МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования
“Оренбургский государственный университет”

Е.Н. ИШАКОВА

ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ И МЕТОДОВ ТРАНСЛЯЦИИ

Рекомендовано Ученым советом государственного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет» в качестве учебного пособия для студентов, обучающихся по программам высшего профессионального образования по специальности «Программное обеспечение вычислительной техники и автоматизированных систем»

Оренбург 2007

УДК 004.43(075.8)
ББК 32.973.26-018.1я73
И 97

Рецензент
доктор технических наук, профессор Пищухин А.М.

Ишакова Е.Н.
И 97 **Теория языков программирования и методов трансляции: учебное пособие / Е.Н. Ишакова. – Оренбург: ГОУ ОГУ, 2007. – 137 с.**

ISBN

В пособии рассмотрены основы классической теории формальных языков, грамматик и автоматов; принципы и методы разработки трансляторов, а также формальные методы описания перевода. Теоретический материал дополнен примерами и задачами с решениями, вопросами и тестами для проверки усвоения материала.

Учебное пособие предназначено для студентов, обучающихся по программам высшего профессионального образования по специальности 230105.65 «Программное обеспечение вычислительной техники и автоматизированных систем», при изучении дисциплины «Теория языков программирования и методов трансляции».

И 2404010000 ББК 32.973.26-018.1я73

© Ишакова Е.Н., 2007
© ГОУ ОГУ, 2007

ISBN

Содержание

Введение ... 5
1 Формальные языки и грамматики ... 6
1.1 Операции над цепочками символов ... 6
1.2 Формальные грамматики ... 7
1.2.1 Определение формальной грамматики и языка ... 7
1.2.2 Классификация языков и грамматик по Хомскому ... 8
1.3 Формы Бэкуса - Наура ... 10
1.4 Диаграммы Вирта ... 11
1.5 Механизмы распознавания языков ... 12
1.5.1 Схема работы распознавателя ... 12
1.5.2 Классификация распознавателей ... 14
Вопросы и задания для самоконтроля по разделу 1 ... 14
Тесты проверки усвоения материала по разделу 1 ... 15
2 Регулярные грамматики и языки ... 19
2.1 Регулярные выражения ... 19
2.2 Конечные автоматы ... 21
2.2.1 Определение конечного автомата ... 21
2.2.2 Распознавание строк конечным автоматом ... 22
2.2.3 Преобразование конечных автоматов ... 22
2.3 Взаимосвязь способов определения грамматик ... 27
Вопросы и задания для самоконтроля по 2 разделу ... 29
Тесты проверки усвоения материала по 2 разделу ... 29
3 Контекстно-зависимые языки и грамматики ... 35
3.1 Задача разбора цепочек ... 35
3.1.2 Дерево разбора ... 35
3.1.3 Однозначность грамматик ... 36
3.2 Преобразование КС-грамматик ... 38
3.2.1 Проверка существования языка грамматики ... 38
3.2.2 Удаление бесполезных символов грамматики ... 39
3.2.3 Устранение ε-правил грамматики ... 41
3.2.4 Устранение цепных правил ... 42
3.2.5 Устранение левой факторизации правил ... 43
3.2.6 Устранение прямой левой рекурсии ... 44
3.3 Автомат с магазинной памятью ... 44
3.3.1 Общая схема функционирования МП-автомата ... 45
3.3.2 Взаимосвязь МП-автоматов и КС-грамматик ... 46
3.4 Нисходящие распознаватели КС-языков ... 49
3.4.1 Рекурсивный спуск ... 49
3.4.2 Распознаватели LL(k)-грамматик ... 51
3.5 Восходящие распознаватели языков ... 56
3.5.1 Распознаватели LR(k)-грамматик ... 56
3.5.2 Грамматики простого предшествования ... 59

3.6 Соотношение классов КС-грамматик и КС-языков 65
Вопросы и задания для самоконтроля по 3 разделу 66
Тесты контроля усвоения материала по 3 разделу 67
4 Принципы построения трансляторов 73
4.1 Определение и виды трансляторов 73
4.2 Общая схема работы транслятора 73
4.3 Лексический анализ 75
4.4 Синтаксический анализ 80
4.5 Семантический анализ 81
4.6 Формы внутреннего представления программы 82
4.6.1 Тетрады и триады 83
4.6.2 Синтаксические деревья 84
4.6.3 Польская инверсная запись 85
4.6.4 Ассемблерный код и машинные команды 89
4.7 Генерация кода 89
4.8 Оптимизация кода 91
4.8.1 Сущность оптимизации кода 91
4.8.2 Методы оптимизации кода 92
4.8.3 Оптимизация линейных участков программ 93
4.8.4 Оптимизация логических выражений 97
4.8.5 Оптимизация циклов 97
Вопросы и задания для самоконтроля по 4 разделу 99
Тесты контроля усвоения материала по 4 разделу 100
5 Формальные методы описания перевода 104
5.1 Синтаксически управляемый перевод 104
5.1.1 Схемы компиляции 104
5.1.2 СУ-схемы 106
5.1.3 МП-преобразователи 107
5.1.4 Практическое применение СУ-схем 109
5.2 Транслирующие грамматики 111
5.3 Атрибутные транслирующие грамматики 112
5.3.1 Синтезируемые и наследуемые атрибуты 112
5.3.2 Определение и свойства АТ-грамматики 114
Вопросы и задачи для самоконтроля по 5 разделу 117
Тесты проверки усвоения материала по 5 разделу 122
6 Итоговые тесты проверки усвоения материала 131
Список использованных источников 136

Введение

Предлагаемый материал посвящен основам классической теории формальных языков, грамматик и компиляторов – одной из важнейших составных частей инженерного образования по информатике и вычислительной технике.

Теория формальных языков, грамматик и автоматов составляет фундамент синтаксических методов. Основы этой теории были заложены Н. Хомским в 40–50-е годы XX столетия в связи с его лингвистическими работами, посвященными изучению естественных языков. Но уже в следующем десятилетии синтаксические методы нашли широкое практическое применение в области разработки и реализации языков программирования.

В настоящее время искусственные языки, использующие для описания предметной области текстовое представление, широко применяются не только в программировании, но и в других областях. С их помощью описывается структура всевозможных документов, трехмерных виртуальных миров, графических интерфейсов пользователя и многих других объектов, используемых в моделях и в реальном мире. Для того чтобы эти текстовые описания были корректно составлены, а затем правильно распознаны и интерпретированы, применяются специальные методы их анализа и преобразования. В основе данных методов лежит теория формальных языков, грамматик и автоматов.

Теория формальных языков, грамматик и автоматов дала новый стимул развитию математической лингвистики и методам искусственного интеллекта, связанных с естественными и искусственными языками. Кроме того, ее элементы успешно применяются, например, при описании структур данных, файлов, изображений, представленных не в текстовом, а двоичном формате. Эти методы полезны при разработке своих трансляторов даже там, где уже имеются соответствующие аналоги.

Несмотря на то, что к настоящему времени разработаны тысячи различных языков и их компиляторов, процесс создания новых приложений в этой области не прекращается. Это связно как с развитием технологии производства вычислительных систем, так и с необходимостью решения все более сложных прикладных задач. Такая разработка может быть обусловлена различными причинами, в частности, функциональными ограничениями, отсутствием локализации, низкой эффективностью существующих компиляторов. Поэтому основы теории языков и формальных грамматик, а также практические методы разработки компиляторов лежат в фундаменте инженерного образования по информатике и вычислительной технике.

В пособии отражены все разделы обязательного минимума содержания основной образовательной программы по дисциплине: основы теории формальных языков и грамматик; распознаватели и преобразователи (конечные автоматы и преобразователи, автоматы и преобразователи с магазинной памятью); связь между грамматиками и автоматами; формальные методы описания перевода (СУ-схемы, Т-грамматики, атрибутные Т-грамматики); алгоритмы синтаксического анализа для $LL(k)$-грамматик, $LR(k)$-грамматик, грамматик предшествования; включение семантики в алгоритмы синтаксического анализа.

1 Формальные языки и грамматики

1.1 Операции над цепочками символов

В основе каждого языка лежит алфавит.

Определение 1.1. Алфавитом V называется конечное множество символов.

Определение 1.2. Цепочкой α в алфавите V называется любая конечная последовательность символов этого алфавита.

Определение 1.3. Цепочка, которая не содержит ни одного символа, называется пустой цепочкой и обозначается ε.

Определение 1.4. Формальное определение цепочки символов в алфавите V:

1) ε - цепочка в алфавите V;

2) если α - цепочка в алфавите V и a – символ этого алфавита, то αa – цепочка в алфавите V;

3) β - цепочка в алфавите V тогда и только тогда, когда она является таковой в силу утверждений 1) и 2).

Определение 1.5. Длиной цепочки α называется число составляющих ее символов (обозначается $|\alpha|$).

Определение 1.6. Конкатенацией (сцеплением) цепочек α и β называется цепочка $\gamma=\alpha\beta$, в которой символы данных цепочек записаны друг за другом.

Для любой цепочки α справедливо утверждение $\alpha\varepsilon=\varepsilon\alpha$.

Определение 1.7. Степенью n цепочки α называется конкатенация n цепочек α (обозначается: α^n).

Для любой цепочки α справедливы утверждения $\alpha^0=\varepsilon$ и $\alpha^n=\alpha^{n-1}\alpha=\alpha\alpha^{n-1}$ для $n\geq 1$.

Определение 1.8. Реверсом (обращением) цепочки α называется цепочка α^R, составленная из символов цепочки α, записанных в обратном порядке.

Пример 1.1. Пусть алфавит $V=\{a, b, c, d\}$, тогда для цепочек этого алфавита $\alpha=ab$ и $\beta=bcd$ будет справедливо $|\alpha|=2$, $|\beta|=3$, $\alpha\beta= abbcd$, $\alpha^2=abab$, $\beta^R=dcb$.

Обозначим через V^* множество, содержащее все цепочки в алфавите V, включая пустую цепочку ε, а через V^+ - множество, содержащее все цепочки в алфавите V, исключая пустую цепочку ε.

Формальным языком L в алфавите V называют некоторое подмножество множества V^*.

Пример 1.2. Пусть задан алфавит двоичных цифр $V=\{1, 0\}$, тогда $V^*=\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$, а $V^+=\{0, 1, 00, 01, 10, 11, 000, \ldots\}$.

Задать язык L в алфавите V можно тремя способами:

1) перечислением всех допустимых цепочек языка (на языке множеств);

2) указанием способа порождения (генерации) цепочек языка (грамматики, формы Бэкуса-Наура и диаграммы Вирта);

3) определением метода распознавания цепочек языка (распознаватели).

Пример 1.3. Язык L в алфавите $V = \{1, 0\}$, состоящий из пустой строки и всевозможных строк, каждая из которых содержит строку нулей и последующую строку единиц той же длины, можно описать с помощью формальной системы определения множеств как $L=\{0^n1^n \mid n\geq 0\}$.

1.2 Формальные грамматики

1.2.1 Определение формальной грамматики и языка

Формальная грамматика – это математическая система, определяющая язык посредством порождающих правил.

Определение 1.9. Формальной грамматикой называется четверка вида:

$$G = (V_T, V_N, P, S), \tag{1.1}$$

где V_N - конечное множество нетерминальных символов грамматики (обычно прописные латинские буквы);

V_T - множество терминальных символов грамматики (обычно строчные латинские буквы, цифры и т.п.), $V_T \cap V_N = \varnothing$;

P - множество правил вывода грамматики, являющееся конечным подмножеством множества $(V_T \cup V_N)^+ \times (V_T \cup V_N)^*$; элемент (α, β) множества P называется правилом вывода и записывается в виде $\alpha \to \beta$ (читается: «из цепочки α выводится цепочка β»);

S - начальный символ грамматики, $S \in V_N$.

Для записи правил вывода с одинаковыми левыми частями вида $\alpha \to \beta_1, \alpha \to \beta_2, \ldots, \alpha \to \beta_n$ используется сокращенная форма записи $\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$.

Пример 1.4. Грамматика G_1=({0, 1}, {A, S}, P_1, S), где множество P состоит из правил вида: 1) $S \to 0A1$; 2) $0A \to 00A1$; 3) $A \to \varepsilon$.

Определение 1.10. Цепочка $\beta \in (V_T \cup V_N)^*$ непосредственно выводима из цепочки $\alpha \in (V_T \cup V_N)^+$ в грамматике $G = (V_T, V_N, P, S)$ (обозначается: $\alpha \Rightarrow \beta$), если $\alpha = \xi_1\gamma\xi_2$ и $\beta = \xi_1\delta\xi_2$, где $\xi_1, \xi_2, \delta \in (V_T \cup V_N)^*$, $\gamma \in (V_T \cup V_N)^+$ и правило вывода $\gamma \to \delta$ содержится во множестве P.

Определение 1.11. Цепочка $\beta \in (V_T \cup V_N)^*$ выводима из цепочки $\alpha \in (V_T \cup V_N)^+$ в грамматике $G = (V_T, V_N, P, S)$ (обозначается $\alpha \Rightarrow^* \beta$), если существует последовательность цепочек $\gamma_0, \gamma_1, \ldots, \gamma_n$ ($n \geq 0$) такая, что $\alpha = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \ldots \Rightarrow \gamma_n = \beta$.

Пример 1.5. В грамматике G_1 $S \Rightarrow^* 000111$, т.к. существует вывод $S \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000111$.

Определение 1.12. Языком, порожденным грамматикой $G = (V_T, V_N, P, S)$, называется множество всех цепочек в алфавите V_T, которые выводимы из начального символа грамматики S с помощью правил множества P, т.е. множество $L(G) = \{\alpha \in V_T^* \mid S \Rightarrow^* \alpha\}$.

Пример 1.6. Для грамматики G_1 язык $L(G_1)=\{0^n1^n \mid n>0\}$.

Определение 1.13. Цепочка $\alpha \in (V_T \cup V_N)^*$, для которой существует вывод $S \Rightarrow^* \alpha$, называется сентенциальной формой или сентенцией в грамматике $G = (V_T, V_N, P, S)$.

Определение 1.14. Языком, порожденным грамматикой G называется множество терминальных сентенциальных форм грамматики.

1.2.2 Классификация языков и грамматик по Хомскому

Классификация грамматик в иерархии американского математика Хомского осуществляется по структуре правил вывода. В расширенной иерархии Хомского выделяется четыре типа грамматик.

Тип 0. Грамматика $G = (V_T, V_N, P, S)$ называется грамматикой типа 0 (грамматикой без ограничений или грамматикой с фразовой структурой), если на ее правила вывода не наложено никаких ограничений, кроме тех, которые указаны в определении грамматики.

Тип 1. Грамматика $G = (V_T, V_N, P, S)$ называется контекстно-зависимой грамматикой (КЗ-грамматикой), если каждое правило вывода из множества P имеет вид $\alpha \to \beta$, где $\alpha \in (V_T \cup V_N)^+$, $\beta \in (V_T \cup V_N)^*$ и $|\alpha| \le |\beta|$. Расширение допускает не более одного ε-правила, т.е. правила вида $A \to \varepsilon$, $A \in V_N$.

Тип 2. Грамматика $G = (V_T, V_N, P, S)$ называется контекстно-свободной грамматикой (КС-грамматикой), если ее правила вывода имеют вид: $A \to \beta$, где $A \in V_N$ и $\beta \in V^*$.

Тип 3. Грамматика $G = (V_T, V_N, P, S)$ называется регулярной грамматикой (Р-грамматикой) выровненной вправо, если ее правила вывода имеют вид $A \to aB \mid a$, где $a \in V_T$; $A, B \in V_N$.

Грамматика $G = (V_T, V_N, P, S)$ называется регулярной грамматикой (Р-грамматикой) выровненной влево, если ее правила вывода имеют вид $A \to Ba \mid a$, где $a \in V_T$; $A, B \in V_N$.

Расширение допускает единственное ε-правило вида $S \to \varepsilon$, но в этом случае начальный символ грамматики S не должен встречаться в правых частях правил.

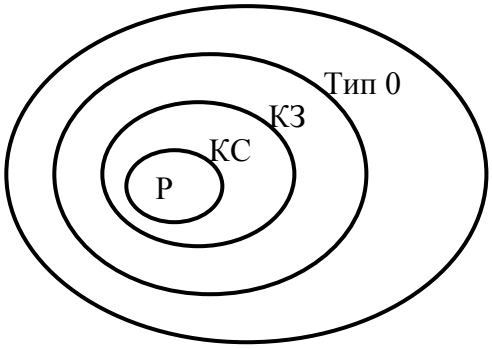
Определение 1.15. Язык $L(G)$ называется языком типа k, если его можно описать грамматикой типа k, где k – максимально возможный номер типа грамматики.

Пример 1.7. Язык вида $L=\{0^n1^n \mid n>0\}$ порождается КЗ-грамматикой (тип 1) G_1 (пример 1.4) и КС-грамматикой (тип 2) G_2= ({0, 1}, {S}, P_2, S), где

множество правил вывода P_2 содержит правила вида $S \to 0S1 \mid 01$.

Так как не существует регулярной грамматики (тип 3), порождающей данный язык, то язык L является языком типа 2 или КС-языком.

Соотношение типов грамматик и языков представлено на рисунке 1.1.



Р – регулярная грамматика;
КС – контекстно-свободная грамматика;
КЗ – контекстно-зависимая грамматика;
Тип 0 – грамматика типа 0.

Рисунок 1.1 – Соотношение типов формальных языков и грамматик

Пример 1.8. Примеры различных типов формальных языков и грамматик по классификации Хомского. Терминалы будем обозначать строчными символами, нетерминалы – прописными буквами, начальный символ грамматики – S.

а) Язык типа 0 $L(G)=\{a^2b^{n^2-1} \mid n \geq 1\}$ определяется грамматикой с правилами вывода:

1) $S \to aaCFD$; 2) $AD \to D$;
3) $F \to AFB \mid AB$; 4) $Cb \to bC$;
5) $AB \to bBA$; 6) $CB \to C$;
7) $Ab \to bA$; 8) $bCD \to \varepsilon$.

б) Контекстно-зависимый язык $L(G)=\{a^nb^nc^n \mid n \geq 1\}$ определяется грамматикой с правилами вывода:

1) $S \to aSBC \mid aBC$; 2) $CB \to BC$;
3) $aB \to ab$; 4) $bB \to bb$;
5) $bC \to bc$; 6) $cC \to cc$.

в) Контекстно-свободный язык $L(G)=\{(ac)^n(cb)^n \mid n>0\}$ определяется грамматикой с правилами вывода:

1) $S \to aQb \mid accb$;
2) $Q \to cSc$.

г) Регулярный язык $L(G)=\{\omega\perp \mid \omega \in \{a, b\}^+$, где нет двух рядом стоящих a$\}$ определяется грамматикой с правилами вывода:

1) $S \to A\perp \mid B\perp$;
2) $A \to a \mid Ba$;
3) $B \to b \mid Bb \mid Ab$.

Определение 1.16. Грамматики G_1 и G_2 называются эквивалентными, если они определяют один и тот же язык, т.е. $L(G_1) = L(G_2)$.

Определение 1.17. Грамматики G_1 и G_2 называются почти эквивалентными, если заданные ими языки различаются не более чем на пустую цепочку символов, т.е. $L(G_1) \cup \{\varepsilon\} = L(G_2) \cup \{\varepsilon\}$.

Пример 1.9. Для грамматики G_1 эквивалентной будет грамматика $G_2 = (\{0, 1\}, \{S\}, P_3, S)$, где P_2: $S \to 0S1 \mid 01$, т.к. $L(G_1)=L(G_2)= \{0^n1^n \mid n>0\}$ (пример 1.7).

Почти эквивалентной для грамматики G_1 будет грамматика $G_3 = (\{0, 1\}, \{S\}, P_3, S)$, где множество правил вывода P_3 содержит правила вида $S \to 0S1 \mid \varepsilon$, т.к. $L(G_3)= \{0^n1^n \mid n\geq 0\}$.

1.3 Формы Бэкуса - Наура

Цепочки языка могут содержать метасимволы, имеющие особое назначение. Метаязык, предложенный Бэкусом и Науром (БНФ) использует следующие обозначения:

- символ «::=» отделяет левую часть правила от правой (читается: «определяется как»);
- нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки «<» и «>»;
- терминалы - это символы, используемые в описываемом языке;
- правило может определять порождение нескольких альтернативных цепочек, отделяемых друг от друга символом вертикальной черты «|» (читается: «или»).

Для повышения удобства и компактности описаний, в расширенных БНФ вводятся следующие дополнительные конструкции (метасимволы):

- квадратные скобки «[» и «]» означают, что заключенная в них синтаксическая конструкция может отсутствовать;
- фигурные скобки «{» и «}» означают повторение заключенной в них синтаксической конструкции ноль или более раз;
- сочетание фигурных скобок и косой черты «{/» и «/}» используется для обозначения повторения один и более раз;
- круглые скобки «(» и «)» используются для ограничения альтернативных конструкций;
- кавычки используются в тех случаях, когда один из метасимволов нужно включить в цепочку обычным образом.

Пример 1.10. Правила, определяющие понятие «идентификатор» языка программирования, могут выглядеть следующим образом:

<буква> ::= *a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* | *j* | *k* | *l* | *m* | *n* | *o* | *p* | *q* | *r* | *s* | *t* | *u* | *v* | *w* | *x* | *y* | *z*

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

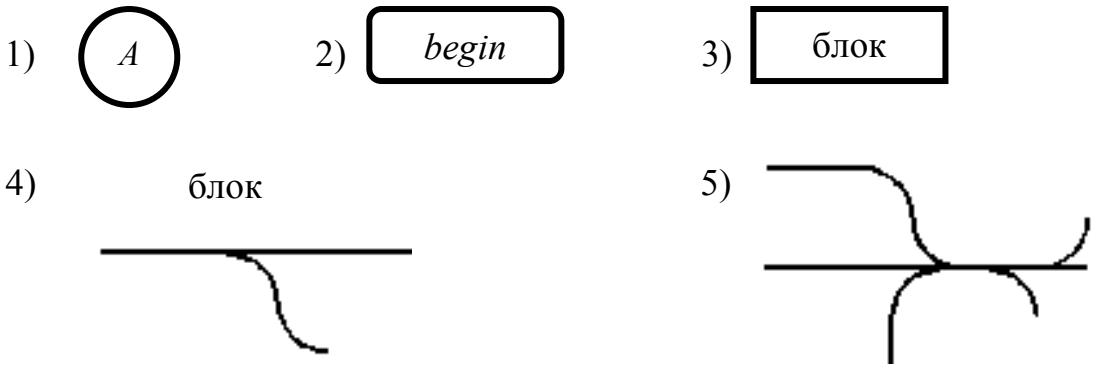
<идентификатор> ::= <буква> {<буква> | <цифра>}

1.4 Диаграммы Вирта

В метаязыке диаграмм Вирта используются графические примитивы, представленные на рисунке 1.2.

При построении диаграмм учитывают следующие правила:

- каждый графический элемент, соответствующий терминалу или нетерминалу, имеет по одному входу и выходу, которые обычно изображаются на противоположных сторонах;
- каждому правилу соответствует своя графическая диаграмма, на которой терминалы и нетерминалы соединяются посредством дуг;
- альтернативы в правилах задаются ветвлением дуг, а итерации - их слиянием;
- должна быть одна входная дуга (располагается обычно слева или сверху), задающая начало правила и помеченная именем определяемого нетерминала, и одна выходная, задающая его конец (обычно располагается справа или снизу);
- стрелки на дугах диаграмм обычно не ставятся, а направления связей отслеживаются движением от начальной дуги в соответствии с плавными изгибами промежуточных дуг и ветвлений.



1) – терминальный символ, принадлежащий алфавиту языка;
2) – постоянная группа терминальных символов, определяющая название лексемы;
3) – нетерминальный символ, определяющий название правила;
4) – входная дуга с именем правила, определяющая его название;
5) – соединительные линии, обеспечивающие связь между терминальными и нетерминальными символами в правилах.

Рисунок 1.2 – Графические примитивы диаграмм Вирта

Пример 1.11. Возможные правила, определяющие понятие «идентификатор» языка программирования, показаны на рисунке 1.3.

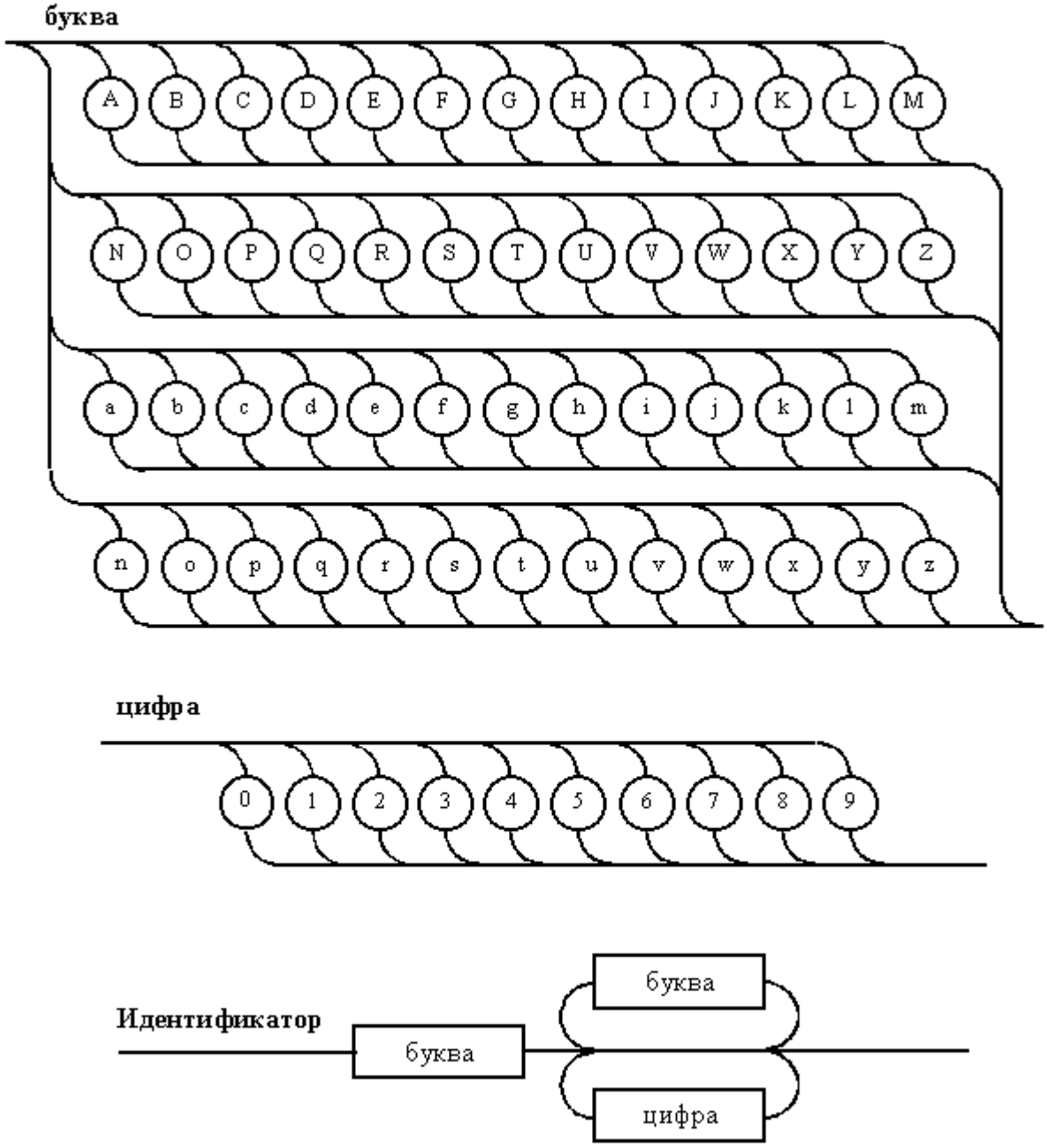


Рисунок 1.3 – Диаграмма Вирта понятия «идентификатор»

1.5 Механизмы распознавания языков

1.5.1 Схема работы распознавателя

Обратной к задаче генерации строк языка является задача проверки принадлежности исходной строки заданному языку. Для решения этой проблемы создается механизм распознавания или распознаватель.

Определение 1.18. Распознаватель – это специальный алгоритм, который позволяет определить принадлежность цепочки символов некоторому языку.

Распознаватель схематично можно представить в виде совокупности входной ленты, управляющего устройства и вспомогательной памяти (рисунок 1.4).

Входная лента представляет собой последовательность ячеек, каждая из которых содержит один символ некоторого конечного алфавита.

Входная головка в каждый момент обозревает одну ячейку. За один шаг работы распознавателя головка сдвигается на одну ячейку влево (вправо) или остается неподвижной. Распознаватель, не перемещающий входную головку влево, называется односторонним.

Управляющее устройство – это программа управления распознавателем. Она задает конечное множество состояний распознавателя и определяет переходы из состояния в состояние в зависимости от прочитанного символа входной ленты и содержимого вспомогательной памяти.

Вспомогательная память служит для хранения информации, которая зависит от состояния распознавателя. У некоторых типов распознавателей она может отсутствовать.

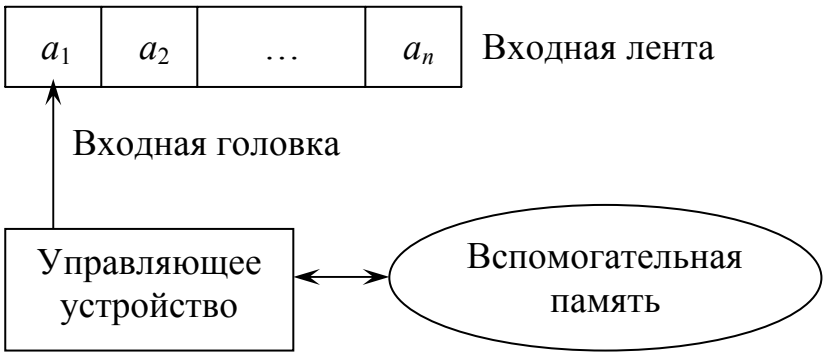


Рисунок 1.4 – Схема распознавателя

Поведение распознавателя можно представить с помощью его конфигураций.

Определение 1.19. Конфигурация распознавателя есть совокупность трех элементов:

- состояния управляющего устройства;
- содержимого входной ленты и положения входной головки;
- содержимого вспомогательной памяти.

Определение 1.20. Управляющее устройство называется детерминированным, если для каждой конфигурации распознавателя существует не более одного возможного следующего шага, в противном случае управляющее устройство называется недетерминированным.

Определение 1.21. Конфигурация распознавателя называется начальной, если управляющее устройство находится в заданном начальном состоянии, входная головка обозревает самый левый символ на входной ленте и вспомогательная память имеет заранее установленное начальное содержимое.

Определение 1.22. Конфигурация распознавателя называется заключительной, если управляющее устройство находится в одном из заранее выделен-

ных заключительных состояний, а входная головка сошла с правого конца входной ленты. Содержимое вспомогательной памяти удовлетворяет некоторому заранее установленному условию.

Определение 1.23. Входная строка ω допускается распознавателем, если от начальной конфигурации, в которой цепочка ω записана на входной ленте, распознаватель может выполнить последовательность шагов, заканчивающуюся заключительной конфигурацией.

Определение 1.24. Множество всех строк, допускаемых распознавателем, называется языком распознавателя.

1.5.2 Классификация распознавателей

Для каждого класса грамматик из иерархии Хомского существует класс распознавателей, определяющих тот же класс языков. Чем шире класс грамматик, тем сложнее класс соответствующих распознавателей.

Для языков типа 0 распознавателями являются машины Тьюринга.

Распознаватели КЗ-языков называются линейными ограниченными автоматами (машины Тьюринга с конечным объемом ленты).

Распознавателями для КС-языков являются автоматы с магазинной памятью (МП-автоматы).

Для регулярных языков распознавателями служат конечные автоматы (КА).

Вопросы и задания для самоконтроля по 1 разделу

1 Дайте формальное определение цепочки в алфавите V.

2 Перечислите основные операции над цепочками символов.

3 Назовите способы формального задания языка.

4 Какими элементами задается формальная грамматика?

5 Как формально определяется выводимость цепочек в грамматике?

6 Назовите типы формальных грамматик по классификации Хомского. Как соотносятся эти типы между собой?

7 Какие грамматики называются эквивалентными; почти эквивалентными?

8 Перечислите основные метасимволы форм Бэкуса-Наура.

9 Какие графические примитивы используются в метаязыке диаграмм Вирта?

10 Изобразите схематично распознаватель языков и поясните принцип его функционирования.

11 Какие классы распознавателей языков выделяют?

12 Определите класс грамматики и порождаемого языка по классификации Хомского:

а) $S \to AB$; $AB \to aABB$; $B \to b$; $A \to a$;

б) $S \to aAbB$; $AbB \to aABbB$; $bBb \to bb$; $A \to \varepsilon$;

в) $S \to aA \mid bB$; $A \to aA \mid a \mid b$; $B \to bB \mid b \mid \varepsilon$.

13 Задайте с помощью грамматик, БНФ и диаграмм Вирта языки, состоящие из:

а) идентификаторов произвольной длины, начинающихся с буквы;

б) вещественных констант;

в) всех цепочек из нулей и единиц, имеющих чётное число нулей и единиц либо нечётное число нулей и единиц.

14 Построить грамматику, определяющую следующий язык:

а) $\{a^n p^n r^n \mid n \geq 1\}$ (Три мушкетёра);

б) $\{\omega\omega \mid \omega \in \{a, b\}^*\}$ (Два лебедя);

в) $\{a^m b^n a^m b^n \mid m, n \geq 1\}$(Две калоши).

15 Эквивалентны ли грамматики:

а) $S \to aAb$; $A \to BB$; $B \to ab \mid A \mid \varepsilon$;

б) $S \to aAb$; $A \to AaAb \mid \varepsilon$;

в) $S \to aB$; $B \to aBB \mid b$.

Тесты проверки усвоения материала по 1 разделу

1 Даны цепочки $\alpha=ab$ и $\beta=bcd$. Порядок следования составных цепочек по возрастанию их длины:

а) α^2; β^R; α^R; $\alpha\beta$; β^2;

б) β^R; α^R; α^2; β^2; $\alpha\beta$;

в) $\alpha\beta$; α^R; α^2; β^R; β^2;

г) α^R; β^R; α^2; $\alpha\beta$; β^2;

д) $\alpha\beta$; α^R; β^R; β^2; α^2.

2 Любая конечная последовательность символов алфавита:

а) цепочка;

б) конкатенация;

в) реверс;

г) язык;

д) грамматика.

3 Язык, состоящий из пустой строки и всевозможных строк, содержащих строку нулей и последующую строку единиц той же длины:

а) $L=\{(01)^n \mid n>0\}$;

б) $L=\{(01)^n \mid n\geq 0\}$;

в) $L=\{0^n 1^n \mid n\geq 0\}$;

г) $L=\{0^n 1^n \mid n>0\}$;

д) $L=\{0^n 1^m \mid n, m\geq 0\}$.

4 Строка вида *abababab* принадлежит языку:

а) $L=\{ab^n \mid n>0\}$;

б) $L=\{(ab)^n \mid n\geq 0\}$;

в) $L=\{a^n b^n \mid n\geq 0\}$;

г) $L=\{a^n b^n \mid n>0\}$;
д) $L=\{ab^n \mid n\geq 0\}$.

5 Порядок следования понятий от общего к частному:
а) сентенция в грамматике; цепочка алфавита; строка языка грамматики;
б) строка языка грамматики; цепочка алфавита; сентенция в грамматике;
в) цепочка алфавита; строка языка грамматики; сентенция в грамматике;
г) сентенция в грамматике; строка языка грамматики; цепочка алфавита;
д) цепочка алфавита; сентенция в грамматике; строка языка грамматики.

6 Для грамматики с правилами вывода вида $S \to aSb|ab$ эквивалентной будет грамматика:
а) $S\to AB$; $A\to a|Aa$; $B\to b|Bb$;
б) $S\to Sab \mid ab$;
в) $S \to SA|ab$; $A\to ab|\varepsilon$;
г) $S\to aAb$; $aA\to aaAb$; $A\to\varepsilon$;
д) $S\to AS|SB|AB$; $A\to a$; $B\to b$.

7 Почти эквивалентной для грамматики с правилами вида $S\to AB$; $A\to a|Aa$; $B\to b|Bb$ является грамматика:
а) $S \to SA|ab|\varepsilon$; $A\to ab|\varepsilon$;
б) $S \to SA|ab$; $A\to ab|\varepsilon$;
в) $S \to SA|ab|\varepsilon$; $A\to ab$;
г) $S\to AS|SB|AB$; $A\to a$; $B\to b$;
д) $S\to AS|SB|AB|\varepsilon$; $A\to a$; $B\to b$.

8 Сентенция языка грамматики $S \to aQb \mid accb$; $Q \to cSc$:
а) $aScb$;
б) $acaccbcbb$;
в) $aaccab$;
г) $acaQbcb$;
д) $aaaQbcb$.

9 Для грамматики с правилами $S\to Sab \mid ab$ справедливо:
а) $S\to ababab$;
б) $S\Rightarrow ababab$;
в) $S\Rightarrow^* ababab$;
г) $S\Rightarrow^* aaabbb$;
д) $S\Rightarrow aaabbb$.

10 Символы вспомогательного алфавита грамматики, обозначающие допустимые комбинации символов языка в строках:
а) аксиомы;
б) терминалы;

в) нетерминалы;
г) сентенции;
д) правила вывода.

11 Множество правил вывода грамматики $G = (V_T, V_N, P, S)$ является конечным подмножеством множества:
а) $(V_T \cup V_N) \times (V_T \cup V_N)$;
б) $(V_T \cup V_N)^+ \times (V_T \cup V_N)^*$;
в) $(V_T \cup V_N)^* \times (V_T \cup V_N)^+$;
г) $V_N{}^* \times V_T{}^+$;
д) $V_T{}^+ \times V_N{}^*$.

12 Грамматика с правилами вывода вида $S \to aAb$; $aA \to aaAb$; $A \to \varepsilon$ по иерархии Хомского является:
а) грамматикой без ограничений;
б) КС-грамматикой;
в) регулярной грамматикой, выровненной право;
г) КЗ-грамматикой;
д) регулярной грамматикой, выровненной влево.

13 Порядок следования типов грамматик от частного к общему:
а) фразовые, регулярные, контекстно-свободные, контекстно-зависимые;
б) контекстно-свободные, контекстно-зависимые, регулярные, фразовые;
в) регулярные, фразовые, контекстно-свободные, контекстно-зависимые;
г) контекстно-зависимые, контекстно-свободные, регулярные, фразовые;
д) регулярные, контекстно-свободные, контекстно-зависимые, фразовые.

14 Грамматика, правила вывода которой имеют вид $A \to aB \mid a$, где $a \in V_T$; $A, B \in V_N$, является:
а) фразовой;
б) КС-грамматикой;
в) регулярной грамматикой, выровненной право;
г) КЗ-грамматикой;
д) регулярной грамматикой, выровненной влево.

15 Правила вывода КЗ-грамматики имеют вид:
а) $\alpha \to \beta$, где $\alpha \in (V_T \cup V_N)^*$, $\beta \in (V_T \cup V_N)^+$ и $|\alpha| \le |\beta|$;
б) $\alpha \to \beta$, где $\alpha \in (V_T \cup V_N)^+$, $\beta \in (V_T \cup V_N)^*$ и $|\alpha| \le |\beta|$;
в) $\alpha \to \beta$, где $\alpha \in (V_T \cup V_N)^+$, $\beta \in (V_T \cup V_N)^*$ и $|\alpha| < |\beta|$;
г) $\alpha \to \beta$, где $\alpha \in (V_T \cup V_N)^*$, $\beta \in (V_T \cup V_N)^+$ и $|\alpha| \ge |\beta|$;
д) $\alpha \to \beta$, где $\alpha \in V_N{}^+$, $\beta \in V_T{}^*$ и $|\alpha| \le |\beta|$.

16 Порядок следования значений конструкций БНФ от общему к частному:
а) {0 | 1}; {/0 | 1/}; (0 | 1);
б) {/0 | 1/}; {0 | 1}; (0 | 1);

в) (0 | 1); {0 | 1}; {/0 | 1/};
г) {0 | 1}; (0 | 1); {/0 | 1/};
д) {/0 | 1/}; (0 | 1); {0 | 1}.

17 Синтаксическая конструкция, которая может отсутствовать, в БНФ заключается в:
а) ( );
б) { };
в) « »;
г) < >;
д) [ ].

18 Ограничители вида {/ /} задают в расширенных БНФ:
а) итерацию ноль и более раз;
б) итерацию один и более раз;
в) ограничение нетерминала;
г) операцию альтернатива;
д) ограничение терминала.

19 Множеству строк, задаваемых конструкцией БНФ вида {/01/} принадлежит строка:
а) ε;
б) 000111;
в) 010101;
г) 001100;
д) 001110.

20 В БНФ в угловые скобки < > заключаются:
а) терминалы;
б) нетерминалы;
в) альтернативы;
г) итерации;
д) необязательные конструкции.

21 Нетерминальным символам на диаграммах Вирта соответствуют:
а) кружки;
б) скругленные прямоугольники;
в) ветвления дуг;
г) прямоугольники;
д) слияния дуг.

22 Ветвления дуг на диаграммах Вирта задают:
а) нетерминальные символы;
б) терминальные символы;
в) постоянную группу терминальных символов;

г) итерации;
д) альтернативы.

23 Если для каждой конфигурации распознавателя существует не более одного возможного следующего шага, то управляющее устройство называется:
а) односторонним;
б) ограниченным;
в) детерминированным;
г) недетерминированным;
д) неограниченным.

24 Распознавателями для КС-языков являются:
а) МП-автоматы;
б) конечные автоматы;
в) машины Тьюринга;
г) линейные ограниченные автоматы;
д) линейные неограниченные автоматы.

25 Конечные автоматы являются распознавателями:
а) фразовых языков;
б) регулярных языков;
в) КС-языков;
г) КЗ-языков;
д) естественных языков.

2 Регулярные грамматики и языки

2.1 Регулярные выражения

Регулярный язык L в некотором алфавите Σ представляет собой регулярное множество строк.

Определение 2.1. Регулярное множество есть $\varnothing$, либо $\{\varepsilon\}$, либо $\{a\}$ для некоторого $a \in \Sigma$, либо множество, которое можно получить из указанных множеств путем применения к ним конечного числа операций сцепления, объединения и итерации.

В основе метода определения регулярности заданного языка лежит лемма о разрастании языка.

Лемма о разрастании языка. В достаточно длинной строке регулярного языка всегда можно найти непустую подстроку, повторение которой произвольное количество раз порождает новые строки того же самого языка.

