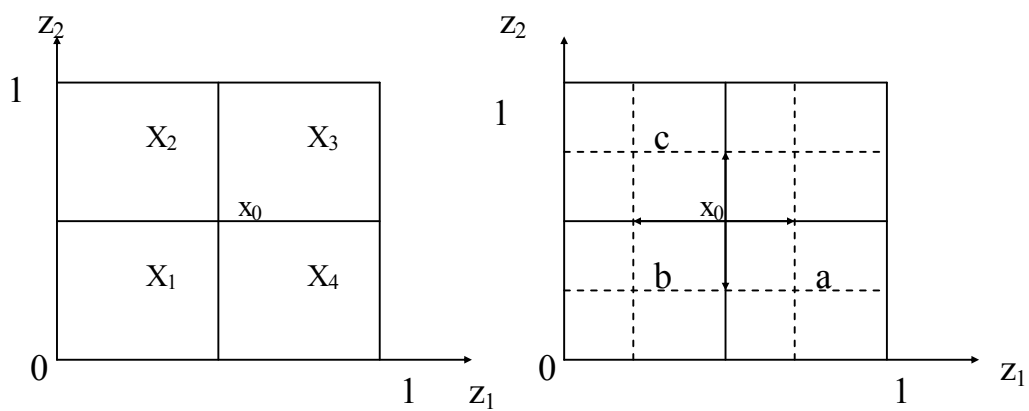


А.С. СЕМЕНОВ

**ПРАКТИЧЕСКИЙ КУРС ПО ТЕОРИИ АВТОМАТОВ И
ФОРМАЛЬНЫХ ЯЗЫКОВ**

С ПРИМЕРАМИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C#



Москва 2015

СОДЕРЖАНИЕ

Введение	4
Глава 1. Классификация грамматик и автоматов	5
1.1. Способы описания синтаксиса языков	6
1.2. Классификация Хомского	9
Глава 2. Автоматные грамматики и конечные автоматы	16
2.1. Определение автоматных грамматик и конечных автоматов ...	30
2.2. Свойства регулярных языков: лемма о накачке	30
2.3. Способы задания конечных автоматов	33
2.4. Пример проектирования грамматики и распознавателя	39
2.5. Способы реализации (по заданному регулярному выражению составлены автоматы)	52
	53
Глава 3. Контекстно-свободные грамматики и МП-автоматы	55
3.1. Определение КС-грамматик и МП-автоматов	60
3.2. Преобразование КС-грамматик	65
3.3. Пример преобразования КС-грамматик к приведенной форм ..	77
3.4. Определение МП и расширенного МП-автоматов. Примеры построения	88
3.5. Способы реализации синтаксических анализаторов	99
3.6. LL(k)-грамматики и пример построения LL(k)-анализатора ...	00
3.7. LR(k)-грамматики и пример построения LR(k)-анализатора ...	22
3.8. Грамматики предшествования	33
3.9. Пример применения алгоритм “перенос-свертка”	33
	63
Глава 4. Грамматики общего вида и машины Тьюринга	65
4.1. Грамматики общего вида и машина Тьюринга	65
4.2. Контекстно-зависимые грамматики и ленточные автоматы ...	65
4.3. Соотношение между грамматиками и языками	66
4.4. Способы реализации	66
Глава 5. Динамические модели и грамматические системы	67
5.1. Клеточные автоматы	22
5.2. Динамические автоматы распознающие КС-языки	22
5.3. L-системы	22
5.4. Способы реализации	
Приложение А. Порядок выполнения задач	
Приложение В. Задания для усвоения материала	
Заключение	
Библиографический список	

Введение

Пособие направлено на освоение и применение теории компиляции. Приводятся примеры построения синтаксических анализаторов для регулярных и контекстно-свободных грамматик, дается реализация анализаторов на объектно-ориентированном языке программирования С# [1].

