была ли выполнена подстановка, и если была, то какая, терминальная или нет. Алгоритм $\mathfrak{P}_3$ убирает лишнее из конечной строки: спецсимволы и код программы.

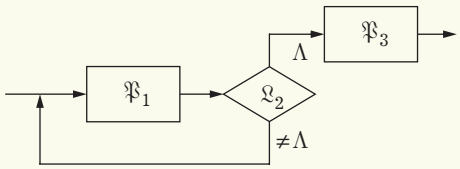

Рис. 9.6

Конкретизируем каждый из трех описанных алгоритмов. В ходе работы будем использовать спецсимволы p, q, u, t для фиксирования результатов работы: символ p будет указывать текущую подстановку в программе, находясь перед левой частью подстановки, символ q будет указывать текущий символ этой подстановки, а символы u и t — выделять фрагмент данных. Если в результате работы алгоритма все подстановки просмотрены и подходящей нет, символы pq будут находиться перед символом δ (за последней подстановкой программы). Если подходящая подстановка найдена и реализована в данных, то символ p будет располагаться перед этой подстановкой, а символ q — перед разделителем α или β в этой подстановке. Алгоритм $\mathfrak{L}_2$ должен проверять эти условия. Схема такого алгоритма может быть такой:

$$\begin{cases} \eta r \to r, & \eta \in Apq\alpha\beta\gamma, \\ r\eta \to r, & \eta \in Apq\alpha\beta\gamma\delta, \\ r \to\cdot\ \Lambda, \\ pq\delta \to r, \\ q\alpha \to\cdot\ q\alpha, \\ q\beta \to r. \end{cases}$$

Алгоритм $\mathfrak{P}_3$ просто удаляет спецсимволы p, q, u, t и все слева от δ (включая δ):

$$\begin{cases} \eta \to \Lambda, & \eta \in pqut, \\ \eta\delta \to \delta, & \eta \in \tilde{A}p, \\ \delta \to\cdot\, \Lambda. \end{cases}$$

Остановимся на алгоритме $\mathfrak{P}_1$. Его схема показана на рис. 9.7. Он начинается с инициализирующего алгоритма $\mathfrak{P}_{11}$, за которым следует повторение алгоритма $\mathfrak{P}_{13}$ под управлением $\mathfrak{L}_{12}$. Алгоритм $\mathfrak{P}_{14}$ завершает работу $\mathfrak{P}_1$. Алгоритм $\mathfrak{P}_{11}$ устанавливает спецсимволы p, q, u, t в начальное положение: символы pq перед первой подстановкой, символы ut в начале данных (вслед за разделителем δ). Алгоритм $\mathfrak{L}_{12}$ проверяет конфигурацию спецсимволов, выявляя два условия: если подходящая подстановка найдена или если все подстановки просмотрены, формируется пустое значение. Все подстановки просмотрены, если символы pq расположены непосредственно перед разделителем δ. Текущая подстановка, отмеченная символом p подходящая, если символ q предшествует разделителю α или β этой подстановки. Алгоритм $\mathfrak{P}_{14}$ реализует подходящую подстановку или не выполняет никаких действий, если подходящая подстановка не найдена.

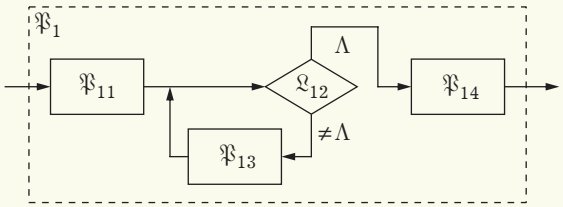

Рис. 9.7

Алгоритм $\mathfrak{P}_{11}$ можно реализовать как композицию $\mathfrak{P}_{113}\circ\mathfrak{P}_{112}\circ\mathfrak{P}_{111}$ трех простых алгоритмов: алгоритм $\mathfrak{P}_{111}$ удаляет спецсимволы (его схема $\eta\Lambda$, $\eta \in pqut$), алгоритм $\mathfrak{P}_{112}$ устанавливает в начальное положение символы pq (его схема $\gamma \to\cdot\, \gamma pq$), а алгоритм $\mathfrak{P}_{113}$ выполняет то же самое для символов ut (его схема $\delta \to\cdot\, \delta ut$).