Пример 2.1. Язык $L_1 = \{0^m1^n \mid m, n \geq 0\}$ – регулярный, т.к., например, в строке 00111 повторение любой подстроки, образованной только из нулей или единиц, порождает строки (0000111, 000111, 001111, 00111111 и т.д.) языка L_1.

Язык $L_2 = \{0^n1^n \mid n \geq 1\}$ – не регулярный, т.к. любая итерация подстроки, состоящей только из нулей или единиц, нарушает баланс нулей и единиц. Подобные действия со смешанными подстроками, содержащими нули и единицы, приводят к нарушению порядка следования нулей и единиц. Таким образом, для языка L_2 нет строк, удовлетворяющих условиям леммы.

Удобным средством формального определения регулярных языков являются регулярные выражения.

Определение 2.2. Регулярные выражения над алфавитом Σ определяются следующим образом:

1) $\varnothing$ - регулярное выражение (обозначает пустоте регулярное множество $\varnothing$);

2) ε - регулярное выражение (обозначает регулярное множество $\{\varepsilon\}$, состоящее из пустой строки);

3) $a \in \Sigma$ - регулярное выражение (обозначает множество $\{a\}$);

4) если p и q – регулярные выражения, обозначающие множества P и Q, то посредством операций над выражениями определяются выражения следующих трех типов:

а) $p|q$ или $p+q$ – регулярное выражение (обозначает объединение $P \cup Q$), где символ «|» или «+» называют операцией *или* (*альтернативой*);

б) pq или $p \cdot q$ – регулярное выражение (обозначает множество $PQ = \{xy \mid x \in P, y \in Q\}$), где символ «точка» (возможно умалчиваемый) называют операцией *сцепления* (*конкатенации*);

в) p^* - регулярное выражение (обозначает множество P^*), где символ «*» называют операцией итерации.

Пример 2.2. Примеры регулярных выражений и их значений представлены в таблице 2.1.

Таблица 2.1 – Примеры регулярных выражений

Регулярное выражение	Значение регулярного выражения
01	единственная строка 01
0|1	две строки: 0 и 1
1^*	строки, образованные из единиц, включая пустую строку
$(0\|1)^*$	строки, образованные из символов 0 и 1, включая пустую строку
$0\|1^*$	строка, состоящая из нуля и любые строки из единиц, включая пустую
01^*	строки, начинающиеся с нуля и любой последующей строки единиц, включая пустую
$(0\|1)^*011$	строки, образованные из символов 0 и 1, включая пустую, обязательно оканчивающиеся строкой 011

В практических приложениях вводятся дополнительные соглашения относительно записи регулярных выражений, например, запись вида p^+ используется для обозначения выражения pp^*.

Соотношение между регулярными языками и регулярными выражениями устанавливает теорема Клини.

Теорема Клини. Каждому регулярному языку из Σ^* соответствует регулярное выражение над множеством Σ.

2.2 Конечные автоматы

2.2.1 Определение конечного автомата

Конечный автомат – это простейший распознаватель без вспомогательной памяти. Он является эффективным способом определения регулярных языков.

Определение 2.3. Детерминированным конечным автоматом (ДКА) называется пятерка объектов:

$$M = (Q, T, F, H, Z), \tag{2.1}$$

где Q - конечное множество состояний автомата;

T- конечное множество допустимых входных символов;

F - функция переходов, отображающая множество $Q \times T$ во множество состояний Q;

H - конечное множество начальных состояний автомата;

Z - множество заключительных состояний автомата, $Z \subseteq Q$.

Определение 2.4. Недетерминированным конечным автоматом (НКА) называется конечный автомат, в котором в качестве функции переходов используется отображение $Q \times T$ во множество всех подмножеств множества состояний автомата $P(Q)$, т.е. функция переходов неоднозначна, так как текущей паре (q, t) соответствует множество очередных состояний автомата $q' \in P(Q)$.

Существуют следующие три способа представления функции переходов.

Командный способ. Каждую команду КА записывают в форме $F(q, t) = p$, где $q, p \in Q, t \in T$.

Табличный способ. Строки таблицы переходов соответствуют входным символам автомата $t \in T$, а столбцы – состояниям Q. Ячейки таблицы заполняются новыми состояниями, соответствующими значению функции $F(q, t)$. Неопределенным значениям функции переходов соответствуют пустые ячейки таблицы.

Графический способ. Строится диаграмма состояний автомата – неупорядоченный ориентированный помеченный граф. Вершины графа помечены именами состояний автомата. Дуга ведет из состояния q в состояние p и помечается списком всех символов $t \in T$, для которых $F(q, t) = p$. Вершина, соот-

ветствующая входному состоянию автомата, снабжается стрелкой. Заключительное состояние на графе обозначается двумя концентрическими окружностями.

2.2.2 Распознавание строк конечным автоматом

Определение 2.5. Пара $(q, \tau) \in Q \times T^*$, где τ - текущий остаток входной строки, называется конфигурацией автомата M.

Определение 2.6. Конфигурация (H, τ), где τ - полная входная строка, называется начальной, а пара (q, ε), где $q \in Z$, называется заключительной конфигурацией.

Шаг работы автомата можно представить отношением непосредственного следования конфигураций |-. Отношение достижимости конфигураций будем обозначать |-*. Тогда, если $F(q, t)=p$, где $p \in Q$, то можно записать $(q, t\tau)$ |- (p, τ) для всех $\tau \in T^*$. Эта запись означает то, что если автомат находится в состоянии q и входная головка обозревает входной символ t, то автомат может сделать шаг, за который он переходит в состояние p и сдвигает головку на одну ячейку вправо.

Определение 2.7. Автомат M допускает (принимает) строку τ, если существует путь по конфигурациям (H, τ) |-* (q, ε) для некоторого $q \in Z$.

Определение 2.8. Язык L, принимаемый конечным автоматом M, формально определяется как множество:

$$L(M) = \{\tau \mid \tau \in T^* \text{ и } (H, \tau) \text{ |-* } (q, \varepsilon) \text{ для некоторого } q \in Z\}. \qquad (2.2)$$

2.2.3 Преобразование конечных автоматов

Возникают две основных задачи эквивалентного преобразования КА:

1) преобразование недетерминированного конечного автомата в детерминированный конечный автомат;

2) минимизация КА.

Преобразование конечного автомата к детерминированному виду

В теории КА доказано, что для любого НКА существует ДКА, принимающий тот же регулярный язык. Рассмотрим этот алгоритм.

Алгоритм 2.1. Преобразование НКА в ДКА.

Вход: НКА $M = (Q, T, F, H, Z)$.

Выход: ДКА $M' = (Q', T, F', H, Z')$.

Шаг 1. Пометить первый столбец таблицы переходов M' ДКА начальным состоянием (множеством начальных состояний) НКА M.

Шаг 2. Заполняем очередной столбец таблицы переходов M', помеченный символами множества D, для этого определяем те состояния M, которые могут быть достигнуты из каждого символа строки D при каждом входном символе x. Поместить каждое найденное множество R (в том числе $\varnothing$) в соответствующие позиции столбца D таблицы M', т.е.:

$$F'(D, x) = \{R \mid R \supseteq F(t, x) \text{ для некоторого } t \in D\} \qquad (2.3)$$

Шаг 3. Для каждого нового множества R (кроме $\varnothing$), полученного в столбце D таблицы переходов M', добавить новый столбец в таблицу, помеченный R.

Шаг 4. Если в таблице переходов КА M' есть столбец с незаполненными позициями, то перейти к шагу 2.

Шаг 5. Во множество Z' ДКА M' включить каждое множество, помечающее столбец таблицы переходов M' и содержащее $q \in Z$ НКА M.

Шаг 6. Составить таблицу новых обозначений множеств состояний и определить ДКА M' в этих обозначениях.

Конец алгоритма.

Пример 2.3. Дан НКА $M = (Q, T, F, H, Z)$, где $Q = \{S, A, B, N\}$, $T = \{a, b\}$, $H = \{S\}$, $Z = \{N\}$ и функция переходов задана таблицей 2.2.

Таблица 2.2 – Функция переходов автомата M

F	S	A	B	N
a	A, B	A	N	$\varnothing$
b	$\varnothing$	N	B	$\varnothing$

Граф исходного НКА M представлен на рисунке 2.1.

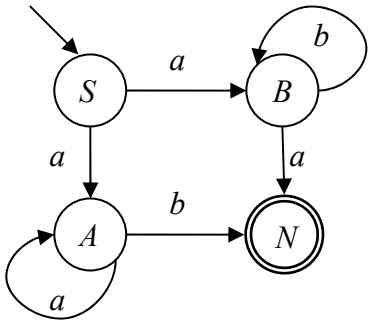


Рисунок 2.1 - Граф НКА

Построим по НКА M ДКА M'.

1 Строим таблицу переходов для ДКА M' (таблица 2.3).

Таблица 2.3 – Построение функции переходов для ДКА M'

Шаг	1	2	3	4	5	6	7
F	S	A, B	A, N	B, N	A	N	B
a	A, B	A, N	A	N	A	$\varnothing$	N
b	$\varnothing$	B, N	N	B	N	$\varnothing$	B

2 Во множество заключительных состояний автомата M' включим элементы $Z' = \{(A, N), (B, N), N\}$.

3 Введем следующие новые обозначения состояний автомата M': $(A, B)=C$, $(A, N)=D$, $(B, N)=E$.

4 Искомый ДКА определяется следующей пятеркой объектов: $Q' = \{S, A, B, C, D, E, N\}$, $T = \{a, b\}$, функция переходов задана таблицей 2.4, $H = \{S\}$, $Z' = \{N, D, E\}$.

Таблица 2.4 – Функция переходов для ДКА M'

F'	S	A	B	C	D	E	N
a	C	A	N	D	A	N	$\varnothing$
b	$\varnothing$	N	B	E	N	B	$\varnothing$

Граф полученного ДКА представлен на рисунке 2.2.

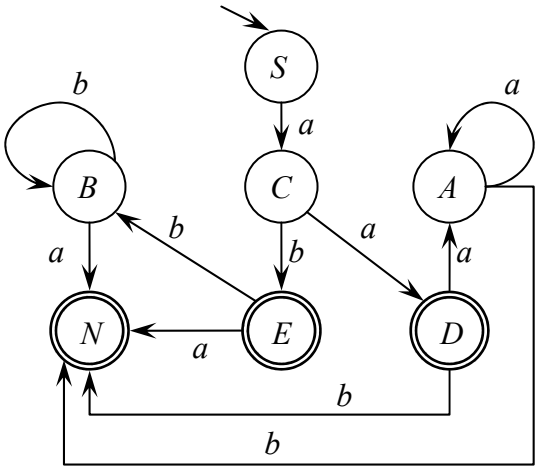


Рисунок 2.2 – Граф ДКА

Минимизация конечного автомата

Конечный автомат может содержать лишние состояния двух типов: недостижимые и эквивалентные состояния.

Определение 2.9. Два различных состояния q и q' в конечном автомате $M = (Q, T, F, H, Z)$ называются n-эквивалентными, $n \in N \cup \{0\}$, если, находясь в одном их этих состояний и получив на вход любую цепочку символов ω: $\omega \in T^*$, $|\omega| \le n$, автомат может перейти в одно и то же множество конечных состояний.

Определение 2.10. Состояние q КА называется недостижимым, если к нему нет пути из начального состояния автомата.

Определение 2.11. КА, не содержащий недостижимых и эквивалентных состояний, называется приведенным или минимальным КА.

В теории КА доказано, что каждое регулярное множество распознается единственным для данного множества ДКА с минимальным числом состояний. Рассмотрим алгоритмы построения минимального ДКА.

Алгоритм 2.2. Устранение недостижимых состояний КА.

Вход: КА $M = (Q, T, F, H, Z)$.

Выход: КА $M' = (Q', T, F', H, Z')$.

Шаг 1. Поместить начальное состояние КА в список достижимых состояний $Q_д$, т.е. $Q_д^0 = \{H\}$.

Шаг 2. Для новых элементов списка достижимых состояний пополнить список группой их состояний-приемников, отсутствующих в списке, т.е.

$$Q_д^i = Q_д^{i-1} \cup \{p \mid \forall q \in Q_д^{i-1} \; \exists \, F(q, t) = p\}. \qquad (2.4)$$

Шаг 3. Повторить шаг 2, пока список достижимых состояний не перестанет меняться. То есть, если $Q_д^i \neq Q_д^{i-1}$, то i:=i+1, иначе $Q_д = Q_д^i$.

Шаг 4. Исключить из множества Q состояний КА все состояния, отсутствующие в списке $Q_д$ достижимых состояний, т.е. $Q' = Q \cap Q_д$.

Шаг 5. Исключить недостижимые заключительные состояния и функции переходов, содержащие недостижимые состояния, т.е. $Z' = Z \cap Q_д$, $F' = F - \{F(q, t) = p \mid q \in (Q - Q_д)\}$.

Конец алгоритма.

Пример 2.4. Устранить недостижимые состояния КА $M = (Q, T, F, H, Z)$, где $Q = \{A, B, C, D, E, F, G\}$, $T = \{a, b\}$, $H = \{A\}$, $Z = \{D, E\}$ и функция переходов задана таблицей 2.5.

Таблица 2.5 – Функция переходов конечного автомата M

F	A	B	C	D	E	F	G
a	B			C	B	D	F
b	C	D	E	E	D	G	E

Граф исходного КА M представлен на рисунке 2.3.

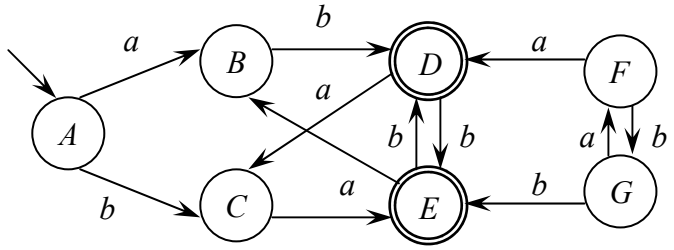


Рисунок 2.3 – Граф исходного конечного автомата M

Последовательность устранения недостижимых состояний КА имеет вид:

$Q_0 = \{A\}$;
$Q_1 = \{A, B, C\}$;
$Q_2 = \{A, B, C, D, E\}$;
$Q_3 = \{A, B, C, D, E\}$; т.к. $Q_2 = Q_3$, то $Q_д = \{A, B, C, D, E\}$.
$Q_н = \{F, G\}$; $Q' = \{A, B, C, D, E\}$; $Z' = \{D, E\}$.

Функция переходов автомата M' представлена в таблице 2.6.

Таблица 2.6 - Функция переходов автомата M'

F	A	B	C	D	E
a	B			C	B
b	C	D	E	E	D

Граф КА M' после устранения недостижимых состояний представлен на рисунке 2.4.

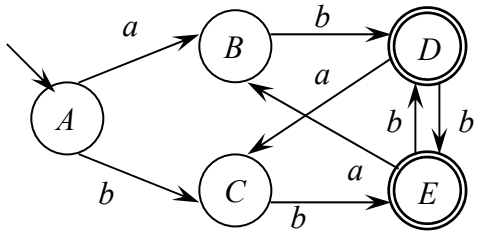


Рисунок 2.4 - Граф КА M' после устранения недостижимых состояний

Объединение эквивалентных состояний КА

Алгоритм 2.3. Объединение эквивалентных состояний КА.

Вход: КА $M' = (Q', T, F', H, Z')$ без недостижимых состояний.

Выход: минимальный КА $M'' = (Q'', T, F'', H, Z'')$.

Шаг 1. На первом шаге строим нулевое разбиение $R(0)$, состоящее из двух классов эквивалентности: заключительные состояния КА - Z и не заключительные - Q-Z.

Шаг 2. На очередном шаге построения разбиения $R(n)$ в классы эквивалентности включить те состояния, которые по одинаковым входным символам переходят в n-1 эквивалентные состояния, т.е.

$$R(n) = \{r_i(n) : \{q_{ij} \in Q : \forall t \in T \; F(q_{ij}, t) \subseteq r_j(n-1)\} \; \forall i, j \in N\}. \qquad (2.5)$$

Шаг 3. До тех пор, пока $R(n) \neq R(n\text{-}1)$ полагаем n:=n+1 и идем к шагу 2.

Шаг 4. Переобозначить оставшиеся неразбитые группы состояний и включить их в таблицу новых обозначений состояний автомата.

Шаг 5. Определить эквивалентный КА M'' в новых обозначениях.

Пример 2.5. Минимизировать конечный автомат из примера 2.4.

Последовательность построения разбиений будет иметь вид:
$R(0) = \{\{A, B, C\}, \{D, E\}\}$, $n = 0$;
$R(1) = \{\{A\}, \{B, C\}, \{D, E\}\}$, $n = 1$;
$R(2) = \{\{A\}, \{B, C\}, \{D, E\}\}$, $n=2$.
Т.к. $R(1) = R(2)$, то искомое разбиение построено.

Переобозначим оставшиеся неразбитые группы состояний:
$X=\{B, C\}$, $Y=\{D, E\}$.
Получим минимальный автомат M'', где $Q''=\{A, X, Y\}$, $Z''=\{Y\}$.
Функция переходов автомата M'' представлена в таблице 2.7.

Таблица 2.7 - Функция переходов автомата M''

F''	A	X	Y
a	X		X
b	X	Y	Y

Граф переходов конечного автомата после его минимизации показан на рисунке 2.5.

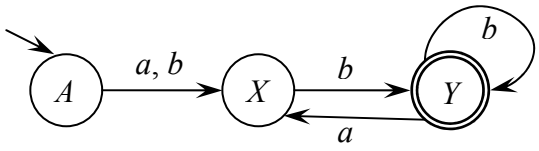


Рисунок 2.5 – Граф минимального ДКА M''

2.3 Взаимосвязь способов определения грамматик

Регулярные грамматики, регулярные выражения и конечные автоматы – три основных способа определения регулярных языков, между которыми существует полное соответствие. Разработаны алгоритмы преобразования одной формы определения в другую. При работе с языками программирования наибольший практический интерес представляют преобразования регулярной грамматики в конечный автомат. Рассмотрим их.

Алгоритм 2.4. Построение КА по регулярной грамматике.
Вход: Регулярная грамматика $G = (V_T, V_N, P, S)$.
Выход: КА $M = (Q, T, F, H, Z)$.

Шаг 1. Пополнить грамматику правилом $A \to aN$, где $A \in V_N, a \in V_T$ и N - новый нетерминал, для каждого правила вида $A \to a$, если в грамматике нет соответствующего ему правила $A \to aB$, где $B \in V_N$.

Шаг 2. Начальный символ грамматики S принять за начальное состояние КА H. Из нетерминалов образовать множество состояний автомата $Q = V_N \cup \{N\}$, а из терминалов – множество символов входного алфавита $T = V_T$.

Шаг 3. Каждое правило $A \to aB$ преобразовать в функцию переходов $F(A, a) = B$, где $A, B \in V_N, a \in V_T$.

Шаг 4. Во множество заключительных состояний включить все вершины, помеченные символами $B \in V_N$ из правил вида $A \to aB$, для которых имеются соответствующие правила $A \to a$, где $A, B \in V_N, a \in V_T$.

Шаг 5. Если в грамматике имеется правило $S \to \varepsilon$, где S - начальный символ грамматики, то поместить S во множество заключительных состояний.

Шаг 6. Если получен НКА, то преобразовать его в ДКА.

Конец алгоритма.

Пример 2.6. Дана регулярная грамматика G=($\{a, b, c, \perp\}$, $\{S, A, B, C\}$, P, S), где P: 1) $S \to aA \mid bB \mid \varepsilon$ 2) $B \to bB \mid cC$ 3) $A \to aA \mid cC$ 4) $C \to cC \mid aS \mid \perp$. Построить по регулярной грамматике конечный автомат.

Решение задачи состоит из следующей последовательности действий.

1 Пополним грамматику правилом $C \to \perp N$, где N - новый нетерминал.

2 Начальное состояние конечного автомата $H = \{S\}$. Множество состояний автомата $Q = V_N = \{S, A, B, C, N\}$, множество символов входного алфавита $T = V_T = \{a, b, c, \perp\}$.

3 Значения сформированной функции переходов даны в таблице 2.8.

Таблица 2.8 – Функция переходов автомата M

F	S	A	B	C	N
a	A	A	$\varnothing$	S	$\varnothing$
b	B	$\varnothing$	B	$\varnothing$	$\varnothing$
c	$\varnothing$	C	C	C	$\varnothing$

4 Множество заключительных состояний $Z = \{N\}$.

5 Для начального символа грамматики есть ε-правило, значит добавляем его во множество заключительных состояний $Z = \{S, N\}$.

Конечный автомат M - детерминированный, граф КА представлен на рисунке 2.6.

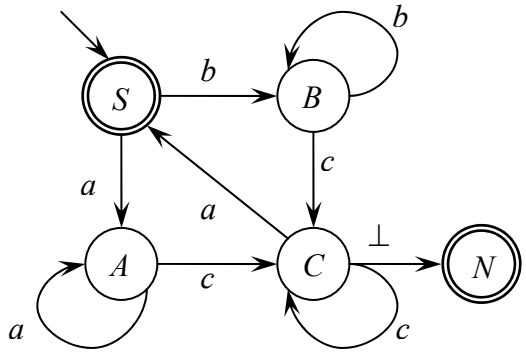


Рисунок 2.6 - Граф искомого ДКА

Вопросы и задания для самоконтроля по 2 разделу

1 Какое множество называется регулярным?

2 Какой метод лежит в основе определения регулярности заданного языка?

3 Назовите способы задания регулярных языков.

4 Какое выражение называется регулярным?

5 Сформулируйте теорему Клини.

6 Дайте формальное определение конечного автомата.

7 Какой конечный автомат называется недетерминированным?

8 Какие существуют способы представления функции переходов конечного автомата?

9 Расскажите алгоритм функционирования конечного автомата.

10 Определите формально язык, принимаемый конечным автоматом.

11 Изложите алгоритм преобразования конечного автомата к детерминированному виду.

12 Какие состояния конечного автомата называются недостижимыми, эквивалентными?

13 Сформулируйте алгоритмы минимизации конечного автомата.

14 Как по заданной регулярной грамматике построить конечный автомат, принимающий тот же язык?

15 Записать регулярное выражение, задающее язык, состоящий из строк множества $\{0, 1\}^*$, таких что:

а) начинаются с 011 и оканчиваются на 110 или 101;

б) содержат каждую из подстрок 011 и 101;

в) каждый третий символ есть 0 или каждый второй – 1.

16 Является ли язык, состоящий из всех цепочек чётной длины в алфавите $\{a, b, c\}$, регулярным?

17 Определите язык, состоящий из всех идентификаторов, с помощью:

а) регулярного выражения;

б) конечного автомата;

в) регулярной грамматики.

18 Дана регулярная грамматика $G = (\{S, C, D\}, \{0, 1\}, P, S)$, где P: 1) $S \to 1C \mid 0D$; 2) $C \to 0D \mid 0S \mid 1$; 3) $D \to 1C \mid 1S \mid 0$. Построить по ней конечный автомат и преобразовать его к детерминированному виду.

19 Минимизировать конечный автомат, приведенный на рисунке 2.7.

Тесты проверки усвоения материала по 2 разделу

1 Согласно лемме о разрастании языка, регулярным является язык:

а) $L = \{a^n b^n c^n \mid n \geq 0\}$;

б) $L = \{(ac)^n (cb)^n \mid n \geq 0\}$;

в) $L = \{\perp c^n b^n \perp \mid n \geq 0\}$;

г) $L = \{a^m b^n c^k \mid m, n, k \geq 0\}$;

д) $L(G) = \{a^n + b^n \mid n \geq 0\}$.

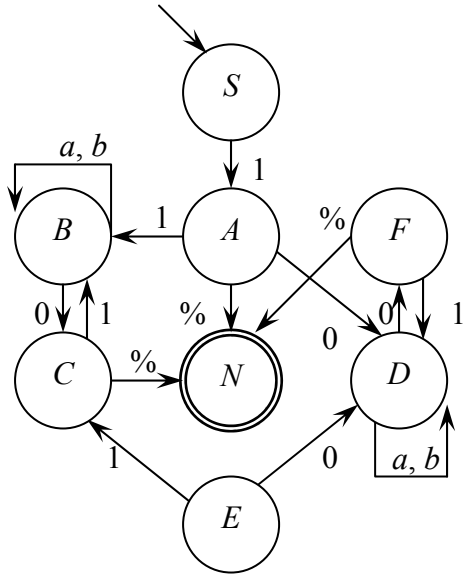


Рисунок 2.7 – Исходный КА

2 В регулярных выражениях знак «|» определяет операцию:
а) реверс;
б) итерация;
в) альтернатива;
г) конкатенация;
д) пересечение.

3 Операция итерация в регулярных выражениях обозначается знаком:
а) *;
б) +;
в) |;
г) ·;
д) -.

4 Строки, образованные из символов 0 и 1, включая пустую строку, начинающиеся с 0, описываются регулярным выражением:
а) $0+(0|1)^*$;
б) $0(0|1)^*$;
в) $0(01)^*$;
г) $0(0^*1^*)$;
д) $0^*(0|1)$.

5 Операции регулярных выражений в порядке уменьшения приоритета:
а) ( ), ·, *, |;
б) ( ), |, ·, *;
в) |, ( ), ·, *;
г) ·, ( ), *, |;
д) ( ), *, ·, |.

6 Языку, описываемому регулярным выражением $(0|1)^*01$, принадлежит строка:

а) 01011;
б) 10110;
в) 00101;
г) 00010;
д) 11100.

7 Конечный автомат, представленный на рисунке 2.6 допускает строку:

а) *bcabb*;
б) *accaa*;
в) *acab*⊥;
г) *bbca*⊥;
д) *bbcc*⊥.

8 Для конечного автомата, представленного на рисунке 2.6, справедливо:

а) $(S, bbc\perp) \vdash^* (B, \perp)$;
б) $(C, aac\perp) \vdash^* (C, c\perp)$;
в) $(B, cabc\perp) \vdash^* (S, c\perp)$;
г) $(A, acc\perp) \vdash^* (N, \varepsilon)$;
д) $(B, bccc\perp) \vdash^* (N, \perp)$.

9 Автомат $M(Q, T, F, H, Z)$ допускает строку τ, если существует путь по конфигурациям:

а) $(H, \tau) \vdash^* (q, \varepsilon)$ для некоторого $q \in Z$;
б) $(H, \tau) \vdash (q, \varepsilon)$ для некоторого $q \in Z$;
в) $(q, \tau) \vdash^* (q, \varepsilon)$ для некоторого $q \in Z$;
г) $(H, \varepsilon) \vdash^* (q, \tau)$ для некоторого $q \in Z$;
д) $(H, \varepsilon) \vdash (q, \tau)$ для некоторого $q \in Z$.

10 Язык L, принимаемый конечным автоматом $M(Q, T, F, H, Z)$ формально определяется как множество:

а) $L(M) = \{\tau \mid \tau \in Q^*$ и $(H, \tau) \vdash (q, \varepsilon)$ для некоторого $q \in Z\}$;
б) $L(M) = \{\tau \mid \tau \in T^*$ и $(H, \tau) \vdash^* (q, \varepsilon)$ для некоторого $q \in Z\}$;
в) $L(M) = \{\tau \mid \tau \in T^+$ и $(q, \tau) \vdash^* (q, \varepsilon)$ для некоторого $q \in Z\}$;
г) $L(M) = \{\tau \mid \tau \in T^*$ и $(q, \tau) \vdash^* (H, \varepsilon)$ для некоторого $q \in Z\}$;
д) $L(M) = \{\tau \mid \tau \in T^*$ и $(H, \varepsilon) \vdash^* (q, \tau)$ для некоторого $q \in Z\}$.

11 Функция переходов НКА задана таблицей 2.9. Для эквивалентного ДКА справедливо:

а) $F'(AB, a)=AAN$;
б) $F'(AB, a)=BN$;
в) $F'(AB, a)=A$;

г) $F'(AB, a)=AN$;
д) $F'(AB, a)=N$.

Таблица 2.9 – Функция переходов НКА

F	S	A	B	N
a	AB	A	AN	$\varnothing$
b	$\varnothing$	BN	B	$\varnothing$

12 Функция переходов НКА задана таблицей 2.9. Порядок следования вершин графа КА по возрастанию количества исходящих дуг:
а) N, A, S;
б) N, S, A;
в) S, N, A;
г) S, A, N;
д) A, S, N.

13 Граф НКА, функция переходов которого задана таблицей 2.9, будет содержать:
а) 6 дуг и 3 петли;
б) 9 дуг и ниодной петли;
в) 6 дуг и 1 петлю;
г) 6 дуг и 2 петли;
д) 3 дуги и две петли.

14 Для КА, граф которого задан на рисунке 2.8, справедливо:
а) $F(Q, 0)=P$;
б) $F(R, 1)=P$;
в) $F(Q, \sim)=R$;
г) $F(U,))=R$;
д) $F(M, 0)=R$.

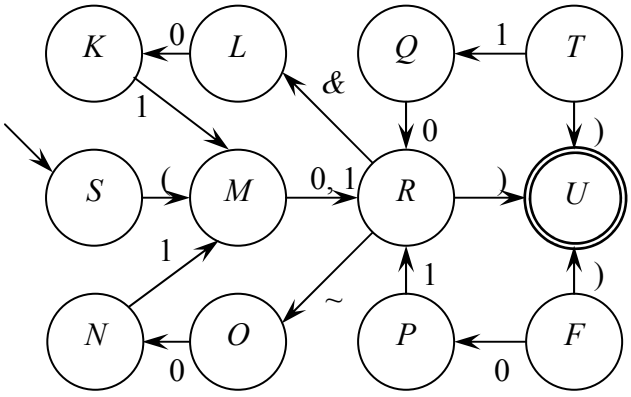


Рисунок 2.8 – Граф ДКА

15 Для КА $M=(Q, T, F, H, Z)$, граф которого задан на рисунке 2.8, справедливо:

а) $H=\{S\}$, $Z=\{U\}$, $T=\{0, 1, (,), \sim, \&\}$;
б) $H=\{U\}$, $Z=\{S\}$, $T=\{0, 1, (,), \sim, \&\}$;
в) $H=\{S\}$, $Z=\{U\}$, $T=\{0, 1, \sim, \&\}$;
г) $H=\{S, U\}$, $Z=\{U\}$, $T=\{0, 1, (,), \sim, \&\}$;
д) $H=\{S\}$, $Z=\{U\}$, $T=\{(,), \sim, \&\}$.

16 Граф КА $M=(Q, T, F, H, Z)$ задан на рисунке 2.8. Недостижимыми состояниями автомата являются:

а) Q, T, P, F, U;
б) Q, T, P, F, R;
в) Q, T, P, F;
г) Q, T;
д) P, F.

17 Граф КА $M=(Q, T, F, H, Z)$ задан на рисунке 2.8. Эквивалентным являются состояния:

а) K, M;
б) N, M;
в) Q, P;
г) T, F;
д) L, O.

18 Граф КА $M=(Q, T, F, H, Z)$ задан на рисунке 2.8. Состояния автомата N и K являются:

а) недостижимыми;
б) начальными;
в) заключительными;
г) эквивалентными;
д) приведенными.

19 Дана грамматика с правилами P: 1) $E\to 0A \mid \varepsilon$; 2) $A\to aB \mid aD$; 3) $B\to bB \mid 1C \mid c$; 4) $D\to aD \mid 0C \mid c$. В эквивалентном ей КА заключительными будут состояния:

а) новый нетерминал N;
б) E и новый нетерминал N;
в) E;
г) D;
д) E и D.

20 Дана грамматика с правилами P: 1) $E\to 0A \mid \varepsilon$; 2) $A\to aB \mid aD$; 3) $B\to bB \mid 1C \mid c$; 4) $D\to aD \mid 0C \mid c$. Для эквивалентного ей КА справедливо:

а) $F(A, a)=BD$;

б) $F(B, b)=BC$;
в) $F(D, 0)=DC$;
г) $F(D, a)=C$;
д) $F(A, a)=D$.

21 Дана грамматика с правилами P: 1) $E\to 0A \mid \varepsilon$; 2) $A\to aB \mid aD$; 3) $B\to bB \mid 1C \mid c$; 4) $D\to aD \mid 0C \mid c$. Множество входных символов эквивалентного ей КА:

а) {0, 1, a, b, c, ε};
б) {a, b, c};
в) {0, 1, a, b, c};
г) {E, A, B, C, D};
д) {E, A, B, C, D, N}.

22 Дана грамматика с правилами P: 1) $E\to 0A \mid \varepsilon$; 2) $A\to aB \mid aD$; 3) $B\to bB \mid 1C \mid c$; 4) $D\to aD \mid 0C \mid c$. Множество {E, A, B, C, D, N} в эквивалентном КА будет определять:

а) входной алфавит КА;
б) функцию переходов КА;
в) начальные состояния КА;
г) состояния КА;
д) заключительные состояния КА.

23 Если во множестве правил вывода регулярной грамматики есть ε-правило для начального символа S, то

а) S будет конечным состоянием эквивалентного КА;
б) S будет начальным состоянием эквивалентного КА;
в) S будет начальным и конечным состоянием эквивалентного КА;
г) S будет отсутствовать в эквивалентном КА;
д) S будет входным символом в эквивалентном КА.

24 В НКА $M=(Q, T, F, H, Z)$ функция переходов F задает отображение:

а) F: $Q\times T\to Q$;
б) F: $Q\times T\to P(T)$;
в) F: $Q\times T\to T$;
г) F: $Q\times T\to Q\times T$;
д) F: $Q\times T\to P(Q)$.

25 Если функция переходов КА текущей паре (q, t) сопоставляет множество очередных состояний автомата $q' \in P(Q)$, то автомат:

а) детерминированный;
б) недетерминированный;
в) приведенный;
г) неприведенный;
д) минимальный.

3 Контекстно-зависимые языки и грамматики

3.1 Задача разбора цепочек

Процесс порождения предложения языка на основе правил, определяющих язык грамматики, называется выводом.

Определение 3.1. Последовательность непосредственно выводимых цепочек называется выводом или цепочкой вывода (см. определения 1.10 и 1.11).

Определение 3.2. Вывод называется правосторонним (левосторонним), если в нем на каждом шаге вывода правило грамматики применяется всегда к крайнему правому (левому) нетерминальному символу в цепочке.

Вывод можно рассматривать также как процесс получения одной строки из другой. С понятием вывода тесно связано понятие разбора строки языка.

Определение 3.3. Разбор строки - это задача выяснения принадлежности заданной строки языку, порождаемому данной грамматикой. А также это последовательность правил грамматики, определенным образом соответствующая выводу.

Пример 3.1. В грамматике G=($\{a, b, +\}$, $\{S, T\}$, $\{S \to T|T+S;\ T \to a|b\}$, S) построим левосторонний и правосторонний вывод строки $a+b+a$.

Левосторонний вывод:

$$S \to \underline{T}+S \to a+\underline{S} \to a+\underline{T}+S \to a+b+\underline{S} \to a+b+\underline{T} \to a+b+a.$$

Правосторонний вывод:

$$S \to T+\underline{S} \to T+T+\underline{S} \to T+T+\underline{T} \to T+\underline{T}+a \to \underline{T}+b+a \to a+b+a.$$

3.1.2 Дерево разбора

Графическим способом отображения процесса разбора цепочек является дерево разбора (или дерево вывода).

Определение 3.4. Деревом разбора грамматики $G = (V_T, V_N, P, S)$ называется дерево, которое соответствует некоторой цепочке вывода и удовлетворяет следующим условиям:

- каждая вершина дерева обозначается символом грамматики $A \in (V_T \cup V_N \cup \varepsilon)$;
- корнем дерева является вершина, обозначенная начальным символом грамматики S;
- листьями дерева (концевыми вершинами) являются вершины, обозначенные терминальными символами грамматики или символом пустой строки ε;

- если некоторый узел дерева обозначен символом $A \in V_N$, а связанные с ним узлы – символами $b_1, b_2, ..., b_n; n > 0, b_i \in (V_T \cup V_N \cup \{\varepsilon\}), 1 \le i \le n$, то в грамматике $G = (V_T, V_N, P, S)$ существует правило $A \to b_1 b_2 ... b_n \in P$.

Дерево разбора можно построить двумя способами: сверху вниз и снизу вверх.

Нисходящее дерево разбора

При построении дерева вывода сверху вниз построение начинается с начального символа грамматики, который помещается в корень дерева. Затем в грамматике выбирается необходимое правило, и на первом шаге вывода корневой символ раскрывается на несколько символов первого уровня. На втором шаге среди всех концевых вершин дерева выбирается крайняя (крайняя левая для левостороннего вывода, крайняя правая для правостороннего) вершина, обозначенная нетерминальным символом, для этой вершины выбирается нужное правило грамматики, и она раскрывается на несколько вершин следующего уровня. Построение дерева заканчивается, когда все концевые вершины обозначены терминальными символами, в противном случае надо вернуться ко второму шагу и продолжить построение.

Восходящее дерево разбора

Построение дерева вывода снизу вверх начинается с листьев дерева. В качестве листьев выбираются терминальные символы конечной цепочки вывода, которые на первом шаге построения образуют последний уровень (слой) дерева. Построение дерева идет по слоям. На втором шаге построения в грамматике выбирается правило, правая часть которого соответствует крайним символам в слое дерева (крайним правым символам при правостороннем выводе и крайним левым при левостороннем). Выбранные вершины слоя соединяются с новой вершиной, которая выбирается из левой части правила. Новая вершина попадает в слой дерева вместо выбранных вершин. Построение дерева закончено, если достигнута корневая вершина (обозначенная начальным символом), иначе следует вернуться ко второму шагу и повторить его над полученным слоем дерева.

Пример 3.2. Нисходящее дерево вывода строки *a+b+a* в грамматике *G* (пример 3.1) показано на рисунке 3.1.

Процесс построения восходящего дерева разбора показан на рисунке 3.2.

3.1.3 Однозначность грамматик

Определение 3.5. Грамматика называется однозначной, если для каждой цепочки символов языка, заданного этой грамматикой, можно построить единственный левосторонний (и единственный правосторонний) вывод.

Другими словами, для каждой цепочки символов языка, заданного однозначной грамматикой, существует единственное дерево вывода. В противном случае грамматика называется неоднозначной.

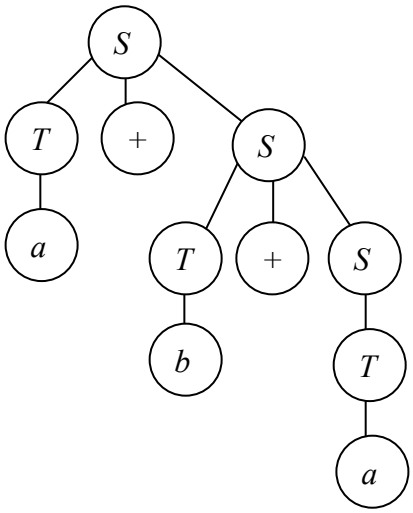
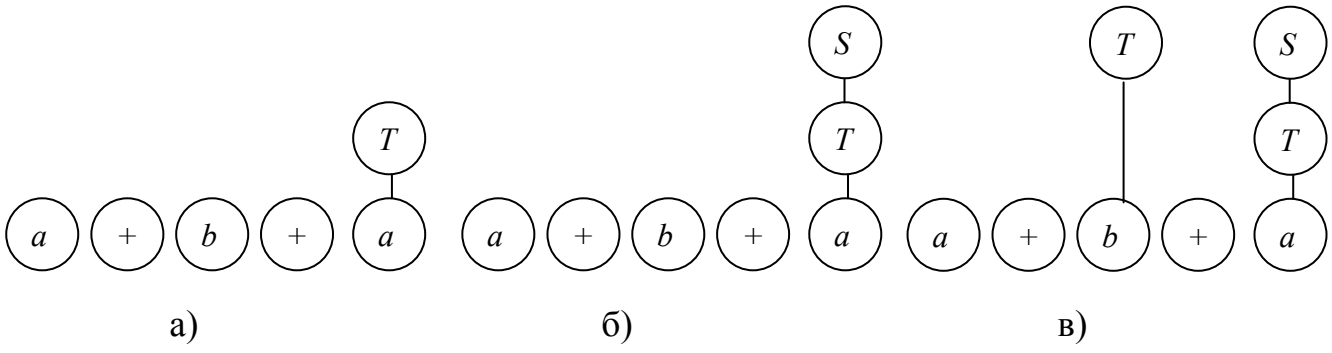
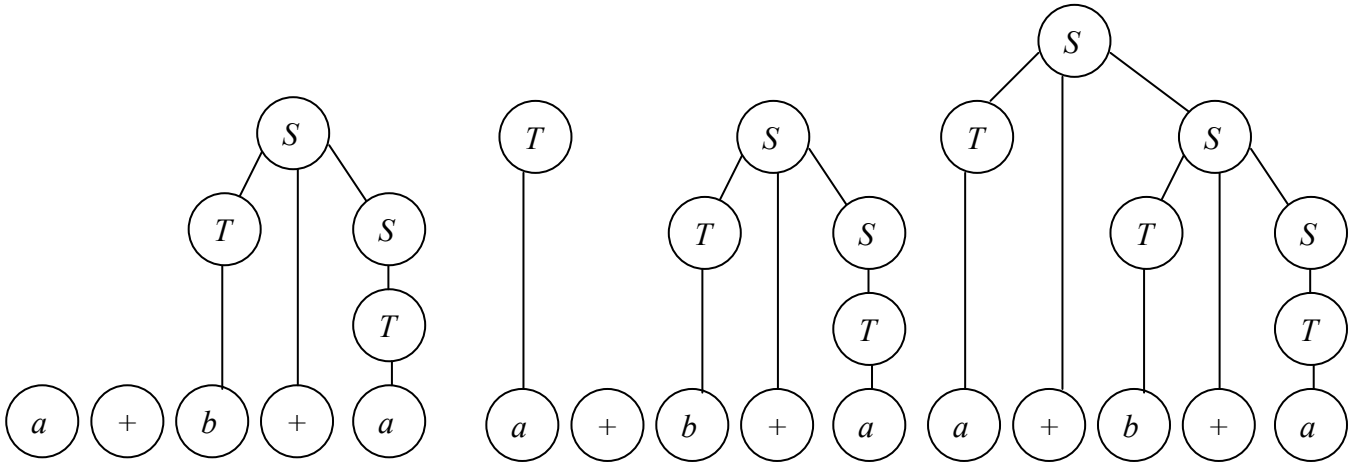


Рисунок 3.1 – Нисходящее дерево разбора



а) б) в)



г) д) е)

Рисунок 3.2 – Процесс построения восходящего дерева разбора

Пример 3.3. Грамматика G из примера 3.1 является однозначной.