Отличительной особенностью пособия является практическое содержание, включая проектирование грамматик и реализацию распознавателей. Изложенный теоретический материал используется для проведения практических работ.

Пособие состоит из трех глав и приложения, в котором приведены основные шаги выполнения практических работ и фрагменты программных текстов. Теоретический материал и практические задания организованы по темам, в соответствии с иерархией Хомского.

В первой главе даются основные определения грамматик и языков, приводятся примеры построения синтаксических анализаторов и приемы программирования.

Основной задачей практических заданий является изучение методов проектирования грамматических правил [3-5], синтаксических анализаторов и их реализации на объектно-ориентированном языке С# [2]. В процессе выполнения практических заданий развивается умение разрабатывать грамматические правила, проводить сравнительный анализ грамматик, определять их вид в соответствии с иерархией Хомского, а также применять объектно-ориентированный подход и реализовывать грамматические правила и синтаксические анализаторы на языке программирования С#.

Во второй главе рассматривается класс клеточных автоматов, которые широко применяются в различных приложениях искусственного интеллекта и при решении дифференциальных уравнений.

В третьей главе рассмотрен класс систем, позволяющий строить различные самоподобные множества по цепочкам порожденным КС-грамматиками. например, дерево Фибоначчи. Система определяет способ генерации совокупностей бесконечных цепочек по шаблону. Все нетерминалы замещаются параллельно. Шаблон содержит терминальные символы и определяет структуру порождаемых слов.

Приводится пример фрактального анализа дерева Фибоначчи, основывающийся на определении самоподобных структур объектов.

Глава 1. Классификация грамматик и автоматов

В главе рассматриваются два способа реализации грамматик: порождающими и распознающими системами см. рис. 1.1.