Алгоритм $\mathfrak{L}_{12}$ проверят два условия: положение символа q в конце левой части подстановки, т.е. перед символом α или β, означает, что эта подстановка применима к данным; положение символов pq перед δ означает, что ни одна подстановка не применима к данным. И то, и другое означает выход из текущего шага работы алгоритма. Алгоритм $\mathfrak{L}_{12}$ можно задать схемой

$$\begin{cases} r\eta \to r, & \eta \in \tilde{A}pq, \\ \eta r \to r, & \eta \in \tilde{A}pq, \\ r \to\cdot\, \Lambda, \\ pq\delta \to r, \\ q\alpha \to r, \\ q\beta \to r. \end{cases}$$

Ключевой составной частью в $\mathfrak{P}_1$ является алгоритм $\mathfrak{P}_{13}$, который должен фактически проверить, является ли текущая подстановка подходящей. Такая проверка сводится к соответствующей расстановке спецсимволов. Если комбинация pq располагается перед δ, то все подстановки просмотрены и подходящей нет. Если символ q предшествует разделителю подстановки α или β, то подходящая подстановка выбрана, при этом символы u и t будут ограничивать вхождение левой части найденной подстановки в строку данных. В остальных случаях символы pq располагаются в начале следующей, еще не просмотренной подстановки. В соответствии со сказанным алгоритм $\mathfrak{P}_{13}$ можно реализовать как разветвление, которое представляет собой выбор одного из нескольких вариантов изменения положения спецсимволов.

Для объяснения сути указанного разветвления придадим следующий условный смысл спецсимволам: p — номер текущей подстановки; q — номер символа в левой части текущей подстановки; u — номер первого символа выделенного фрагмента данных; t — номер последнего символа выделенного фрагмента данных, номера u и t изменяются от 1 до m — длины строки данных. Структура алгоритма $\mathfrak{P}_{13}$ с содержательной интерпретацией структурных элементов показана на рис. 9.8. На рисунке условие $= \Lambda$ равносильно „да“, x_q и y_t обозначают символ подстановки с номером q и символ данных с номером t.

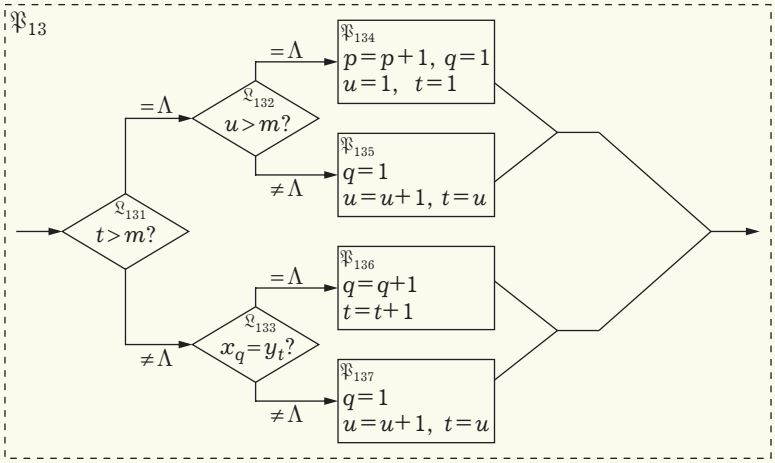

Рис. 9.8

Алгоритмы $\mathfrak{L}_{131}$ и $\mathfrak{L}_{132}$ однотипны и различаются используемым спецсимволом:

$$\mathfrak{L}_{131}\colon \begin{cases} r\eta \to r, & \eta \in \tilde{A}pq, \\ \eta r \to r, & \eta \in \tilde{A}pq, \\ r \to\cdot\ \Lambda, \\ t\xi \to\cdot\ .t\xi, \\ t \to r; \end{cases} \qquad \mathfrak{L}_{132}\colon \begin{cases} r\eta \to r, & \eta \in \tilde{A}pq, \\ \eta r \to r, & \eta \in \tilde{A}pq, \\ r \to\cdot\ \Lambda, \\ u\xi \to\cdot\ u\xi, \\ u \to r; \end{cases}$$