Грамматика условного оператора G_1=({*if*, *then*, *else*, *a*, *b*}, {*S*}, *P*, *S*) P={$S \to$ *if b then S else S* | *if b then S* | *a*} является неоднозначной. Показать ее неоднозначность можно, построив различные деревья разбора, например, для строки вида *if b then if b then a else a*. Неоднозначность в данном случае связана с тем, что одно и тоже *else* может быть отнесено к различным *then*. Но если условиться, что *else* всегда должно относится к ближайшему *then* и преобразовать правила вывода к виду (3.1), то грамматика G_1 становится однозначной.

$$\begin{aligned} P&: S \to \textit{if b then } S \mid \textit{if b then } S' \textit{ else } S \mid a \\ &\quad S' \to \textit{if b then } S' \textit{ else } S' \mid a \end{aligned} \tag{3.1}$$

Задача установления неоднозначности грамматики в общем случае является неразрешимой. Но наличие любого правила вывода вида 3.2 приводит к неоднозначности грамматики.

$$\begin{aligned} P&: A \to AA \mid \alpha \\ &\quad A \to A\alpha A \mid \beta \\ &\quad A \to \alpha A \mid A\beta \mid \gamma \\ &\quad A \to \alpha A \mid \alpha A \beta A \mid \gamma \end{aligned} \tag{3.2}$$

где $A \in V_N$;

$\alpha, \beta, \gamma \in V^*$.

Некоторые неоднозначные грамматики можно преобразовать в однозначные, порождающие тот же язык. Рассмотрим возможные эквивалентные преобразования КС-грамматик.

3.2 Преобразование КС-грамматик

Определение 3.6. КС-грамматика называется приведенной, если она не имеет циклов, ε-правил и бесполезных символов.

Рассмотрим основные алгоритмы приведения КС-грамматик.

Перед всеми другими исследованиями и преобразованиями КС-грамматик выполняется проверка существования языка грамматики.

3.2.1 Проверка существования языка грамматики

Алгоритм 3.1. Проверка существования языка грамматики.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: заключение о существовании или отсутствии языка грамматики.

Определим множество нетерминалов, порождающих терминальные строки $N = \{Z \mid Z \in V_N, Z \Rightarrow *x, x \in V_T^*\}$.

Шаг 1. Положить N_0=Ø.

Шаг 2. Вычислить $N_i = N_{i-1} \cup \{A \mid (A \to \alpha) \in P \text{ и } \alpha \in (N_{i-1} \cup V_T)^*\}$.

Шаг 3. Если $N_i \neq N_{i-1}$, то положить i:=i+1 и перейти к пункту 2, иначе считать $N = N_i$.

Если $S \in N$, то выдать сообщение о том, что язык грамматики существует, иначе сообщить об отсутствии языка.

Конец алгоритма.

Пример 3.4. Дана грамматика $G = (\{0,1\}, \{S, A, B\}, P, S)$, где множество правил P: 1) $S \to AB$; 2) $A \to 0A$; 3) $A \to 0$; 4) $B \to 1$. Построим последовательность приближений множества N:

$N_0 = \varnothing$;

$N_1 = \{A, B\}$;

$N_2 = \{S, A, B\}$;

$N_3 = \{S, A, B\}$.

Т.к. N_2=N_3, то $N = \{S, A, B\}$, следовательно, язык грамматики существует, потому что начальный символ $S \in N$.

3.2.2 Удаление бесполезных символов грамматики

Определение 3.7. Бесполезными символами грамматики называют:

а) нетерминалы, не порождающие терминальных строк, т.е. множество символов вида:

$$\{X \mid X \in V_N, \neg\exists(X \Rightarrow *x), x \in V_T^*\}; \tag{3.3}$$

б) недостижимые нетерминалы, порождающие терминальные строки, т.е. множество символов вида:

$$\{X \mid X \in V_N, \neg\exists(S \Rightarrow *\alpha X\beta), \exists(X \Rightarrow *x); \alpha, \beta \in V^*; x \in V_T^*\}; \tag{3.4}$$

в) недостижимые терминалы, т.е. множество символов вида:

$$\{X \mid X \in V_T, \neg\exists(S \Rightarrow *\alpha X\beta); \alpha, \beta \in V^*\}. \tag{3.5}$$

Алгоритм 3.2. Устранение нетерминалов, не порождающих терминальных строк.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: КС-грамматика $G' = (V_T, V_N', P', S)$, такая, что $L(G') = L(G)$ и для всех $Z \in V_N'$ существуют выводы $Z \Rightarrow *x$, где $x \in V_T^*$.

Шаг 1. Определить множество нетерминалов, порождающих терминальные строки, с помощью алгоритма 3.1.

Шаг 2. Вычислить $V'_N = V_N \cap N, N_Б = V_N - V'_N, P' = P - P_Б$, где $P_Б \subseteq P$ - это множество правил, содержащих бесполезные нетерминалы $X \in N_Б$.

Конец алгоритма.

Пример 3.5. Дана грамматика $G = (\{a, b, c\}, \{S, A, B, C\}, P, S)$ с правилами P: 1) $S \to ab$; 2) $S \to AC$; 3) $A \to AB$; 4) $B \to b$; 5) $C \to cb$.

Преобразуем ее в эквивалентную грамматику G' по алгоритму 3.2:

$N_0 = Ø$;

$N_1 = \{S, B, C\}$;

$N_2 = \{S, B, C\}$.

Т.к. $N_1 = N_2$, то $N = \{S, B, C\}$. После удаления бесполезных нетерминалов и правил вывода, получим грамматику $G' = (\{a, b, c\}, \{S, B, C\}, P', S)$ с правилами P': 1) $S \to ab$; 2) $B \to b$; 3) $C \to cb$.

Алгоритм 3.3. Устранение недостижимых символов грамматики.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: КС-грамматика $G' = (V'_T, V'_N, P', S)$, такая, что $L(G') = L(G)$ и для всех $Z \in V'$ существует вывод $S \Rightarrow {}^*\alpha Z\beta$, где $\alpha, \beta \in (V')^*$.

Определим множество достижимых символов Z грамматики G, т.е. множество:

$$W = \{Z \mid Z \in V, \exists (S \Rightarrow {}^*\alpha Z\beta); \alpha, \beta \in V^*\}. \qquad (3.6)$$

Шаг 1. Положить $W_0 = \{S\}$.

Шаг 2. Вычислить очередное приближение следующим образом:

$$W_i = W_{i-1} \cup \{X \mid X \in V, (A \to \alpha X\beta) \in P, A \in W_{i-1}; \alpha, \beta \in V^*\}. \qquad (3.7)$$

Шаг 3. Если $W_i \neq W_{i-1}$, то положить i:=i+1 и перейти к шагу 2, иначе считать $W = W_i$.

Шаг 4. Вычислить $V'_N = V_N \cap W, V'_T = V_T \cap W, V_Б = V - W, P' = P - P_Б$, где $P_Б \subseteq P$ - это множество правил, содержащих недостижимые символы $X \in V_Б$.

Конец алгоритма.

Пример 3.6. Дана грамматика $G = (\{a, b, c\}, \{S, B, C\}, P, S)$ с правилами P': 1) $S \to ab$; 2) $B \to b$; 5) $C \to cb$.

Преобразуем ее в эквивалентную грамматику G' по алгоритму 3.3:

$W_0 = \{S\}$;
$W_1 = \{S, a, b\}$;
$W_2 = \{S, a, b\}$.

Т.к. $W_1 = W_2$, то $W = \{S, a, b\}$. Множество недостижимых символов $V_Б = \{B, C, c\}$. Тогда после удаления недостижимых символов, получим грамматику $G' = (\{a, b\}, \{S\}, P', S)$ с правилом P': $S \to ab$.

3.2.3 Устранение ε-правил грамматики

Алгоритм 3.4. Устранение ε-правил.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V'_N, P', S')$ без ε-правил для всех нетерминальных символов, кроме начального, который в этом случае не должен встречаться в правых частях правил грамматики.

Шаг 1. В исходной грамматике G найти ε-порождающие нетерминальные символы $A \in V_N$, такие, что $A \Rightarrow {}^*\varepsilon$.

1.1 Положить $N_0 = \{A \mid (A \to \varepsilon) \in P\}$.

1.2 Вычислить $N_i = N_{i-1} \cup \{B \mid (B \to \alpha) \in P, \alpha \in N_{i-1}^*\}$.

1.3 Если $N_i \neq N_{i-1}$, то положить i:=i+1 и перейти к пункту 1.2, иначе считать $N = N_i$.

Шаг 2. Из множества P правил исходной грамматики G перенести во множество P' все правила, за исключением ε-правил, т.е. $P' = P - \{(A \to \varepsilon) \in P$ для всех $A \in V_N\}$.

Шаг 3. Пополнить множество P' правилами, которые получаются из каждого правила этого множества путем исключения всевозможных комбинаций ε-порождающих нетерминалов в правой части. Полученные при этом ε-правила во множество P' не включать.

Шаг 4. Если $S \in N$, то $P' = P \cup \{S' \to \varepsilon, S' \to S\}, V'_N = V_N \cup S'$, где $V \cap \{S'\} = \varnothing$; иначе $V'_N = V_N, S' = S$.

Конец алгоритма.

Пример 3.7. Дана грамматика $G = (\{0, 1\}, \{S, A, B\}, P, S)$ с правилами P: 1) $S \to AB$; 2) $A \to 0A \mid \varepsilon$; 3) $B \to 1B \mid \varepsilon$. Преобразуем ее в эквивалентную грамматику по алгоритму 3.4.

Шаг 1. $N_0 = \{A, B\}$;
$N_1 = \{S, A, B\}$;
$N_2 = \{S, A, B\}$.

Т.к. $N_1 = N_2$, то искомое множество построено и $N = \{S, A, B\}$.

Шаг 2, 3. Множество P': 1) $S \to AB \mid A \mid B$; 2) $A \to 0A \mid 0$; 3) $B \to 1B \mid 1$.

Шаг 4. Т.к. $S \in N$, то введем новый нетерминал C и пополним множество P' правилом вида $C \to S \mid \varepsilon$. Результирующая грамматика будет иметь вид:

$G' = (\{0,1\}, \{S, A, B, C\}, P', C)$ с правилами P': 1) $C \to S \mid \varepsilon$; 2) $S \to AB \mid A \mid B$; 3) $A \to 0A \mid 0$; 4) $B \to 1B \mid 1$.

3.2.4 Устранение цепных правил

Алгоритм 3.5. Устранение цепных правил.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V'_N, P', S')$ без цепных правил, т.е. правил вида $A \to B$, где $A, B \in V_N$.

Шаг 1. Для каждого нетерминала A вычислить множество выводимых из него нетерминалов, т.е. множество $N^A = \{B \mid A \Rightarrow^* B, \text{ где } B \in V_N\}$.

1.1 Положить $N_0^A = \{A\}$.

1.2 Вычислить $N_i^A = N_{i-1}^A \cup \{C \mid (B \to C) \in P, B \in N_{i-1}^A, C \in V_N\}$.

1.3 Если $N_i^A \neq N_{i-1}^A$, то положить i:=i+1 и перейти к пункту 1.2, иначе считать $N^A = N_i^A$.

Шаг 2. Построить множество P' так: если $(B \to \alpha) \in P$ не является цепным правилом $(\alpha \notin V_N)$, то включить в P' правило $A \to \alpha$ для каждого A, такого, что $B \in N^A$.

Конец алгоритма.

Пример 3.8. Грамматика $G = (\{+, n\}, \{L, M, N\}, P, L)$ с правилами P: 1) $L \to M$; 2) $M \to N$; 3) $N \to N+ \mid n$. Преобразуем ее в эквивалентную грамматику G' по алгоритму 3.5.

Шаг 1. $N_0^L = \{L\}$;

$N_1^L = \{L, M\}$;

$N_2^L = \{L, M, N\}$;

$N_3^L = \{L, M, N\}$.

Т.к. $N_2^L = N_3^L$, то $N^L = \{L, M, N\}$.

$N_0^M = \{M\}$;

$N_1^M = \{M, N\}$;

$N_2^M = \{M, N\}$.

Т.к. $N_1^M = N_2^M$, то $N^M = \{M, N\}$.

$N_0^N = \{N\}$;

$N_1^N = \{N\}$.

Т.к. $N_1^N = N_0^N$, то $N^N = \{N\}$.

Шаг 2. Преобразовав правила вывода грамматики, получим грамматику $G' = (\{+, n\}, \{L, M, N\}, P', L)$ с правилами P':

$$1)\ L \to N+ \mid n; \quad 2)\ M \to N+ \mid n; \quad 3)\ N \to N+ \mid n.$$

3.2.5 Устранение левой факторизации правил

Алгоритм 3.6. Устранение левой факторизации правил.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V_N', P', S')$ без одинаковых префиксов в правых частях правил, определяющих нетерминалы.

Шаг 1. Записать все правила для нетерминала X, имеющие одинаковые префиксы $\alpha \in V^*$, в виде одного правила с альтернативами: $X \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n;\ \beta_1, \beta_2, \ldots, \beta_n \in V^*$.

Шаг 2. Вынести за скобки влево префикс α в каждой строке-альтернативе: $X \to \alpha(\beta_1 \mid \beta_2 \mid \ldots \mid \beta_n)$.

Шаг 3. Обозначить новым нетерминалом Y выражение, оставшееся в скобках: $X \to \alpha Y, Y \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$.

Шаг 4. Пополнить множество нетерминалов новым нетерминалом Y и заменить правила, подвергшиеся факторизации, новыми правилами для X и Y.

Шаг 5. Повторить шаги 1-4 для всех нетерминалов грамматики, для которых это возможно и необходимо.

Конец алгоритма.

Пример 3.9. Дана грамматика $G = (\{k,l,m,n\}, \{S\}, P, S)$ с правилами P: $1)\ S \to kSl; \quad 2)\ S \to kSm; \quad 3)\ S \to n$. Преобразуем ее в эквивалентную грамматику G' по алгоритму 3.6:

Шаг 1. $S \to kSl \mid kSm \mid n$.

Шаг 2. $S \to kS(l \mid m) \mid n$.

Шаг 3,4. Пополнив множество нетерминалов новым нетерминалом C и заменив правила, подвергшиеся факторизации, получим грамматику $G' = (\{k,l,m,n\}, \{S, C\}, P', S)$ с правилами P': $1)\ S \to kSC; \quad 2)\ S \to n; \quad 3)\ C \to l; \quad 4)\ C \to m.$

3.2.6 Устранение прямой левой рекурсии

Алгоритм 3.7. Устранение прямой левой рекурсии.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V_N', P', S')$ без прямой левой рекурсии, т.е. без правил вида $A \to A\alpha, A \in V_N, \alpha \in V^*$.

Шаг 1. Вывести из грамматики все правила для рекурсивного нетерминала X:

$$\begin{aligned} &X \to X\alpha_1 \mid X\alpha_2 \mid \ldots \mid X\alpha_m \quad (X \in V_N; \alpha_1, \alpha_2, \ldots, \alpha_m \in V^*) \\ &X \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \quad (\beta_1, \beta_2, \ldots, \beta_n \in V^*). \end{aligned} \tag{3.8}$$

Шаг 2. Внести новый нетерминал Y так, чтобы он описывал любой «хвост» строки, порождаемой рекурсивным нетерминалом X:

$$\begin{aligned} &Y \to \alpha_1 Y \mid \alpha_2 Y \mid \ldots \mid \alpha_m Y \\ &Y \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m. \end{aligned} \tag{3.9}$$

Шаг 3. Заменить в рекурсивном правиле для X правую часть, используя новый нетерминал и все нерекурсивные правила для X так, чтобы генерируемый язык не изменился:

$$\begin{aligned} &X \to \beta_1 Y \mid \beta_2 Y \mid \ldots \mid \beta_n Y \\ &X \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \\ &Y \to \alpha_1 Y \mid \alpha_2 Y \mid \ldots \mid \alpha_m Y \\ &Y \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m. \end{aligned} \tag{3.10}$$

Шаг 4. Пополнить множество нетерминалов грамматики новым нетерминалом Y. Пополнить множество правил грамматики правилами, полученными на шаге 3.

Шаг 5. Повторить действия шагов 1-4 для всех рекурсивных нетерминалов грамматики, после чего полученные множества нетерминалов и правил принять в качестве V_N' и P'.

Конец алгоритма.

Пример 3.10. Дана грамматика $G = (\{a, b, c, d, z\}, \{S, A, B, C\}, P, S)$ с правилами P: 1) $S \to Aa$; 2) $A \to Bb$; 3) $B \to Cc \mid d$; 4) $C \to Ccbz \mid dbz$. После устранения прямой левой рекурсии получим эквивалентную грамматику $G' = (\{a, b, c, d, z\}, \{S, A, B, C, Z\}, P', S)$ с правилами P':

1) $S \to Aa$; 2) $A \to Bb$; 3) $B \to Cc \mid d$; 4) $C \to dbzZ \mid dbz$; 5) $Z \to cbzZ \mid cbz$.

3.3 Автомат с магазинной памятью

КС-языки можно распознавать с помощью автомата с магазинной памятью (МП-автомата).

3.3.1 Общая схема функционирования МП-автомата

Определение 3.8. МП-автомат можно представить в виде семерки:

$$M = (Q, T, N, F, q_0, N_0, Z), \tag{3.11}$$

где Q – конечное множество состояний автомата;
T – конечный входной алфавит;
N – конечный магазинный алфавит;
F – магазинная функция, отображающая множество $(Q \times (T \cup \{\varepsilon\}) \times N)$ во множество всех подмножеств множества $Q \times N^*$, т.е. $F: (Q \times (T \cup \{\varepsilon\}) \times N) \to P(Q \times N^*)$;
q_0 – начальное состояние автомата, $q_0 \in Q$;
N_0– начальный символ магазина, $N_0 \in N$;
Z – множество заключительных состояний автомата, $Z \subseteq Q$.

Определение 3.9. Конфигурацией МП-автомата называется тройка вида:

$$(q, \omega, \alpha) \in (Q \times T^* \times N^*), \tag{3.12}$$

где q - текущее состояние автомата, $q \in Q$;
ω - часть входной строки, первый символ которой находится под входной головкой, $\omega \in T^*$;
α - содержимое магазина, $\alpha \in N^*$.
Общая схема МП-автомата представлена на рисунке 3.3.

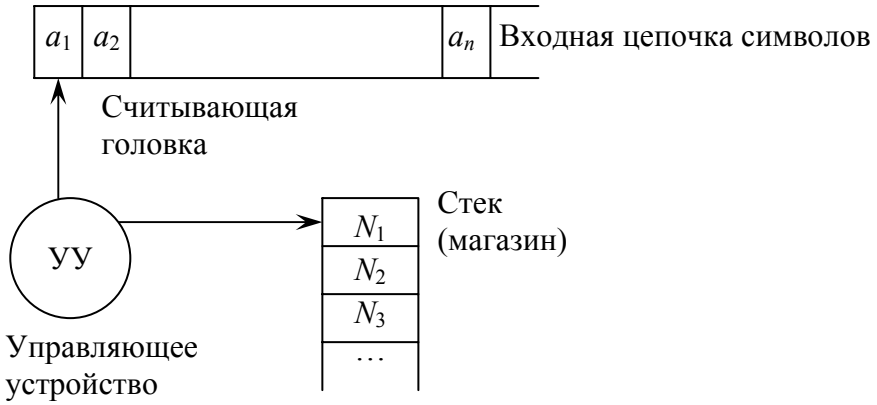


Рисунок 3.3 – Схема МП-автомата

Алгоритм 3.8. Функционирование МП-автомата.

Начальной конфигурацией МП-автомата является конфигурация (q_0, ω, N_0).

Шаг работы МП-автомата будем представлять в виде отношения непосредственного следования конфигураций (обозначается «|=») и отношения достижимости конфигураций (обозначается «|=*»). Если одним из значений магазинной функции $F(q \in Q, t \in (T \cup \{\varepsilon\}), S \in N)$ является $(q' \in Q, \gamma \in N^*)$, то записывается $(q, t\omega, S\alpha) \models (q', \omega, \gamma\alpha)$. При этом возможны следующие варианты.

1) Случай $t \in T$. Автомат находится в текущем состоянии q, читает входной символ t, имеет в вершине стека символ S. Он переходит в очередное состояние q', сдвигает входную головку на ячейку вправо и заменяет верхний символ S строкой γ магазинных символов. Вариант $\gamma = \varepsilon$ означает, что S удаляется из стека.

2) Случай $t = \varepsilon$. Отличается от первого случая тем, что входной символ t просто не принимается во внимание, и входная головка не сдвигается. Такой шаг работы МП-автомата называется ε-шагом, который может выполняться даже после завершения чтения входной строки.

Заключительной конфигурацией МП-автомата является конфигурация (q, ε, α), где $q \in Z$.

Определение 3.10. МП-автомат допускает входную стоку ω, если существует путь по конфигурациям $(q_0, \omega, N_0) \models *(q, \varepsilon, \alpha)$ для некоторых $q \in Z$ и $\alpha \in N^*$.

Определение 3.11. Язык L, распознаваемый (принимаемый) МП-автоматом M, определяется как множество вида:

$$L(M) = \{\omega \mid \omega \in T^* \text{ и } (q_0, \omega, N_0) \models *(q, \varepsilon, \alpha) \text{ для некоторых } q \in Z \text{ и } \alpha \in N^*\}.$$

Определение 3.12. МП-автомат с магазинной функцией $F : (Q \times (T \cup \{\varepsilon\}) \times N^*) \to P(Q \times N^*)$ называется расширенным МП-автоматом, т.е. автоматом, который может заменять цепочку символов конечной длины в верхушке стека на другую цепочку символов конечной длины.

3.3.2 Взаимосвязь МП-автоматов и КС-грамматик

В теории автоматов доказано, что существуют КС-грамматики, МП-автоматы и расширенные МП-автоматы, определяющие один и тот же язык.

Алгоритм 3.9. Построение МП-автомата по КС-грамматике.

Построим МП-автомат, выполняющий левосторонний разбор. Данный автомат обладает только одним состоянием и принимает входную строку опустошением магазина. Стек используется для размещения текущей сентенции, первоначально это начальный символ грамматики. Очередная сентенция получается заменой верхнего нетерминала стека.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: МП-автомат $M = (Q, T, N, F, q_0, N_0, Z)$ такой, что $L(M) = L(G)$.

Шаг 1. Положить $Q = \{q\}$, $q_0 = q$, $Z = \varnothing$, $N = V_T \cup V_N$, $T = V_T$, $N_0 = S$.

Шаг 2. Для каждого правила вида $(A \to \beta) \in P$, где $\beta \in V^*$, сформировать магазинную функцию вида $F(q, \varepsilon, A) = (q, \beta)$. Эти функции предписывают замещать нетерминал в вершине стека по правилу грамматики.

Шаг 3. Для каждого $t \in V_T$ сформировать магазинную функцию вида $F(q, t, t) = (q, \varepsilon)$, которая выталкивает из стека символ, совпадающий с входным, и перемещает читающую головку. Эти функции обеспечивают опустошение стека.

Конец алгоритма.

Пример 3.11. Дана КС-грамматика:

$G(\{+, (,), a\}, \{S, A\}, \{S \to S+A \mid A, A \to (S) \mid a\}, \{S\})$. Последовательность построения МП-автомата будет иметь вид.

1) $Q = \{q\}$, $q_0 = q$, $T = \{+, (,), a\}$, $N = \{+, (,), a, S, A\}$, $N_0 = S$, $Z = \varnothing$.

2) $F(q, \varepsilon, S) = (q, S+A)$, $F(q, \varepsilon, S) = (q, A)$, $F(q, \varepsilon, A) = (q, (S))$; $F(q, \varepsilon, A) = (q, a)$.

3) $F(q, t, t) = (q, \varepsilon)$ для каждого $t \in \{+, (,), a\}$.

Распознавание строки (a) построенным МП-автоматом представлено в таблице 3.1. Полученный МП-автомат является недетерминированным.

Таблица 3.1 – Распознавание МП-автоматом строки (a)

Номер конфигурации	Текущее состояние	Входная строка	Содержимое магазина
1	q	(a)	S
2	q	(a)	A
3	q	(a)	(S)
4	q	$a)$	$S)$
5	q	$a)$	$A)$
6	q	$a)$	$a)$
7	q	$)$	$)$
8	q	ε	ε

Алгоритм 3.10. Построение расширенного МП-автомата по КС-грамматике.

Построим МП-автомат, выполняющий правосторонний разбор. Данный автомат имеет единственное текущее состояние и одно заключительное состояние, в котором стек пуст. Стек содержит левую часть текущей сентенции. Первоначально в стек помещается специальный магазинный символ, маркер пустого стека #. На каждом шаге автомат по правилу грамматики замещает нетерми-

налом строку верхних символов стека или дописывает в вершину входной символ.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: расширенный МП-автомат $M = (Q, T, N, F, q_0, N_0, Z)$ такой, что $L(M) = L(G)$.

Шаг 1. Положить $Q = \{q, r\}$, $q_0 = q$, $Z = \{r\}$, $N = V_T \cup V_N \cup \{\#\}$, $T = V_T$, $N_0 = \#$.

Шаг 2. Для каждого правила вида $(A \to \beta) \in P$, где $\beta \in V^*$, сформировать магазинную функцию вида $F(q, \varepsilon, \beta) = (q, A)$, предписывающую заменять правую часть правила в вершине стека нетерминалом из левой части, независимо от текущего символа входной строки.

Шаг 3. Для каждого терминала $t \in T$ сформировать магазинную функцию вида $F(q, t, \varepsilon) = (q, t)$, которая помещает символ входной строки в вершину стека, если там нет правой части правила, и перемещает читающую головку.

Шаг 4. Предусмотреть магазинную функцию для перевода автомата в заключительное состояние $F(q, \varepsilon, \#S) = (r, \varepsilon)$.

Конец алгоритма.

Пример 3.12. Для грамматики из примера 3.11 построить расширенный МП-автомат. Последовательность построения МП-автомата будет иметь вид.

1) $Q = \{q, r\}$, $q_0 = q$, $T = \{+, (,), a\}$, $N = \{+, (,), a, S, A, \#\}$, $N_0 = \#$, $Z = r$.

2) $F(q, \varepsilon, S{+}A) = (q, S)$, $F(q, \varepsilon, A) = (q, S)$, $F(q, \varepsilon, (S)) = (q, A)$, $F(q, \varepsilon, a) = (q, A)$.

3) $F(q, t, \varepsilon) = (q, t)$ для каждого $t \in \{+, (,), a\}$.

4) $F(q, \varepsilon, \#S) = (r, \varepsilon)$.

Распознавание строки (a) расширенным МП-автоматом представлено в таблице 3.2. Полученный МП-автомат является детерминированным.

Таблица 3.2 – Распознавание расширенным МП-автоматом строки (a)

Номер конфигурации	Текущее состояние	Входная строка	Содержимое магазина
1	q	(a)	#
2	q	$a)$	#(
3	q	$)$	#(a
4	q	$)$	#(A
5	q	$)$	#(S
6	q	ε	#(S)
7	q	ε	#A
8	q	ε	#S
9	r	ε	ε

3.4 Нисходящие распознаватели КС-языков

3.4.1 Рекурсивный спуск

В основе метода рекурсивного спуска лежит левосторонний разбор строки языка. Исходной сентенциальной формой является начальный символ грамматики, а целевой – заданная строка языка. На каждом шаге разбора правило грамматики применяется к самому левому нетерминалу сентенции. Данный процесс соответствует построению дерева разбора цепочки сверху вниз (от корня к листьям).

Метод рекурсивного спуска реализует разбор цепочки сверху вниз следующим образом. Для каждого нетерминального символа грамматики создается своя процедура, носящая его имя. Задача этой процедуры – начиная с указанного места исходной цепочки, найти подцепочку, которая выводится из этого нетерминала. Если такую подцепочку считать не удается, то процедура завершает свою работу вызовом процедуры обработки ошибок, которая выдает сообщение о том, что цепочка не принадлежит языку грамматики и останавливает разбор. Если подцепочку удалось найти, то работа процедуры считается нормально завершенной и осуществляется возврат в точку вызова. Тело каждой такой процедуры составляется непосредственно по правилам вывода соответствующего нетерминала, при этом терминалы распознаются самой процедурой, а нетерминалам соответствуют вызовы процедур, носящих их имена.

Пример 3.13. Дана грамматика $G(\{a, b, c, \perp\}, \{S, A, B\}, P, S)$ с правилами P: 1) $S \to AB\perp$; 2) $A \to a$; 3) $A \to cA$; 4) $B \to bA$. Требуется выполнить анализ строки $cabca\perp$.

Левосторонний вывод цепочки имеет вид:

$$S \Rightarrow AB\perp \Rightarrow cAB\perp \Rightarrow caB\perp \Rightarrow cabA\perp \Rightarrow cabcA\perp \Rightarrow cabca\perp.$$

Нисходящее дерево разбора цепочки представлено на рисунке 3.4.

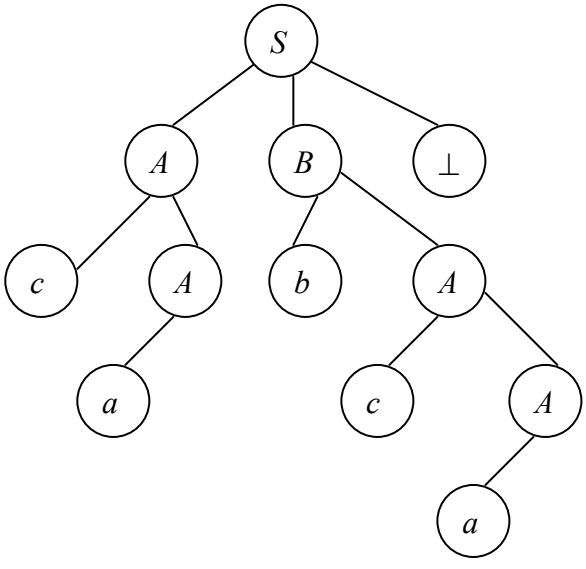


Рисунок 3.4 – Дерево нисходящего разбора цепочки $cabca\perp$

Построим распознаватель для грамматики G методом рекурсивного спуска.

Введем следующие обозначения:

1) *CH* – текущий символ исходной строки;

2) *gc* – процедура считывания очередного символа исходной строки в переменную *CH*;

3) *Err* - процедура обработки ошибок, возвращающая по коду соответствующее сообщение об ошибке.

С учетом введенных обозначений, процедуры разбора будут иметь вид.

```
procedure S;
begin
  A; B;
  if CH<>'⊥' then ERR
end;
```

```
procedure A;
begin
  if CH='a' then gc
  else if CH='c'
    then begin
           gc; A
         end
    else Err
end;
```

```
procedure B;
begin
  if CH= 'b' then
    begin
      gc; A
    end
  else Err
end;
```

Теорема 3.1. Достаточные условия применимости метода рекурсивного спуска.

Метод рекурсивного спуска применим к грамматике, если правила вывода грамматики имеют один из следующих видов:

1) $A\to\alpha$, где $\alpha\in(V_T\cup V_N)^*$, и это единственное правило вывода для этого нетерминала;

2) $A\to a_1\alpha_1 \mid a_2\alpha_2 \mid ... \mid a_n\alpha_n$, где $a_i \in V_T$ для каждого i=1, 2,…, n; $a_i\neq a_j$ для $i\neq j$, $\alpha_i\in(V_T\cup V_N)^*$, т.е. если для нетерминала A несколько правил вывода, то они должны начинаться с терминалов, причем эти терминалы должны быть различными.

Данные требования являются достаточными, но не являются необходимыми. Можно применить эквивалентные преобразования КС-грамматик, которые способствуют приведению грамматики к требуемому виду, но не гарантируют его достижения (см. пп.3.2).

При описании синтаксиса языков программирования часто встречаются правила, которые задают последовательность однотипных конструкций, отделенных друг от друга каким-либо разделителем. Общий вид таких правил:

$$L\to a \mid a,L \text{ или в сокращенной форме } L\to a\{,a\}.$$

Формально здесь не выполняются условия метода рекурсивного спуска, т.к. две альтернативы начинаются одинаковыми терминальными символами. Но если принять соглашения, что в подобных ситуациях выбирается самая длинная подцепочка, выводимая из нетерминала L, то разбор становится детерминиро-

ванным, и метод рекурсивного спуска будет применим к данному правилу грамматики. Соответствующая правилу процедура будет иметь вид:

```
procedure L;
begin
  if CH<>'a' then Err else gc;
  while CH=',' do
    begin
      gc;
      if CH<>'a' then Err else gc
    end
end;
```

3.4.2 Распознаватели *LL*(*k*)-грамматик

Определение 3.12. КС-грамматика обладает свойством $LL(k)$ для некоторого целого $k>0$, если на каждом шаге вывода для однозначного выбора очередной альтернативы МП-автомату достаточно знать символ на верхушке стека и рассмотреть первые k символов от текущего положения считывающей головки во входной строке.

Определение 3.13. КС-грамматика называется $LL(k)$-грамматикой, если она обладает свойством $LL(k)$ для некоторого целого $k>0$.

В основе распознавателя $LL(k)$-грамматик лежит левосторонний разбор строки языка. Исходной сентенциальной формой является начальный символ грамматики, а целевой – заданная строка языка. На каждом шаге разбора правило грамматики применяется к самому левому нетерминалу сентенции. Данный процесс соответствует построению дерева разбора цепочки сверху вниз (от корня к листьям). Отсюда и произошла аббревиатура $LL(k)$: первая «*L*» (от слова «*left*») означает левосторонний ввод исходной цепочки символов, вторая «*L*» - левосторонний вывод в процессе работы распознавателя.

Определение 3.14. Для построения распознавателей для $LL(k)$-грамматик используются два вспомогательных множества:

- $FIRST(k, \alpha)$ – множество терминальных цепочек, выводимых из цепочки $\alpha \in (V_T \cup V_N)^*$, укороченных до k символов;

- $FOLLOW(k, A)$ – множество укороченных до k символов терминальных цепочек, которые могут следовать непосредственно за $A \in V_N$ в цепочках вывода.

Формально эти множества можно определить следующим образом:

- $FIRST(k, \alpha) = \{\omega \in V_T^* \mid \exists$ вывод $\alpha \Rightarrow^* \omega$ и $|\omega| \le k$ или $\exists$ вывод $\alpha \Rightarrow^* \omega x$ и $|\omega| = k$; x, $\alpha \in (V_T \cup V_N)^*$, $k > 0\}$;

- $FOLLOW(k, A) = \{\omega \in V_T^* \mid \exists$ вывод $S \Rightarrow^* \alpha A \gamma$ и $\omega \in FIRST(k, \gamma)$; $\alpha, \gamma \in V^*$, $A \in V_N$, $k>0\}$.

Рассмотрим класс $LL(k)$-грамматик на примере $LL(1)$-грамматик.

Теорема 3.2. Необходимое и достаточное условие *LL*(1)-грамматики.

Для того чтобы грамматика $G(V_N, V_T, P, S)$ была $LL(1)$-грамматикой необходимо и достаточно, чтобы для каждого символа $A \in V_N$, у которого в грамматике существует более одного правила вида $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, выполнялось требование:

$$FIRST(1,\alpha_i FOLLOW(1,A)) \cap FIRST(1,\alpha_j FOLLOW(1,A)) = \varnothing, \quad \forall\, i \neq j,\ 0 < i \le n,\ 0 < j \le n. \tag{3.13}$$

Т.е. если для символа A отсутствует правило вида $A \to \varepsilon$, то все множества $FIRST(1, \alpha_1)$, $FIRST(1, \alpha_2)$,…, $FIRST(1, \alpha_n)$ должны попарно не пересекаться, если же присутствует правило $A \to \varepsilon$, то они не должны также пересекаться с множеством $FOLLOW(1, A)$.

Для построения распознавателей для $LL(1)$-грамматик необходимо построить множества $FIRST(1, x)$ и $FOLLOW(1, A)$. Причем, если строка x будет начинаться с терминального символа a, то $FIRST(1, x)=a$, и если она будет начинаться с нетерминального символа A, то $FIRST(1, x)=FIRST(1, A)$. Следовательно, достаточно рассмотреть алгоритмы построения множеств $FIRST(1, A)$ и $FOLLOW(1, A)$ для каждого нетерминального символа A.

Алгоритм 3.11. Построение множества *FIRST*(1, *A*).

Для выполнения алгоритма необходимо предварительно преобразовать исходную грамматику G в грамматику G', не содержащую ε-правил (см. алгоритм 3.4). Алгоритм построения множества $FIRST(1, A)$ использует грамматику G'.

Шаг 1. Первоначально внести во множество первых символов для каждого нетерминального символа A все символы, стоящие в начале правых частей правил для этого нетерминала, т.е.

$$\forall A \in V_N\ FIRST_0(1,A) = \{X \mid A \to X\alpha \in P,\ X \in (V_T \cup V_N),\ \alpha \in (V_T \cup V_N)^*\} \tag{3.14}$$

Шаг 2. Для всех $A \in V_N$ положить:

$$FIRST_{i+1}(1,A) = FIRST_i(1,A) \cup FIRST_i(1,B), \forall B \in (FIRST_i(1,A) \cap V_N) \tag{3.15}$$

Шаг 3. Если существует $A \in V_N$, такой что $FIRST_{i+1}(1, A) \neq FIRST_i(1, A)$, то присвоить $i:=i+1$ и вернуться к шагу 2, иначе перейти к шагу 4.

Шаг 4. Исключить из построенных множеств все нетерминальные символы, т.е.

$$\forall\ A \in V_N\ FIRST(1, A) = FIRST_i(1, A)_{\setminus N} \tag{3.16}$$

Конец алгоритма.

Алгоритм 3.12. Построение множества *FOLLOW*(1, *A*).

Алгоритм основан на использовании правил вывода грамматики *G*.

Шаг 1. Первоначально внести во множество последующих символов для каждого нетерминального символа *A* все символы, которые в правых частях правил вывода встречаются непосредственно за символом *A*, т.е.

$$\forall A \in V_N \; FOLLOW_0(1, A) = \{X \mid \exists B \to \alpha A X \beta \in P, B \in V_N, \; X \in (V_T \cup V_N), \alpha, \beta \in (V_T \cup V_N)^*\}. \quad (3.17)$$

Шаг 2. Внести пустую строку во множество *FOLLOW*(1, *S*), т.е.

$$FOLLOW(1, S) = FOLLOW(1, S) \cup \{\varepsilon\}. \quad (3.18)$$

Шаг 3. Для всех $A \in V_N$ вычислить:

$$FOLLOW'_i(1,A) = FOLLOW_i(1,A) \cup FIRST(1,B), \quad \forall B \in (FOLLOW_i(1,A) \cap V_N). \quad (3.19)$$

Шаг 4. Для всех $A \in V_N$ положить:

$$FOLLOW''_i(1, A) = FOLLOW'_i(1, A) \cup FOLLOW'_i(1, B), \quad \forall B \in (FOLLOW'_i(1, A) \cap V_N), \text{ если } \exists \text{ правило } B \to \varepsilon. \quad (3.20)$$

Шаг 5. Для всех $A \in V_N$ определить:

$$FOLLOW_{i+1}(1, A) = FOLLOW''_i(1, A) \cup FOLLOW''_i(1, B), \quad \forall B \in V_N, \exists B \to \alpha A, \; \alpha \in (V_T \cup V_N)^*. \quad (3.21)$$

Шаг 6. Если существует $A \in V_N$ такой, что $FOLLOW_{i+1}(1,A) \neq FOLLOW_i(1,A)$, то положить $i := i+1$ и вернуться к шагу 3, иначе перейти к шагу 7.

Шаг 7. Исключить из построенных множеств все нетерминальные символы, т.е. $\forall A \in V_N \; FOLLOW(1, A) = FOLLOW_i(1, A)_{\setminus N}$.

Конец алгоритма.

Алгоритм 3.13. Функционирование распознавателя цепочек для *LL*(1)-грамматик.

Шаг 1. Помещаем в стек начальный символ грамматики *S*, а во входной буфер - исходную цепочку символов.

Шаг 2. До тех пор пока в стеке и во входном буфере останется только пустая строка ε либо будет обнаружена ошибка в алгоритме разбора, выполняем одно из следующих действий:

- если на верхушке стека находится нетерминальный символ *A* и очередной символ входной строки символ *a*, то выполняем операцию «свертка» по правилу $A \to x$ при условии, что $a \in FIRST(1, x)$, т.е. извлекаем из стека символ *A* и заносим в стек строку *x*, не меняя содержимого входного буфера;
- если на верхушке стека находится нетерминальный символ *A* и очередной символ входной строки символ *a*, то выполняем операцию «свертка» по правилу $A \to \varepsilon$ при условии, что $a \in FOLLOW(1, A)$, т.е. извлекаем из стека символ *A* и заносим в стек строку ε, не меняя содержимого входного буфера;

- если на верхушке стека находится терминальный символ a, совпадающий с очередным символом входной строки, то выполняем операцию «выброс», т.е. удаляем из стека и входного буфера данный терминальный символ;
- если содержимое стека и входного буфера пусто, то исходная строка прочитана полностью, и разбор завершен удачно;
- если ни одно из данных условий не выполнено, то цепочка не принадлежит заданному языку, и алгоритм завершает свою работу с ошибкой.

Пример 3.14. Дана грамматика G ($\{S, T, R\}$, $\{+, -, (,), a, b\}$, P, S), с правилами P: 1) $S \to TR$; 2) $R \to \varepsilon \mid +TR \mid -TR$; 3) $T \to (S) \mid a \mid b$. Построить распознаватель для строки $(a+(b-a))$ языка грамматики G.

Этап 1. Преобразуем грамматику G в грамматику G', не содержащую ε-правил:

$N_0 = \{R\}$;

$N_1 = \{R\}$, т.к. $N_0 = N_1$, то во множество P' войдут правила:

1) $S \to TR \mid T$; 2) $R \to +TR \mid +T \mid -TR \mid -T$; 3) $T \to (S) \mid a \mid b$.

Этап 2. Построение множеств $FIRST(1, A)$ для каждого нетерминала A представлено в таблице 3.3.

Таблица 3.3 – Построение множеств $FIRST(1, A)$

$FIRST_i(1, A)$	0	1	2	$FIRST(1, A)$
S	T	T, (, a, b	T, (, a, b	(, a, b
R	+, -	+, -	+, -	+, -
T	(, a, b	(, a, b	(, a, b	(, a, b

Этап 3. Построение множеств $FOLLOW(1, A)$ для каждого нетерминала A представлено в таблице 3.4.