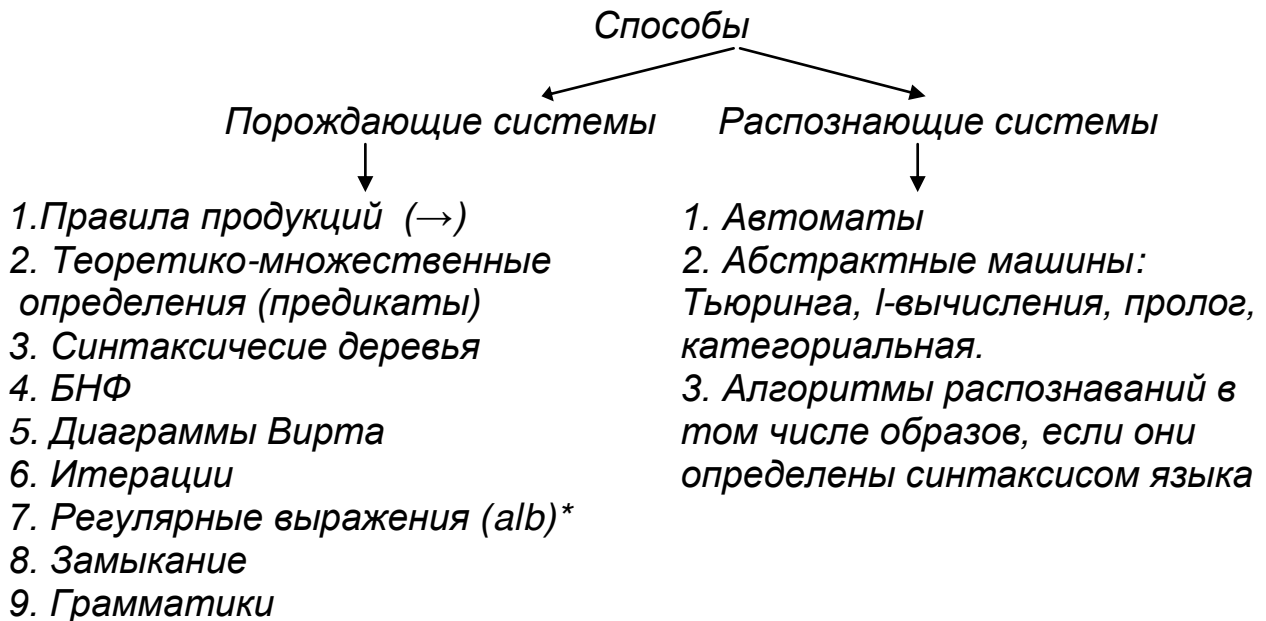


Рис. 1.1. Способы описания синтаксиса языков

Дается классификация Хомского, вводятся определения грамматик и эквивалентных им автоматов. Приводятся соотношения между типами грамматик и языков

1.1. Способы описания синтаксиса языков

Язык, будь то естественный (естественная речь) или язык программирования, задается множеством правил, образующих грамматику языка. Синтаксис языка отделяется от семантики. Появление формальных грамматик было обусловлено стремлением формализовать естественные языки и языки программирования.

Фигурные скобки используются для обозначения множеств, например $\{1,2,3\}$ обозначает множество, содержащее целые числа 1, 2 и 3.

Объединение \cup и пересечение \cap множеств определяются как обычно:

$$\{1,2,3\} \cup \{3,4,5\} = \{1,2,3,4,5\}$$

$$\{1,2,3\} \cap \{3,4,5\} = \{3\}$$

Множество A включает \supseteq множество B , если каждый элемент B является элементом A , например

$$\{3,4,5\} \supseteq \{3\}$$

Если A есть множество и мы определяем B как

$$B = \{x \mid x \in \mathbb{N}, x \text{ есть четное число}\}$$

то B будет множеством $\{2,4,8\}$ задается с помощью предиката, в данном случае это " $x \in \mathbb{N}, x$ есть четное число".

Правила описывают те последовательности слов, которые являются корректными, т.е. допустимыми предложениями языка. Грамматика

позволяет осуществить грамматический анализ предложения, а значит, и явно описать его структуру.

Одно из назначений грамматики – это определить бесконечное число предложений языка с помощью конечного числа правил.

Грамматика языка может быть реализована двумя способами: в виде порождающей системы и в виде устройств, называемых распознавателями.

1. Порождающие системы. Для описания синтаксиса языков программирования наибольшее распространение получили следующие порождающие системы: формальные грамматики, форма Бекуса-Наура и ее модификации, диаграммы Вирта [3].

В порождающих системах правила грамматики задаются продукциями, состоящими из двух частей: левой и правой. В левой части, записывается определение, в правой части варианты, из которых может состоять определение. Например, предложение “он смотрит кино” может быть определено синтаксическими правилами:

предложение состоит из *подлежащее*, *группа сказуемого*
подлежащее - он
группа сказуемого - *сказуемое*, *дополнение*
сказуемое - смотрит
дополнение – кино

Синтаксис и семантика

a b c
“он смотрит кино”

В рассматриваемых правилах порядок элементов фиксирован. Для определения грамматического формализма, эквивалентного синтаксическому, введем правило замены, которое обозначается символом “→”. Тогда порождающая грамматика, для рассматриваемого предложения, примет вид:

Предложение → *подлежащее*, *группа сказуемого*
подлежащее → он
группа сказуемого → *сказуемое*, *дополнение*
сказуемое → смотрит
дополнение → кино

Выполнив правило замены “→”, получим предложение “он смотрит кино”. Предложения языка, порождаются из исходного символа применением продукций. Шаги замены можно интерпретировать как шаги построения дерева вывода см. рис. 1.