Алгоритм $\mathfrak{L}_{133}$ можно реализовать следующим образом:

$$\begin{cases} r\eta \to r, & \eta \in \tilde{A}pqut, \\ \eta r \to r, & \eta \in \tilde{A}pqut, \\ r \to\cdot\ \Lambda, \\ s\xi\eta \to \eta s\xi, & \xi \in A,\ \ \eta \in \tilde{A}u, \\ s\xi t\xi \to r, & \xi \in A, \\ s\xi t \to\cdot\ s\xi t, & \xi \in A, \\ q \to qs. \end{cases}$$

Алгоритм $\mathfrak{P}_{134}$ можно реализовать как композицию двух алгоритмов: $\mathfrak{P}_{134} = \mathfrak{P}_{1342} \circ \mathfrak{P}_{1341}$. Алгоритм $\mathfrak{P}_{1341}$ удаляет символ q, передвигает символ p на следующую подстановку и заканчивает работу вставкой символа q сразу после p. Алгоритм $\mathfrak{P}_{1342}$ удаляет символы u и t, а затем вставляет после δ:

$$\mathfrak{P}_{1341}\colon \begin{cases} q \to \Lambda, \\ p\eta \to \eta p, & \eta \in A\alpha\beta, \\ p\gamma \to\cdot\ \gamma pq; \end{cases} \qquad \mathfrak{P}_{1342}\colon \begin{cases} u \to \Lambda, \\ t \to \Lambda, \\ \delta \to\cdot\ \delta ut. \end{cases}$$

Алгоритм $\mathfrak{P}_{135}$ также удобно разделить в композицию: $\mathfrak{P}_{135} = \mathfrak{P}_{1352} \circ \mathfrak{P}_{1351}$, где $\mathfrak{P}_{1351}$ отрабатывает ситуацию с символами p и q, а $\mathfrak{P}_{1352}$ — с символами u и t:

$$\mathfrak{P}_{1351}\colon \begin{cases} q \to \Lambda, \\ p \to\cdot\, pq; \end{cases} \qquad \mathfrak{P}_{1352}\colon \begin{cases} t \to \Lambda, \\ u\xi \to\cdot\, \xi ut, \quad \xi \in A. \end{cases}$$

Алгоритм $\mathfrak{P}_{136}$ также разделяем в композицию алгоритма $\mathfrak{P}_{1361}$, передвигающего символ q, и алгоритма $\mathfrak{P}_{1362}$, передвигающего символ t. Каждый реализуется одной групповой подстановкой: первый $q\xi \to\cdot\, \xi q$, второй $t\xi \to\cdot\, \xi t$.

Осталось описать алгоритм $\mathfrak{P}_{14}$. Этот алгоритм можно представить в виде разветвления $S(\mathfrak{L}_{141}, \mathfrak{P}_{142}, \mathfrak{P}_{143})$, в котором выполняется $\mathfrak{P}_{142}$, если $\mathfrak{L}_{141}(X) = \Lambda$, и $\mathfrak{P}_{143}$ в противном случае. Алгоритм $\mathfrak{L}_{141}$ проверяет, следует ли в данных выполнить замену. Его схема:

$$\begin{cases} r\eta \to r, \\ \eta r \to r, \\ r \to\cdot\, \Lambda, \\ pq\delta \to r. \end{cases}$$

Алгоритм $\mathfrak{P}_{142}$ ничего не делает: $\Lambda \to\cdot\, \Lambda$, а алгоритм $\mathfrak{P}_{143}$ выполняет подстановку, заменяя фрагмент данных между u и t правой частью подстановки. Его можно представить как композицию алгоритма $\mathfrak{P}_{1431}$, который стирает фрагмент между символами u и t, и алгоритма $\mathfrak{P}_{1432}$, который копирует правую часть подстановки в место между u и t. Алгоритм $\mathfrak{P}_{1431}$ можно задать схемой

$$\begin{cases} u\xi \to u, \quad \xi \in A, \\ ut \to\cdot\, ut. \end{cases}$$

Алгоритм $\mathfrak{P}_{1432}$ можно представить схемой

$$\begin{cases} s\xi\eta \to \eta s\xi, \quad \xi \in A, \ \ \eta \in \tilde{A}u, \\ s\xi t \to \xi t, \quad \xi \in A, \\ q\gamma \to\cdot\, \gamma, \\ q\xi \to \xi q s\xi, \quad \xi \in A, \\ q\alpha \to q\alpha q, \\ q\beta \to q\beta q. \end{cases}$$