Таблица 3.4 – Построение множеств $FOLLOW(1, A)$

Шаг	Нетерминалы	$FOLLOW_i(1, A)$	$FOLLOW_i'(1, A)$	$FOLLOW_i''(1, A)$
0	S	)	), ε	), ε
	R	$\varnothing$	$\varnothing$	$\varnothing$
	T	R	R, +, -	R, +, -
1	S	), ε	), ε	), ε
	R	), ε	), ε	), ε
	T	R, +, -	R, +, -	R, +, -,), ε
2	S	), ε	), ε	), ε
	R	), ε	), ε	), ε
	T	R, +, -,), ε	R, +, -,), ε	R, +, -,), ε
$FOLLOW(1, S)$		), ε		
$FOLLOW(1, R)$		), ε		
$FOLLOW(1, T)$		+, -,), ε		

Этап 4. Множества *FIRST*(1, *A*) и *FOLLOW*(1, *A*) для каждого нетерминала *A* сведены в таблицу 3.5.

Таблица 3.5 – Множества *FIRST*(1, *A*) и *FOLLOW*(1, *A*)

A	*FIRST*(1, *A*)	*FOLLOW*(1, *A*)
S	(, *a*, *b*	), ε
R	+, -	), ε
T	(, *a*, *b*	+, -,), ε

Грамматика *G* является *LL*(1)-грамматикой, т.к. для каждого нетерминала *A*, имеющего альтернативные выводы, множества *FIRST*(1, *A*) попарно не пересекаются, а для нетерминала *R* они также не пересекаются со множеством *FOLLOW*(1, *R*).

Шаг 5. Разбор строки (*a*+(*b*-*a*)) для грамматики *G* показан в таблице 3.6.

Таблица 3.6 - Разбор строки (*a*+(*b*-*a*)) для грамматики *G*

Стек	Входной буфер	Действие
S	(*a*+(*b*-*a*))	свертка $S \to TR$, т.к. ($\in$ *FIRST*(1, *TR*)
TR	(*a*+(*b*-*a*))	свертка $T \to (S)$, т.к. ($\in$ *FIRST*(1, (*S*))
(*S*)*R*	(*a*+(*b*-a))	выброс
S)*R*	*a*+(*b*-*a*))	свертка $S \to TR$, т.к. $a \in$ *FIRST*(1, *TR*)
TR)*R*	*a*+(*b*-*a*))	свертка $T \to a$, т.к. $a \in$ *FIRST*(1, *a*)
aR)*R*	*a*+(*b*-*a*))	выброс
R)*R*	+(*b*-*a*))	свертка $R \to +TR$, т.к. + $\in$ *FIRST*(1, *TR*)
+*TR*)*R*	+(*b*-*a*))	выброс
TR)*R*	(*b*-*a*))	свертка $T \to (S)$, т.к. ($\in$ *FIRST*(1, (*S*))
(*S*)*R*)*R*	(*b*-*a*))	выброс
S)*R*)*R*	*b*-*a*))	свертка $S \to TR$, т.к. $b \in$ *FIRST*(1, *TR*)
TR)*R*)*R*	*b*-*a*))	свертка $T \to b$, т.к. $b \in$ *FIRST*(1, *b*)
bR)*R*)*R*	*b*-*a*))	выброс
R)*R*)*R*	-*a*))	свертка $R \to -TR$, т.к. - $\in$ *FIRST*(1, -*TR*)
-*TR*)*R*)*R*	-*a*))	выброс
TR)*R*)*R*	*a*))	свертка $T \to a$, т.к. $a \in$ *FIRST*(1, *a*)
aR)*R*)*R*	*a*))	выброс
R)*R*)*R*	))	свертка $R \to \varepsilon$, т.к.) $\in$ *FOLLOW*(1, *R*)
)*R*)*R*	))	выброс
R)*R*	)	свертка $R \to \varepsilon$, т.к.) $\in$ *FOLLOW*(1, *R*)
)*R*	)	выброс
R	ε	свертка $R \to \varepsilon$, т.к. $\varepsilon \in$ *FOLLOW*(1, *R*)
ε	ε	строка принята полностью

Шаг 6. Получили следующую цепочку вывода:

$S \Rightarrow TR \Rightarrow (S)R \Rightarrow (TR)R \Rightarrow (aR)R \Rightarrow (a+TR)R \Rightarrow (a+(S)R)R \Rightarrow (a+(TR)R)R \Rightarrow$
$\Rightarrow (a+(bR)R)R \Rightarrow (a+(b-TR)R)R \Rightarrow (a+(b-aR)R)R \Rightarrow (a+(b-a)R)R \Rightarrow (a+(b-a))R \Rightarrow$
$\Rightarrow (a+(b-a))$.

Нисходящее дерево разбора цепочки представлено на рисунке 3.5.

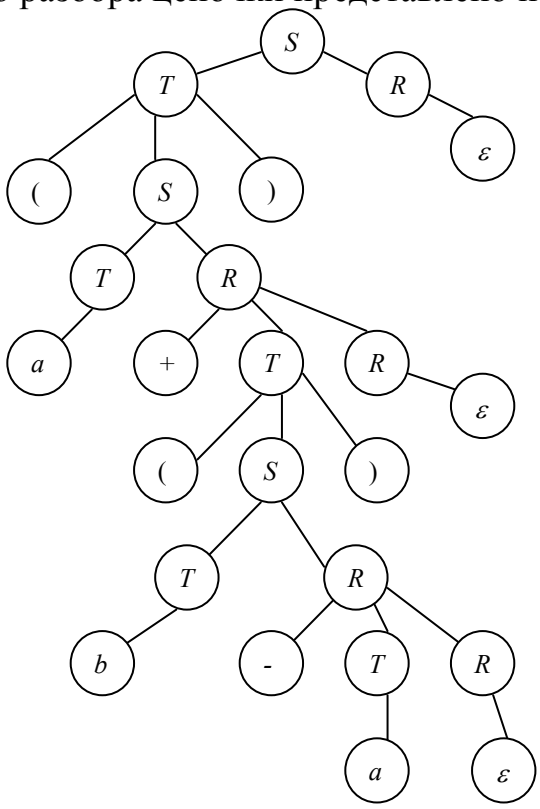


Рисунок 3.5 – Дерево вывода для цепочки (a+(b-a)) в грамматике G

3.5 Восходящие распознаватели языков

3.5.1 Распознаватели *LR*(*k*)-грамматик

Распознаватели $LR(k)$-грамматик основаны на построении дерева разбора снизу вверх и правостороннем выводе цепочек.

Определение 3.15. КС-грамматика обладает свойством $LR(k)$, $k \geq 0$, если на каждом шаге вывода для однозначного решения вопроса о выполняемом действии расширенному МП-автомату достаточно знать содержимое верхней части стека и рассмотреть первые k символов от текущего положения считывающей головки автомата во входной цепочке символов.

Определение 3.16. Грамматика называется $LR(k)$-грамматикой, если она обладает свойством $LR(k)$ для некоторого целого $k \geq 0$.

Первая литера «*L*» происходит от слова «*left*» и обозначает порядок чтения входной цепочки символов: слева - направо. Вторая литера «*R*» про-

исходит от слова «*right*» и означает, что в результате работы распознавателя получается правосторонний вывод.

Для того чтобы формально определить $LR(k)$ свойство для КС-грамматик, введем понятие пополненной КС-грамматики.

Определение 3.17. Грамматика G' является пополненной грамматикой, построенной на основании исходной грамматики $G(V_T, V_N, P, S)$, если выполняются следующие условия:

1) грамматика G' совпадает с грамматикой G, если начальный символ S не встречается нигде в правых частях правил грамматики G;

2) грамматика G' строится как грамматика $G(V_T, V_N \cup \{S'\}, P \cup \{S' \to S\}, S')$, если начальный символ S встречается в правой части хотя бы одного правила из множества P в исходной грамматике G.

Понятие «пополненной грамматики» введено для того, чтобы в процессе работы МП-автомата выполнение свертки к начальному символу пополненной грамматики S' служило сигналом к завершению алгоритма.

Дадим формальное определение $LR(k)$ свойства.

Определение 3.18. Если для произвольной КС-грамматики G в ее пополненной грамматике G' для двух произвольных цепочек вывода из условий:

1) $S' \Rightarrow^* \alpha A \omega \Rightarrow a \beta \omega$;
2) $S' \Rightarrow^* \gamma B x \Rightarrow \alpha \beta y$;
3) $FIRST(k, \omega) = FIRST(k, y)$;

следует, что $\alpha A \omega = \gamma B x$ (то есть $\alpha = \gamma$, $A = B$ и $\omega = y$), то доказано, что грамматика G обладает $LR(k)$ свойством. Очевидно, что тогда и пополненная грамматика G' также обладает $LR(k)$ свойством.

Распознаватель для $LR(k)$-грамматик функционирует на основе управляющей таблицы. Эта таблица состоит из двух частей, называемых «действия» и «переходы». По строкам таблицы распределены все цепочки символов на верхушке стека, которые могут приниматься во внимание в процессе работы распознавателя. По столбцам в области «действия» распределены все части входной цепочки символов длиной не более k (аванцепочки), которые могут следовать за считывающей головкой автомата в процессе выполнения разбора. В части «переходы» помещены все терминальные и нетерминальные символы грамматики, которые могут появляться на верхушке стека автомата при выполнении действий (сдвигов или сверток).

Клетки управляющей таблицы в части «действия» могут содержать один из следующих вариантов:

1) «сдвиг» - если в данной ситуации требуется выполнение сдвига (переноса текущего символа из входной цепочки в стек);

2) «успех» - если возможна свертка к начальному символу грамматики S и разбор входной цепочки завершен;

3) целое число («свертка») - если возможно выполнение свертки (число обозначает номер правила грамматики, по которому должна выполняться свертка);

4) «ошибка» - во всех других ситуациях.

Клетки управляющей таблицы в части «переходы» служат для выяснения номера строки таблицы, которая будет использована для определения выполняемого действия на очередном шаге. Эти клетки могут содержать следующие данные:

1) целое число - номер строки управляющей таблицы;

2) «ошибка» - во всех других ситуациях.

Для удобства работы распознаватель $LR(k)$-грамматики использует два специальных символа $\perp_н$ и $\perp_к$. Считается, что входная цепочка символов всегда начинается символом $\perp_н$ и завершается символом $\perp_к$.

Алгоритм 3.14. Функционирование распознавателя *LR*(*k*)-грамматики.

Шаг 1. Поместить в стек символ $\perp_н$ и начальную (нулевую) строку управляющей таблицы. В конец входной цепочки поместить символ $\perp_к$. Перейти к шагу 2.

Шаг 2. Прочитать с вершины стека номер строки управляющей таблицы. Выбрать из этой строки часть «действие» в соответствии с аванцепочкой, обозреваемой считывающей головкой автомата. Перейти к шагу 3.

Шаг 3. В соответствии с типом действия выполнить одну из четырех операций:

1) «сдвиг» - если входная цепочка не прочитана до конца, прочитать и запомнить как «новый символ» очередной символ из входной цепочки, сдвинуть считывающую головку на одну позицию вправо, иначе прервать выполнение алгоритма и сообщить об ошибке;

2) целое число («свертка») - выбрать правило вывода в соответствии с номером, удалить из стека цепочку символов, составляющую правую часть выбранного правила, взять символ из левой части правила и запомнить его как «новый символ»;

3) «ошибка» - прервать выполнение алгоритма, сообщить об ошибке;

4) «успех» - выполнить свертку к начальному символу S, завершить выполнение алгоритма, сообщить об успешном разборе входной цепочки символов, если входная цепочка прочитана до конца, иначе сообщить об ошибке.

Конец выбора. Перейти к шагу 4.

Шаг 4. Прочитать с вершины стека номер строки управляющей таблицы. Выбрать из этой строки часть «переход» в соответствии с символом, который был запомнен как «новый символ» на предыдущем шаге. Перейти к шагу 5.

Шаг 5. Если часть «переход» содержит вариант «ошибка», тогда прервать выполнение алгоритма и сообщить об ошибке, иначе положить в стек «новый символ» с выбранным номером строки управляющей таблицы. Вернуться к шагу 2.

Конец алгоритма.

Рассмотрим пример применения конкретной управляющей таблицы.

Пример 3.15. Дана КС-грамматика $G(\{a, b\}, \{S\}, \{S \to SaSb \mid \varepsilon\}, S)$. Пополненная грамматика для нее будет иметь вид $G(\{a, b\}, \{S, S'\}, \{S' \to S,$

$S \to SaSb \mid \varepsilon\}, S')$. Эта грамматика является $LR(1)$-грамматикой. Управляющая таблица для нее приведена в таблице 3.7.

Таблица 3.7 - Управляющая таблица для $LR(1)$-грамматики

Номер	Стек	Действие			Переход		
		a	b	$\perp_к$	a	b	S
0	$\perp_н$	свертка, 3		свертка, 3			1
1	S	сдвиг		успех,1	2		
2	Sa	свертка, 3	свертка, 3				3
3	SaS	сдвиг	сдвиг		4	5	
4	$SaSa$	свертка, 3	свертка, 3				6
5	$SaSb$	свертка, 2		свертка, 2			
6	$SaSaS$	сдвиг	сдвиг		4	7	
7	$SaSaSb$	свертка, 2	свертка, 2				

Разбор цепочки *ababb* распознавателем показан в таблице 3.8.

Таблица 3.8 – Разбор цепочки *ababb*

Стек	Буфер	Действие
$\{\perp_н, 0\}$	$ababb\perp_к$	свертка $S \to \varepsilon$
$\{\perp_н, 0\}\{S, 1\}$	$ababb\perp_к$	сдвиг
$\{\perp_н, 0\}\{S, 1\}\{a, 2\}$	$babb\perp_к$	свертка $S \to \varepsilon$
$\{\perp_н, 0\}\{S, 1\}\{a, 2\}\{S, 3\}$	$babb\perp_к$	сдвиг
$\{\perp_н, 0\}\{S, 1\}\{a, 2\}\{S, 3\}\{b, 5\}$	$abb\perp_к$	свертка $S \to SaSb$
$\{\perp_н, 0\}\{S, 1\}$	$abb\perp_к$	сдвиг
$\{\perp_н, 0\}\{S, 1\}\{a, 2\}$	$bb\perp_к$	свертка $S \to \varepsilon$
$\{\perp_н, 0\}\{S, 1\}\{a, 2\}\{S, 3\}$	$bb\perp_к$	сдвиг
$\{\perp_н, 0\}\{S, 1\}\{a, 2\}\{S, 3\}\{b, 5\}$	$b\perp_к$	ошибка

На практике $LR(k)$-грамматики при $k>1$ не применяются. Доказано, что для любого языка, определяемого $LR(k)$-грамматикой, существует $LR(1)$-грамматика, определяющая тот же язык.

3.5.2 Грамматики простого предшествования

Определение 3.19. Приведенная КС-грамматика G (V_N, V_T, P, S) называется грамматикой простого предшествования, если выполняются следующие условия.

1) Для каждой упорядоченной пары терминальных и нетерминальных символов выполняется не более чем одно из трех отношений предшествования:

а) $B_i =\cdot B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \to xB_iB_jy \in P$, где $x, y \in V^*$;

б) $B_i <\cdot B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \to xB_iDy \in P$ и вывод $D \Rightarrow^* B_jz$, где $A, D \in V_N$, $x, y, z \in V^*$;

в) $B_i \cdot> B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \to xCB_jy$ и вывод $C \Rightarrow^* zB_i$ или существует правило $A \to xCDy \in P$ и вывод $C \Rightarrow^* zB_i$ и $D \Rightarrow^* B_jw$, где $A, C, D \in V_N$, $x, y, z, w \in V^*$.

2) Различные правила в грамматике имеют разные правые части.

Отношения $=\cdot$, $<\cdot$, $\cdot>$ называют отношениями простого предшествования для символов грамматики.

В основе распознавателя для грамматик простого предшествования лежит правосторонний разбор строки языка. Исходной сентенциальной формой является заданная строка языка, а целевой – начальный символ грамматики. На каждом шаге разбора в исходной цепочке символов пытаются выделить подцепочку, совпадающую с правой частью некоторого правила вывода грамматики, и заменить ее нетерминалом, стоящим в левой части этого правила. Данная операция называется сверткой к нетерминалу, а заменяемая подстрока – основой сентенции. Описанный процесс разбора соответствует построению дерева вывода цепочки снизу вверх (от листьев к корню).

Метод предшествования основан на том факте, что отношения между двумя соседними символами распознаваемой строки соответствуют трем следующим вариантам:

- $B_i =\cdot B_{i+1}$, если символы B_i и B_{i+1} принадлежат основе;
- $B_i <\cdot B_{i+1}$, если B_{i+1} – крайний левый символ некоторой основы;
- $B_i \cdot> B_{i+1}$, если B_i – крайний правый символ некоторой основы.

Алгоритм 3.15. Поиск основы сентенции грамматики.

Если грамматика является грамматикой простого предшествования, то для поиска основы каждой ее сентенции надо просматривать элементы сентенции слева направо и найти самую левую пару символов x_j и x_{j+1}, такую что $x_j \cdot> x_{j+1}$. Окончанием основы сентенции будет x_j. Далее просматривать элементы сентенции справа налево, начиная с символа x_j до тех пор, пока не будет найдена самая правая пара символов x_{i-1} и x_i, такая что $x_{i-1} <\cdot x_i$. Заголовком основы будет символ x_i. Таким образом, будет найдена основа сентенции, имеющая вид $x_i\, x_{i+1} \ldots x_{j-1}\, x_j$. Схема поиска основы сентенции грамматики представлена на рисунке 3.6.

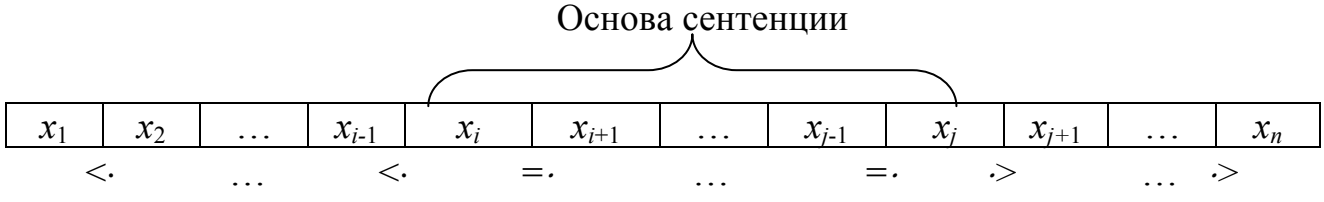


Рисунок 3.6 – Схема поиска основы сентенции грамматики

На основе отношений предшествования строят матрицу предшествования грамматики. Строки и столбцы матрицы предшествования помечаются символами грамматики. Пустые клетки матрицы указывают на то, что данные символы не связаны отношением предшествования.

Определение 3.20. Построение матрицы предшествования основано на двух вспомогательных множествах, определяемых следующим образом:

- $L(A) = \{X \mid \exists\ A \Rightarrow^* Xz\}$, $A \in V_N$, $X \in V$, $z \in V^*$ - множество крайних левых символов относительно нетерминального символа A;
- $R(A) = \{X \mid \exists\ A \Rightarrow^* zX\}$, $A \in V_N$, $X \in V$, $z \in V^*$ - множество крайних правых символов относительно нетерминального символа A.

Определение 3.21. Отношения предшествования можно определить с помощью введенных множеств следующим образом:

- $B_i =\cdot\ B_j$ ($\forall\ B_i,\ B_j \in V$), если и только если существует правило $A \to xB_iB_jy \in P$, где $A \in V_N$, $x, y \in V^*$;

- $B_i <\cdot\ B_j$ ($\forall\ B_i,\ B_j \in V$), если и только если существует правило $A \to xB_iDy \in P$ и $B_j \in L(D)$, где $A, D \in V_N$, $x, y \in V^*$;

- $B_i \cdot> B_j$ ($\forall\ B_i,\ B_j \in V$), если и только если существует правило $A \to xCB_jy$ и $B_i \in R(C)$ или существует правило $A \to xCDy \in P$ и $B_i \in R(C)$, $B_j \in L(D)$, где A, C, $D \in V_N$, $x, y \in V^*$.

Матрицу предшествования дополняют символами $\perp_н$ и $\perp_к$ (начало и конец цепочки). Для них определены следующие отношения предшествования:

- $\perp_н <\cdot\ X$, $\forall\ X \in V$, если $X \in L(S)$;
- $\perp_к \cdot> X$, $\forall\ X \in V$, если $X \in R(S)$.

Алгоритм 3.16. Построение множеств *L*(*A*) и *R*(*A*).

Шаг 1. Для каждого нетерминального символа A ищем все правила, содержание A в левой части. Во множество $L(A)$ включаем самый левый символ из правой части правил, а во множество $R(A)$ – самый крайний правый символ из правой части, т.е.

$$\forall\ A \in V_N\colon L_0(A) = \{X \mid A \to Xy, X \in V, y \in V^*\}, \qquad R_0(A) = \{X \mid A \to yX, X \in V, y \in V^*\}. \tag{3.22}$$

Шаг 2. Для каждого нетерминального символа A: если множество $L(A)$ содержит нетерминальные символы грамматики A', A'', …, то множество $L(A)$ надо дополнить символами, входящими в соответствующие множества $L(A')$,

$L(A'')$ и т.д., и не входящими в $L(A)$. Аналогичную операцию выполнить для множеств $R(A)$, т.е.

$$\forall A \in V_N: L_i(A) = L_{i-1}(A) \cup L_{i-1}(B), \forall B \in (L_{i-1}(A) \cap V_N), \quad (3.23)$$
$$R_i(A) = R_{i-1}(A) \cup R_{i-1}(B), \forall B \in (R_{i-1}(A) \cap V_N).$$

Шаг 3. Если на предыдущем шаге хотя бы одно множество $L(A)$ или $R(A)$ для некоторого символа грамматики изменилось, то вернуться к шагу 2, иначе построение закончено. Т.е. если существует $A \in V_N$: $R_i(A) \neq R_{i-1}(A)$ или $L_i(A) \neq L_{i-1}(A)$, то положить $i:=i+1$ и вернуться к шагу 2, иначе построение закончено и $R(A) = R_i(A)$ и $L(A) = L_i(A)$.

Конец алгоритма.

Алгоритм 3.17. Функционирование распознавателя для грамматик простого предшествования.

Шаг 1. Поместить в верхушку стека символ $\perp_н$, считывающую головку – в начало входной цепочки символов.

Шаг 2. До тех пор, пока не будет обнаружена ошибка, либо успешно завершен алгоритм разбора, сравниваем отношение простого предшествования символа на верхушке стека и очередного символа входной строки. При этом возможны следующие ситуации:

- если самый верхний символ стека имеет меньшее или равное предшествование, чем очередной символ входной строки, то производим операцию «сдвиг» (перенос текущего символа из входной цепочки в стек и перемещение считывающей головки на один символ вправо);
- если самый верхний символ стека имеет большее предшествование, чем очередной символ входной строки, то выполняем операцию «свертка». Для этого находим на верхушке стека «основу» сентенции, т.е. все символы, имеющие равное предшествование или один символ на верхушке стека. Символы основы удаляем из стека, выбираем правило вывода грамматики, имеющее правую часть, совпадающую с основой, и помещаем в стек левую часть выбранного правила. Если такого правила вывода найти не удалось, то выдается сообщение об ошибке, и разбор завершается неудачно;
- если не установлено ни одно отношение предшествования между текущим символом входной цепочки и самым верхним символом в стеке, то алгоритм прерывается сообщением об ошибке;
- если в стеке остаются символы $\perp_н S$, а во входном буфере только символ $\perp_к$, то входная строка прочитана полностью, и алгоритм разбора завершен успешно.

Пример 3.16. Дана грамматика $G(\{a, (,)\}, \{S, R\}, P, S)$, с правилами P:
1)$S \rightarrow (R|\ a$; 2)$R \rightarrow Sa)$. Построить распознаватель для строки $\perp_н(((aa)a)a)\perp_к$.

Этап 1. Построим множества крайних левых и крайних правых символов $L(A)$ и $R(A)$ относительно всех нетерминальных символов грамматики (таблица 3.9).

Таблица 3.9 – Построение множеств $L(A)$ и $R(A)$ для грамматики G

Шаг	$L_i(A)$	$R_i(A)$
0	$L_0(S)=\{(, a\}$ $L_0(R)=\{S\}$	$R_0(S)=\{R, a\}$ $R_0(R)=\{)\}$
1	$L_1(S)=\{(, a\}$ $L_1(R)=\{S, (, a\}$	$R_1(S)=\{R, a,)\}$ $R_1(R)=\{)\}$
2	$L_2(S)=\{(, a\}$ $L_2(R)=\{S, (, a\}$	$R_2(S)=\{R, a,)\}$ $R_2(R)=\{)\}$
Результат	$L(S)=\{(, a\}$ $L(R)=\{S, (, a\}$	$R(S)=\{R, a,)\}$ $R(R)=\{)\}$

Этап 2. На основе построенных множеств и правил вывода грамматики составим матрицу предшествования символов (таблица 3.10).

Поясним заполнение матрицы предшествования. В правиле грамматики $S\rightarrow(R$ символ (стоит слева от нетерминального символа R. Во множестве $L(R)$ входят символы S, (, a. Ставим знак $<\cdot$ в клетках матрицы, соответствующих этим символам, в строке для символа (.

В правиле грамматики $R\rightarrow Sa)$ символ a стоит справа от нетерминального символа S. Во множество $R(S)$ входят символы R, a,). Ставим знак $\cdot>$ в клетках матрицы, соответствующих этим символам, в столбце для символа a.

В строке символа $\perp_н$ ставим знак $<\cdot$ в клетках символов, входящих во множество $L(S)$, т.е. символов (, a. В столбце символа $\perp_к$ ставим знак $\cdot>$ в клетках, входящих во множество $R(S)$, т.е. символов R, a,).

В клетках, соответствующих строке символа S и столбцу символа a, ставим знак $=\cdot$, т.к. существует правило $R\rightarrow Sa)$, в котором эти символы стоят рядом. По тем же соображениям ставим знак $=\cdot$ в клетках строки a и столбца), а также строки (и столбца R.

Таблица 3.10 – Матрица предшествования символов грамматики

Символы	S	R	a	(	)	$\perp_к$
S			$=\cdot$			
R			$\cdot>$			$\cdot>$
a			$\cdot>$		$=\cdot$	$\cdot>$
(	$<\cdot$	$=\cdot$	$<\cdot$	$<\cdot$		
)			$\cdot>$			$\cdot>$
$\perp_н$			$<\cdot$	$<\cdot$		

Шаг 3. Функционирование распознавателя для цепочки (((*aa*)*a*)*a*) показано в таблице 3.11.

Таблица 3.11 – Алгоритм работы распознавателя цепочки (((*aa*)*a*)*a*)

Шаг	Стек	Входной буфер	Действие
1	$\perp_н$	(((*aa*)*a*)*a*)$\perp_к$	сдвиг
2	$\perp_н$(	((*aa*)*a*)*a*)$\perp_к$	сдвиг
3	$\perp_н$((	(*aa*)*a*)*a*)$\perp_к$	сдвиг
4	$\perp_н$(((	*aa*)*a*)*a*)$\perp_к$	сдвиг
5	$\perp_н$(((*a*	*a*)*a*)*a*)$\perp_к$	свертка $S \to a$
6	$\perp_н$(((*S*	*a*)*a*)*a*)$\perp_к$	сдвиг
7	$\perp_н$(((*Sa*	)*a*)*a*)$\perp_к$	сдвиг
8	$\perp_н$(((*Sa*)	*a*)*a*)$\perp_к$	свертка $R \to Sa$)
9	$\perp_н$(((*R*	*a*)*a*)$\perp_к$	свертка $S \to (R$
10	$\perp_н$((*S*	*a*)*a*)$\perp_к$	сдвиг
11	$\perp_н$((*Sa*	)*a*)$\perp_к$	сдвиг
12	$\perp_н$((*Sa*)	*a*)$\perp_к$	свертка $R \to Sa$)
13	$\perp_н$((*R*	*a*)$\perp_к$	свертка $S \to (R$
14	$\perp_н$(*S*	*a*)$\perp_к$	сдвиг
15	$\perp_н$(*Sa*	)$\perp_к$	сдвиг
16	$\perp_н$(*Sa*)	$\perp_к$	свертка $R \to Sa$)
17	$\perp_н$(*R*	$\perp_к$	свертка $S \to (R$
18	$\perp_н S$	$\perp_к$	строка принята

Шаг 4. Получили следующую цепочку вывода:
$S \Rightarrow (R \Rightarrow (Sa) \Rightarrow ((Ra) \Rightarrow ((Sa)a) \Rightarrow (((Ra)a) \Rightarrow (((Sa)a)a) \Rightarrow (((aa)a)a)$.
Восходящее дерево вывода цепочки представлено на рисунке 3.7.

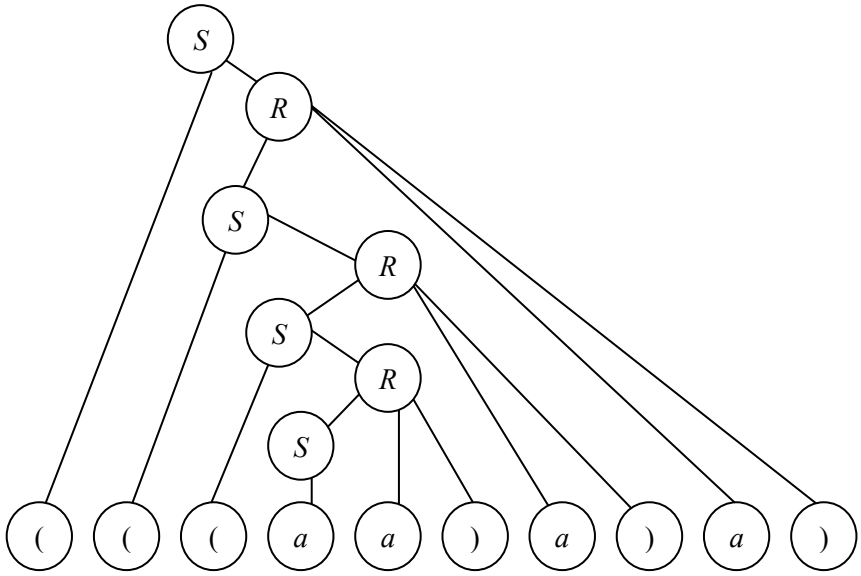


Рисунок 3.7 – Дерево вывода для цепочки (((*aa*)*a*)*a*) в грамматике *G*

3.6 Соотношение классов КС-грамматик и КС-языков

В общем случае можно выделить правоанализируемые КС-грамматики (ПГ), предполагающие построение левостороннего (восходящего) распознавателя, и левоанализируемые (ЛГ), использующие правосторонние (нисходящие) распознаватели.

Соотношение классов левоанализируемых и правоанализируемых КС-грамматик показано на рисунке 3.8.

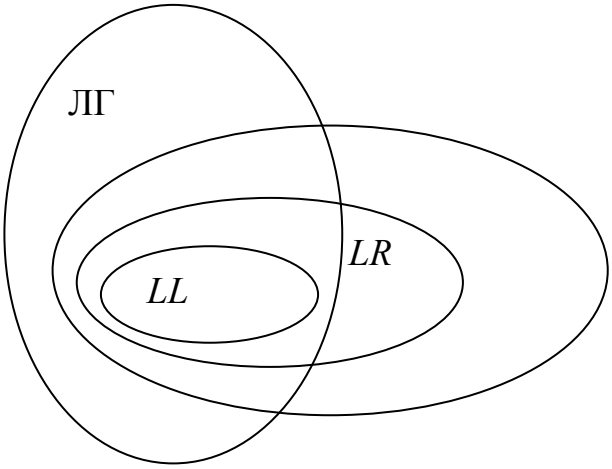


Рисунок 3.8 - Соотношение классов левоанализируемых и правоанализируемых КС-грамматик

Классы левоанализируемых и правоанализируемых грамматик являются несопоставимыми. Класс левоанализируемых *LL*-грамматик является собственным подмножеством класса *LR*-грамматик.

Так как проблема преобразования и эквивалентности КС-грамматик не имеет формализованного решения, то возможно задать один и тот же КС-язык двумя несопоставимыми грамматиками. Поэтому соотношение классов КС-языков не совпадает с соотношением задающих их классов КС-грамматик.

Соотношение классов КС-языков показано на рисунке 3.9.

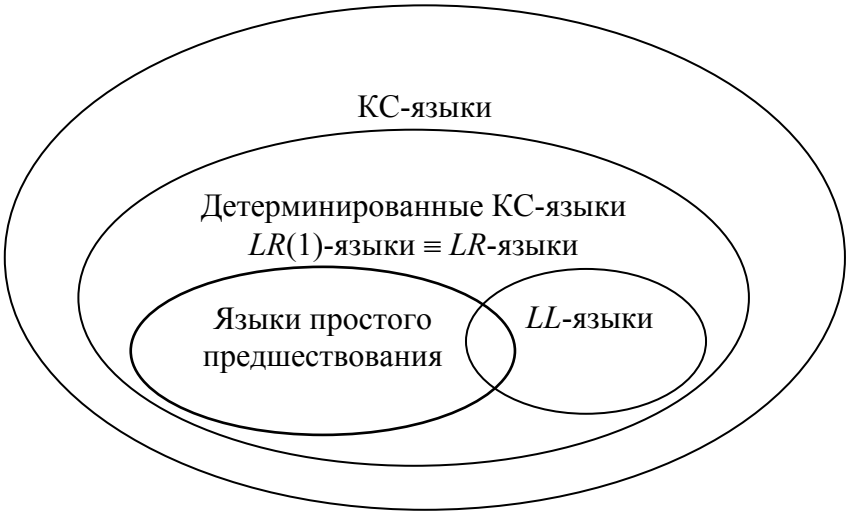


Рисунок 3.9 – Соотношение между различными классами КС-языков

Класс детерминированных КС-языков полностью совпадает с классом *LR*-языков и, более того, с классом *LR*(1)-языков. То есть, доказано, что для любого детерминированного КС-языка существует задающая его *LR*(1)-грамматика. Проблема состоит в том, что не всегда возможно найти такую грамматику, и нет формализованного алгоритма, как ее построить в общем случае.

Также *LL*-языки являются собственным подмножеством *LR*-языков. Языки простого предшествования, в свою очередь, также являются собственным подмножеством *LR*-языков. Языки простого предшествования и *LL*-языки несопоставимы между собой.

Вопросы и задания для самоконтроля по 3 разделу

1 Какой вывод цепочки называется правосторонним (левосторонним)?

2 Какое дерево называют деревом разбора строки?

3 Чем отличаются стратегии нисходящего и восходящего построения дерева разбора строки?

4 Запишите правила вывода, наличие которых приводит к неоднозначности грамматики.

5 Какими свойствами должна обладать приведенная КС-грамматика?

6 Как проверить существование языка грамматики?

7 Какие символы грамматики являются бесполезными и как от них избавиться?

8 Поясните методику удаления ε-правил грамматики.

9 Приведите примеры цепных правил и их исключения из грамматики.

10 Как избавиться от леворекурсивных правил грамматики?

11 В чем основная идея устранения левой факторизации правил грамматики?

12 Какими элементами формально задается МП-автомат?

13 Расскажите алгоритм функционирования МП-автомата.

14 Какой МП-автомат называется расширенным?

15 Поясните взаимосвязь между КС-грамматиками, МП-автоматами и расширенными МП-автоматами.

16 Поясните сущность метода рекурсивного спуска?

17 Запишите достаточные условия применимости метода рекурсивного спуска.

18 При каких условиях КС-грамматика обладает свойством *LL*(k)?

19 Как найти множества *FIRST*(1, A) и *FOLLOW*(1, A)?

20 Как функционирует распознаватель цепочек для *LL*(1)-грамматик?

21 Какая КС-грамматика называется *LR*(k)-грамматикой?

22 Каким образом используется управляющая таблица в работе распознавателя *LR*(k)-грамматики?

23 Какая приведенная КС-грамматика называется грамматикой простого предшествования?

24 Как строится матрица предшествования символов грамматики?

25 Поясните алгоритм функционирования распознавателя грамматик простого предшествования.

26 Почему соотношение классов КС-языков не совпадает с соотношением задающих их классов КС-грамматик? Подтвердите примерами.

27 Является ли однозначной грамматика с правилами вида: $S \to SbS \mid ScS \mid a$? Постройте возможные деревья вывода для цепочки *abaca*.

28 Укажите множество бесполезных символов в грамматике $G = (\{S, A, B, C\}, \{0, 1, 2\}, P, S)$, где P: $S \to SbS \mid ScS \mid a$; $A \to 0A1 \mid C \mid 0$; $B \to 0C \mid 1$; $C \to BC \mid CS$.

29 Выполнить эквивалентные преобразования КС-грамматики $G = (\{S, A, B, D, E\}, \{a, b, c, e\}, P, S)$, где P: 1) $S \to AB \mid \varepsilon$; 2) $A \to Aa \mid S \mid a$; 3) $B \to bD \mid bS \mid b$; 4) $D \to ccD$; 5) $E \to eE \mid e$.

30 Дана грамматика арифметических выражений $G(\{+, -, /, *, a, b\}, \{S, T, E\}, P, S)$, $P = \{S \to S+T \mid S-T \mid T,\ T \to T*E \mid T/E \mid E,\ E \to (S) \mid a \mid b\}$. Построить распознаватель и выполнить разбор строки вида $a*(a+b)-c$ методом:

а) рекурсивного спуска;

б) $LL(1)$-грамматик;

в) простого предшествования.

Тесты контроля усвоения материала по 3 разделу

1 В грамматике $G=(\{a, b, c, d\}, \{A, C\}, \{A \to aACd \mid b;\ C \to c \mid \varepsilon\}, A)$ левосторонним является вывод:

а) $A \Rightarrow aACd \Rightarrow abCd \Rightarrow abdd$;

б) $A \Rightarrow aACd \Rightarrow aAd \Rightarrow abd$;

в) $A \Rightarrow aACd \Rightarrow aAcd \Rightarrow abcd$;

г) $A \Rightarrow aACd \Rightarrow abCd \Rightarrow abcd$;

д) $A \Rightarrow aACd \Rightarrow aAcd \Rightarrow acd$.

2 В однозначной грамматике для каждой цепочки языка существует:

а) единственное правило вывода;

б) единственное дерево вывода;

в) единственная последовательность правил вывода;

г) единственная основа;

д) единственная свертка.

3 Грамматика является неоднозначной, если имеются правила вида ($A \in V_N$, $\alpha, \beta, \gamma \in V^*$):

а) $A \to A\alpha \mid \alpha$;

б) $A \to A\alpha A \mid A\beta$;

в) $A \to \alpha A \mid A\beta \mid \gamma$;

г) $A \to \alpha A \mid \alpha A\beta \mid \gamma$;

д) $A \to \alpha A \mid \beta$.

4 КС-грамматика, не имеющая циклов, ε-правил и бесполезных символов, называется:

а) приведенной;

б) однозначной;

в) автоматной;

г) нормированной;

д) регулярной.

5 Грамматика, для которой существует язык:

а) $S \to A \mid B$; $B \to aB \mid C$; $A \to AA \mid aA$; $C \to cC$;

б) $S \to A \mid B$; $B \to cB \mid A$; $A \to AA \mid aB$; $C \to cC \mid c$;

в) $S \to AB$; $B \to aB \mid bC$; $A \to AA \mid a$; $C \to cC$;

г) $S \to AB$; $B \to aB \mid b \mid C$; $A \to aAA$; $C \to cC \mid c$;

д) $S \to A \mid B$; $B \to aB \mid b \mid C$; $A \to AA \mid a$; $C \to cC$.

6 Множество бесполезных символов грамматики с правилами $P = \{S \to A|B$; $B \to AB \mid Ba$; $A \to aB \mid bS \mid b$; $C \to AS \mid b\}$:

а) $\{B\}$;

б) $\{C\}$;

в) $\{A, C\}$;

г) $\{B, C\}$;

д) $\{A, B\}$.

7 Символ C в грамматике с правилами $P = \{S \to SS \mid A; \quad B \to 0C \mid 1; A \to 0A1 \mid C \mid 0; C \to BC \mid CS\}$ является:

а) нетерминалом, не порождающим терминальных строк;

б) недостижимым нетерминалом;

в) недостижимым терминалом;

г) терминалом, не порождающим нетерминальных строк;

д) достижимым терминалом.

8 После удаления ε-правил, грамматика с правилами $P = \{S \to AB$; $A \to C|ab$; $C \to c \mid \varepsilon$; $B \to aAa\}$ примет вид:

а) $P = \{S \to AB \mid B \mid A$; $A \to C|ab$; $C \to c$; $B \to aAa \mid aa\}$;

б) $P = \{S \to AB \mid B$; $A \to ab$; $C \to c$; $B \to aAa \mid aa\}$;

в) $P = \{S \to AB \mid B$; $A \to C|ab$; $C \to c$; $B \to aAa\}$;

г) $P = \{S \to AB$; $A \to C|ab$; $C \to c$; $B \to aAa\}$;

д) $P = \{S \to AB \mid B$; $A \to C|ab$; $C \to c$; $B \to aAa \mid aa\}$.

9 Правила вида $A \to B$, где $A, B \in V_N$ называются:

а) леворекурсивными;

б) цепными;

в) факторизованными;

г) бесполезными;
д) праворекурсивными.