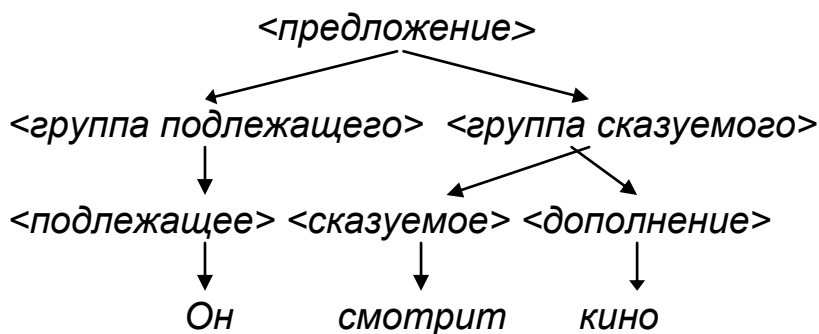


Рис. 1. Древоподобная структура предложения

Бэкусовая нормальная форма (БНФ) или форма Бэкуса-Наура была предложена Д.Бэкусом в 1959 году и впервые применена П.Науром для описания языка Алгол-60. БНФ - *метаязык*, который используется для описания других языков.

Обозначение правил в форме записи $\xi ::= \eta$ относится к нотации БНФ. В этой нотации используются обозначения:

$::=$ - это есть, $|$ - или, $< >$ - угловые скобки, в которых записываются металингвистическая переменная, т.е. определяемое понятие;

$[]$ - факультативный элемент (необязательная часть), то есть конструкция в скобках может присутствовать или отсутствовать во фразе языка;

$\{ \}$ - множественный элемент (одно из, элемент выбора), то есть во фразе языка используется один из элементов внутри скобки.

Правило вида $\alpha ::= \beta \mid \beta\gamma$ можно представить $\alpha ::= \beta [\gamma]$.

БНФ для рассматриваемого примера.

$\langle \text{предложение} \rangle ::= \langle \text{подлежащее} \rangle \langle \text{группа сказуемого} \rangle$

$\langle \text{подлежащее} \rangle ::= \text{он} \mid \text{она}$

$\langle \text{группа сказуемого} \rangle ::= \langle \text{сказуемое} \rangle \langle \text{дополнение} \rangle$

$\langle \text{сказуемое} \rangle ::= \text{смотрит} \mid \text{обожает}$

$\langle \text{дополнение} \rangle ::= \text{кино}$

Используя множество правил, *выведем* или *породим* предложение по следующей схеме. Начнем с начального символа грамматики - $\langle \text{предложение} \rangle$, найдем правило, в котором $\langle \text{предложение} \rangle$ слева от $::=$, и подставим вместо $\langle \text{предложение} \rangle$ цепочку, которая расположена справа от $::=$, т.е.

$\langle \text{предложение} \rangle \Rightarrow \langle \text{подлежащее} \rangle \langle \text{группа сказуемого} \rangle$

Символ " \Rightarrow " означает, что один символ слева от \Rightarrow в соответствии с правилом грамматики заменяется цепочкой, находящейся справа от \Rightarrow . Обозначение \Rightarrow называют также непосредственное следование.

Заменим синтаксическое понятие на одну из цепочек, из которых оно может состоять. Повторим процесс. Возьмем один из метасимволов в цепочке $\langle \text{подлежащее} \rangle \langle \text{группа сказуемого} \rangle$, например $\langle \text{подлежащее} \rangle$; найдем правило, где $\langle \text{подлежащее} \rangle$ находится слева от $::=$, и заменим

<подлежащее> в исходной цепочке на соответствующую цепочку, которая находится справа от $::=$. Получим вывод:

<подлежащее> < группа сказуемого > \Rightarrow он < группа сказуемого >

Полный вывод одного предложения будет таким:

<предложение> \Rightarrow <подлежащее> <группа сказуемого> \Rightarrow он <группа сказуемого> \Rightarrow он <сказуемое> <дополнение> \Rightarrow он *смотрит* <дополнение> \Rightarrow он *смотрит* кино

Этот вывод предложения запишем сокращенно, используя символ \Rightarrow^+ , который означает, что цепочка выводима *нетривиальным способом* :

<предложение> \Rightarrow^+ он *смотрит* кино

Две последовательности связаны отношением \Rightarrow^* , когда вторая получается из первой применением некоторой последовательности продукции $\alpha \Rightarrow^* \alpha_m$ тогда, и только тогда, когда $\alpha \Rightarrow^+ \alpha_m$.