Отметим, что этот алгоритм начинает работу в ситуации, когда символ q находится перед разделителем подстановки α или β. А алгоритме формируется второй символ q, помещаемый в правую часть подстановки. После завершения работы первое вхождение символа q остается перед разделителем, сигнализируя, что алгоритм $\mathfrak{P}_1$ выполнил подстановку, и обозначая тип этой подстановки. ►

Замечание 9.2. Из доказательства теоремы 9.14 ясно, что универсальный алгоритм устроен достаточно сложно, и привести конкретную схему этого алгоритма непросто. Однако такие построения были выполнены разными исследователями.

Приведем пример нормального алгорифма, построенного В.Г. Жаровым*. Этот алгорифм однократно применяем схему нормального алгорифма к исходному слову. Чтобы построить

*Вообще-то этот алгоритм построен для двухбуквенного алфавита. Однако он является рабочим и для алфавита с большим числом букв. Кроме того, он приписывает слева к слову спецсимвол π.

полновесные универсальный алгорифм, достаточно устроить его повторение. В нем использованы символы подстановки ξ и η, причем $\xi \in A$, а η пробегает алфавит, явно указанный. Кроме того, введено обозначение $\tilde{A} = A\alpha\beta\gamma$):

1. $\overline{\xi}\eta \to \eta\overline{\xi}, \quad \eta \in \tilde{A}u\delta;$
2. $\overline{\xi}t\xi \to \xi t;$
3. $\overline{\xi}t \to s;$
4. $\overline{\xi}r \to \xi r;$
5. $\eta s \to s\eta, \quad \eta \in \tilde{A}\delta;$
6. $us \to s\overline{u};$
7. $pqs \to u;$
8. $ps \to pq;$
9. $\overline{p}s\gamma \to \gamma pq;$
10. $qs \to$
11. $s \to pq$
12. $\overline{u}\xi \to s\xi ut$
13. $\overline{u} \to$
14. $v \to s$
15. $\overline{p}\eta \to \eta\overline{p}, \quad \eta \in A\alpha\beta;$
16. $\overline{p}q\eta \to \overline{\eta}\,\overline{q}, \quad \eta \in \alpha\beta;$
17. $pq\gamma \to \gamma pq;$
18. $q\xi \to \xi q\overline{\xi};$
19. $p \to \overline{p};$
20. $u\eta \to u, \quad \eta \in \tilde{A};$
21. $ut \to r;$
22. $u\delta \to \pi;$
23. $\overline{q} \to q;$
24. $r \to s;$
25. $\overline{\beta} \to sv;$
26. $\delta \to s\delta ut.$

Свойства приведенного алгоритма:

1) $\mathfrak{A}^{\text{и}}\delta Q \vDash pq\mathfrak{A}^{\text{и}}s\delta utQ$. Это начало работы алгоритма, состоящее в применении подстановки 26.

2) $pq\gamma X \vDash \gamma pqX$, где $X \in (\tilde{A}u\delta t)^*$. Этот этап реализуется в перемещении s в начало слова (подстановка 5), затем его замене на pq (подстановка 11), и смещении вставленной пары букв на одну позицию вправо (подстановка 17).

3) если $Q \not\subset S$, то $LpqQM\gamma N\delta utR \vDash LQM\gamma pqN\delta S$, где L, $N \in \tilde{A}$, $M \in A\alpha\beta$, Q, $S \in A$. Это основная часть работы алгоритма, соответствующая побуквенному сравнению, изложенному выше. Очередная буква-двойник смещается вправо до символа t с помощью подстановки 1, затем применяется подстановка 2 (сравнение успешное) или 3 (сравнение неуспешное)

4) если $Q \subset S$, то $LpqQ\alpha R\gamma N\delta utS \vDash LqQ\alpha R\gamma N\delta S_R^Q$ и $LpqQ\beta R\gamma N\delta utS \vDash \pi S_R^Q$, где L, $N \in \tilde{A}$, Q, R, $S \in A$.