10 После устранения факторизации грамматика с правилами $P = \{S \to aA | b;$ $A \to abA \mid abAc \mid \varepsilon\}$ примет вид:
а) $S \to aA \mid b;\ A \to abAC;\ C \to c \mid \varepsilon;$
б) $S \to aA \mid b;\ A \to abAC \mid \varepsilon;\ C \to c;$
в) $S \to aA \mid b;\ A \to abAC \mid \varepsilon;\ C \to c \mid \varepsilon;$
г) $S \to aA | b;\ A \to c \mid abAc \mid \varepsilon;$
д) $S \to aA \mid b;\ A \to abAC \mid \varepsilon;\ C \to Ac \mid A.$

11 Магазинная функция МП-автомата $M = (Q, T, N, F, q_0, N_0, Z)$ задает отображение вида:
а) $F:\ (Q \times (T \cup \{\varepsilon\}) \times N) \to P(Q \times N^*);$
б) $F:\ (Q \times (T^* \cup \{\varepsilon\}) \times N) \to P(Q \times N^*);$
в) $F:\ (Q \times T \times N) \to P(Q \times N^*);$
г) $F:\ (Q \times (T \cup \{\varepsilon\}) \times Z) \to P(Q \times N^*);$
д) $F:\ (Q \times (T \cup \{\varepsilon\}) \times N^*) \to P(Q \times N^*).$

12 Заключительная конфигурация МП-автомата $M = (Q, T, N, F, q_0, N_0, Z)$:
а) $(q,\ \varepsilon,\ \alpha)$, где $q \in Q,\ \alpha \in T^*;$
б) $(q,\ \varepsilon,\ \alpha)$, где $q \in Z,\ \alpha \in N^*;$
в) $(q, Z,\ \alpha)$, где $q \in Q,\ \alpha \in N^*;$
г) $(q,\ \varepsilon,\ \alpha)$, где $q \in Z,\ \alpha \in T^*;$
д) $(\varepsilon, Z,\ \alpha)$, где $Z \in Q,\ \alpha \in N^*;$

13 МП-автомат $M = (Q, T, N, F, q_0, N_0, Z)$ допускает входную стоку ω, если существует путь по конфигурациям:
а) $(q_0, \omega, N_0) \mid\!= (q, \varepsilon, \alpha)$, где $q \in Z$ и $\alpha \in N^*;$
б) $(q, \omega, N_0) \mid\!= {}^*(q, \varepsilon, \alpha)$, где $q \in Z$ и $\alpha \in N^*;$
в) $(q_0, \omega, N_0) \mid\!= {}^*(q, \varepsilon, \alpha)$, где $q \in Z$ и $\alpha \in T^*;$
г) $(q_0, \varepsilon, N_0) \mid\!= {}^*(q, \omega, \alpha)$, где $q \in Z$ и $\alpha \in T^*$
д) $(q_0, \omega, N_0) \mid\!= {}^*(q, \varepsilon, \alpha)$, где $q \in Z$ и $\alpha \in N^*.$

14 Процедура *procedure B*; *begin if CH= 'b' then begin gc*; *B end else Err end*; реализует рекурсивный спуск для:
а) правила $B \to bB \mid b;$
б) правила $B \to bB;$

в) правила $B \to b$;

г) правила $B \to Bb$;

д) правила $B \to Bb \mid b$.

15 Достаточные условия применимости метода рекурсивного спуска выполняются в правиле:

а) $S \to aS \mid aB$;

б) $S \to Sa \mid aB$;

в) $S \to Sa \mid a$;

г) $S \to aS \mid bB$;

д) $S \to aS \mid a$.

16 Множество $\{\omega \in V_T^* \mid \exists$ вывод $\alpha \Rightarrow^* \omega$ и $|\omega| \le k$ или $\exists$ вывод $\alpha \Rightarrow^* \omega x$ и $|\omega| = k$; x, $\alpha \in (V_T \cup V_N)^*$, $k > 0\}$ формально определяет:

а) $FOLLOW(k, A)$;

б) $LL(k)$-свойство;

в) $L(\alpha)$;

г) $R(\alpha)$;

д) $FIRST(k, \alpha)$.

17 Формальное определение множества $FOLLOW(k, A)$:

а) $FOLLOW(k, A) = \{\omega \in V_T \mid \exists$ вывод $S \Rightarrow \alpha A \gamma$ и $\omega \in FIRST(k, \gamma)$; α, $\gamma \in V^*$, $A \in V_N$, $k > 0\}$;

б) $FOLLOW(k, A) = \{\omega \in V_T^* \mid \exists$ вывод $S \Rightarrow^* \alpha A \gamma$ и $\omega \in FIRST(k, \alpha)$; α, $\gamma \in V^*$, $A \in V_N$, $k > 0\}$;

в) $FOLLOW(k, A) = \{\omega \in V_T^* \mid \exists$ вывод $S \Rightarrow^* \alpha A \gamma$ и $\omega \in FIRST(k, \gamma)$; α, $\gamma \in V^*$, $A \in V_N$, $k > 0\}$;

г) $FOLLOW(k, A) = \{\omega \in V_T^* \mid \exists$ вывод $S \Rightarrow^* \alpha A \omega$ и $|\omega| \le k$; $\alpha \in V^*$, $A \in V_N$, $k > 0\}$;

д) $FOLLOW(k, A) = \{\omega \in V_T^* \mid \exists$ вывод $S \Rightarrow \alpha A \omega$ и $|\omega| \le k$; $\alpha \in V$, $A \in V_N$, $k > 0\}$.

18 В грамматике с правилами $P = \{S \to S{+}T \mid T;\ T \to a \mid S[S]\}$ справедливо:

а) $FIRST(T) = \{a\}$;

б) $FIRST(T) = \{S, T\}$;

в) $FIRST(T) = \{a, +\}$;

г) $FIRST(T) = \{S, T, a\}$;

д) $FIRST(T) = \{a, [,], +\}$.

19 Если на верхушке стека находится нетерминал A и очередной входной символ a, $a \in FIRST(1, x)$, то МП-автомат для $LL(1)$-грамматики выполняет операцию:

а) «свертка» по правилу $A \to x$;

б) «свертка» по правилу $A \to \varepsilon$;

в) сдвиг;
г) выброс;
д) ошибка.

20 Если в магазине находится строка «R)R», а во входном буфере – «)», то МП-автомат для $LL(1)$-грамматики выполняет действие:
а) свертка $R \to \varepsilon$, если «)» $\in FIRST(1, R)$;
б) свертка $R \to \varepsilon$, если «)» $\in FOLLOW(1, R)$;
в) свертка $R \to)R$, если «)» $\in FIRST(1, R)$;
г) выброс;
д) ошибка.

21 Распознаватели $LR(k)$-грамматик основаны на построении:
а) дерева разбора снизу вверх и левостороннем выводе цепочек;
б) дерева разбора сверху вниз и левостороннем выводе цепочек;
в) дерева разбора снизу вверх и правостороннем выводе цепочек;
г) дерева разбора сверху вниз и правостороннем выводе цепочек;
д) дерева разбора сверху вниз и комбинированном выводе цепочек.

22 Пополненная грамматика для $G(\{a, b\}, \{S\}, \{S \to AS \mid \varepsilon; A \to a \mid b\}, S)$ имеет вид:
а) $G'(\{a, b\}, \{S, S'\}, \{S' \to S; S \to AS \mid \varepsilon; A \to a \mid b\}, S)$;
б) $G'(\{a, b\}, \{S, S'\}, \{S \to S'; S' \to AS \mid \varepsilon; A \to a \mid b\}, S')$;
в) $G'(\{a, b\}, \{S, S'\}, \{S' \to S; S \to AS; A \to a \mid b\}, S')$;
г) $G'(\{a, b\}, \{S, S'\}, \{S \to S'; S \to AS' \mid \varepsilon; A \to a \mid b\}, S)$;
д) $G'(\{a, b\}, \{S, S'\}, \{S' \to S; S \to AS \mid \varepsilon; A \to a \mid b\}, S')$.

23 Если существует правило $A \to xB_iDy \in P$ и вывод $D \Rightarrow^* B_jz$, где $A, D \in V_N$, $x, y, z \in V^*$, то:
а) $B_i \cdot\!> B_j$ $(\forall\ B_i, B_j \in V)$;
б) $B_i =\!\cdot\ B_j$ $(\forall\ B_i, B_j \in V)$;
в) $B_i <\!\cdot\ B_j$ $(\forall\ B_i, B_j \in V_T)$;
г) $B_i <\!\cdot\ B_j$ $(\forall\ B_i, B_j \in V)$;
д) $B_i >\!\cdot\ B_j$ $(\forall\ B_i, B_j \in V_T)$.

24 Основой сентенции $x_1 <\!\cdot x_2 <\!\cdot\ x_3 \cdot\!> x_4 <\!\cdot x_5 =\!\cdot\ x_6 \cdot\!> x_7 \cdot\!> x_8$ является строка:
а) $x_5\,x_6\,x_7$;
б) $x_5\,x_6$;
в) x_3x_4;
г) $x_2x_3x_4$;
д) $x_5x_6x_7x_8$.

25 Формальное определение множества крайних левых символов относительно нетерминального символа A:

а) $L(A) = \{X \mid \exists\, A \Rightarrow^* Xz\}$, $A \in V_N$, $X \in V$, $z \in V^*$;
б) $L(A) = \{X \mid \exists\, A \Rightarrow Xz\}$, $A \in V_N$, $X \in V$, $z \in V^*$;
в) $L(A) = \{X \mid \exists\, A \Rightarrow^* zX\}$, $A \in V_N$, $X \in V$, $z \in V^*$;
г) $L(A) = \{X \mid \exists\, A \rightarrow Xz \in P\}$, $A \in V_N$, $X \in V$, $z \in V^*$;
д) $L(A) = \{X \mid \exists\, A \Rightarrow^* Xz\}$, $A \in V_N$, $X \in V_T$, $z \in V^*$.

26 Для отношения простого предшествования справедливо:
а) $\perp_н \cdot\!> X$, $\forall\, X \in V$, если $X \in L(S)$;
б) $\perp_н <\!\cdot\, X$, $\forall\, X \in V$, если $X \in L(S)$;
в) $\perp_н <\!\cdot\, X$, $\forall\, X \in V$, если $X \in R(S)$;
г) $\perp_н \cdot\!> X$, $\forall\, X \in V$, если $X \in R(S)$;
д) $\perp_н =\!\cdot\, X$, $\forall\, X \in V$, если $X \in L(S)$.

27 Для грамматики $G(\{t, i, ,\}, \{O, S, P\}, \{O \rightarrow t\ C;\ C \rightarrow i\ P \mid ,i\ P \mid i;\ P \rightarrow ,i\}, O\}$ справедливо:
а) $R(O) = \{\text{C}, P, i, t\}$;
б) $R(O) = \{t\}$;
в) $R(O) = \{C\}$;
г) $R(O) = \{P, i\}$;
д) $R(O) = \{\text{C}, P, i\}$.

28 Если символ в верхушке стека имеет меньшее или равное простое предшествование, чем очередной входной символ, то:
а) МП-автомат выполняет свертку;
б) МП-автомат выполняет выброс;
в) МП-автомат выполняет сдвиг;
г) МП-автомат выдает ошибку;
д) МП-автомат успешно завершает работу.

29 Порядок следования классов грамматик от частного к общему:
а) *LL*, *ПГ*, *LR*;
б) *LR*, *LL*, *ПГ*;
в) *ПГ*, *LL*, *LR*;
г) *ПГ*, *LR*, *LL*;
д) *LL*, *LR*, *ПГ*.

30 Порядок следования классов языков от общему к частному:
а) *LL*-языки, КС-языки, *LR*-языки;
б) *LR*-языки, *LL*-языки, КС-языки;
в) КС-языки, *LR*-языки, *LL*-языки;
г) *LR*-языки, КС-языки, *LL*-языки;
д) КС-языки, *LL*-языки, *LR*-языки.

4 Принципы построения трансляторов

4.1 Определение и виды трансляторов

Определение 4.1. Транслятор – это программа, которая переводит входную программу на исходном (входном) языке в эквивалентную ей выходную программу на результирующем (выходном) языке.

Результатом работы транслятора будет результирующая программа, но только в том случае, если текст исходной программы является правильным, т.е. не содержит ошибок с точки зрения синтаксиса и семантики входного языка. Если исходная программа неправильная, то результатом работы транслятора будет сообщение об ошибке (как правило, с дополнительными пояснениями и указанием места ошибки в исходной программе).

Определение 4.2. Компилятор – это транслятор, который осуществляет перевод исходной программы в эквивалентную ей объектную программу на языке машинных команд или на языке ассемблера.

Компилятор отличается от транслятора тем, что его результирующая программа всегда должна быть написана на языке машинных кодов или на языке ассемблера. Результирующая программа транслятора, в общем случае, может быть написана на любом языке - возможен, например, транслятор программ с языка *Pascal* на язык *C*. Примерами компилируемых языков программирования являются *Pascal*, *C*.

Определение 4.3. Ассемблер – компилятор, который переводит каждую команду исходной программы в одну машинную команду.

Определение 4.4. Интерпретатор - это программа, которая воспринимает входную программу на исходном языке и выполняет ее.

Интерпретатор, так же как и компилятор, анализирует текст исходной программы, но не порождает результирующей программы, а сразу же выполняет исходную в соответствии с ее смыслом, заданным семантикой входного языка. Таким образом, результатом работы интерпретатора будет результат, заданный смыслом исходной программы, в том случае, если эта программа правильная, или сообщение об ошибке, если исходная программа неверна. Примерами интерпретируемых языков программирования являются *Java Script*, *VB Script*.

4.2 Общая схема работы транслятора

В компиляторах и интерпретаторах протекают схожие процессы преобразования исходного текста. Несмотря на то, что взаимодействие этих процессов может быть организовано различным путем, можно выделить функции, выполнение которых приводит к одинаковым результатам. Учитывая схожесть процессов, рассмотрим фазы, существующие в компиляторе.

Основные функции компилятора:

1) проверка исходной цепочки символов на принадлежность к входному языку;

2) генерация выходной цепочки символов на языке машинных команд или ассемблере.

Процесс компиляции состоит из двух основных этапов: анализа и синтеза. На этапе анализа выполняется распознавание текста исходной программы и заполнение таблиц идентификаторов. Результатом этапа служит некоторое внутреннее представление программы, понятное компилятору.

На этапе синтеза на основании внутреннего представления программы и информации, содержащейся в таблицах идентификаторов, порождается текст результирующей программы. Результатом этого этапа является объектный код.

Данные этапы состоят из более мелких стадий, называемых фазами. Состав фаз и их взаимодействие зависит от конкретной реализации компилятора. Но в том или ином виде в каждом компиляторе выделяются следующие фазы:

1) лексический анализ;
2) синтаксический анализ;
3) семантический анализ;
4) подготовка к генерации кода;
5) генерация кода.

Общая схема работы компилятора представлена на рисунке 4.1.

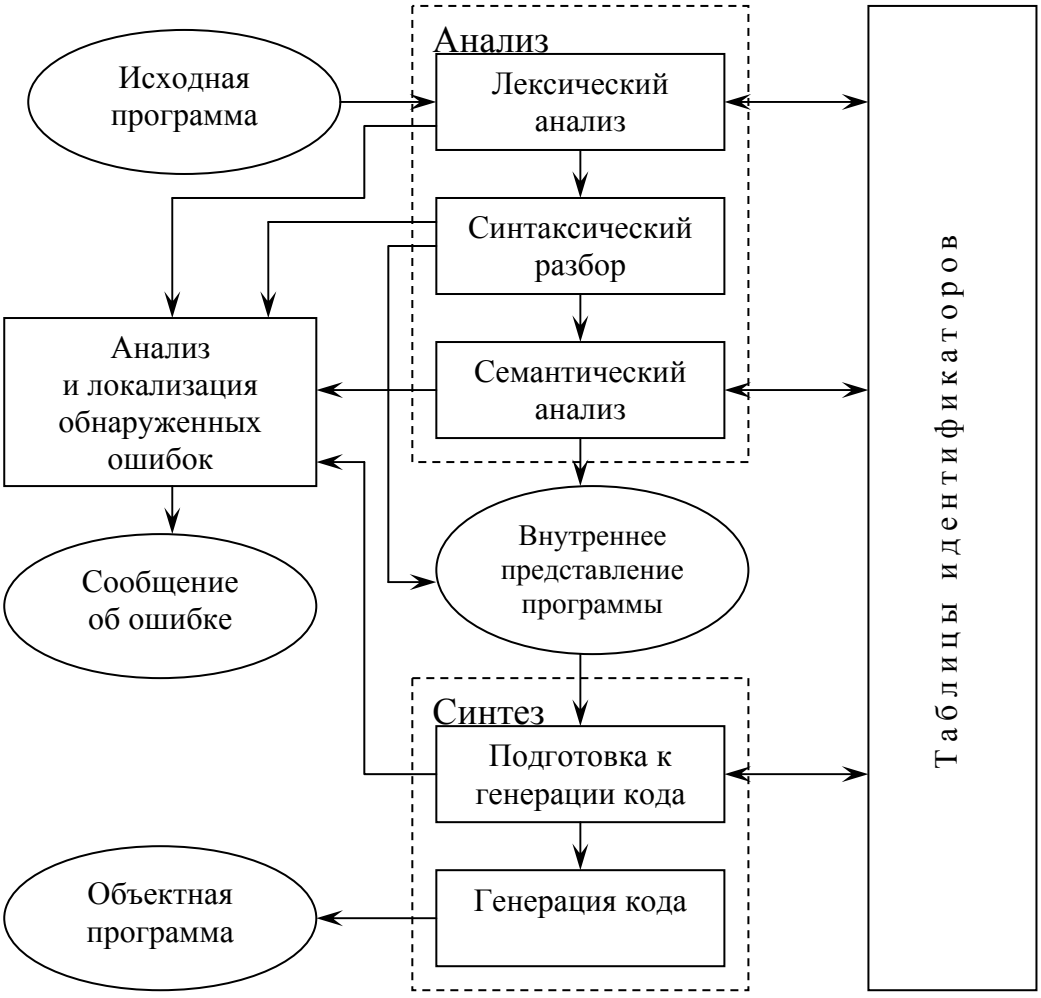


Рисунок 4.1 – Общая схема работы компилятора

Определение 4.5. Процесс последовательного чтения компилятором данных из внешней памяти, их обработки и помещения результатов во внешнюю память, называется проходом компилятора.

По количеству проходов выделяют одно-, двух-, трех- и многопроходные компиляторы.

4.3 Лексический анализ

Определение 4.6. Лексический анализ (ЛА) – это первый этап процесса компиляции, на котором символы, составляющие исходную программу, группируются в отдельные минимальные единицы текста, несущие смысловую нагрузку – лексемы.

Задача лексического анализа - выделить лексемы и преобразовать их к виду, удобному для последующей обработки. ЛА необязательный этап компиляции, но желательный по следующим причинам:

1) замена идентификаторов, констант, ограничителей и служебных слов лексемами делает программу более удобной для дальнейшей обработки;

2) ЛА уменьшает длину программы, устраняя из ее исходного представления несущественные пробелы и комментарии;

3) если будет изменена кодировка в исходном представлении программы, то это отразится только на ЛА.

В процедурных языках лексемы обычно делятся на классы:

1) служебные слова;
2) ограничители;
3) числа;
4) идентификаторы.

Каждую лексему можно представить парой чисел вида (n, k), где n – номер таблицы лексем, k - номер лексемы в таблице. Тогда входными данными ЛА будет текст транслируемой программы на входном языке, а выходными - последовательность лексем в числовом представлении.

Пример 4.1. Пусть для некоторого языка программирования таблица служебных слов имеет вид:

1) *program*; 2) *var*; 3) *int*; 4) *bool*; 5) *begin*; 6) *end*; 7) *if*; 8) *then*; 9) *else*; 10) *while*; 11) *do*; 12) *read*; 13) *write*; 14) *true*; 15) *false*.

Таблица ограничителей содержит:

1) . ; 2) ; ; 3) , ; 4) : ; 5) := ; 6) (; 7)) ; 8) + ; 9) - ; 10) * ; 11) / ; 12) $\vee$; 13) $\wedge$; 14) $\neg$; 15) = ; 16) > ; 17) <.

Таблицы идентификаторов и чисел формируются в ходе лексического анализа.

Фрагмент исходной программы имеет вид: *program var k*, *sum*: *int*; *begin k*:=0;…Тогда результатом работы ЛА будет следующая последовательность лексем: (1, 1) (1, 2) (4, 1) (2, 3) (4, 2) (2, 4) (1, 3) (2, 2) (1, 5) (4, 1) (2, 5) (3, 1) (2, 2)…

Анализ текста исходной программы проводится путем разбора по регулярным грамматикам и опирается на способ разбора по конечному автомату, снабженному дополнительными пометками-действиями. Каждая дуга такого автомата имеет вид, представленный на рисунке 4.2. Смысл этой конструкции: если текущим является состояние A и очередной входной символ совпадает с t_i для какого либо i, то осуществляется переход в новое состояние B, при этом выполняются действия $D_1, D_2, \ldots, D_m$.

Для удобства разбора вводится дополнительное состояние автомата ER, попадание в которое соответствует появлению ошибки в алгоритме разбора. Переход по дуге, не помеченной ни одним символом, осуществляется по любому другому символу, кроме тех, которыми помечены все другие дуги, выходящие из данного состояния.

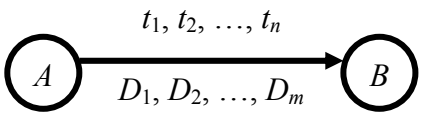


Рисунок 4.2 – Дуга конечного автомата с действиями

Выделяются методы непрямого и прямого лексического анализа.

Непрямой лексический анализ, или лексический анализ с возвратами, заключается в последовательной проверке версий о классах лексем. Если проверка текущей версии не подтверждается, то происходит откат назад по цепочке символов и осуществляется проверка следующей версии.

Непрямой лексический анализатор состоит из отдельных автоматов, каждый из которых распознает одну заданную лексему. Все автоматы имеют одинаковую структуру и отличаются только внутренними состояниями, что связано с различием распознаваемых лексем.

Прямой лексический анализ позволяет определить значение лексемы без откатов назад по цепочке символов. Прямой лексический анализатор строится на основе одного детерминированного автомата, объединяющего множество автоматов, распознающих отдельные лексемы. Такой автомат на каждом шаге читает один входной символ и переходит в следующее состояние, приближающее его к распознаванию текущей лексемы или формированию ошибки. Для лексем, имеющих одинаковые подцепочки, автомат имеет общие фрагменты, реализующие единое множество состояний. Отличающиеся части реализуются своими фрагментами.

Обычно лексический анализатор строится в два этапа:

1) построение конечного автомата с действиями для распознавания и формирования внутреннего представления лексем;

2) написание по конечному автомату с действиями функции сканирования текста исходной программы.

Рассмотрим реализацию данной методики на примере ЛА для числовых констант.

Пример 4.2. Конечный автомат с действиями, распознающий числовую константу и преобразующий ее в машинную форму, показан на рисунке 4.3. На рисунке использованы следующие обозначения.

Переменные:

1) *m* – текущее значение мантиссы в машине;

2) *p* – текущее значение показателя порядка в машине;

3) *n* – счетчик числа десятичных цифр после точки;

4) *s* – знак порядка (множитель +1 или -1);

5) *d* – очередная десятичная цифра;

6) *r* – число с плавающей точкой в машине.

7) *CH* – очередной входной символ;

8) *CS* - текущее состояние автомата.

Функции:

1) *digit* - проверка очередного символа на цифру 0..9;

2) *sign* – проверка очередного символа на принадлежность к множеству ограничителей констант, в нашем случае это множество {+, - , *, /, (,), ;}.

Процедуры:

1) *P* – добавление к предыдущим очередного символа лексемы для ее сборки, перемещение считывающей головки к следующему символу;

2) *B* – помещение собранной лексемы в таблицу и вывод ее внутреннего представления в выходную последовательность;

3) *D*0 – инициализация переменных $m=p=n=0$; $s=+1$ и считывание первого символа исходного текста;

4) *D*1 – вычисление текущего значения мантиссы при чтении очередной цифры мантиссы до обнаружения десятичной точки $m=10\times m+d$;

5) *D*2 – подсчет десятичных цифр после точки и вычисление текущего значения мантиссы при чтении очередной цифры мантиссы $n=n+1$; $m=10\times m+d$;

6) *D*3 – вычисление текущего значения показателя порядка при чтении очередной цифры порядка $p=10\times p+d$;

7) *D*4 – $s=-1$ при обнаружении отрицательного знака порядка;

8) *D*5 – получение машинной формы числа по завершению просмотра числа $r=m\times 10^{(s\times p-n)}$.

По конечному автомату можно написать функцию сканирования исходной строки символов на каком-либо языке программирования. В нашем случае функция *scanner* на языке *Pascal* будет иметь вид.

```
function scanner: boolean;
var CS: (S0, S1, S2, S3, S4, S5, S6, S7, V, ER);
begin D; CS:=S0;
  repeat
    case CS of
    S0: if CH='·' then
          begin
            P; CS:=S1
          end
```

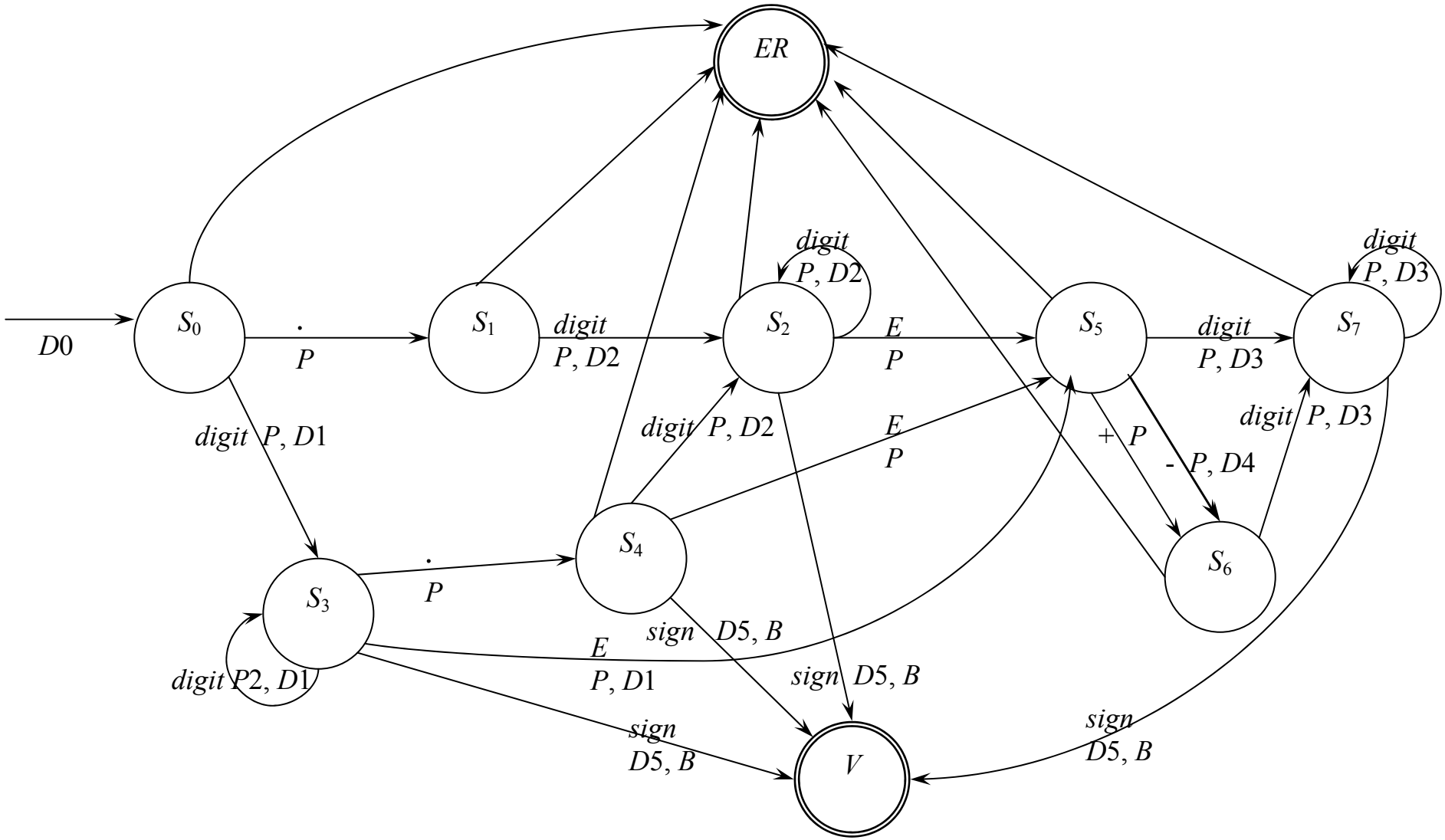


Рисунок 4.3 – Конечный автомат с действиями для распознавания и преобразования числовой константы

```
    else if digit then
         begin
           P, D1;
           CS:= S3
         end
    else ER;
S1: if digit then
        begin
          P; D2; CS:= S2
        end
        else ER;
S2: if digit then
        begin
          P; D2
        end
        else if sign then
        begin
          D5; B; CS:= V
        end
        else if CH='E' then
        begin
          P; CS:= S5
        end
        else ER;
S3: if digit then
        begin
          P; D1
        end
        else if sign then
        begin
          D5; B; CS:= V
        end
        else if CH='E' then
        begin
          P; D1; CS:= S5
        end
        else if CH='.' then
        begin
          P; CS:= S4
        end;
S4: if digit then
        begin
          P; D2; CS:= S2
        end
        else if sign then
        begin
          D5; B; CS:= V
        end
        else if CH='E' then
        begin
          P; CS:= S5
        end
        else ER;
    S5: if digit then
        begin
          P; D3; CS:= S7
        end
        else if CH='+' then
        begin
          P; CS:= S6
        end
        else if CH='-' then
        begin
          P; D4; CS:= S6
        end
        else ER;
    S6: if digit then
        begin
          P; D3; CS:= S7
        end
        else ER;
    S7: if digit then
        begin
          P; D3
        end
        else if sign then
        begin
          D5; B; CS:= V
        end
        else ER;
    end {case}
  until (CS=V) or (CS=ER);
  scanner:= CS=V
end;
```

4.4 Синтаксический анализ

Задача синтаксического анализа - провести проверку правильности конструкций программы (выражений, описаний, операторов и др.), образованных из лексем. Если формализовать эту задачу, то она будет ставиться следующим образом. Дан язык $L(G)$ с грамматикой G, в которой S - начальный символ. Построить дерево разбора входной цепочки $\alpha = a_1a_2a_3...a_n$.

Синтаксический разбор осуществляется с применением более сложных грамматик, обеспечивающих иерархическое определение одних правил через другие - КС-грамматик. Поэтому, для построения распознавателей, мощности конечных автоматов, используемых при лексическом анализе, уже не хватает. Необходим более мощный и универсальный автомат, поддерживающий построение дерева разбора и выстраивающий его как сверху вниз, так и снизу вверх – МП-автомат.

Общая классификация возможных вариантов построения распознавателя представлена на рисунке 4.4.

Рисунок 4.4 – Классификация методов организации синтаксического разбора

Разработано множество алгоритмов построения МП-автоматов по КС-грамматикам, некоторые из которых рассмотрены нами в главе 3. В качестве примера приведем еще один простейший распознаватель скобочных выражений.

Пример 4.3. Построим одну из возможных реализация МП-автомата для задачи проверки корректности вложенности круглых скобок. Кратко опишем общий принцип работы данного автомата. Первоначально в стек помещается признак дна #, а на входную ленту исходная строка символов, заканчиваемая признаком конца ⊥. Если входная головка читает «(», то в магазин заталкивается символ A. Если входная головка читает «)», то из магазина выталкивается содержащийся там символ. В обоих случаях головка перемещается на одну позицию вправо. Строка принимается автоматом, если стек пуст и входная цепочка прочитана до конца.

Формально данный МП-автомат определим следующим образом.
1) Множество входных символов $T = \{(,), \perp\}$.
2) Множество магазинных символов $N = \{A, \#\}$.
3) Множество состояний состоит из единственного состояния q.
4) Начальный магазинный символ – маркер дна $N_0 = \#$.
5) Магазинная функция задана управляющей таблицей 4.1.

Таблица 4.1 – Управляющая таблица МП-автомата

Магазинные символы	Входные символы		
	(	)	$\perp$
A	заталкивание	выброс	ошибка
#	заталкивание	ошибка	успех

Такая таблица является типичной формой представления одного внутреннего состояния для любого МП-автомата. Если у автомата имеется несколько внутренних состояний, то для каждого из них строится такая таблица переходов. В большинстве реальных случаев МП-автоматы имеют только одно состояние.

Работу данного МП-автомата можно рассмотреть на примере распознавания цепочки вида (( ) ( )). Осуществляемые автоматом переходы представлены в виде последовательности текущих состояний его устройств в таблице 4.2

Таблица 4.2 – Синтаксический разбор строки вида (( ) ( ))

Номер конфигурации	Текущее состояние	Содержимое магазина	Входная строка	Действие
1	q	#	(( ) ( )) $\perp$	заталкивание
2	q	#A	( ) ( )) $\perp$	заталкивание
3	q	# A A	) ( )) $\perp$	выброс
4	q	# A	( )) $\perp$	заталкивание
5	q	# A A	)) $\perp$	выброс
6	q	# A	) $\perp$	выброс
7	q	#	$\perp$	успех

В приведенном примере цепочка оказалась распознанной.

4.5 Семантический анализ

Семантика языков программирования изменяется в очень широких пределах. Она отличается не только по особенностям реализации отдельных операций, но и по парадигмам программирования, определяющим принципиальные различия в методах разработки программ. Специфика реализации операций может касаться как структуры обрабатываемых данных, так и правил обработки

одних и тех же типов данных. Такие языки, как *PL*/1 и *APL* поддерживают выполнение матричных и векторных операций. Большинство же языков работают в основном со скалярами, предоставляя для обработки массивов процедуры и функции, написанные программистами. Но даже при выполнении операции сложения двух целых чисел такие языки, как *C* и *Pascal* могут вести себя поразному.

Семантика определяет смысл операций языка, корректность операндов. Цепочки, имеющие одинаковую синтаксическую структуру в различных языках программирования, могут различаться по семантике (что, например, наблюдается в *C*++, *Pascal*, *Basic* для арифметических выражений).

Описание семантики и распознавание ее корректности обычно является самой трудоемкой и объемной частью транслятора, так как необходимо осуществить перебор и анализ множества вариантов допустимых комбинаций операций и операндов.

На этапе семантического анализа информация о свойствах именованных объектов программы помещается в таблицу имен, созданную на этапе лексического анализа. В ходе семантического анализа проверяются отдельные правила записи исходных программ, которые не описываются КС-грамматикой. Эти правила носят контекстно-зависимый характер, их называют семантическими соглашениями и контекстными условиями. Типичными контекстными условиями являются требования:

1) единственности объявления имени в блоке, параметра в списке параметров, списке значений и т.д.;

2) соответствия между описанием и использованием имени;

3) соответствия видов величин в синтаксических конструкциях (операндов в выражениях, формальных и фактических параметров и т.п.);

4) выполнения ограничений реализации языка в трансляторе (например, количество измерений массива или глубины вложенности блоков).

Проверка контекстных условий является простейшим контролем семантической правильности программы. Контроль позволяет обнаружить и зафиксировать в таблице ошибок очевидные ошибки, такие как, повторное объявление имени, противоречивые или недопустимые характеристики объекта программы, количественное несоответствие формальных и фактических параметров, несоответствие границ измерения массива и т.д.

На этапе семантического анализа может также решаться задача перевода программы на промежуточный язык. Рассмотрим существующие формы внутреннего представления программы.

4.6 Формы внутреннего представления программы

Возможны различные формы внутреннего представления синтаксических конструкций исходной программы, которые удобны на различных этапах компиляции.

Выделяют следующие общепринятые способы внутреннего представления программы:

1) многоадресный код с явно именуемым результатом (тетрады);
2) многоадресный код с неявно именуемым результатом (триады);
3) синтаксические деревья;
4) постфиксная запись;
5) машинные команды или ассемблерный код.

4.6.1 Тетрады и триады

Определение 4.7. Тетрады - это запись операций в форме из четырех составляющих: операции, двух операндов и результата операции.

Например, тетрады могут выглядеть так:

<операция>(<операнд1>, <операнд2>, <результат>).

Каждая тетрада в последовательности вычисляется так: операция, заданная тетрадой, выполняется над операндами и результат ее выполнения помещается в переменную, заданную результатом тетрады.

Определение 4.8. Триады – это запись операций в форме из трех составляющих: операция и два операнда.

Например, триады могут иметь вид:

<операция>(<операнд1>, <операнд2>).

Особенностью триад является то, что один или оба операнда могут быть ссылками на другую триаду в том случае, если в качестве операнда данной триады выступает результат выполнения другой триады. Каждая триада в последовательности вычисляется так: операция, заданная триадой, выполняется над операндами, а если в качестве одного из операндов (или обоих операндов) выступает ссылка на другую триаду, то берется результат вычисления той триады. Результат вычисления триады хранится во временной памяти.

Если какой-то из операндов в тетраде или триаде отсутствует, то он опускается или заменяется пустым операндом (в зависимости от принятой формы записи и ее реализации). Результат вычисления тетрады никогда опущен быть не может, иначе тетрада теряет смысл.

Тетрады и триады представляют собой линейную последовательность команд. При вычислении выражения, записанного в форме тетрад или триад, они вычисляются одна за другой последовательно. Порядок их вычисления может быть изменен, если допустить наличие тетрад и триад специального вида, например, вызывающих безусловный переход или переход по условию.

Линейную последовательность тетрад или триад проще преобразовывать в последовательность команд результирующей программы, чем синтаксические деревья. В отличие от команд ассемблера тетрады и триады являются машинно-независимой формой внутреннего представления программы.

Тетрады требуют больше памяти для своего представления, чем триады. Триады, в отличие от тетрад, явно отражают взаимосвязь операций между со-

бой. Возможны сложности с преобразованием тетрад в машинный код, так как они плохо отображаются в двухадресные команды ассемблера и машинные коды.

В отличие от тетрад, триады требуют разработки алгоритма, отвечающего за распределение памяти, необходимой для хранения промежуточных результатов вычисления, так как временные переменные для этой цели не используются.

Пример 4.4. Дано арифметическое выражение *X*:=*Y*/2-*Z*+*X**5.

Выражение, записанное в виде тетрад, будет иметь вид:

/ (*Y*, 2, *T*1);
- (*T*1, *Z*, *T*2);
* (*X*, 5, *T*3);
+ (*T*2, *T*3, *T*4);
:= (*X*, *T*4,).

Идентификаторы *T*1,…,*T*4 обозначают временные переменные, используемые для хранения результатов вычисления тетрад. В последней тетраде, которая требует только одного операнда, в качестве второго операнда выступает пустой операнд.

То же выражение, записанное в виде триад, будет иметь вид:

1) / (*Y*, 2);
2) - (^1, *Z*);
3) * (*X*, 5);
4) + (^2, ^3);
5) := (*X*, ^4).

Здесь знак ^ означает ссылку операнда одной триады на результат другой.

4.6.2 Синтаксические деревья

Результатом синтаксического разбора является дерево вывода. Оно содержит избыточную информацию, которая для дальнейшей работы компилятора не требуется. Эта информация включает в себя все нетерминальные символы и терминалы, не несущие смысловой нагрузки (например, «(» и «)», ε). После построения дерева эти символы не представляют интереса, так как им не соответствует никакой объектный код.

В синтаксическом дереве внутренние узлы (вершины) соответствуют операциям, а листья представляют собой операнды. Как правило, листья синтаксического дерева связаны с записями в таблице идентификаторов. Структура синтаксического дерева отражает синтаксис языка программирования, на котором написана исходная программа.

Синтаксические деревья могут быть построены компилятором для любой части входной программы. Не всегда синтаксическому дереву должен соответ-

ствовать фрагмент кода результирующей программы - например, возможно построение синтаксических деревьев для декларативной части языка. В этом случае операции, имеющиеся в дереве, не требуют порождения объектного кода, но несут информацию о действиях, которые должен выполнить сам компилятор над соответствующими элементами. В том случае, когда синтаксическому дереву соответствует некоторая последовательность операций, влекущая порождение фрагмента объектного кода, говорят о дереве операций.

Дерево операций можно непосредственно построить по дереву вывода, порожденному синтаксическим анализатором.

Алгоритм 4.1. Преобразование дерева разбора в дерево операций.

Шаг 1. Если в дереве больше нет узлов, помеченных нетерминальными символами, то выполнение алгоритма завершено; иначе - перейти к шагу 2.

Шаг 2. Выбрать крайний левый узел дерева, помеченный нетерминальным символом грамматики и сделать его текущим. Перейти к шагу 3.

Шаг 3. Если текущий узел имеет только один нижележащий узел, то текущий узел необходимо удалить из дерева, а связанный с ним узел присоединить к узлу вышележащего уровня (исключить из дерева цепочку) и вернуться к шагу 1; иначе - перейти к шагу 4.

Шаг 4. Если текущий узел имеет нижележащий узел (лист дерева), помеченный терминальным символом, который не несет семантической нагрузки, тогда этот лист нужно удалить из дерева и вернуться к шагу 3; иначе - перейти к шагу 5.

Шаг 5. Если текущий узел имеет один нижележащий узел (лист дерева), помеченный терминальным символом, обозначающим знак операции, а остальные узлы помечены как операнды, то лист, помеченный знаком операции, надо удалить из дерева, текущий узел пометить этим знаком операции и перейти к шагу 1; иначе - перейти к шагу 6.

Шаг 6. Если среди нижележащих узлов для текущего узла есть узлы, помеченные нетерминальными символами грамматики, то необходимо выбрать крайний левый среди этих узлов, сделать его текущим узлом и перейти к шагу 3; иначе - выполнение алгоритма завершено.

Конец алгоритма.

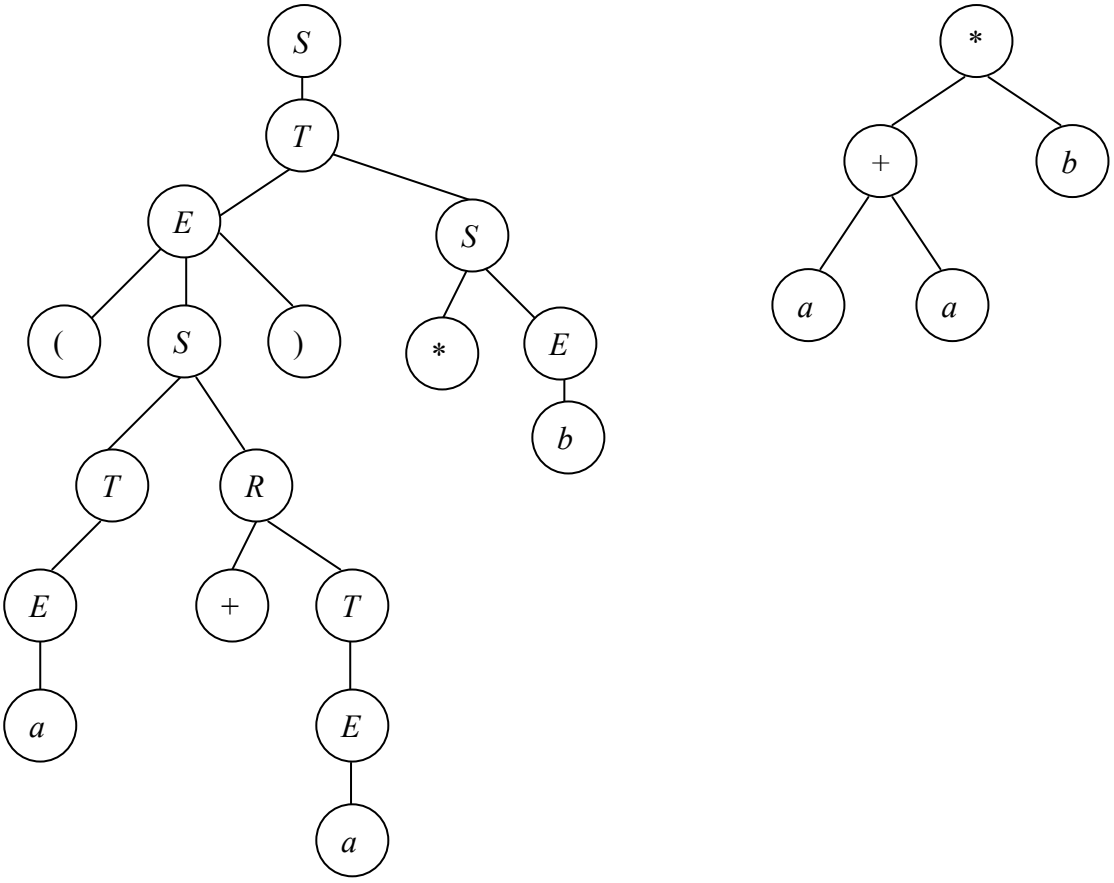
Пример 4.5. Пусть в результате синтаксического разбора получено дерево разбора для цепочки (*a*+*a*)***b*, показанное на рисунке 4.5 а). В результате применения алгоритма 4.1 получим дерево операций, представленное на рисунке 4.5 б).