На каждом шаге можно заменить любую метaperеменную. В приведенном выше выводе всегда заменялся самый левый из них. Если в процессе вывода цепочки правила грамматики применяются только к самому левому нетерминалу, говорят, что получен *левый вывод* цепочки. Аналогично определяется правый вывод.

Рассматриваемая грамматика определяет несколько предложений (цепочек) языка:

он *смотрит* кино он *обожает* кино
она *смотрит* кино она *обожает* кино

Диаграммы Вирта позволяют визуально представить правила порождения. Каждому правилу соответствует диаграмма.

Элементы ИЛИ обозначаются разветвлениями и вершинами, элементы И последовательными вершинами графа, метасимволы {...} циклом. Вершины могут быть двух типов: терминальные – кружки, и нетерминальные – прямоугольники. Каждому правилу соответствует диаграмма см. рис. 2.

Правила- И, цикл, ИЛИ: $T \rightarrow \{ F \} \mid (E)$

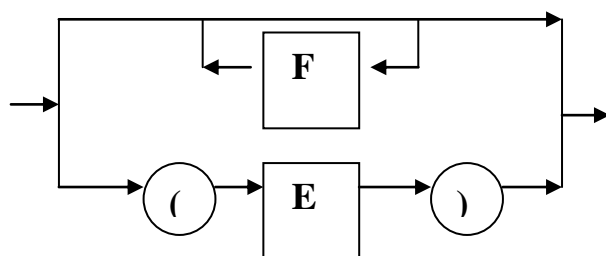


Рис. 2. Диаграммы Вирта

Итерационная форма описания. Вводится операция итерации – обозначается парой круглых скобок со звездочкой.

Итерация вида $(a)^*$ определяется как множество, включающее цепочки всевозможной длины, построенные с использованием символа **a**.

$(a)^* = \{a, aa, aaa, aaaa, \dots\}$

Определим понятия, используемые при определении грамматик формальных языков и распознавателей.

Алфавит (или словарный состав) – конечное множество символов. Например, *алфавит* $V_1 = \{a,b\}$ – содержит два символа, *алфавит* $V_2 = \{01,11,10,00\}$ – содержит четыре символа.

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

Пустой цепочкой символов ε называется цепочка, не содержащая ни одного символа.

Длина цепочки α – число составляющих ее символов, обозначается $|\alpha|$. Например, если цепочка символов $\alpha = \text{defgabck}$, то длина α равна 8, $|\alpha| = 8$. Длина ε равна 0, $|\varepsilon| = 0$.

Замыкание Клини (или звезда Клини) в математической логике и информатике – унарная операция над множеством строк либо символов. Замыкание Клини множества V обозначается V^* . Широко применяется в регулярных выражениях, на примере которых было введено Стивеном Клини для описания некоторых автоматов.

Множество всех строк (включая пустую), которые могут быть построены из **символов** алфавита V , называется **замыканием** V , и обозначается V^* .

Пусть V – множество строк, тогда V^* – минимальное надмножество множества V , которое содержит ε (пустую строку) и замкнуто относительно конкатенации.

Надмножество (superset) – множество, подмножеством которого является данное. Иначе супермножество, объемлющее множество (множество множеств).

V^* (замыкание V) – обозначается множество всех цепочек составленных из символов алфавита V , включая пустую цепочку ε . Это также множество всех строк, полученных конкатенацией ε или более строк из V . Например, если $V = \{0,1\}$, то $V^* = \{\varepsilon, 0, 1, 00, 11, 01, 10, 000, 001, 011, \dots\}$.