Работу алгоритма Жарова проиллюстрируем на примере алгоритма со схемой, состоящей из единственной подстановки $ab \to c$, и исходного слова aabc. В представленной последователь-

ности для наглядности выполнены замены α на $>$, β на $\geqslant$, γ на _ и δ на $+$, номер примененной подстановки указан слева:

_ab⩾c_+aabc
(26) _ab⩾c_s+utaabc
(5) _ab⩾cs_+utaabc
(5) _ab⩾sc_+utaabc
(5) _abs⩾c_+utaabc
(5) _asb⩾c_+utaabc
(5) _sab⩾c_+utaabc
(5) s_ab⩾c_+utaabc
(11) pq_ab⩾c_+utaabc
(17) _pqab⩾c_+utaabc
(18) _paqAb⩾c_+utaabc
(1) _paqbA⩾c_+utaabc
(1) _paqb⩾Ac_+utaabc
(1) _paqb⩾cA_+utaabc
(1) _paqb⩾c_A+utaabc
(1) _paqb⩾c_+Autaabc
(1) _paqb⩾c_+uAtaabc
(2) _paqb⩾c_+uatabc
(18) _pabqB⩾c_+uatabc
(1) _pabq⩾Bc_+uatabc
(1) _pabq⩾cB_+uatabc
(1) _pabq⩾c_B+uatabc
(1) _pabq⩾c_+Buatabc
(1) _pabq⩾c_+uBatabc
(1) _pabq⩾c_+uaBtabc
(3) _pabq⩾c_+uasabc
(5) _pabq⩾c_+usaabc
(6) _pabq⩾c_+sUaabc
(5) _pabq⩾c_s+Uaabc
(5) _pabq⩾cs_+Uaabc
(5) _pabq⩾sc_+Uaabc
(5) _pabqs⩾c_+Uaabc
(10) _pab⩾c_+Uaabc
(12) _pab⩾c_+sautabc
(5) _pab⩾c_s+autabc
(5) _pab⩾cs_+autabc
(5) _pab⩾sc_+autabc
(5) _pabs⩾c_+autabc
(5) _pasb⩾c_+autabc
(5) _psab⩾c_+autabc
(8) _pqab⩾c_+autabc
(18) _paqAb⩾c_+autabc
(1) _paqbA⩾c_+autabc
(1) _paqb⩾Ac_+autabc
(1) _paqb⩾cA_+autabc
(1) _paqb⩾c_A+autabc
(1) _paqb⩾c_+Aautabc
(1) _paqb⩾c_+aAutabc
(1) _paqb⩾c_+auAtabc
(2) _paqb⩾c_+auatbc
(18) _pabqB⩾c_+auatbc
(1) _pabq⩾Bc_+auatbc
(1) _pabq⩾cB_+auatbc
(1) _pabq⩾c_B+auatbc
(1) _pabq⩾c_+Bauatbc
(1) _pabq⩾c_+aBuatbc
(1) _pabq⩾c_+auBatbc
(1) _pabq⩾c_+auaBtbc
(2) _pabq⩾c_+auabtc
(19) _Pabq⩾c_+auabtc
(15) _aPbq⩾c_+auabtc
(15) _abPq⩾c_+auabtc
(16) _ab⩾Qc_+auabtc
(20) _ab⩾Qc_+aubtc
(20) _ab⩾Qc_+autc
(21) _ab⩾Qc_+arc
(23) _ab⩾qc_+arc
(18) _ab⩾cqC_+arc
(1) _ab⩾cq_C+arc
(1) _ab⩾cq_+Carc
(1) _ab⩾cq_+aCrc
(4) _ab⩾cq_+acrc
(24) _ab⩾cq_+acsc
(5) _ab⩾cq_+ascc
(5) _ab⩾cq_+sacc
(5) _ab⩾cq_s+acc
(5) _ab⩾cqs_+acc
(10) _ab⩾c_+acc
(25) _absvc_+acc
(5) _asbvc_+acc
(5) _sabvc_+acc
(5) s_abvc_+acc
(11) pq_abvc_+acc
(14) pq_absc_+acc
(5) pq_asbc_+acc
(5) pq_sabc_+acc
(5) pqs_abc_+acc
(7) u_abc_+acc
(20) uabc_+acc
(20) ubc_+acc
(20) uc_+acc
(20) u_+acc
(20) u+acc
(22) wacc
wacc

ОГЛАВЛЕНИЕ