4.6.3 Польская инверсная запись

Польская инверсная запись (ПОЛИЗ) - это постфиксная запись операций. Она была предложена польским математиком Я. Лукашевичем, откуда и происходит ее название.

В этой записи знаки операций записываются непосредственно за операндами. По сравнению с обычной (инфиксной) записью операций в польской записи операнды следуют в том же порядке, а знаки операций - строго в порядке

их выполнения. Этот факт делает ее чрезвычайно удобной для вычисления выражений с помощью стека на компьютере. Польская запись не требует учёта приоритета операций, в ней не употребляются скобки, и в этом ее основное преимущество.



а) дерево разбора б) дерево операций

Рисунок 4.5 – Графическое представление строки (*a*+*a*)**b*

Главный недостаток обратной польской записи заключается в том, что выражения в форме обратной польской записи почти не поддаются оптимизации.

Перевод в ПОЛИЗ выражений.

Справедливы следующие формальные определения.

Определение 4.9. Если E является единственным операндом, то ПОЛИЗ выражения E – это этот операнд.

Определение 4.10. ПОЛИЗом выражения $E_1\ \theta\ E_2$, где θ - знак бинарной операции, E_1 и E_2 – операнды для θ, является запись $E_1'E_2'\theta$, где E_1', E_2' - ПОЛИЗ выражений E_1 и E_2 соответственно.

Определение 4.11. ПОЛИЗ выражения θE, где θ - знак унарной операции, а E – операнд θ, есть запись $E_1'\theta$, где E' - ПОЛИЗ выражения E.

Определение 4.12. ПОЛИЗ выражения (E) есть ПОЛИЗ выражения E.

Пример 4.6. Для выражения в обычной (инфиксной записи) *a**(*b*+*c*)-(*d*-*e*)/*f* ПОЛИЗ будет иметь вид: *abc*+**de*-*f*/-.

Перевод в ПОЛИЗ операторов.

Каждый оператор языка программирования может быть представлен как *n*-местная операция с семантикой, соответствующей семантике оператора. Рассмотрим примеры перевода в ПОЛИЗ операторов некоторого модельного языка программирования.

Оператор присваивания *I:=E* в ПОЛИЗе записывается:

$$\underline{I}E:=, \tag{4.1}$$

где «:=» - двуместная операция,

$\underline{I}$, E – операнды операции присваивания;

$\underline{I}$ – означает, что операндом операции «:=» является адрес переменной *I*, а не ее значение.

Пример 4.7. Оператор *x*:=*x*+9 в ПОЛИЗе имеет вид: $\underline{x}\ x\ 9\ +\ :=$.

Оператор перехода в терминах ПОЛИЗа означает, что процесс интерпретации необходимо продолжить с того элемента ПОЛИЗа, который указан как операнд операции перехода. Чтобы можно было ссылаться на элементы ПОЛИЗа, будем считать, что все они пронумерованы, начиная с единицы (например, последовательные элементы одномерного массива). Пусть ПОЛИЗ оператора, помеченного меткой *L*, начинается с номера *p*, тогда оператору безусловного перехода *goto L* в ПОЛИЗе будет соответствовать:

p!, где ! – операция выбора элемента ПОЛИЗа, номер которого равен *p*.

Условный оператор. Введем вспомогательную операцию – условный переход «по лжи» с семантикой *if* (*not B*) *then goto L*. Это двуместная операция с операндами *B* и *L*. Обозначим ее !*F*, тогда в ПОЛИЗе она будет записываться:

B p !*F*, где *p* – номер элемента, с которого начинается ПОЛИЗ оператора, помеченного меткой *L*.

С использованием введенной операции условный оператор *if B then* S_1 *else* S_2 в ПОЛИЗе будет записываться:

B p_1 !*F* S_1 p_2 ! S_2, где p_1 – номер элемента, с которого начинается ПОЛИЗ оператора S_2, а p_1 – оператора, следующего за условным оператором.

Оператор цикла. С учетом введенных операций оператор цикла *while B do S* в ПОЛИЗе будет записываться:

B p_1 !*F S* p_o !, где p_o – номер элемента, с которого начинается ПОЛИЗ выражения *B*, а p_1 – оператора, следующего за данным оператором цикла.

Пример 4.8. ПОЛИЗ оператора *if* $x>0$ *then* $x:=x+8$ *else* $x:=x-3$ представлен в таблице 4.3.

Таблица 4.3 – ПОЛИЗ оператора *if*

лексема	$\underline{x}$	0	>	13	!*F*	$\underline{x}$	x	8	+	:=	18	!	$\underline{x}$	x	3	-	:=	…
номер	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Пусть **операторы ввода и вывода** одноместные. Тогда R – обозначение операции ввода, а W – обозначение операции вывода, значит оператор *read*(I) в ПОЛИЗе запишется как $\underline{I}\ R$, а оператор *write*(E) – $E\ W$.

Составной оператор *begin* S_1; S_2;...; S_n *end* в ПОЛИЗе записывается как $S_1\ S_2 ...\ S_n$.

Пример 4.9. ПОЛИЗ оператора *while* $n>3$ *do begin write*($n*n$-1); $n:=n$-1 *end* представлен в таблице 4.4.

Таблица 4.4 – ПОЛИЗ оператора *while*

лексема	n	3	>	19	!*F*	n	n	*	1	-	W	$\underline{n}$	n	1	-	:=	1	!	…
номер	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Запись программы в форме ПОЛИЗа удобна для последующей интерпретации (выполнения программы) с помощью стека. Массив ПОЛИЗа просматривается один раз слева направо, при этом:

1) если очередной элемент ПОЛИЗа является операндом, то его значение заносят в стек;

2) если очередной элемент – операция, то на «верхушке» стека находятся ее операнды, которые извлекаются из стека, над ними выполняется соответствующая операция, результат которой заносится в стек;

3) интерпретация продолжается до тех пор, пока не будет считан признак конца ПОЛИЗа, стек при этом должен быть пуст (в случае выражения, содержать значение выражения).

Пример 4.10. ПОЛИЗ выражения (4+2)/3+2*(4+6) имеет вид 42+3/246+*+. Состояния стека при интерпретации этого выражения показаны на рисунке 4.6. Состояния стека пронумерованы. Состояния 0, 1, 3, 5, 6, 7 соответствуют ситуации «операнд», остальные состояния – ситуации «операция». После завершения интерпретации в стеке остался результат вычисления выражения – 22.

							6			
						4	4	10		
	2		3		2	2	2	2	20	
4	4	6	6	2	2	2	2	2	2	22
0	1	2	3	4	5	6	7	8	9	10

Рисунок 4.6 – Состояния стека при интерпретации ПОЛИЗа

4.6.4 Ассемблерный код и машинные команды

Команды ассемблера представляют собой лишь форму записи машинных команд, а потому в качестве формы внутреннего представления программы практически ничем не отличаются от них. Машинные команды удобны тем, что при их использовании внутреннее представление программы полностью соответствует объектному коду, и сложные преобразования не требуются.

Использование команд ассемблера или машинных команд для внутреннего представления программы требует дополнительных структур для отображения взаимосвязи операций. Это машинно-зависимые формы представления программы, поэтому при ориентации компилятора на другой результирующий код потребуется перестраивать как само внутреннее представление программы, так и методы его обработки (при использовании триад или тетрад этого не требуется).

Любой компилятор работает с представлением результирующей программы в форме машинных команд, однако их обработка происходит, как правило, на завершающих этапах фазы генерации кода.

4.7 Генерация кода

Определение 4.13. Непосредственная генерация кода – это процесс порождения машинных команд по записи программы на промежуточном языке.

Задача генерации машинного кода обычно делится на две более простые задачи:

1) расчленение операций промежуточного языка на операции, эквивалентные машинным, и представление программы в виде псевдокода;

2) формирование компьютерных команд по псевдокоду.

При формировании псевдокода по промежуточной программе можно использовать заготовки. Продемонстрируем данную методику на примере преобразования дерева операций в код на языке ассемблера.

В качестве языка ассемблера возьмем язык ассемблера процессоров типа *Intel* 80x86. При этом будем считать, что операнды могут быть помещены в 16-разрядные регистры процессора и в коде результирующей объектной программы могут использоваться регистры *AX* (аккумулятор) и *DX* (регистр данных), а также стек для хранения промежуточных результатов.

Введем обозначения:

1) *Code*(*U*) - функция, реализующая перевод данного узла *U* дерева в последовательность команд ассемблера;

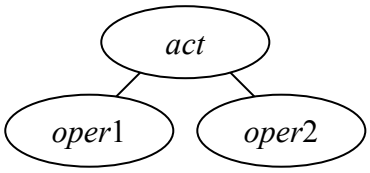
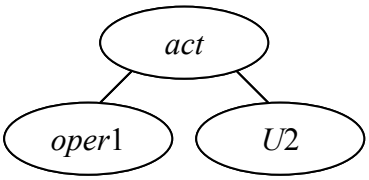
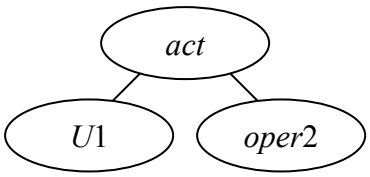
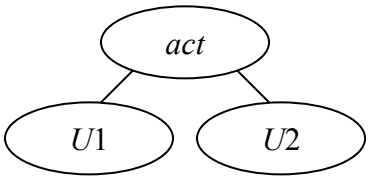
2) *oper*l, *oper*2 - операнды (листья дерева);

3) *U*1, *U*2 – узлы дерева, не являющиеся листьями;

4) *act* - обозначение одной из арифметических операций - сложения (+), вычитания (-), умножения (*) и деления (/), которым будут соответствовать команды ассемблера *add*, *sub*, *mul* и *div*.

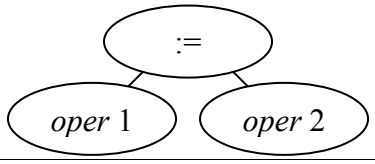
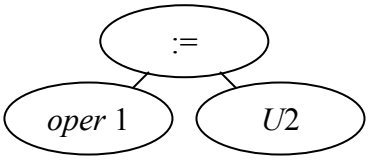
Тогда четырём возможным формам текущего узла дерева для каждой арифметической операции будут соответствовать фрагменты кода на языке ассемблера, приведенные в таблице 4.5.

Таблица 4.5 - Преобразование узлов дерева операций в код на языке ассемблера для арифметических операций

Вид узла дерева	Результирующий код
	*mov ax*, *oper*1 *act ax*, *oper*2
	*Code* (*U*2) *mov dx*, *ax* *mov ax*, *oper*1 *act ax*, *dx*
	*Code* (*U*1) *act ax*, *oper*2
	*Code* (*U*1) *push ax* *Code* (*U*2) *mov dx*, *ax* *pop ax* *act ax*, *dx*

Семантика оператора присвоения требует, чтобы в левой части операции всегда был операнд, поэтому для нее возможны только два типа узлов дерева, влекущие порождение кода (два других типа узлов должны приводить к сообщению об ошибке). Заготовки для этой операции приведены в таблице 4.6.

Таблица 4.6 - Преобразование узлов дерева операций в код на языке ассемблера для операции присвоения

Вид узла дерева	Результирующий код
	*mov ax*, o*per*2 *mov oper*l, *ax*
	*Code* (*U*2) *mov oper*l, *ax*

Пример 4.11. Рассмотрим в качестве примера выражение *X*:=*Y**2-*Z*/3. Соответствующее ему дерево операций приведено на рисунке 4.7.

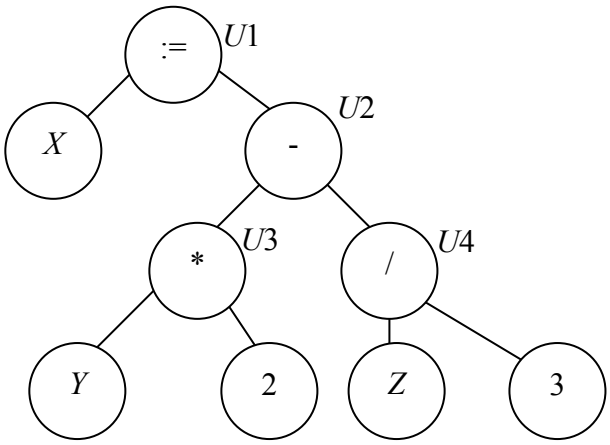


Рисунок 4.7 - Дерево операций для выражения *X*:=*Y**2-*Z*/3

Построим последовательность команд языка ассемблера, соответствующую дереву операций. Пронумеруем узлы дерева согласно методу рекурсивного спуска по направлению от корня к листьям в порядке слева направо. Последовательность построения цепочки команд языка ассемблера по шагам рекурсии имеет вид.

Шаг 1. *Code*(*U*2)
mov X, *ax*

Шаг 2. *Code*(*U*3)
push ax
Code(*U*4)
mov dx, *ax*
pop ax
sub ax, *dx*
mov X, *ax*

Шаг 3. *mov ax*, *Y*
mul 2
push ax
Code(*U*4)
mov dx, *ax*
pop ax
sub ax, *dx*
mov X, *ax*

Шаг 4. *mov ax*, *Y*
mul 2
push ax
mov ax, *Z*
div 3
mov dx, *ax*
pop ax
sub ax, *dx*
mov X, *ax*

4.8 Оптимизация кода

4.8.1 Сущность оптимизации кода

Полученный в результате генерации объектный код может содержать лишние команды и данные. Это снижает эффективность выполнения результирующей программы, поэтому большинство современных компиляторов выполняют еще один этап компиляции - оптимизацию результирующей программы.

Определение 4.14. Оптимизация программы - это обработка, связанная с переупорядочиванием и изменением операций в компилируемой программе с целью получения более эффективной результирующей объектной программы.

Оптимизация выполняется на этапах подготовки к генерации и непосредственно при генерации объектного кода.

Компилятор вынужден производить оптимизацию построенного кода, т.к. он не может выполнить семантический анализ всей входной программы в целом, оценить ее смысл и, исходя из него, построить результирующую программу. Оптимизация нужна, поскольку результирующая программа строится не вся сразу, а поэтапно.

Оптимизация - это необязательный этап компиляции. Без оптимизации результирующая программа будет правильной, а сам компилятор будет полностью выполнять свои функции. Но практически все компиляторы выполняют оптимизацию, поскольку их разработчики стремятся завоевать хорошие позиции на рынке средств разработки программного обеспечения.

Выделяют два критерия эффективности результирующей программы:

1) объем памяти, необходимый для выполнения результирующей программы (для хранения всех ее данных и кода);

2) скорость выполнения (быстродействие) программы.

Выполнить оптимизацию по обоим критериям не всегда удается, поэтому для оптимизации обычно выбирается либо один из них, либо комплексный критерий, основанный на них. Выбор критерия оптимизации обычно выполняет непосредственно пользователь в настройках компилятора.

Проблема нахождения самой короткой или самой быстрой результирующей программы, эквивалентной заданной исходной программе, в общем случае алгоритмическим неразрешима. Все, что можно сделать на этапе оптимизации, - это выполнить над заданной программой последовательность преобразований в надежде сделать ее более эффективной.

На практике для оценки эффективности результирующей программы, полученной с помощью компилятора, сравнивают ее с эквивалентной программой, полученной из исходной программы, написанной на языке ассемблера. Обычно объектная программа, построенная с помощью компилятора с языка высокого уровня, содержит на 10-30 % больше команд, чем эквивалентная ей объектная программа, построенная с помощью ассемблера, а также выполняется на 10-30 % медленнее.

4.8.2 Методы оптимизации кода

Выделяют два основных вида оптимизирующих преобразований.

1 Преобразования исходной программы (в форме ее внутреннего представления в компиляторе). Данный вид преобразований не зависит от результирующего объектного языка и архитектуры целевой вычислительной системы; обычно основан на выполнении известных математических и логических тождеств.

2 Преобразования результирующей объектной программы. Данные преобразования могут зависеть не только от свойств объектного языка, но и от архитектуры целевой вычислительной системы. В большинстве случаев они

сильно зависят от реализации компилятора и являются «ноу-хау» производителей компилятора.

По типу синтаксических конструкций методы оптимизации делятся на методы преобразования:

- линейных участков программы;
- логических выражений;
- вызовов процедур и функций;
- других конструкций входного языка.

Во всех случаях могут использоваться как машинно-зависимые, так и машинно-независимые методы оптимизации.

4.8.3 Оптимизация линейных участков программ

Для операций, составляющих линейный участок программы, могут применяться следующие виды оптимизирующих преобразований:

- удаление бесполезных присваиваний;
- исключение избыточных вычислений (лишних операций);
- свертка операций объектного кода;
- перестановка операций;
- арифметические преобразования.

Определение 4.15. Если в составе линейного участка программы имеется операция присвоения значения некоторой переменной A с номером i, а также операция присвоения значения той же переменной A с номером j, $i<j$ и ни в одной операции между i и j не используется значение переменной A, то операция присвоения с номером i является бесполезной.

Проблемы с удалением бесполезных присваиваний могут возникнуть, если между двумя операциями присваивания в линейном участке выполняются действия над указателями или вызовы функций, имеющих «побочный эффект».

Пример 4.12. Даны два фрагмента программы.

X:= *Y*/*Z*;	*P*:= @*X*;
D:= *Y*-*Z*;	*X*:= *Y*/*Z*;
X:= *D*/*Z*;	*D*:= *P*^-*Z*;
	X:= *D*/*Z*;

Во фрагменте программы слева операция присвоения *X*:= *Y*/*Z* является бесполезной и может быть удалена. Вместе с ней удаляется и операция деления, которая в результате также окажется бесполезной. Во фрагменте программы справа таже операция присвоения уже не является бесполезной.

Определение 4.16. Операция линейного участка с порядковым номером j считается лишней, если существует идентичная ей операция с порядковым номером i, $i<j$ и никакой операнд, обрабатываемый этой операцией, не изменялся никакой операцией, имеющей порядковый номер между i и j.

Определение 4.17. Свертка объектного кода - это выполнение во время компиляции тех операций исходной программы, для которых значения операндов уже известны.

Пример 4.13. Тривиальный вариант свертки - это вычисление выражений, все операнды которых являются константами.

Выражение *X*:=5**Y***Z**2 в результате свертки может быть преобразовано к виду *X*:=10**Y***Z*.

Хорошим стилем программирования является объединение вместе операций, производимых над константами, чтобы облегчить компилятору выполнение свертки.

Определение 4.18. Перестановка операций – это изменение порядка следования операций, которое может повысить эффективность программы, не изменяя конечный результат вычислений.

Пример 4.13. Операции умножения в выражении *X*:=5**Y***Z**2 можно переставить без изменения конечного результата и выполнить в порядке *X*:=(5*2)*(*Y***Z*). Тогда возможно выполнить свертку и сократить количество операций.

Арифметические преобразования представляют собой изменение характера и порядка следования операций на основании известных алгебраических и логических тождеств.

Пример 4.14. Выражение *X*:=2**Y*+2**Z* может быть заменено на *X*:=2*(*Y*+*Z*). Конечный результат при этом не изменится, но объектный код будет содержать на одну операцию умножения меньше.

К арифметическим преобразованиям относятся и такие действия, как замена возведения в степень на умножение; целочисленного умножения на константы, кратные 2, - выполнением операций сдвига. В этих случаях быстродействие программы повышается за счет замены сложных операций более простыми.

Рассмотрим подробнее два из перечисленных методов оптимизации линейных участков программ.

Алгоритм 4.2. Свертка объектного кода программы.

Алгоритм свертки для линейного участка программы работает со специальной таблицей, которая содержит пары (<переменная>, <константа>) для всех переменных, значения которых уже известны.

Рассмотрим выполнение алгоритма свертки объектного кода для триад. Для пометки операций, не требующих порождения объектного кода, будем использовать триады специального вида *C*(*k*, 0).

Алгоритм свертки триад последовательно просматривает триады линейного участка и для каждой триады делает следующее.

1 Если операнд триады - переменная, которая содержится в таблице, то операнд заменяется на соответствующее значение константы.

2 Если операнд триады - ссылка на триаду специального вида *C*(*k*, 0), то операнд заменяется на значение константы *k*.

3 Если все операнды триады - константы, то триада сворачивается. Т.е. данная триада вычисляется и вместо нее помещается триада специального вида *C*(*k*,0), где *k* - константа, являющаяся результатом выполнения свернутой триа-

ды. При генерации кода для триад специального вида объектный код не порождается, потому она в дальнейшем исключается.

4 Если триада является присваиванием типа *A*:=*B*, тогда:

- если *B* - константа, то переменная *A* со значением константы заносится в таблицу (если там уже было старое значение для *A*, то это старое значение исключается);

- если *B* - не константа, то переменная *A* вообще исключается из таблицы, если она там есть.

Конец алгоритма.

Пример 4.15. Пусть фрагмент исходной программы (записанной на языке типа *Pascal*) имеет вид:

A:=2+3;

A:=6;

B:=3**A*+*A*;

Его внутреннее представление в форме триад показано в первом столбце таблицы 4.7. Процесс выполнения алгоритма свертки объектного кода по шагам отражен в таблице 4.7.

Таблица 4.7 - Пример работы алгоритма свертки

Триада	Шаг 1	Шаг 2	Шаг 3	Шаг 4	Шаг 5	Шаг 6
1) +(2, 3)	*C*(5, 0)	*C*(5, 0)	*C*(5, 0)	*C*(5, 0)	*C*(5, 0)	*C*(5, 0)
2) :=(*A*, ^1)	:=(*A*, ^1)	:=(*A*, 5)	:=(*A*, 5)	:=(*A*, 5)	:=(*A*, 5)	:=(*A*, 5)
3) :=(*A*, 6)	:=(*A*, 6)	:=(*A*, 6)	:=(*A*, 6)	:=(*A*, 6)	:=(*A*, 6)	:=(*A*, 6)
4) *(3, *A*)	*(3, *A*)	*(3, *A*)	*(3, *A*)	*C*(18, 0)	*C*(18, 0)	*C*(18, 0)
5) +(^4, *A*)	+(^4, *A*)	+(^4, *A*)	+(^4, *A*)	+(^4, *A*)	*C*(24, 0)	*C*(24, 0)
6) :=(*B*,^5)	:=(*B*,^5)	:=(*B*,^5)	:=(*B*,^5)	:=(*B*,^5)	:=(*B*,^5)	:=(*B*,24)
Таблица пар	(,)	(*A*, 5)	(*A*, 6)	(*A*, 6)	(*A*, 6)	(*A*, 6) (*B*, 24)

Если исключить триады специального вида *C*(*k*, 0) (которые не порождают объектного кода), то в результате выполнения свертки получим следующую последовательность триад:

1) :=(*A*, 5);

2) :=(*A*, 6);

3) :=(*B*, 24).

Алгоритм 4.3. Исключение лишних операций.

Рассмотрим алгоритм исключения лишних операций для триад, т.к. они полностью отражают взаимосвязь операций.

Чтобы следить за внутренней зависимостью переменных и триад, присвоим им числа зависимости по следующим правилам:

1) первоначально число зависимости каждой переменной равно 0, так как в начале работы программы ее значение не зависит ни от какой триады;

2) после обработки *i*-й триады, в которой переменной *A* присваивается некоторое значение, число зависимости *A* (*dep*(*A*)) становится равным *i*, так как значение *A* теперь зависит от *i*-й триады;

3) при обработке *i*-й триады ее число зависимости (*dep*(*i*)) принимается равным значению 1+(максимальное из чисел зависимости её операндов).

Алгоритм исключения лишних операций использует триады особого вида *SAME*(*i*, 0). Если такая триада встречается в позиции с номером *j*, то это означает, что в исходной последовательности триад некоторая триада *j* идентична триаде *i*.

Триады линейного участка последовательно просматриваются, и для каждой из них выполняются следующие шаги.

1 Если какой-то операнд триады ссылается на особую триаду вида *SAME*(*i*, 0), то он заменяется ссылкой на триаду с номером *i*(^*i*).

2 Вычисляется число зависимости текущей триады с номером *j*, исходя из чисел зависимости ее операндов.

3 Если в просмотренной части списка триад существует идентичная *i*-я триада, причем *i*<*j* и *dep*(*i*)=*dep*(*j*), то текущая триада *j* заменяется на триаду особого вида *SAME*(*i*, 0).

4 Если текущая триада - присвоение, то вычисляется число зависимости соответствующей переменной.

Конец алгоритма.

Пример 4.16. Дан фрагмент программы:

K:=*K*-*Z*/*Y*;

X:=*K*-*Z*/*Y*;

Z:=*K*-*Z*/*Y*;

Работа алгоритма исключения лишних операций отражена в таблице 4.8.

Таблица 4.8 - Пример работы алгоритма исключения лишних операций

Обрабатываемая триада *j*	Числа зависимости переменных				Числа зависимости триад *dep*(*j*)	Выходная последовательность триад
	X	*Y*	*Z*	*K*		
1) /(*Z*, *Y*)	0	0	0	0	1	1) /(*Z*, *Y*)
2) -(*K*, ^l)	0	0	0	0	2	2) -(*K*, ^1)
3) := (*K*, ^2)	0	0	0	3	3	3) :=(*K*, ^2)
4) /(*Z*, *Y*)	0	0	0	3	1	4) *SAME*(1,0)
5) - (*K*, ^4)	0	0	0	3	4	5) -(*K*, ^1)
6):= (*X*, ^5)	6	0	0	3	5	6) :=(*X*, ^5)
7) /(*Z*, *Y*)	6	0	0	3	1	7) *SAME*(1, 0)
8) -(*K*, ^7)	6	0	0	3	4	8) *SAME*(5, 0)
9) :=(*Z*,^8)	6	0	9	3	5	9) :=(*Z*, ^5)

После исключения триад особого вида *SAME*(*i*, 0) и перенумерации оставшихся, получим следующую последовательность триад:

1) /(*Z*, *Y*);
2) -(*K*, ^1);
3) :=(*K*, ^2);
4) -(*K*, ^1);
5) :=(*X*, ^4);
6) :=(*Z*, ^4).

4.8.4 Оптимизация логических выражений

Компиляторы строят объектный код вычисления логических выражений таким образом, что вычисление выражения прекращается сразу же, как только его значение становится предопределенным. В сочетании с преобразованиями логических выражений на основе тождеств булевой алгебры и перестановкой операций эффективность данного метода может быть несколько увеличена.

Определение 4.19. Операция называется предопределенной для некоторого значения операнда, если ее результат зависит только от этого операнда и остается неизменным (инвариантным) относительно значений других операндов.

Пример 4.17. Операция логического сложения (*or*) является предопределенной для логического значения «истина» (*true*), а операция логического умножения (*and*) предопределена для логического значения «ложь» (*false*).

Поэтому, например, выражение *A and B and C and D* не имеет смысла вычислять, если известно, что значение переменной *A* есть *false.*

Не всегда такие преобразования инвариантны к смыслу программы. Если операнды являются функциями с «побочными эффектами», то семантика программы может измениться.

Хорошим стилем программирования считается располагать операнды в логических выражениях таким образом, чтобы в первую очередь вычислялись те из них, которые чаще определяют всё значение выражения. Кроме того, значения функций лучше вычислять в конце, а не в начале логического выражения, чтобы избежать лишних обращений к ним.

4.8.5 Оптимизация циклов

Для оптимизации циклов используются следующие методы:

- вынесение инвариантных вычислений из циклов;
- замена операций с индуктивными переменными;
- слияние и развертывание циклов.

Вынесение инвариантных вычислений из циклов заключается в вынесении за пределы циклов тех операций, операнды которых не изменяются в процессе выполнения цикла. Такие операции могут быть выполнены один раз до начала цикла, а полученные результаты потом могут использоваться в теле цикла.

Пример 4.18. Рассмотрим цикл *for i*:=l *to* 100 *do K*[*i*]:= *K*[*i*]/(*A***C*).

После применения оптимизирующего преобразования цикл будет иметь вид:

C:= *A***B*;
for i:=l *to* 100 *do K*[*i*]:= *K*[*i*]/*C*.

Замена операций с индуктивными переменными заключается в изменении сложных операций с индуктивными переменными в теле цикла более простыми операциями. Как правило, выполняется замена умножения на сложение.

Определение 4.20. Переменная называется индуктивной в цикле, если ее значения в процессе выполнения цикла образуют арифметическую прогрессию.

Если индуктивных переменных в цикле несколько, то в теле цикла их можно заменить на одну, а реальные значения для каждой переменной будут вычисляться с помощью соответствующих коэффициентов соотношения. Простейшей индуктивной переменной является переменная-счетчик цикла.

Пример 4.18. Рассмотрим цикл *for i*:=l *to N do K*[*i*]:=*i**100.

После применения оптимизирующего преобразования, цикл будет иметь вид:

```
T:= 100; i:= 1;
while i<=N do
begin
    K[i]:= T;
    T:= T+100;
    i:= i +1;
end;
```

Слияние циклов предусматривает замену двух вложенных циклов одним.

Пример 4.19. Рассмотрим фрагмент программы, инициализирующий двумерный массив:

```
for i:=l to N do
  for j:=l to M do A[i, j]:=0.
```

С точки зрения объектного кода (но не с позиции семантики исходного языка) двумерный массив - это область памяти размером *N***M*, поэтому данную операцию можно представить так:

```
K:= N*M;
for i:=1 to K do A[i]:= 0.
```

Развертывание циклов – это замена цикла линейной последовательностью операций. Данную замену можно выполнить для циклов, кратность выполнения которых известна уже на этапе компиляции. Тогда цикл, кратностью *N*, можно заменить линейной последовательность *N* операций, содержащих тело цикла.

Пример 4.20. Цикл *for i*:=1 *to* 3 *do A*[*i*]:= *i** *i* можно заменить операциями:

```
A[1] := 1;
A[2] := 4;
A[3] := 9.
```

Вопросы и задания для самоконтроля по 4 разделу

1 Как соотносятся понятия транслятор, компилятор, интерпретатор и ассемблер? Приведите примеры.

2 Изобразите общую схему работы компилятора и поясните ее элементы.

3 Почему лексический анализ является желательным этапом компиляции?

4 Какой тип грамматик и распознавателей лежит в основе лексического анализатора?

5 Какие методы лексического анализа существуют?

6 Составьте конечный автомат для разбора идентификаторов и служебных слов программы. Запишите по нему функцию сканирования.

7 Формализуйте задачу синтаксического анализатора.

8 На основе какого типа распознавателей функционирует синтаксический анализатор?

9 Классифицируйте возможные варианты построения синтаксического анализатора.

10 Какие условия проверяются на этапе семантического анализа?

11 Выполните сравнительный анализ существующих форм внутреннего представления программы.

12 Запишите правила перевода в ПОЛИЗ операторов на примере некоторого языка программирования.

13 Как выполнить интерпретацию программы, записанной в ПОЛИЗе?

14 В чем сущность процесса генерации кода?

15 Назовите критерии эффективности результирующей программы.

16 Какие виды оптимизирующих преобразований программы выделяют?

17 В чем суть метода свертки объектного кода программы?

18 На чем основан алгоритм исключение лишних операций линейного участка программы?

19 Какова специфика оптимизации логических выражений?

20 Охарактеризуйте методы оптимизации циклов.

21 Записать данное выражение $a-a*(c-d)+(a-b)/d$ в виде:

а) тетрад;

б) триад;

в) ПОЛИза;

г) дерева операций;

д) последовательности ассемблерных команд.

22 Записать в форме ПОЛИЗа программу вычисления факториала числа.

23 Используя стек, вычислить выражение *a b or not a not b not or and a not or* при $a = b = true$.

24 Удалить бесполезные присваивания и выполнить свертку кода линейного участка программы:

$A:=4;$

$C:=A+7;$

$B:=A*2-3;$

$C:=A+B;$

Тесты контроля усвоения материала по 4 разделу

1 Программа, которая переводит исходную программу в эквивалентную ей объектную программу на языке машинных команд или на языке ассемблера, называется:

а) транслятором;
б) интерпретатором;
в) ассемблером;
г) компилятором;
д) эмулятором.

2 Интерпретируемым языком программирования является:

а) *Pascal*;
б) *C*;
в) *Java Script*;
г) Ада;
д) Фортран.

3 На этапе синтеза компилятор выполняет:

а) группировку символов в лексемы;
б) синтаксический разбор программы;
в) проверку семантической корректности программы;
г) заполнение таблиц идентификаторов;
д) генерацию объектной программы.

4 Процесс последовательного чтения компилятором данных из внешней памяти, их обработки и помещения результатов во внешнюю память, называется:

а) этапом компиляции;
б) проходом компилятора;
в) фазой компиляции;
г) результатом компиляции;
д) методологией компиляции.

5 Этап компиляции, на котором символы исходной программы, группируются в отдельные минимальные единицы текста, несущие смысловую нагрузку:

а) лексический анализ;
б) синтаксический анализ;
в) семантический анализ;
г) генерация кода;
д) оптимизация кода.

6 Лексический анализ проводится путем разбора по:

а) МП-автомату;
б) конечному автомату;

в) машине Тьюринга;
г) ограниченному линейному автомату;
д) МП-машине.

7 Синтаксический разбор осуществляется с применением:
а) регулярных грамматик, выровненных влево;
б) КЗ-грамматик;
в) фразовых грамматик;
г) КС-грамматик;
д) регулярных грамматик, выровненных вправо.

8 Многоадресный код с явно именуемым результатом называется:
а) тетрадами;
б) триадами;
в) ПОЛИЗом;
г) синтаксическим деревом;
д) машинной командой.

9 При записи выражения *X*:=*Y**2/*Z*+*X**10 в виде тетрад, последней будет тетрада вида:
а) +(*T*2, *T*3, *T*4);
б) *(*X*, 10, *T*4);
в) *(*X*, 10, *T*5);
г) :=(*X*, *T*3,);
д) :=(*X*, *T*4,).

10 Последовательности триад 1) *(*Y*, 2); 2) -(*Z*, ^1); 3) /(^2, 2); соответствует выражение:
а) (*Y**2-*Z*)/2;
б) 2/(*Z*-*Y**2);
в) 2/(*Y**2-*Z*);
г) (*Z*-*Y**2)/2;
д) (*Z*-*Y*/2)*2.

11 Результат вычисления последовательности триад 1) /(8, 2); 2) -(6, ^1); 3) *(^2, 3):
а) 6;
б) -6;
в) 4;
г) -4;
д) 8.

12 Дерево операций арифметического выражения может содержать вершины, помеченные:
а) начальным символом грамматики;
б) нетерминалами грамматики;

в) (и);
г) идентификаторами;
д) ε.

13 ПОЛИЗ выражения (X-2)/3-4*(Z+2) имеет вид:
а) X2-/34Z2+*-;
б) X2-3/4Z*2+-;
в) X2-3/4Z2+*-;
г) X2-3/-4Z2+*;
д) X2-3/4*Z2+-.

14 ПОЛИЗ условного оператора *if B then* S_1 *else* S_2 имеет вид:
а) $B\ p_2\ !F\ S_1\ p_1\ !\ S_2\ \ldots$;
$p_1\ p_2$
б) $B\ p_1\ !F\ S_1\ p_2\ !\ S_2\ \ldots$;
$p_1\ p_2$
в) $B\ p_1\ !\ S_1\ p_2\ !F\ S_2\ \ldots$;
$p_1\ p_2$
г) $B\ p_1\ !F\ S_1\ S_2\ p_2\ !\ \ldots$;
$p_1\ p_2$
д) $p_1\ B!F\ S_1\ p_2\ !\ S_2\ \ldots$
$p_1\ p_2$

15 Оператор цикла *while B do S* в ПОЛИЗе записывается:
а) $B\ p_0\ !F\ S\ p_1\ !...$;
p_o p_1
б) $B\ p_1\ !F\ S\ p_o\ !...$;
p_o p_1
в) $B\ p_1\ !\ S\ p_o\ !\ F...$;
p_o p_1
г) $B\ p_1\ !\ S\ p_o\ !...$;
p_o p_1
д) $p_1\ B\ !F\ S\ p_o\ !...$
p_o p_1

16 Значение выражения, записанного в ПОЛИЗе 1 2 + 3 / 2 4 6 + * + равно:
а) 18;
б) 51;
в) 20;
г) 53,5;
д) 21.

17 При интерпретации ПОЛИЗа 7 0 > 11 !F $\underline{x}$ 5 := 13 ! $\underline{x}$ 9 := переменная x получит значение, равное:
а) 5;
б) 7;

в) 9;
г) 13;
д) 11.

18 Фрагмент кода *mov ax*, *oper*1; *add ax*, *oper*2; соответствует узлу дерева операций, у которого:
а) оба потомка – узлы;
б) левый потомок – узел, правый – лист;
в) оба потока – листья;
г) правый потомок – узел, левый – лист;
д) один потомок – узел.

19 Если левый потомок узла дерева для операции «+» – лист, а правый – узел, то ему соответствует псевдокод:
а) *add ax*, *oper*2; *Code*(*U*);
б) *Code*(*U*); *add ax*, *oper*2;
в) *Code*(*U*); *mov ax*, *oper*l; *add ax*, *dx*; *mov dx*, *ax*;
г) *Code*(*U*); *mov ax*, *oper*l; *mov dx*, *ax*; *add ax*, *dx*;
д) *Code*(*U*); *mov dx*, *ax*; *mov ax*, *oper*l; *add ax*, *dx*.

20 В линейном участке программы 1) *P*:= @*X*; 2) *A*:= *X*; 3) *X*:= *Y*/*Z*; 4) *D*:=*P*^-*Z*; 5) *X*:= *D*/*Z*; 6) *A*:= *X*+*Y*; лишним являются присваивания :
а) 3;
б) 2;
в) 6;
г) 5;
д) 2 и 3.

21 Выполнение во время компиляции тех операций исходной программы, для которых значения операндов уже известны, называется:
а) чисткой циклов;
б) удалением бесполезных операций;
в) перестановкой операций;
г) сверткой кода;
д) арифметическим преобразованием.

22 Алгоритм свертки кода выдал промежуточный результат: 1) *C*(3, 0); 2):=(*A*, 2); 3) :=(*A*, 4); 4) *C*(15, 0); 5) +(^4, *A*); 6) :=(*B*,^5); Таблица: (*A*, 4).
а) после обработки 5-ая триада будет +(15; *A*);
б) после обработки 5-ая триада будет *C*(19; 0);
в) после обработки 5-ая триада будет +(^4; 4);
г) после обработки 5-ая триада будет +(15; 4);
д) после обработки 5-ая триада будет *C*(15; 4).

23 В таблице 4.9 дан фрагмент алгоритма исключения лишних операций.

Таблица 4.9 – Фрагмент алгоритма исключения лишних операций

Обрабатываемая триада *j*	Числа зависимости переменных			Числа зависимости триад *dep*(*j*)	Выходная последовательность триад
	A	*B*	*C*		
…	…	…	…	…	…
2) +(*A*, ^l)	0	0	1	2	2) +(*A*, ^1)
3) := (*C*, ^2)					

После обработки 3-ей триады получаем:
а) *dep*(*C*) = 3;
б) *dep*(*C*) = 1;
в) *dep*(*C*) = 2;
г) *dep*(*C*) = 4;
д) *dep*(*C*) = 5.

24 В таблице 4.9 дан фрагмент алгоритма исключения лишних операций. После обработки 3-ей триады получаем:
а) *dep*(3) = 2;
б) *dep*(3) = 1;
в) *dep*(3) = 3;
г) *dep*(3) = 4;
д) *dep*(3) = 5.

25 Значение *true* предопределяет логическую операцию:
а) конъюнкция;
б) дизъюнкция;
в) импликация;
г) отрицание;
д) эквивалентность.

5 Формальные методы описания перевода

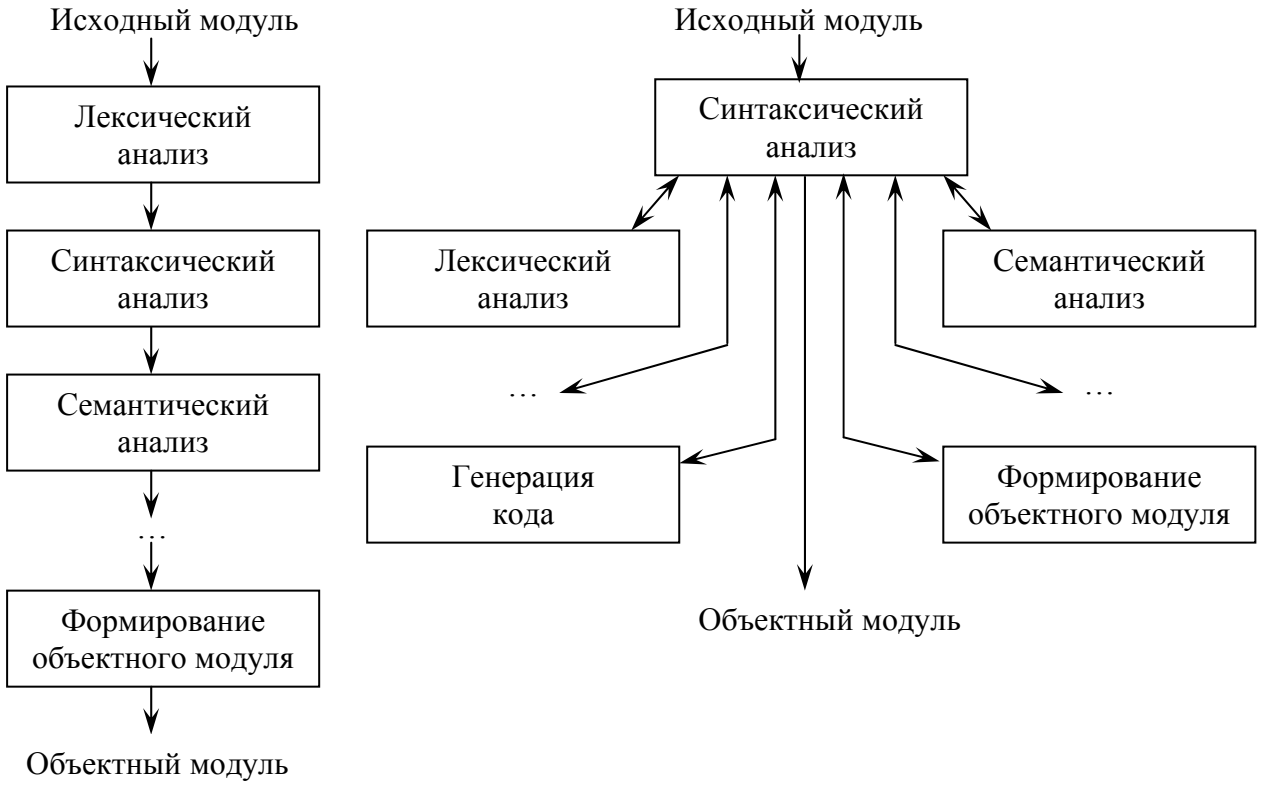
5.1 Синтаксически управляемый перевод

5.1.1 Схемы компиляции

Выделяют две основные схемы компиляции:
- последовательную;
- интегрированную.