V^+ – обозначается множество всех цепочек составленных из алфавита V , исключая пустую цепочку ε . Следовательно, $V^* = V^+ \cup \{\varepsilon\}$.

Декартовым произведением $A \times B$ множеств A и B называется множество $\{(a,b) \mid a \in A, b \in B\}$.

\emptyset – обозначается пустое множество.

2. Распознающие системы (автоматы). Грамматика может быть реализована в виде некоторого алгоритма, производящего действия и отвечающего на вопрос, принадлежит ли данная фраза языку.

Функцией перехода δ называется отношение, управляющее действиями распознавателя. Функция перехода δ определяет для каждой пары (входной символ, состояние) множество состояний, в которых он будет находится на следующем шаге.

Например, принадлежит ли предложение “он смотрит кино” языку заданному рассмотренными грамматическими правилами? Для этого необходимо построить распознаватель.

1.2. Классификация Хомского

Синтаксис языка L можно специфицировать, пользуясь системой изображения множеств, например:

$$L = \{0^n 1^n \mid n \geq 0\}$$

Язык L включает строки, состоящие из нуля или нулей и того же числа последующих единиц. Пустая строка включается в язык.

Такое задание синтаксиса намного проще синтаксисов большинства языков специфицируемых с помощью *грамматики*.

Определение 1. Грамматика $G = (T, V, P, S_0)$,

где T - конечное множество терминальных символов (терминалов) алфавита;

V - конечное множество нетерминальных символов алфавита, не пересекающихся с T , $T \cap V = \emptyset$,

S_0 - начальный символ (или аксиома), $S_0 \in V$;

P - конечное множество правил порождения (продукций), $P = (T \cup V)^+ \times (T \cup V)^*$. Элемент (α, β) множества P называется *правилом вывода* и записывается в виде $\alpha \rightarrow \beta$.

Цепочка $\beta \in (T \cup V)^*$ *непосредственно выводима* ($\alpha \rightarrow \beta$) из цепочки $\alpha \in (T \cup V)^+$ в грамматике G , если $\alpha = \xi_1 \gamma \xi_2$, $\beta = \xi_1 \delta \xi_2$, где $\xi_1, \xi_2, \delta \in (T \cup V)^*$, $\gamma \in (T \cup V)^+$ и правило вывода $\gamma \rightarrow \delta$ содержится в P .

Цепочка $\beta \in (T \cup V)^*$ *выводима* из цепочки $\alpha \in (T \cup V)^+$ в грамматике G , (обозначим $\alpha \Rightarrow \beta$), если существуют цепочки $\gamma_0, \gamma_1, \dots, \gamma_n$ ($n \geq 0$), такие, что $\alpha = \gamma_0 \rightarrow \gamma_1 \rightarrow \dots \rightarrow \gamma_n = \beta$. Последовательность $\gamma_0, \gamma_1, \dots, \gamma_n$ называется *выводом длины n* .

Правила вывода с одинаковыми левыми частями $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$, записывают сокращенно $\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$, где β_i , $i = 1, 2, \dots, n$ и называют *альтернативой* правил вывода из цепочки α .

Для обозначения терминалов грамматики мы будем употреблять прописные буквы (или строки прописных букв), а для обозначения не терминалов — заглавные буквы (или строки заглавных букв).

Примеры грамматик.

Пример 1. Для примера “он смотрит кино” имеем грамматику $G = (\{\text{“он”}, \text{“кино”}, \text{“смотрит”}\}, \{\text{Предложение, подлежащее, группа сказуемого, сказуемое, дополнение}\}, P, S_0)$, где $S_0 = \text{Предложение}$, P состоит из правил $P = \{p_1, p_2, p_3, p_4, p_5\}$;

p_1 : Предложение \rightarrow подлежащее, группа сказуемого

p_2 : подлежащее \rightarrow он

p_3 : группа сказуемого \rightarrow сказуемое, дополнение

p_4 : сказуемое \rightarrow смотрит

p_5 : дополнение \rightarrow кино

Пример 2. $G = (\{c, d\}, \{B, S_0\}, P, S_0)$, где P состоит из правил

$S_0 \rightarrow cBd$

$cB \rightarrow ccBd$

$B \rightarrow \varepsilon$

Цепочка $ccBdd$ непосредственно выводима из cBd в грамматике G .