Определение 5.1. Последовательная схема компиляции – это совокупность последовательно выполняемых программных компонентов, каждый из которых соответствует одному этапу компиляции.

Последовательная схема предполагает не менее одного прохода программы на каждом этапе (рисунок 5.1 а). Построение промежуточной программы не требует наличия разбора, и синтаксический анализатор может не выполнять никакого преобразования программы. Последовательная схема проста и понятна, но она громоздка (по объему занимаемой памяти и времени компиляции программы) и применяется редко.



а – последовательная схема *б* – интегрированная схема

Рисунок 5.1 - Схемы компиляции

Определение 5.2. Интегрированная схема компиляции – это схема, в которой компоненты выполняются под управлением синтаксического анализатора.

Синтаксический анализатор, осуществляя разбор программы, вызывает лексический анализатор для сборки лексемы, когда в этом возникает необходимость. После завершения распознавания каждой синтаксической единицы программы вызывается семантический анализатор для ее обработки. Главная идея состоит в том, чтобы в ходе построения дерева разбора решать задачи последующих этапов компиляции для каждого вновь сформированного узла.

Интеграция приводит к усложнению алгоритмов компиляции. Поэтому выбор подходящей структуры компилятора - сложная задача. Обычно строят компиляторы так, что на выходе синтаксического анализатора получается, как минимум, промежуточная программа в той или иной форме.

5.1.2 СУ-схемы

Интегрированные схемы компиляции базируются на теории перевода языков, основными понятиями которой являются схема синтаксически управляемого перевода (СУ-схема), синтаксически управляемый перевод (СУ - перевод), преобразователь с магазинной памятью (МП - преобразователь). Рассмотрим эти понятия.

Определение 5.3. СУ-схемой T называется пятерка следующих объектов:

$$T = \{V_T, V_N, \Delta, R, S\}, \tag{5.1}$$

где V_T - конечный входной алфавит, терминалы;

V_N - конечное множество нетерминалов;

Δ - конечный выходной алфавит;

R - конечное множество правил вида $A \to \alpha, \beta$, где $\alpha \in V^*$, $\beta \in (V_N \cup \Delta)^*$;

S - начальный символ схемы.

Определение 5.4. СУ-схема называется простой, если в каждом правиле $A \to \alpha, \beta$ одноименные нетерминалы встречаются в α и β в одном и том же порядке.

Определение 5.5. СУ-схема называется постфиксной, если $\beta \in V_N{}^*\Delta^*$ в каждом правиле $A \to \alpha, \beta \in R$.

Определение 5.6. СУ-переводом τ, генерируемым СУ-схемой T, называется множество пар:

$$\tau(T) = \{(w, y) \mid (S, S) \Rightarrow *(w, y), w \in V_T^*, y \in \Delta^*\} \tag{5.2}$$

Определение 5.7. Грамматика $G_{\text{ВХ}} = \{V_T, V_N, P, S\}$, где $P = \{A \to \alpha \mid A \to \alpha, \beta \in R\}$, называется входной грамматикой СУ-схемы. Грамматика $G_{\text{ВЫХ}} = \{\Delta, V_N, P', S\}$, где $P' = \{A \to \beta \mid A \to \alpha, \beta \in R\}$, называется выходной грамматикой СУ-схемы.

Пример 5.1. Рассмотрим СУ-схему T_1 перевода логических выражений в польскую обратную запись. В основе этой схемы лежит соответствие правил записи логических выражений в инфиксной форме и в ПОЛИЗе.

Правила инфиксной формы:	Правила ПОЛИЗа:
$S \to E$	$S \to E$
$E \to E \vee T$	$E \to ET \vee$
$E \to T$	$E \to T$
$T \to T \wedge F$	$T \to TF \wedge$
$T \to F$	$T \to F$
$F \to (E)$	$F \to E$
$F \to \neg F$	$F \to F \neg$
$F \to$ имя	$F \to$ имя

СУ-схема T_1 представляется пятеркой $T_1 = \{V_T, V_N, \Delta, R, S\}$, где S – начальный символ, $V_T = \{\vee, \wedge, \neg,$ имя, $(,)\}$; $\Delta = \{\vee, \wedge, \neg,$ имя$\}$; $V_N = \{S, E, T, F\}$ и множество R содержит следующие правила:

1) $S \to E, E$ 2) $E \to E \vee T, ET\vee$ 3) $E \to T, T$ 4) $T \to T \wedge F, TF\wedge$
5) $T \to F, F$ 6) $F \to (E), E$ 7) $F \to \neg F, F\neg$ 7) $F \to$ имя, имя

Здесь имя – любой символ латинского алфавита. Входной грамматикой СУ-схемы T_1 является $G_{\text{ВХ}} = (V_T, V_N, P, S)$, где множество правил P представлено правилами инфиксной формы. $G_{\text{ВЫХ}} = (\Delta, V_N, P', S)$ - выходная грамматика СУ-схемы T_1, а ее правила P' - это правила ПОЛИЗа. T_1 - простая постфиксная СУ-схема. Вывод элемента $(a \vee b \wedge c, abc\wedge\vee)$ имеет вид:

$(S, S) \Rightarrow (E \vee T, ET\vee) \Rightarrow (E \vee T \wedge F, ETF\wedge\vee) \Rightarrow (E \vee T \wedge c, ETc\wedge\vee) \Rightarrow (E \vee F \wedge c, EFc\wedge\vee)$
$\Rightarrow (E \vee b \wedge c, Ebc\wedge\vee) \Rightarrow (T \vee b \wedge c, Tbc\wedge\vee) \Rightarrow (F \vee b \wedge c, Fbc\wedge\vee) \Rightarrow (a \vee b \wedge c, abc\wedge\vee)$.

СУ-схема T_1 порождает элемент перевода $(a \vee b \wedge c, abc\wedge\vee) \in \tau(T_1)$.

5.1.3 МП-преобразователи

МП-преобразователи позволяют формально описать действия, связанные не только с распознаванием синтаксических конструкций, но и с построением дерева разбора и его преобразованием, а также действия, которые имеют как синтаксический, так и семантический характер.

Определение 5.6. МП - преобразователем называют восьмерку вида:

$$P = (Q, T, N, \Delta, F, q_0, N_0, Z), \tag{5.3}$$

где Q - конечное множество состояний преобразователя;
T - конечный входной алфавит;
N - конечный магазинный алфавит;
Δ - конечный выходной алфавит;
F - отображение множества $(Q \times (T \cup \{\varepsilon\}) \times N)$ во множество всех подмножеств множества $(Q \times N^* \times \Delta^*)$, т.е.
F: $(Q \times (T \cup \{\varepsilon\}) \times N) \to P(Q \times N^* \times \Delta^*)$;
q_0 - начальное состояние преобразователя, $q_0 \in Q$;
N_0 - начальное содержимое магазина, $N_0 \in N$;
Z - множество заключительных состояний преобразователя, $Z \subseteq Q$.

Определение 5.7. Конфигурация МП-преобразователя - это четверка вида:

$$(q, \omega, \alpha, y) \in (Q \times T^* \times N^* \times \Delta^*). \tag{5.4}$$

Начальная конфигурация - $(q_0, \omega, N_0, \varepsilon)$, заключительная конфигурация - $(q, \varepsilon, \alpha, y)$, $q \in Z$.

Если одним из значений функции $F(q, t, n)$ является (q', γ, r), то шаг работы преобразователя можно представить в виде отношения на конфигурациях:

$$(q, t\omega, n\alpha, y) \vdash (q', \omega, \alpha\gamma, yr), \forall(\omega \in T^*, \gamma \in N^*, y \in \Delta^*). \qquad (5.5)$$

Определение 5.8. Строка y будет выходом МП-преобразователя для строки ω, если существует путь от начальной до заключительной конфигурации:

$$(q_0, \omega, N_0, \varepsilon) \vdash^* (q, \varepsilon, \alpha, y), q \in Z, \alpha \in N^*. \qquad (5.6)$$

Определение 5.9. Переводом (преобразованием) τ, определяемым МП-преобразователем P, называется множество:

$$\tau(P) = \{(\omega, y) | (q_0, \omega, N_0, \varepsilon) \vdash^* (q, \varepsilon, \alpha, y), q \in Z, \alpha \in N^*\}. \qquad (5.7)$$

Общая схема МП-преобразователя показана на рисунке 5.1.

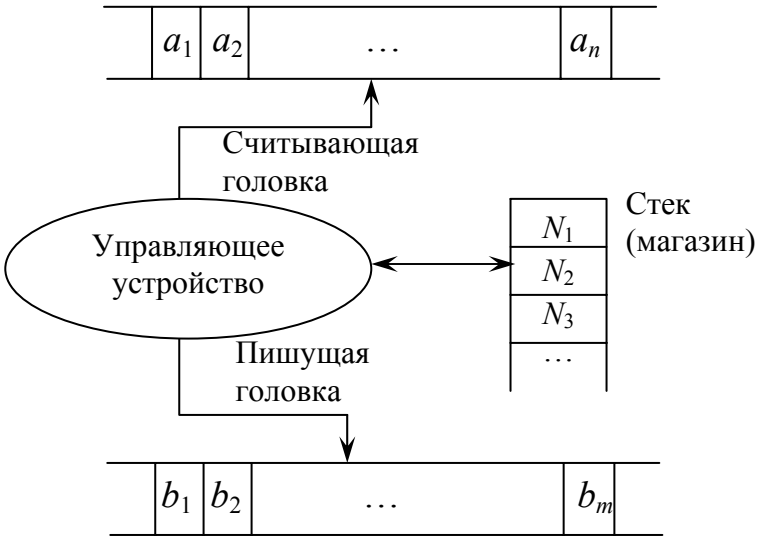


Рисунок 5.1 – Общая схема МП-преобразователя

Определение 5.10. МП-преобразователь будет детерминированным (ДМП-преобразователем), если он имеет не более одной возможной очередной конфигурации.

Определение 5.11. Расширенный МП-преобразователь есть МП-преобразователь, магазинная функция которого имеет вид:

$$F: (Q \times (T \cup \{\varepsilon\}) \times N^*) \to P(Q \times N^* \times \Delta^*). \qquad (5.8)$$

Пример 5.2. Рассмотрим перевод τ_2, отображающий каждую цепочку $x \in \{a, b\}^* \perp$, в которой число вхождений символа a равно числу вхождений сим-

вола b, в цепочку $y = (ab)^n$, где n – число вхождений символа a и b в цепочку x. Например, $\tau_2(abbaab\perp) = ababab$.

Этот перевод может быть реализован ДМП-преобразователем $P = (\{q_0, q_z\}, \{a, b, \perp\}, \{Z, a, b\}, \{a, b\}, F, q_0, Z, \{q_z\})$ с функцией переходов вида:

$F(q_0, X, Z) = \{(q_0, XZ, \varepsilon)\}, X \in \{a, b\};$
$F(q_0, \perp, Z) = \{(q_z, Z, \varepsilon)\};$
$F(q_0, X, X) = \{(q_0, XX, \varepsilon)\}, X \in \{a, b\};$
$F(q_0, X, Y) = \{(q_0, \varepsilon, ab)\}, X \in \{a, b\}, Y \in \{a, b\}, X \neq Y.$

В теории перевода доказаны две теоремы, имеющие важное практическое значение.

Теорема 5.1. Если $T = \{V_T, V_N, \Delta, R, S\}$ - простая СУ-схема с входной грамматикой $LL(k)$, то СУ-перевод $\tau(T)$ можно осуществить детерминированным МП-преобразователем.

Теорема 5.2. Если $T = \{V_T, V_N, \Delta, R, S\}$ - простая постфиксная СУ-схема с входной грамматикой $LR(k)$, то перевод $\tau(T)$ можно выполнить детерминированным МП - преобразователем.

Существуют алгоритмы, позволяющие построить детерминированный МП-преобразователь по заданной СУ-схеме перевода. В их основе лежат алгоритмы построения таблиц разбора.

Простые СУ-переводы образуют важный класс переводов, т.к. для каждого из них можно построить детерминированный МП-преобразователь, отображающий входную строку в выходную строку. Такие переводы называют цепочечными.

5.1.4 Практическое применение СУ-схем

СУ-схемы предполагают существование отображения вида:

$$f: P_1 \rightarrow P_2, \tag{5.9}$$

где P_1 - множества правил грамматики исходного языка;

P_2 - множество правил грамматики объектного языка.

Выделяют два метода построения объектной программы путем преобразования исходной программы.

1 СУ-компиляция – строит объектную программу по ходу синтаксического анализа. В этом случае строить полное дерево разбора исходной программы и сохранять его после синтаксического анализа не требуется.

2 СУ-перевод – строит объектную программу после синтаксического анализа по его результату, представленному в виде дерева разбора.

Рассмотрим метод построения объектной программы на основе СУ-перевода. В основу метода положено преобразование дерева разбора строки во входной грамматике $G_{\text{ВХ}}$ в дерево разбора выходной строки y в грамматике $G_{\text{ВЫХ}}$. Метод позволяет получить перевод (ω, y) любой входной строки ω путем последовательного решения следующих трех задач:

1) построение дерева разбора строки ω в грамматике $G_{\text{ВХ}}$;

2) преобразование полученного дерева в дерево разбора соответствующей строки y в грамматике $G_{\text{ВЫХ}}$, используя правила СУ-схемы;

3) получение выходной строку y по кроне дерева ее разбора.

Первая и третья задачи уже разобраны. Рассмотрим решение первой.

Алгоритм 5.1. Преобразование дерева разбора входной строки.

Введем обозначения:

1) A - произвольный нетерминал СУ-схемы;

2) $A \to \alpha$ - правило входной грамматики, где $\alpha = n_1 \ldots n_k$.

Пусть нетерминалу A соответствует узел A дерева разбора. Тогда узлы n_1, n_2, …, n_k - прямые потомки узла A. Преобразование дерева разбора начинается с его корня и состоит в следующем.

Шаг 1. Выбрать очередной нетерминальный узел A дерева разбора входной строки; если все узлы обработаны, завершить алгоритм.

Шаг 3. Устранить листья из множества узлов n_1, …, n_k (вершины, помеченные терминалами или ε);

Шаг 3. Найти в СУ-схеме правило вида ($A \to \alpha, \beta$) и переставить оставшихся прямых потомков узла A в соответствии с их размещением в строке β (поддеревья перемещать вместе с их корнями).

Шаг 4. Добавить в качестве прямых потомков узла A листья так, чтобы метки всех его прямых потомков образовали цепочку β.

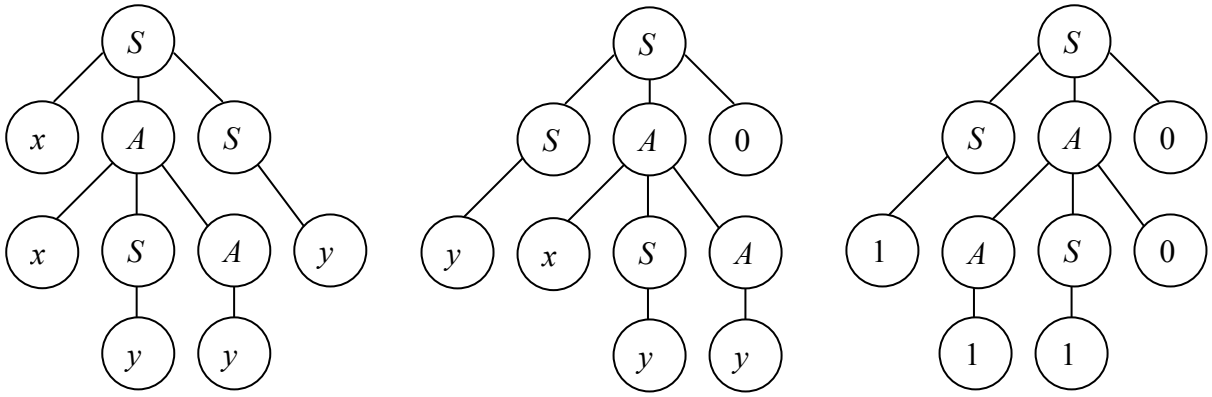
Повторять шаги 1-4 для всех прямых нетерминальных потомков узла A по порядку слева направо.

Конец алгоритма.

Пример 5.3. Дана СУ-схема $T_2 = (\{x, y\}, \{S, A\}, \{0, 1), R, S)$, где R содержит правила:

1) $S \to xAS, SA0$ 2) $A \to xSA, AS0$ 3) $S \to y, 1$ 4) $A \to y, 1$.

На рисунке 5.2 а показано дерево разбора входной строки $xxyyy$. Применение алгоритма преобразования к корню S этого дерева устраняет левый лист, помеченный x (шаг 2).



а) б) в)

Рисунок 5.2 - Преобразование дерева разбора: *а* - дерево входной строки;
б - дерево после однократного применения алгоритма;
в - дерево выходной строки

Далее, так как корню соответствует правило $S \to xAS$ и для этого правила $\beta = SA0$, нужно поменять местами оставшихся прямых потомков корня (шаг 3). Затем следует добавить 0 в качестве самого правого прямого потомка (шаг 4). Результатом будет дерево, показанное на рисунке 5.2 б. Далее применяем алгоритм к первым двум потомкам корня. Обработка второго из них приводит еще к двукратному повторению алгоритма. Окончательный результат показан на рисунке 5.2 в, это дерево разбора выходной строки 11100. Правил СУ-схемы T_2 указывают на то, что она является не простой постфиксной схемой.

5.2 Транслирующие грамматики

В простых СУ-схемах можно объединить левые и правые части соответствующих правил входной и выходной грамматик. Получающиеся таким образом грамматики принято называть транслирующими грамматиками (Т-грамматиками). Символы выходного алфавита Δ в Т-грамматике называют операционными символами. При записи Т-грамматики каждый операционный символ во избежание путаницы каким-либо образом выделяют. Будем заключать такой символ в прямоугольные скобки.

Пример 5.4. Определение Т-грамматики для перевода инфиксных логических выражений в ПОЛИЗ: $V_T = \{\vee, \wedge, \neg,$ имя, (,)$\}$; $\Delta = \{[\vee], [\wedge], [\neg],$ [имя]$\}$; S – начальный символ, правила R имеют вид:

1) $S \to E$ 2) $E \to E \vee T[\vee]$ 3) $E \to T$ 4) $T \to T \wedge F[\wedge]$
5) $T \to F$ 6) $F \to (E)$ 7) $F \to \neg F[\neg]$ 7) $F \to$имя[имя]

В результате удаления операционных символов из Т-грамматики будет получена входная грамматика. Если удалить из Т-грамматики терминалы, получим выходную грамматику, или грамматику действий**.**

Т-грамматика описывает правила образования строк из терминалов и операционных символов. Эти строки называют активными цепочками. Если удалить из активной цепочки операционные символы, то получится ее входная часть. Если удалить из активной цепочки терминалы, получится выходная часть, или, иначе, трансляция, перевод входной части. Активную цепочку можно рассматривать как программу управления работой МП-преобразователя, соответствующего Т-грамматике. При этом входной символ можно считать обозначением операции чтения этого символа, а операционный символ - обозначением операции выдачи этого символа.

Пример 5.5. Активная цепочка $c[c] \vee b[b] \wedge a[a][\wedge][\vee]$ описывает процесс трансляции входной цепочки $c \vee b \wedge a$ (исключили из активной цепочки операционные символы) в ПОЛИЗ $[c][b][a][\wedge][\vee]$ (исключили из активной цепочки терминалы). Действительно, существует вывод:

$$S \Rightarrow E \Rightarrow E \vee T[\vee] \Rightarrow T \vee T[\vee] \Rightarrow F \vee T[\vee] \Rightarrow c[c] \vee T[\vee] \Rightarrow c[c] \vee T \wedge F[\wedge][\vee] \Rightarrow$$
$$\Rightarrow c[c] \vee F \wedge F[\wedge][\vee] \Rightarrow c[c] \vee b[b] \wedge F[\wedge][\vee] \Rightarrow c[c] \vee b[b] \wedge a[a][\wedge][\vee].$$

Пара ($c \vee b \wedge a$, [c][b][a][$\wedge$][$\vee$]) принадлежит переводу инфиксных выражений в ПОЛИЗ. В этом примере каждый операционный символ обозначает выдачу соответствующего входного символа в выходную строку.

5.3 Атрибутные транслирующие грамматики

5.3.1 Синтезируемые и наследуемые атрибуты

Каждому символу формальной грамматики присущ класс и значение. Например, если символом E обозначено арифметическое выражение, то с ним может быть связано числовое значение, адрес памяти для числа, тип числа, признак обнаруженной в выражении ошибки и т.д. Все это значения символа E.

Найти значение строки КС-языка можно, вычислив атрибуты в каждом узле дерева ее разбора. Атрибуты вычисляют по формулам, связанным с правилами грамматики языка. Атрибуты подразделяют на синтезируемые и наследуемые.

Синтезируемые атрибуты в некотором узле дерева зависят только от атрибутов в узлах-потомках. Они служат для передачи информации по дереву снизу вверх, а в правилах грамматики - из правой части в левую.

Наследуемые атрибуты в некотором узле являются функциями атрибутов его узла-предка и (или) узлов-потомков этого предка. Такие атрибуты могут распространять информацию в противоположном направлении.

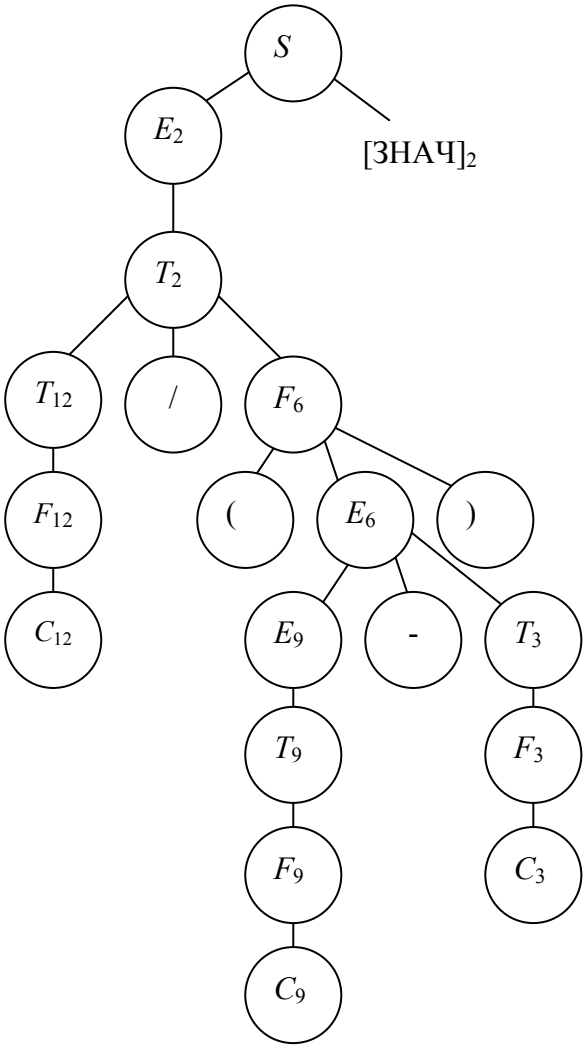
КС-грамматика, снабженная атрибутами, сохраняет свои обычнее свойства и вместе с тем позволяет формально учесть контекстно-зависимые условия языка. А элементы контекстной зависимости присутствуют во всех языках программирования.

Рассмотрим суть синтезируемых атрибутов на примере.

Пример 5.6. Пусть синтаксический анализатор арифметических выражений выдает численное значение выражения. В этом примере операндами могут быть только константы. Т-грамматика выражений содержит единственный операционный символ (обозначим его [ЗНАЧ]) и имеет вид:

1) $S \to E$[ЗНАЧ] 2) $E \to E\text{-}T$ 3) $E \to T$
4) $T \to T/F$ 5) $T \to F$ 6) $F \to (E)$ 7) $F \to C$
где $V_N = \{S, E, T, F\}$,
$V_T = \{\text{-}, /, (,), C\}$,
S – начальный символ;
C - константа, значение которой во внутреннем представлении построил лексический анализатор.

Возьмем строку $C_{12}/(C_9 - C_3)$ в качестве входной для синтаксического анализатора. В этом выражении подстрочными индексами указаны значения констант, которые и будем считать их атрибутами. Построим дерево разбора активной цепочки $C_{12}/(C_9 - C_3)$[ЗНАЧ]$_2$ и снабдим нетерминалы (в узлах дерева) значениями синтезируемых атрибутов (рисунок 5.3 *а*).



а) дерево разбора

1) $S \rightarrow E_q[\text{ЗНАЧ}]_r$
   $r = q$

2) $E_p \rightarrow E_q - T_r$
   $p = q - r$

3) $E_p \rightarrow T_q$
   $p = q$

4) $T_p \rightarrow T_q/F_r$
   $p = q/r$

5) $T_p \rightarrow F_q$
   $p = q$

6) $F_p \rightarrow (E_q)$
   $p = q$

7) $F_p \rightarrow C_q$
   $p = q$

б) АТ-грамматика

Рисунок 5.3 – Разбор активной цепочки $C_{12}/(C_9 - C_3)[\text{ЗНАЧ}]_2$

Значение выражения формируется из значений его подвыражений по правилам (формулам), соответствующим правилам грамматики. Представим эти формулы математически, для чего в каждом правиле грамматики предусмотрим различные имена для разных значений атрибутов, а затем запишем соответствующие формулы. Атрибутом снабжается каждый нетерминал правила грамматики. Имена атрибутов обычно записывают как подстрочные индексы. Например, правилу $E \rightarrow E - T$ транслирующей грамматики соответствуют правило $E_p \rightarrow E_q - T_r$ и формула $p = q - r$. Обозначения атрибутов в различных правилах не имеют между собой ничего общего и могут быть одинаковыми.

Полученные таким образом правила называют атрибутными, а грамматику с атрибутными правилами и формулами для атрибутов - атрибутной транслирующей грамматикой (АТ-грамматикой). Атрибутная транслирующая грамматика для нашего примера дана на рисунке 5.3 б. Заметим, что атрибут операционного символа [ЗНАЧ] не относится к числу синтезируемых, он - наследуемый.

Теперь поясним смысл наследуемых атрибутов на примере.

Пример 5.7. Пусть синтаксический анализатор описания переменных помещает признак типа (*real*, *int* или *bool*) имени в соответствующее поле таблицы имен.

Пусть Т-грамматика описания переменных имеет вид:

1) *описание* $\rightarrow$ *T V* [тип] *список*

2) *список* $\rightarrow$, *V* [тип] *список*

3) *список* $\rightarrow \varepsilon$

В этой грамматике V_N = {«*описание*», «*список*»}; V_T = {*T*, *V*, ,}; Δ = {[тип]}. Здесь *V* - символ класса имен со значением, представленным указателем на элемент таблицы имен; *T* - символ класса ключевых слов со значением, представленным указателем на *real*, *int* или *bool*. Запятая - символ класса ограничителей. Операционный символ [тип] располагается в правилах вслед за именем *V* и предписывает поместить признак типа *T* в поле «тип» записи переменной *V* таблицы имен. Поэтому значение [тип] определяется парой атрибутов («указ», «тип»), где «указ» - указатель на запись *V* в таблице имен, «тип» - значение *T*.

Введем атрибуты и формулы для их вычисления в грамматику описания переменных. Будем использовать имена t_i, где i = 0, 1, 2, ..., для обозначения атрибута «тип» (значение *T*), именами p_i обозначим атрибут «указатель» (значение *V*). Для передачи типа переменной из правила 1 в последующие снабдим нетерминал «список» атрибутом типа. Тогда АТ-грамматика описания переменных будет иметь вид:

1) *описание* $\rightarrow T_{t0}\ V_{P0}$ [тип]$_{p1,\ t1}$ *список*$_{t2}$
$t_2,\ t_1 = t_0;\ p_1 = p_0$

2) *список*$_{t0}$ $\rightarrow$, V_{p0} [тип]$_{p1,\ t1}$ *список*$_{t2}$
$t_2,\ t_1 = t_0;\ p_1 = p_0$

3) *список*$_{t0}$ $\rightarrow \varepsilon$

Формула $t_2,\ t_1 = t_0$ означает, что атрибуты t_2 и t_1 получают значение атрибута t_0. Согласно формуле $p_1 = p_0$ атрибут p_1 получит значение атрибута p_0.

Возьмем в качестве примера описание *int x*, *y*1, *max*. Синтаксический анализатор получит входную строку, которая в наших обозначениях выглядит так: $T_{int}\ V_1,\ V_2,\ V_3$. Здесь 1, 2, 3 - номера позиций имен в таблице имен. Дерево разбора соответствующей активной цепочки $T_{int}\ V_1$[тип]$_{1,\ int}$, V_2[тип]$_{2,\ int}$, V_3[тип]$_{3,\ int}$ показано на рисунке 5.4.

5.3.2 Определение и свойства АТ-грамматики

Определение 5.12. АТ-грамматика - это транслирующая грамматика, дополненная следующим образом.

1 Каждый терминал, нетерминал и операционный символ имеет конечное множество атрибутов, и каждый атрибут имеет множество (возможно, бесконечное) допустимых значений.

2 Все атрибуты нетерминалов и операционных символов делятся на наследуемые и синтезируемые.

3 Наследуемые атрибуты вычисляются следующим образом:

а) значение наследуемого атрибута из правой части правила грамматики вычисляется как функция некоторых других атрибутов символов, входящих в правую или левую часть данного правила;

б) начальные значения наследуемых атрибутов начального символа грамматики полагаются известными.

4 Синтезируемые атрибуты вычисляются так:

а) значение синтезируемого атрибута нетерминала из левой части правила грамматики вычисляется как функция некоторых других атрибутов символов из левой или правой части данного правила;

б) значение синтезируемого атрибута операционного символа вычисляется как функция некоторых других атрибутов этого символа.

5 Значения атрибутов терминалов считаются заданными, они не относятся ни к наследуемым, ни к синтезируемым.

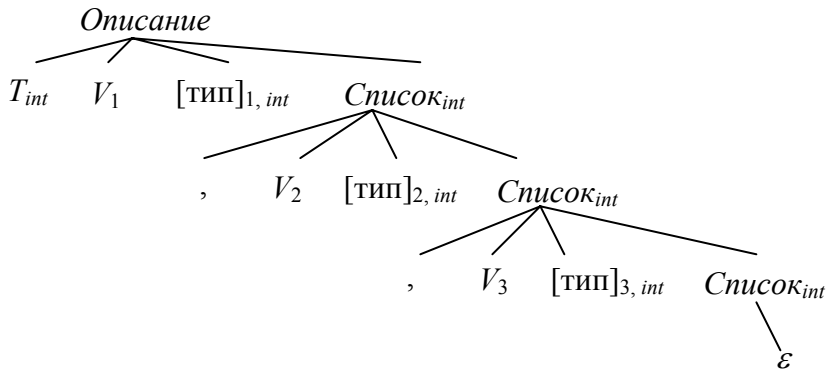


Рисунок 5.4 - Дерево разбора активной цепочки
$T_{int}\ V_1[\text{тип}]_{1,\ int},\ V_2[\text{тип}]_{2,\ int},\ V_3[\text{тип}]_{3,\ int}$

Пример 5.8. В грамматике выражений (пример 5.6) атрибуты всех нетерминалов - синтезируемые, а в грамматике описаний (пример 5.7) - наследуемые. В обоих грамматиках атрибуты операционных символов - наследуемые. Значения атрибутов терминалов устанавливает лексический анализатор. Атрибуты отдельных символов не указывались, поскольку не играли никакой роли.

АТ-грамматики используют для построения атрибутных деревьев разбора, атрибутных активных цепочек и атрибутных переводов (трансляций).

Алгоритм 5.2. Построение атрибутного дерева разбора.

1 По Т-грамматике построить дерево разбора активной цепочки, состоящей из терминалов и операционных символов без атрибутов.

2 Присвоить значения атрибутам терминалов, входящих в дерево разбора.

3 Присвоить начальные значения наследуемым атрибутам начального символа грамматики на дереве разбора.

4 Вычислять значения атрибутов символов на дереве, пока это возможно,

по правилу:

- найти атрибут, которого еще нет на дереве, но аргументы для функции его вычисления уже известны;
- вычислить значение этого атрибута;
- разместить атрибут на дереве.

Конец алгоритма.

Если по завершении пункта 4 значения всех атрибутов всех символов дерева оказываются вычисленными, то такое дерево называют завершенным.

Причиной незавершенности дерева может оказаться круговая зависимость между атрибутами, входящими в одну формулу.

Определение 5.13. АТ-грамматика, обеспечивающая завершенность любого дерева разбора, называется корректной.

Именно корректные грамматики представляют интерес для разработчиков трансляторов.

Рассмотрим подклассы АТ-грамматик, которым соответствуют детерминированные МП-машины.

Определение 5.14. АТ-грамматика называется *L*-АТ-грамматикой, если выполняются следующие условия (ограничения на АТ-грамматику).

1 Значение наследуемого атрибута символа из правой части правила может зависеть только от наследуемых атрибутов левой части правила и от любых атрибутов символов, стоящих в правой части правила левее рассматриваемого символа.

2 Значение синтезируемого атрибута нетерминала из левой части правила может зависеть только от наследуемых атрибутов этого символа и от любых атрибутов символов правой.

3 Значение синтезируемого атрибута операционного символа зависит только от наследуемых атрибутов этого символа.

Условие 1 делает наследуемые атрибуты некоторого узла дерева разбора зависящими (прямо или косвенно) от атрибутов терминалов, расположенных на дереве левее данного узла. Отсюда и наименование грамматики (*Left* - левая). Условия 2 и 3 делают грамматику корректной.

Определение 5.15. АТ-грамматика называется польской, если все операционные символы встречаются только в качестве последних символов правых частей правил грамматики.

Определение 5.16. *L*-АТ-грамматика называется *S*-атрибутной, если все атрибуты нетерминалов - синтезируемые.

Атрибутную трансляцию, описанную формально АТ-грамматикой, можно выполнить с помощью атрибутного МП-преобразователя, который называют также атрибутным МП-процессором или атрибутной МП-машиной. Выполнение атрибутной трансляции предполагает следующие действия МП-преобразователя:

- последовательное чтение входных символов вместе с их атрибутами;
- проверку принадлежности входной строки языку, определенному входной грамматикой;
- выдачу последовательности атрибутных операционных символов, соот-

ветствующей входной строке из активной атрибутной цепочки.

Атрибутный МП-преобразователь представляет собой обычный МП-преобразователь, у которого состояния, входные, выходные и магазинные символы имеют атрибуты. На каждом шаге работы преобразователя значения атрибутов нового состояния, нового верхнего символа магазина (если этот шаг - не выталкивание из магазина) и выходных символов вычисляются как функции атрибутов старого состояния, верхнего символа магазина и входного символа (если это не пустой шаг).

Сформулирован ряд важных теорем, основные из которых дадим без доказательства.

Теорема 5.3. Любая трансляция, определяемая L-АТ-грамматикой, может быть выполнена недетерминированной атрибутной МП-машиной.

Теорема 5.4. Любую трансляцию, определяемую L-АТ-грамматикой, у которой входная грамматика является $LL(k)$-грамматикой, можно выполнить на атрибутной детерминированной МП-машине с концевым маркером.

Теорема 5.5. Всякая трансляция, определяемая некоторой S-атрибутной польской транслирующей грамматикой (S-АПТ-грамматикой) с входной $LR(k)$-грамматикой, может быть выполнена детерминированной атрибутной МП-машиной с концевым маркером.

Вопросы и задачи для самоконтроля по 5 разделу

1 В чем достоинства и недостатки последовательной и интегрированной схемы компиляции?

2 Какими элементами формально определяется СУ-схема?

3 При каких условиях СУ-схема называется простой, постфиксной? Приведите примеры.

4 Запишите формально множество пар, определяющих СУ-перевод, генерируемый СУ-схемой.

5 В чем принципиальные отличия в структуре и функционировании МП-распознавателя и МП-преобразователя?

6 Как формально можно определить перевод, задаваемый МП-преобразователем?

7 Какой МП-преобразователь называется расширенным, детерминированным?

8 Сформулируйте теоремы о соответствии между СУ-схемами и детерминированными МП - преобразователями.

9 Какие методы построения объектной программы по исходной программе существуют?

10 Приведите примеры практического применения СУ-схем.

11 Раскройте сущность метода построения объектной программы на основе СУ-перевода.

12 Как из простых СУ-схем можно получить Т-грамматику?

13 Какую роль выполняют в активных цепочках терминальные и операционные символы?

14 Как можно найти значение строки КС-языка?

15 В чем принципиальные различия между синтезируемыми и наследуемыми атрибутами символов АТ-грамматики?

16 Каким образом дополнить Т-грамматику до АТ-грамматики?

17 Сформулируйте алгоритм построения атрибутного дерева разбора.

18 Охарактеризуйте подклассы АТ-грамматик.

19 В грамматике [целое] $\rightarrow dC$; $C \rightarrow dC \mid \varepsilon$; терминал d имеет атрибут 0 и 1. Определить атрибуты так, чтобы нетерминал [целое] имел атрибут, равный восьмеричному значению выводимого числа.

Тесты проверки усвоения материала по 5 разделу

1 В интегрированной схеме компиляции компоненты выполняются под управлением:

а) лексического анализатора;

б) семантического анализатора;

в) генератора кода;

г) синтаксического анализатора;

д) оптимизатора кода.

2 Множество правил СУ-схемы $T = \{V_T, V_N, \Delta, R, S\}$ имеет вид:

а) $A \rightarrow \alpha, \beta$, где $\alpha \in V_T^*$, $\beta \in (V_N \cup \Delta)^*$;

б) $A \rightarrow \alpha, \beta$, где $\alpha \in V^*$, $\beta \in (V_N \cup \Delta)^*$;

в) $A \rightarrow \alpha$, где $\alpha \in (V_N \cup \Delta)^*$;

г) $A \rightarrow \alpha$, где $\alpha \in V^*$

д) $A \rightarrow \alpha \mid \beta$, где $\alpha \in V^*$, $\beta \in (V_N \cup \Delta)^*$.

3 Если в каждом правиле $A \rightarrow \alpha, \beta$ одноименные нетерминалы встречаются в α и β в одном и том же порядке, то СУ-схема называется:

а) простой;

б) постфиксной;

в) прямой;

г) детерминированной;

д) инфиксной.

4 СУ-схема называется постфиксной, если в каждом правиле $A \rightarrow \alpha, \beta \in R$ выполняется условие:

а) $\alpha \in V_N^* \Delta^*$;

б) $\alpha \in \Delta^* V_N^*$;

в) $\beta \in \Delta^* V_N^*$;

г) $\beta \in \Delta^* V_T^*$;

д) $\beta \in V_N^* \Delta^*$.

5 СУ-переводом τ, генерируемым СУ-схемой $T = \{V_T, V_N, \Delta, R, S\}$, называется множество пар:

а) $\tau(T) = \{(w, y) \mid (S, S) \Rightarrow (w, y), w \in V_T^*, y \in \Delta^*\}$;

б) $\tau(T) = \{(w, y) \mid (S, S) \Rightarrow *(w, y), w \in V_T, y \in V_N\}$;

в) $\tau(T) = \{(w, y) \mid (S, S) \to (w, y), w \in V_T^*, y \in \Delta^*\}$;

г) $\tau(T) = \{(w, y) \mid (S, S) \Rightarrow *(w, y), w \in V_T^*, y \in \Delta^*\}$;

д) $\tau(T) = \{(w, y) \mid (S, S) \Rightarrow *(w, y), w \in V_T^*, y \in V_N^*\}$.

6 Выходной грамматикой СУ-схемы $T = \{V_T, V_N, \Delta, R, S\}$ называется:

а) $G_{\text{ВЫХ}} = \{\Delta, V_N, P', S\}$, где $P' = \{A \to \beta \mid A \to \alpha, \beta \in R\}$;

б) $G_{\text{ВЫХ}} = \{\Delta, V_N, P', S\}$, где $P' = \{A \to \alpha \mid A \to \alpha, \beta \in R\}$;

в) $G_{\text{ВЫХ}} = \{V_T, V_N, P', S\}$, где $P' = \{A \to \beta \mid A \to \alpha, \beta \in R\}$;

г) $G_{\text{ВЫХ}} = \{V_T, V_N, P', S\}$, где $P' = \{A \to \alpha \mid A \to \alpha, \beta \in R\}$;

д) $G_{\text{ВЫХ}} = \{\Delta, V_T, P', S\}$, где $P' = \{A \to \beta \mid A \to \alpha, \beta \in R\}$.

7 Магазинная функция МП-преобразователя $P = (Q, T, N, \Delta, F, q_0, N_0, Z)$ имеет вид:

а) F: $(Q \times (T \cup \{\varepsilon\}) \times N) \to P(Q \times N^*)$;

б) F: $(Q \times (T \cup \{\varepsilon\}) \times N) \to P(N^* \times \Delta^*)$;

в) F: $(Q \times (T \cup \{\varepsilon\}) \times N) \to P(Q \times N^* \times \Delta^*)$;

г) F: $(Q \times (T \cup \{\varepsilon\}) \times N) \to P(Q \times \Delta^*)$;

д) F: $(Q \times N) \to P(Q \times N^* \times \Delta^*)$.

8 Если $F(q, t, n) = (q', \gamma, r)$, то шаг работы МП-преобразователя $P = (Q, T, N, \Delta, F, q_0, N_0, Z)$ можно представить в виде отношения на конфигурациях:

а) $(q, t\omega, n\alpha, y) \vdash^* (q', \omega, \alpha\gamma, yr)$, $\forall(\omega \in T^*, \gamma \in N^*, y \in \Delta^*)$;

б) $(q, t\omega, n\alpha, y) \vdash (q', \omega, \alpha\gamma, yr)$, $\forall(\omega \in T^*, \gamma \in N^*, y \in \Delta^*)$;

в) $(q, t\omega, n\alpha, y) \vdash (q', \omega, \alpha\gamma, yr)$, $\forall(\omega \in N^*, \gamma \in T^*, y \in \Delta^*)$;

г) $(q, t\omega, n\alpha, y) \vdash^* (q', \omega, \alpha\gamma, yr)$, $\forall(\omega \in T^*, \gamma \notin \Delta^*, y \in N^*)$;

д) $(q, t\omega, n\alpha, ry) \vdash (q', \omega, \alpha\gamma, y)$, $\forall(\omega \in T^*, \gamma \in N^*, y \in \Delta^*)$.

9 Строка y будет выходом МП-преобразователя $P = (Q, T, N, \Delta, F, q_0, N_0, Z)$ для строки ω, если существует путь:

а) $(q, \varepsilon, \alpha, y) \vdash^* (q_0, \omega, N_0, \varepsilon)$, $q \in Z$, $\alpha \in N^*$;

б) $(q_0, \varepsilon, N_0, \omega) \vdash^* (q, \varepsilon, \alpha, y)$, $q \in Z$, $\alpha \in N^*$;

в) $(q_0, \omega, N_0, \varepsilon) \vdash^* (q, \varepsilon, \alpha, y)$, $q \in Z^*$, $\alpha \in N^*$;

г) $(q_0, \omega, N_0, \varepsilon) \vdash^* (q, \varepsilon, \alpha, y)$, $q \in Z$, $\alpha \in N^*$;

д) $(q_0, \omega, N_0, \varepsilon) \vdash^* (q, \varepsilon, \alpha, y)$, $q \in Z$, $\alpha \in N$.

10 Переводом τ, определяемым МП-преобразователем $P = (Q, T, N, \Delta, F, q_0, N_0, Z)$, называется множество:

а) $\tau(P) = \{(\omega, y) \mid (q_0, \omega, N_0, \varepsilon) \vdash^* (q, \varepsilon, \alpha, y), q \in Z, \alpha \in N^*\}$;

б) $\tau(P) = \{(\omega, y) | (q_0, y, N_0, \varepsilon) \text{ |-*} (q, \varepsilon, \alpha, \omega), q \in Z, \alpha \in N^*\}$;
в) $\tau(P) = \{(\omega, y) | (q_0, \omega, N_0, \varepsilon) \text{ |-} (q, \varepsilon, \alpha, y), q \in Z, \alpha \in N^*\}$;
г) $\tau(P) = \{(\omega, y) | (q_0, \omega, N_0, \varepsilon) \text{ |-*} (q, \varepsilon, \alpha, y), q \in Z^*, \alpha \in N\}$;
д) $\tau(P) = \{(\omega, y) | (q_0, y, N_0, \varepsilon) \text{ |-*} (q, \varepsilon, \alpha, \omega), q \in Z, \alpha \in N\}$.

11 Если МП-преобразователь имеет не более одной возможной очередной конфигурации, то он является:
а) минимальным;
б) постфиксным;
в) расширенным;
г) недетерминированным.
д) детерминированным.

12 СУ-перевод $\tau(T)$ можно осуществить детерминированным МП-преобразователем, если:
а) $T = \{V_T, V_N, \Delta, R, S\}$ - простая постфиксная СУ-схема с входной грамматикой $LL(k)$;
б) $T = \{V_T, V_N, \Delta, R, S\}$ - простая СУ-схема с входной грамматикой $LR(k)$;
в) $T = \{V_T, V_N, \Delta, R, S\}$ - постфиксная СУ-схема с входной грамматикой $LL(k)$;
г) $T = \{V_T, V_N, \Delta, R, S\}$ - простая СУ-схема с входной грамматикой $LL(k)$;
д) $T = \{V_T, V_N, \Delta, R, S\}$ - простая СУ-схема с входной грамматикой $LL(1)$.

13 Если $T = \{V_T, V_N, \Delta, R, S\}$ - простая постфиксная СУ-схема с входной грамматикой $LR(k)$, то перевод $\tau(T)$ можно выполнить:
а) недетерминированным МП – преобразователем;
б) расширенным МП – преобразователем;
в) детерминированным МП – преобразователем;
г) МП – преобразователем;
д) детерминированным МП - распознавателем.

14 СУ-компиляция строит объектную программу:
а) по ходу синтаксического анализа;
б) по результатам синтаксического анализа;
в) по ходу семантического анализа;
г) по результатам семантического анализа;
д) по ходу лексического анализа.

15 Метод построения объектной программы после синтаксического анализа по его результату, представленному в виде дерева разбора:
а) СУ-компиляция;
б) СУ-перевод;

в) СУ-трансляция;
г) СУ-интерпретация;
д) СУ-эмуляция.

16 СУ-схема с правилами R: 1) $S \to aSA$, $AS1$; 2) $A \to a$, 1; 3) $S \to b$, 0 является:
а) не простой, не постфиксной;
б) простой, постфиксной;
в) простой, не постфиксной;
г) не простой, расширенной;
д) не простой, постфиксной.

17 СУ-схема с правилами R: 1) $S \to aSA$, $AS1$; 2) $A \to a$, 1; 3) $S \to b$, 0 переводит строку *aabaa* в строку:
а) 00100;
б) *aa*1*aa*;
в) *bb*0*bb*;
г) 11011;
д) *aa*1*bb*.

18 При объединении левых и правых частей соответствующих правил входной и выходной грамматик простых СУ-схем получаются:
а) грамматики предшествования;
б) атрибутные грамматики;
в) транслирующие грамматики;
г) атрибутные транслирующие грамматики;
д) автоматные грамматики.

19 Символы выходного алфавита в Т-грамматике называют:
а) операционными;
б) терминальными;
в) нетерминальными;
г) активными;
д) денатационными.

20 Грамматика действий получается из Т-грамматики путем удаления:
а) нетерминальных символов;
б) терминальных символов;
в) операционных символов;
г) атрибутных символов;
д) активных символов.

21 Активные цепочки Т-грамматики состоят из:
а) терминалов и операционных символов;
б) терминалов и нетерминалов;

в) нетерминалов и операционных символов;
г) операционных символов;
д) терминалов.

22 Выходная строка для активной цепочки $x[x]+y[y]*z[z][*][+]$ имеет вид:
а) $x+y*z$;
б) $[x]+[y]*[z]$;
в) $xyz*+$;
г) $x+yz*$;
д) $xy\ z+*$.

23 Синтезируемые атрибуты в некотором узле дерева зависят от:
а) атрибутов его узла-предка и (или) узлов-потомков этого предка;
б) атрибутов узлов-потомков и узла-предка;
в) атрибутов его узла-предка;
г) атрибутов в узлах-потомках;
д) атрибутов узлов-потомков его предка.

24 Если все операционные символы встречаются только в качестве последних символов правых частей правил АТ-грамматики, то она называется:
а) *S*-грамматикой;
б) *S*-атрибутной;
в) *R*-АТ-грамматикой;
г) *L*-АТ-грамматикой;
д) польской.

25 *L*-АТ-грамматика называется *S*-атрибутной, если:
а) все атрибуты терминалов – синтезируемые;
б) все атрибуты нетерминалов – синтезируемые;
в) все атрибуты нетерминалов – наследуемые;
г) все атрибуты терминалов – наследуемые;
д) все атрибуты операционных символов – наследуемые.

6 Итоговые тесты проверки усвоения материала

1 Даны цепочки $\alpha = 010$ и $\beta = ac$. Порядок следования составных цепочек по возрастанию их длины:
а) α^2; β^R; α^R; $\alpha\beta$; β^2;
б) β^R; α^R; α^2; β^2; $\alpha\beta$;
в) $\alpha\beta$; α^R; α^2; β^R; β^2;
г) β^R; α^R; $\alpha\beta$; α^2; $\alpha^2\beta^R$;
д) $\alpha\beta$; α^R; β^R; β^2; α^2.

2 В регулярных выражениях знак «*» определяет операцию:

а) реверс;

б) итерация;

в) альтернатива;

г) конкатенация;

д) пересечение.

3 В грамматике $G=(\{a, b, c, d\}, \{A, C\}, \{A \to aACd \mid b; C \to c \mid \varepsilon\}, A)$ правосторонним является вывод:

а) $A \Rightarrow aACd \Rightarrow abCd \Rightarrow abdd$;

б) $A \Rightarrow aACd \Rightarrow aAcd \Rightarrow aaACdcd \Rightarrow aaAdcd \Rightarrow aabdcd$;

в) $A \Rightarrow aACd \Rightarrow aaACdCd \Rightarrow aaAcdCd \Rightarrow aaAcdd \Rightarrow aabcdd$;

г) $A \Rightarrow aACd \Rightarrow aAdd \Rightarrow abcd$;

д) $A \Rightarrow aACd \Rightarrow aAcd \Rightarrow acd$.

4 Грамматика является неоднозначной, если имеются правила вида ($A \in V_N$, $\alpha, \beta, \gamma \in V^*$):

а) $A \to A\alpha \mid \alpha$;

б) $A \to A\alpha A \mid A\beta$;

в) $A \to \alpha A \mid \alpha A\beta \mid \gamma$;

г) $A \to A\alpha A \mid \beta$;

д) $A \to \alpha A \mid \beta$.

5 Программа, которая переводит каждую команду исходной программы в одну машинную команду, называется:

а) транслятором;

б) интерпретатором;

в) ассемблером;

г) компилятором;

д) эмулятором.

6 Если для каждого правила $A \to \alpha, \beta \in R$ выполняется условие $\beta \in V_N{}^*\Delta^*$, то СУ-схема называется:

а) простой;

б) постфиксной;

в) прямой;

г) детерминированной;

д) инфиксной.

7 Язык, состоящий из пустой строки и всевозможных строк, содержащих строку нулей и последующую строку единиц произвольной длины:

а) $L=\{(01)^n \mid n>0\}$;

б) $L=\{(01)^n \mid n\geq 0\}$;

в) $L=\{0^n1^n \mid n\geq 0\}$;

г) $L=\{0^n1^n \mid n>0\}$;
д) $L=\{0^n1^m \mid n, m\geq 0\}$.

8 Операция «альтернатива» в регулярных выражениях обозначается знаком:
а) *;
б) ^;
в) |;
г) ·;
д) -.

9 Грамматика, для которой существует язык:
а) $S\rightarrow aB \mid cA$; $A \rightarrow BaA \mid a$; $B \rightarrow A \mid b$;
б) $S\rightarrow aB \mid cA$; $A \rightarrow BaA$; $B \rightarrow A \mid bB$;
в) $S\rightarrow aS \mid cA$; $A \rightarrow BaA$; $B \rightarrow A \mid b$;
г) $S\rightarrow aS \mid cB$; $A \rightarrow BaA \mid a$; $B \rightarrow AB \mid bB$;
д) $S\rightarrow aA \mid cA$; $A \rightarrow BaA \mid aA$; $B \rightarrow A \mid b$.

10 Множество бесполезных символов грамматики с правилами $P=\{S\rightarrow SS|A$; $B \rightarrow 0C \mid 1$; $A \rightarrow 0A1 \mid C \mid 0$; $C \rightarrow BC \mid CS\}$:
а) $\{B\}$;
б) $\{C\}$;
в) $\{A, C\}$;
г) $\{B, C\}$;
д) $\{A, B\}$.

11 Этап компиляции, на котором проводится проверка правильности конструкций программы (выражений, описаний, операторов и др.), образованных из лексем:
а) лексический анализ;
б) синтаксический анализ;
в) семантический анализ;
г) генерация кода;
д) оптимизация кода.

12 Входной грамматикой СУ-схемы $T = \{V_T, V_N, \Delta, R, S)$ называется:
а) $G_{\text{ВХ}} = \{\Delta, V_N, P', S)$, где $P' = \{A\rightarrow \beta \mid A\rightarrow \alpha, \beta \in R\}$;
б) $G_{\text{ВХ}} = \{\Delta, V_N, P', S)$, где $P' = \{A\rightarrow \alpha \mid A\rightarrow \alpha, \beta \in R\}$;
в) $G_{\text{ВХ}} = \{V_T, V_N, P', S)$, где $P' = \{A\rightarrow \beta \mid A\rightarrow \alpha, \beta \in R\}$;
г) $G_{\text{ВХ}} = \{V_T, V_N, P', S)$, где $P' = \{A\rightarrow \alpha \mid A\rightarrow \alpha, \beta \in R\}$;
д) $G_{\text{ВХ}} = \{\Delta, V_T, P', S)$, где $P' = \{A\rightarrow \beta \mid A\rightarrow \alpha, \beta \in R\}$.

13 Строка вида *abbb* принадлежит языку:
а) $L=\{(ab)^n \mid n>0\}$;
б) $L=\{(ab)^n \mid n\geq 0\}$;

в) $L=\{a^n b^n \mid n\geq 0\}$;
г) $L=\{a^n b^n \mid n>0\}$;
д) $L=\{ab^n \mid n\geq 0\}$.

14 Строки, образованные из последовательности 0 и последующей последовательности 1, включая пустую строку, начинающиеся с 0, описываются регулярным выражением:
а) $0+(0|1)^*$;
б) $0(0|1)^*$;
в) $0(01)^*$;
г) $0(0^*1^*)$;
д) $0^*(0|1)$.

15 Символ C в грамматике с правилами $P=\{S\to A|B;\ B\to AB \mid Ba;\ A\to aB \mid bS \mid b;\ C\to AS \mid b\}$ является:
а) нетерминалом, не порождающим терминальных строк;
б) недостижимым нетерминалом;
в) недостижимым терминалом;
г) достижимым нетерминалом;
д) достижимым терминалом.

16 После удаления ε-правил, грамматика с правилами $P = \{S \to aSbS \mid bSaS \mid \varepsilon\}$ примет вид:
а) $P' = \{S \to abS \mid aSb \mid ab \mid baS \mid bSa \mid ba\}$;
б) $P' = \{S'\to S;\ S \to abS \mid aSb \mid ab \mid baS \mid bSa \mid ba \mid aSbS \mid bSaS\}$;
в) $P' = \{S'\to S \mid \varepsilon,\ S \to abS \mid aSb \mid ab \mid baS \mid bSa \mid ba \mid aSbS \mid bSaS\}$;
г) $P' = \{S'\to S \mid \varepsilon,\ S \to abS \mid ab \mid baS \mid ba\}$;
д) $P' = \{S'\to S \mid \varepsilon,\ S \to abS \mid aSb \mid baS \mid bSa\}$.

17 Синтаксический анализ проводится путем разбора по:
а) МП-автомату;
б) конечному автомату;
в) машине Тьюринга;
г) ограниченному линейному автомату;
д) МП-преобразователю.

18 Магазинная функция расширенного МП-преобразователя $P = (Q, T, N, \Delta, F, q_0, N_0, Z)$ имеет вид:
а) F: $(Q\times(T\cup\{\varepsilon\})\times N)\to P(Q\times N^*)$;
б) F: $(Q\times(T\cup\{\varepsilon\})\times N)\to P(N^*\times\Delta^*)$;
в) F: $(Q\times(T\cup\{\varepsilon\})\times N)\to P(Q\times N^*\times\Delta^*)$;
г) F: $(Q\times(T\cup\{\varepsilon\})\times N^*)\to P(Q\times N^*\times\Delta^*)$;
д) F: $(Q\times N)\to P(Q\times N^*\times\Delta^*)$.

19 Для грамматики с правилами вывода вида $S \to Sab \mid ab$ эквивалентной будет грамматика:

а) $S \to AB$; $A \to a|Aa$; $B \to b|Bb$;

б) $S \to aSb|\varepsilon$;

в) $S \to SA|ab$; $A \to ab|\varepsilon$;

г) $S \to aAb$; $aA \to aaAb$; $A \to \varepsilon$;

д) $S \to AS|SB|AB$; $A \to a$; $B \to b$.

20 Языку, описываемому регулярным выражением $1(01)^*0$, принадлежит строка:

а) 10101;

б) 01010;

в) 101010;

г) 10100;

д) 11010.

21 Правила вида $A \to Ab$, где $A \in V_N$, $b \in V_T$, называются:

а) леворекурсивными;

б) цепными;

в) факторизованными;

г) бесполезными;

д) праворекурсивными.

22 После устранения цепных правил грамматика $P = \{S \to A \mid B;\ A \to AA \mid a;\ B \to aB \mid b \mid C;\ C \to cC \mid c\}$ примет вид:

а) $P' = \{S \to AA \mid a \mid aB \mid b;\ A \to AA \mid a;\ B \to aB \mid b \mid cC \mid c;\ C \to cC \mid c\}$;

б) $P' = \{S \to AA \mid a \mid aB \mid b \mid cC \mid c;\ A \to AA \mid a;\ B \to aB \mid b \mid C;\ C \to cC \mid c\}$;

в) $P' = \{S \to AA \mid a \mid aB \mid b \mid cC \mid c;\ A \to AA \mid a;\ B \to aB \mid b \mid cC \mid c;\ C \to cC \mid c\}$;

г) $P' = \{S \to AA \mid a \mid aB \mid b;\ A \to AA \mid a;\ B \to aB \mid b \mid C;\ C \to cC \mid c\}$;

д) $P' = \{S \to AA \mid aB \mid cC;\ A \to AA \mid a;\ B \to aB \mid b \mid cC \mid c;\ C \to cC \mid c\}$.

23 Лексический разбор осуществляется с применением:

а) регулярных грамматик;

б) КЗ-грамматик;

в) фразовых грамматик;

г) КС-грамматик;

д) грамматик предшествования.

24 СУ-перевод $\tau(T)$ можно осуществить детерминированным МП-преобразователем, если:

а) $T = \{V_T, V_N, \Delta, R, S\}$ - простая СУ-схема с входной грамматикой $LR(k)$;

б) $T = \{V_T, V_N, \Delta, R, S\}$ - постфиксная СУ-схема с входной грамматикой $LR(k)$;

в) $T = \{V_T, V_N, \Delta, R, S\}$ - простая постфиксная СУ-схема с входной грамматикой $LR(k)$;

г) $T = \{V_T, V_N, \Delta, R, S\}$ - простая постфиксная СУ-схема с входной грамматикой $LL(k)$;

д) $T = \{V_T, V_N, \Delta, R, S\}$ - постфиксная СУ-схема с входной грамматикой $LL(k)$.

25 Для грамматики с правилами $S \to SaSb \mid \varepsilon$ справедливо:

а) $S \to ababab$;

б) $S \Rightarrow ababab$;

в) $S \Rightarrow^* abababb$;

г) $S \Rightarrow^* abaabb$;

д) $S \Rightarrow aaabbb$.

26 Конечный автомат, представленный на рисунке 6.1 допускает строку:

а) *inmj*;

б) *jn/m*;

в) *jjmj*;

г) *jnmm*;

д) *inmk*.

27 Магазинная функция расширенного МП-автомата $M = (Q, T, N, F, q_0, N_0, Z)$ задает отображение вида:

а) $F: (Q \times (T \cup \{\varepsilon\}) \times N) \to P(Q \times N^*)$;

б) $F: (Q \times (T^* \cup \{\varepsilon\}) \times N) \to P(Q \times N^*)$;

в) $F: (Q \times T \times N) \to P(Q \times N^*)$;

г) $F: (Q \times (T \cup \{\varepsilon\}) \times Z) \to P(Q \times N^*)$;

д) $F: (Q \times (T \cup \{\varepsilon\}) \times N^*) \to P(Q \times N^*)$.

28 Начальная конфигурация МП-автомата $M = (Q, T, N, F, q_0, N_0, Z)$:

а) (q_0, ω, N_0), где $\omega \in T^*$;

б) (q_0, ω, N_0), где $\omega \in N^*$;

в) (q, ω, N_0), где $q \in Q$, $\omega \in T^*$;

г) (q, ε, α), где $q \in Z$, $\alpha \in T^*$;

д) $(q_0, \omega, \varepsilon)$, где $\omega \in T^*$.

29 При записи выражения (*Y**2+*X*/5)**Z* в виде тетрад, последней будет тетрада вида:

а) /(*T*3, 5, *T*4);

б) +(*T*3, *T*4, *T*5);

в) +(*T*2, *T*3, *T*4);

г) *(*T*2, *Z*, *T*3);
д) *(*T*3, *Z*, *T*4).

30 СУ-перевод строит объектную программу:
а) по ходу синтаксического анализа;
б) по результатам синтаксического анализа;
в) по ходу семантического анализа;
г) по результатам семантического анализа;
д) по ходу лексического анализа.

31 Грамматика с правилами вывода вида $S \to bAb$; $A \to AA \mid a$ по иерархии Хомского является:
а) грамматикой без ограничений;
б) КС-грамматикой;
в) регулярной грамматикой, выровненной право;
г) КЗ-грамматикой;
д) регулярной грамматикой, выровненной влево.

32 Для конечного автомата, представленного на рисунке 6.1, справедливо:
а) (*I*, *mnk*) |- (*L*, *nk*);
б) (*I*, *nmk*) |- (*M*, *mk*);
в) (*I*, *jk*) |- (*K*, *k*);
г) (*I*, *nmk*) |- (*L*, *mk*);
д) (*I*, *jk*) |-* (*N*, *k*).

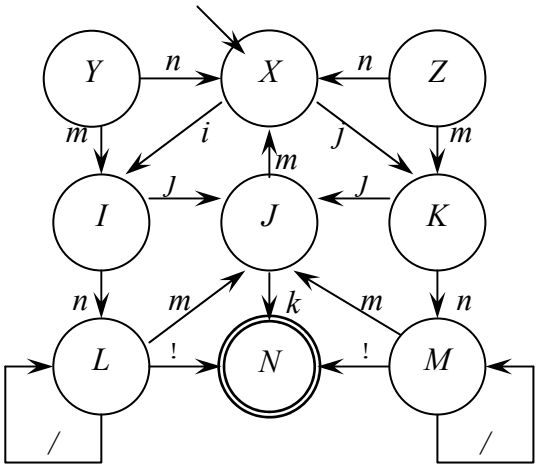


Рисунок 6.1 – Граф конечного автомата

33 Язык, распознаваемый МП-автоматом $M = (Q, T, N, F, q_0, N_0, Z)$, определяется как множество вида:

а) $L(M) = \{\omega \mid \omega \in T^*$ и $(q_0, \omega, N_0) \mid\!= (q, \varepsilon, \alpha)$, где $q \in Z$ и $\alpha \in N^*\}$;

б) $L(M) = \{\omega \mid \omega \in T^*$ и $(q, \omega, N_0) \mid\!= *(q, \varepsilon, \alpha)$, где $q \in Z$ и $\alpha \in N^*\}$;

в) $L(M) = \{\omega \mid \omega \in T^*$ и $(q_0, \omega, N_0) \mid\!= *(q, \varepsilon, \alpha)$, где $q \in Z$ и $\alpha \in T^*\}$;

г) $L(M) = \{\omega \mid \omega \in T^*$ и $(q_0, \varepsilon, N_0) \mid\!= *(q, \omega, \alpha)$, где $q \in Z$ и $\alpha \in T^*\}$;

д) $L(M) = \{\omega \mid \omega \in T^*$ и $(q_0, \omega, N_0) \mid\!= *(q, \varepsilon, \alpha)$ где $q \in Z$ и $\alpha \in N^*\}$.

34 Последовательности триад 1) *(10, *X*); 2) *(*X*, *Y*); 3) -(^1, ^2); 4) /(^3, 2) соответствует выражение:

а) (*X***Y* - 10**X*)/2;

б) 2/(10**X*-*X***Y*);

в) (10**Y*-*X***X*)/2;

г) (10**X*-*X***Y*)/2;

д) 2/(10**Y*-*X***X*).

35 Метод построения объектной программы по ходу синтаксического анализа:

а) СУ-компиляция;

б) СУ-перевод;

в) СУ-трансляция;

г) СУ-интерпретация;

д) СУ-эмуляция.

36 В классификации грамматик по Хомскому справедливо:

а) фразовые грамматики – подмножество КС-грамматик;

б) КС-грамматики – подмножество регулярных грамматик;

в) фразовые грамматики – подмножество КЗ-грамматик;

г) КЗ-грамматики – подмножество КС-грамматик;

д) КС-грамматики – подмножество КЗ грамматик.

37 Для КА, граф которого задан на рисунке 6.1, справедливо:

а) *F*(*J*, *m*)=*M*;

б) *F*(*X*, *m*)=*J*;

в) *F*(*I*, *m*)=*Y*;

г) *F*(*K*, *m*)=*Z*;

д) *F*(*I*, *n*)=*L*.

38 Достаточные условия применимости метода рекурсивного спуска выполняются в правиле:

а) $A \to aS \mid cB \mid c$;

б) $A \to Sa \mid bB \mid c$;

в) $A \to Sa \mid Bb$;

г) $A \to aS \mid bB \mid c$;

д) $A \to aS \mid abB \mid c$.

39 Результат вычисления последовательности триад 1) +(8, 4); 2) /(^1, 4); 3) *(2, 1); 4) –(^2, ^3):

а) 1;
б) -1;
в) 2;
г) -2;
д) 0.

40 СУ-схема с правилами R: 1) $S \to 1AS, ASa$; 2) $A \to 1, a$; 3) $S \to 0, b$ является:
а) не простой, не постфиксной;
б) простой, постфиксной;
в) простой, не постфиксной;
г) не простой, расширенной;
д) не простой, постфиксной.

41 Грамматика, правила вывода которой имеют вид $A \to \beta$, где $A \in V_N$ и $\beta \in V^*$, является:
а) фразовой;
б) КС-грамматикой;
в) регулярной грамматикой, выровненной вправо;
г) КЗ-грамматикой;
д) регулярной грамматикой, выровненной влево.

42 Для КА $M=(Q, T, F, H, Z)$, граф которого задан на рисунке 6.1, справедливо:
а) $H=\{X\}, Z=\{N\}, T=\{n, m, i, j, k, !, /\}$;
б) $H=\{N\}, Z=\{X\}, T=\{n, m, i, j, k, !, /\}$;
в) $H=\{X\}, Z=\{N\}, T=\{n, m, i, j, k\}$;
г) $H=\{X\}, Z=\{N\}, T=\{X, Y, Z, I, J, K, L, M, N\}$;
д) $H=\{N\}, Z=\{X\}, T=\{X, Y, Z, I, J, K, L, M, N\}$.

43 Множество $\{\omega \in V_T^* \mid \exists$ вывод $S \Rightarrow^* \alpha A \gamma$ и $\omega \in FIRST(k, \gamma)$; $\alpha, \gamma \in V^*, A \in V_N, k>0\}$ формально определяет:
а) $FOLLOW(k, A)$;
б) $LL(k)$;
в) $L(A)$;
г) $R(A)$;
д) $LR(k)$ - свойство.

44 ПОЛИЗ выражения $X/(5+X*Y-Z)$ имеет вид:
а) $X\ 5\ /\ X\ Y\ *\ +\ Z\ -$;
б) $X\ 5\ X\ Y\ +\ *\ Z\ -\ /$;
в) $X\ 5\ X\ Y\ *\ +\ Z\ -\ /$;
г) $X\ 5\ X\ Y\ *\ +\ Z\ /\ -$;
д) $X\ 5\ X\ Y\ *\ +\ Z\ -$.

45 СУ-схема с правилами R: 1) $S \to 1AS$, ASa; 2) $A \to 1$, a; 3) $S \to 0$, b переводит строку 11110 в строку:

а) 01111;

б) *aabaa*;

в) *bbabb*;

г) 00001;

д) *aa*1*bb*.

46 Правила вывода регулярной грамматики, выровненной влево, имеют вид:

а) $A \to aB \mid a$, где $a \in V_T$; $A, B \in V_N$;

б) $A \to Ba \mid a$, где $a \in V_T$; $A, B \in V_N$;

в) $A \to \beta$, где $A \in V_N$ и $\beta \in V^*$;

г) $\alpha \to \beta$, где $\alpha \in (V_T \cup V_N)^*$, $\beta \in (V_T \cup V_N)^+$ и $|\alpha| \geq |\beta|$;

д) $\alpha \to \beta$, где $\alpha \in V_N{}^+$, $\beta \in V_T{}^*$ и $|\alpha| \leq |\beta|$.

47 Граф КА M=(Q, T, F, H, Z) задан на рисунке 6.1. Недостижимыми состояниями автомата являются:

а) M, N;

б) K, L;

в) Y, Z;

г) I, J;

д) X, Y, Z.

48 В грамматике с правилами $P = \{S \to S{+}T \mid T;\ T \to a \mid S(S)\}$ справедливо:

а) $FIRST(S) = \{a\}$;

б) $FIRST(T) = \{S, T\}$;

в) $FIRST(T) = \{a, +\}$;

г) $FIRST(T) = \{S, T, a\}$;

д) $FIRST(T) = \{a, (,), +\}$.

49 Значение выражения, записанного в ПОЛИЗе 4 7 3 2 * - 1 + / равно:

а) 10;

б) 3;

в) 28;

г) -2;

д) 2.

50 КС-грамматика, позволяющая формально учесть контекстно-зависимые условия языка и найти значение строки, называется:

а) грамматикой предшествования;

б) КЗ-грамматикой;

в) транслирующей грамматикой;

г) атрибутной грамматикой;

д) автоматной грамматикой.

51 Итерация синтаксической конструкции 0 и более раз в БНФ обозначается:

а) ( );
б) { };
в) « »;
г) < >;
д) [ ].

52 Граф КА $M=(Q, T, F, H, Z)$ задан на рисунке 6.1. Эквивалентным являются состояния:

а) I и K;
б) L и M;
в) I и L, K и M;
г) I и L;
д) I и K, L и M.

53 Основой сентенции $x_1 <\cdot x_2 \cdot> x_3 <\cdot x_4 =\cdot x_5 =\cdot x_6 \cdot> x_7 \cdot> x_8$ является строка:

а) $x_5 x_6 x_7$;
б) $x_2 x_3$;
в) $x_3 x_4$;
г) $x_4 x_5 x_6$;
д) $x_3 x_4 x_5 x_6$.

54 При интерпретации ПОЛИЗа условного оператора 10 7 < 11 !F $\underline{x}$ 20 := 13 ! $\underline{x}$ 15 := переменная x получит значение, равное:

а) 15;
б) 20;
в) 13;
г) 11;
д) 10.

55 Синтезируемые атрибуты передают информацию:

а) по дереву сверху снизу, в правилах грамматики - из правой части в левую;

б) по дереву снизу вверх, в правилах грамматики - из левой части в правую;

в) по дереву снизу вверх, в правилах грамматики - из правой части в левую;

г) по дереву сверху внизу, в правилах грамматики - из левой части в правую;

д) по дереву снизу вверх, в правилах грамматики – произвольно.

56 Множеству строк, задаваемых конструкцией БНФ вида {/0/}|{/1/} принадлежат строки:

а) ε;
б) 000111;

в) 000000;
г) 001100;
д) 001110.

57 Дана грамматика с правилами P: 1) $X \to yY \mid zZ$; 2) $Y \to 1V$; 3) $Z \to 0W \mid 0Y$; 4) $V \to xZ \mid xW \mid 1$; 5) $W \to 1Y \mid 0$. Для эквивалентного ей КА справедливо:
а) $F(Z, 0)=WY$;
б) $F(Z, 0)=W$;
в) $F(Y, 1)=W$;
г) $F(X, y)=YZ$;
д) $F(X, z)=YZ$.

58 Для грамматики $G(\{t, i, ,\}, \{O, S, P\}, \{O \to t\ C; C \to i\ P \mid ,i\ P \mid i; P \to ,i\}, O\}$ справедливо:
а) $L(P) = \{,\}$;
б) $L(P) = \{t\}$;
в) $L(P) = \{i\}$;
г) $L(P) = \{, i\ \}$;
д) $L(P) = \{, i, P\}$.

59 Если оба потомка узла дерева для операции + являются листьями, то ему соответствует псевдокод:
а) *add ax*, *oper*2; *Code*(*U*);
б) *mov ax*, *oper*1; *add ax*, *oper*2;
в) *Code*(*U*); *mov ax*, *oper*l; *add ax*, *dx*; *mov dx*, *ax*;
г) *Code*(*U*); *mov ax*, *oper*l; *mov dx*, *ax*; *add ax*, *dx*;
д) *Code*(*U*); *mov dx*, *ax*; *mov ax*, *oper*l; *add ax*, *dx*.

60 Символами входного алфавита в Т-грамматике являются:
а) операционные символы;
б) терминальные символы;
в) нетерминальные символы;
г) активные символы;
д) терминальные и нетерминальные символы.

61 Терминальным символам на диаграммах Вирта соответствуют:
а) кружки;
б) скругленные прямоугольники;
в) ветвления дуг;
г) прямоугольники;
д) слияния дуг.

62 Дана грамматика с правилами P: 1) $X \to yY \mid zZ$; 2) $Y \to 1V$; 3) $Z \to 0W \mid 0Y$; 4) $V \to xZ \mid xW \mid x$; 5) $W \to 1Y \mid 1$. В эквивалентном ей КА конечными будут состояния:

а) V;
б) W;
в) Y, W;
г) Y, W, Z;
д) X.

63 Если символ в верхушке стека имеет большее простое предшествование, чем очередной входной символ, то:

а) МП-автомат выполняет свертку;
б) МП-автомат выполняет выброс;
в) МП-автомат выполняет сдвиг;
г) МП-автомат выдает ошибку;
д) МП-автомат успешно завершает работу.

64 Алгоритм свертки кода выдал промежуточный результат: 1) C(3, 0); 2):=(A, 2); 3) :=(A, 4); 4) *(3, A); 5) +(^4, A); 6) :=(B,^5); Таблица: (A, 4).

а) после обработки 4-ая триада будет *(12; A);
б) после обработки 4-ая триада будет C(12; 0);
в) после обработки 4-ая триада будет *(3; 4);
г) после обработки 4-ая триада будет *(12; 0);
д) после обработки 4-ая триада будет C(3; 4).

65 Входная строка для активной цепочки $c[c]*(a[a]+b[b])[+][*]$ имеет вид:

а) $[c]*([a]+[b])$;
б) $c\ a\ b+*$;
в) $c\ (a\ b)+*$;
г) $c*(a+b)$;
д) $c*a+b$.

66 Распознавателями для регулярных языков являются:

а) МП-автоматы;
б) конечные автоматы;
в) машины Тьюринга;
г) линейные ограниченные автоматы;
д) линейные неограниченные автоматы.

67 Дана грамматика с правилами P: 1) $X \to yY \mid zZ$; 2) $Y \to 1V$; 3) $Z \to 0W \mid 0Y$; 4) $V \to xZ \mid xW \mid x$; 5) $W \to 1Y \mid 1$. В эквивалентном ей КА входным будут символы:

а) x, y, z;
б) x, y, z, 0, 1;

в) 0, 1;
г) *X*, *Y*, *Z*, 0, 1;
д) *X*, *Y*, *Z*.

68 Справедливо утверждение:
а) *LR*-языки являются подмножеством *LL*-языков;
б) *LL*-языки являются подмножеством *LR*-языков;
в) КС-языки являются подмножеством *LL*-языков;
г) КС-языки являются подмножеством *LR*-языков;
д) языки простого предшествования являются подмножеством *LL*-языков.

69 Значение *false* предопределяет логическую операцию:
а) конъюнкция;
б) дизъюнкция;
в) импликация;
г) отрицание;
д) эквивалентность.

70 Наследуемые атрибуты в некотором узле дерева зависят от:
а) атрибутов его узла-предка и (или) узлов-потомков этого предка;
б) атрибутов узлов-потомков и узла-предка;
в) атрибутов его узла-предка;
г) атрибутов в узлах-потомках;
д) атрибутов узлов-потомков его предка.

Список использованных источников

1 Ахо, А.В. Компиляторы: принципы, технологии и инструменты [Текст]: / А. Ахо, Р. Сети, Д. Ульман; перевод с англ. И.В. Красикова и др. - М.: Вильямс, 2001. - 767 с.: ил.; 24 см. - Библиогр.: с. 742-763. - Предм. указ.: 764-767. - 5000 экз. - ISBN 5-8459-0189-8 (в пер.).

2 Власенко, А.В. Теория языков программирования и методы трансляции [Текст]: учеб. пособие / А.В. Власенко, В.И. Ключко; М-во образования и науки РФ, ГОУ ВПО «Кубан. гос. технол. ун-т». - Краснодар: Изд-во КубГТУ, 2004. - 119 с.: ил.; 21 см. - Библиогр.: с. 118. - 75 экз. - ISBN 5-8333-0176-9.

3 Гавриков, М.М. Основы конструирования компиляторов [Текст]: учеб. пособие / М.М. Гавриков, А.Н. Иванченко, Д.В. Гринченков; М-во общ. и проф. образования РФ, Новочеркас. гос. техн. ун-т. – Новочеркасск: НГТУ, 1997. - 80 с.: ил.; 20 см. - Библиогр.: с. 79. – 75 экз. - ISBN 5-88998-059-9.

4 Гордеев, А.В. Системное программное обеспечение [Текст]: учеб. для вузов / А. В. Гордеев, А.Ю. Молчанов; под. общ. ред. А.В. Гордеева. – СПб.: Питер, 2001. - 734 с.: ил.; 24 см. - Библиогр.: с. 719-724. – 5000 экз. – ISBN 5-272-00341-1 (в пер.).

5 Калинин, А.Г. Универсальные языки программирования: Семантический подход [Текст] / А.Г. Калинин, И.В. Мацкевич. – М.: Радио и связь, 1991. - 398, [1] с.: ил.; 21 см. - Библиогр.: с. 395-398. – 10000 экз. – ISBN 5-256-00638-X (в пер.).

6 Карпов, В.Э. Классическая теория компиляторов [Текст]: учеб. пособие / В.Э. Карпов; М-во образования РФ, Моск. гос. ин-т электрон. и математики (техн. ун-т). - М.: МГИЭМ, 2002. - 78 с.: ил.; 20 см. - Библиогр.: с. 78. - 150 экз. - ISBN 5-230-16344-5.

7 Компаниец, Р.И. Системное программирование: основы построения трансляторов [Текст]: учеб. пособие для высших и средних учебных заведений / Р.И. Компаниец, Е.В. Маньков, Н.Е. Филатов. - СПб.: Корона принт, 2000. - 254, [1] с.: ил.; 23 см. - Библиогр.: с. 255. - 3000 экз. - ISBN 5-7931-0124-1.

8 Кораблин, Ю.П. Семантика языков программирования [Текст]: учеб. пособие по курсу «Семантика языков программирования» / Ю.П. Кораблин; под общ. ред. В.П. Кутепова; М-во науки, высш. шк. и техн. политики Рос. Федерации, Моск. энерг. ин-т. - М.: Изд-во МЭИ, 1992. - 100 с.; 21 см. - Библиогр.: с. 99. – 500 экз.

9 Мозговой, М.В. Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход [Текст] / М.В. Мозговой. – СПб.: Наука и техника, 2006. - 320 с.: ил.; 24 см. - 3000 экз. - ISBN 5-94387-224-8.

10 Пратт, Т. Языки программирования: разработка и реализация [Текст] / Т. Пратт, М. Зелковиц; пер. с англ. - СПб.: Питер принт, 2002. - 688 с.: ил.; 24 см. - Библиогр.: с. 669-674. - Алф. указ.: с. 675-688. - Загл. и авт. ориг.: Programming languages / Terrence W. Pratt, Marvin V. Zelkowitz. - 4000 экз. - ISBN 5-318-00189-0 (в пер.).

11 Рейуорд-Смит, В. Теория формальных языков. Вводный курс [Текст] / В. Рейуорд-Смит; пер. с англ. Б.А. Кузьмина; под ред. И.Г. Шестакова. - М.: Радио и связь, 1988. - 127, [2] с.: ил.; 21 см. - Перевод изд.: A first course in formal language theory / V.J. Rayward Smith (Oxford etc.). - 30000 экз. - ISBN 5-256-00159-0.

12 Семантика языков программирования [Текст]: Сборник статей / Пер. с англ. А.Н. Бирюкова, В.А. Серебрякова; под общ. ред. В.М. Курочкина. – М.: Мир, 1980. – 395 с.; 22 см. - Библиогр. в конце статей. – 8000 экз.

13 Серебряков, В.А. Основы конструирования компиляторов [Текст] / В.А. Серебряков, М.П. Галочкин. - М.: Эдиториал УРСС, 2001. - 221, [1] с.: ил.; 20 см. - Библиогр. в конце кн. – 1000 экз. - ISBN 5-8360-0242-8.

14 Соколов, А.П. Системы программирования: теория, методы, алгоритмы [Текст]: учеб. пособие для студентов, обучающихся по направлению 654600 - Информатика и вычисл. техника / А.П. Соколов. – М.: Финансы и статистика, 2004. – 319, [1] с.: ил.; 21 см. - Библиогр.: с. 309-310. – Предм. указ.: с. 313-320. – 4000 экз. – ISBN 5-279-02770-7.

15 Теория и реализация языков программирования: учебное пособие по курсу теории и реализации языков программирования [Текст] / В.А. Серебряков, М.П. Галочкин, Д.Р. Гончар, М.Г. Фуругян. – М.: МЗ-Пресс, 2006. - 348, [1] с.: ил.; 20 см. - Библиогр.: с. 347-249. – 2000 экз. - ISBN 5-94073-094-9.

16 Хантер, Р. Основные концепции компиляторов [Текст]: / Р. Хантер; пер. с англ. - М.: Вильямс, 2002. - 252 с.: ил.; 21 см. - Библиогр.: с. 247-248. - Предм. указ.: с. 249-252. - 3000 экз. - ISBN 5-8459-0360-2.

17 Хантер, Р. Проектирование и конструирование компиляторов [Текст] / Р. Хантер; пер. с англ. С.М. Круговой; под ред. В.М. Савинкова. - М.: Финансы и статистика, 1984. - 232 с.: ил.; 24 см. - Библиогр.: с. 220-221. - Предм. указ.: с. 222-230. - Перевод изд.: The design and construction of compilers / Robin Hunter (Chichester etc.). - 17000 экз.