Цепочка cccBddd в грамматике G выводима, т.к. существует вывод $S_0 \Rightarrow cBd \Rightarrow ccBdd \Rightarrow cccBddd \Rightarrow cccddd$. Длина вывода равна 4.

Определение 2. Языком, порождаемым грамматикой $G = (T, V, P, S_0)$, называется множество $L(G) = \{\alpha \in T^* \mid S_0 \Rightarrow^* \alpha\}$.

Другими словами, $L(G)$ - это все цепочки в алфавите T , которые выводимы из S_0 с помощью P .

Цепочка $\alpha \in (T \cup V)^*$, для которой $S_0 \Rightarrow^* \alpha$ (то есть цепочка, которая может быть выведена из начального символа), называется *сентенциальной формой* в грамматике G . Язык, порождаемый грамматикой, можно определить как множество терминальных сентенциальных форм.

Например, пусть задана грамматика $G_1 = (\{0,1\}, \{A, S_0\}, P_1, S_0)$, состоящая из правил P_1 :
 $S_0 \rightarrow 0A1$
 $0A \rightarrow 00A1$
 $A \rightarrow \varepsilon$

Тогда язык, порождаемый грамматикой G_1 , это язык $L(G_1) = \{0^n 1^n \mid n > 0\}$.

Грамматики G_1 и G_2 называются *эквивалентными*, если $L(G_1) = L(G_2)$. Например, пусть задана грамматика $G_2 = (\{0,1\}, \{S\}, P_2, S)$, где

$P_2: S \rightarrow 0S1 \mid 01$, тогда

грамматики G_1 и G_2 эквивалентны, т.к. обе порождают язык $L(G_1) = L(G_2) = \{0^n 1^n \mid n > 0\}$.

Грамматики G_2 и G_3 называются *почти эквивалентными* $L(G_2) = L(G_3)$, если $L(G_2) = L(G_3) \cup \{\varepsilon\}$. Другими словами, грамматики почти эквивалентны, если языки, ими порождаемые, отличаются не более чем на ε .

Определим грамматику $G_3 = (\{0,1\}, \{S\}, P_3, S)$ с правилами

$P_3: S \rightarrow 0S1 \mid \varepsilon$

Тогда грамматики G_2 и G_3 *почти эквивалентны*, так как $L(G_2) = \{0^n 1^n \mid n > 0\}$, а $L(G_3) = \{0^n 1^n \mid n \geq 0\}$, т.е. $L(G_3)$ состоит из всех цепочек языка $L(G_2)$ и пустой цепочки, которая в $L(G_2)$ не входит.

Классификация Хомского в таб.1. дает представление о 4 типах грамматик, располагающихся по иерархии от 0 до 3 в порядке убывания их общности, и о эквивалентных формальных автоматах.

Таб. 1. Классификация Хомского

Тип	Ограничения на правила P	Пример правил грамматики $G = (T, V, P, S_0)$	Пример языка	Автомат или абстрактная машина
0	Общего вида, не накладывається никаких ограничений	$S_0 \rightarrow CD, C \rightarrow 0CA, C \rightarrow 1CB, AD \rightarrow 0D, BD \rightarrow 1D, A0 \rightarrow 0A, A1 \rightarrow 1A, B0 \rightarrow 0B, B1 \rightarrow 1B, C \rightarrow \varepsilon, D \rightarrow \varepsilon$	$\{\omega\omega \omega \in \{0,1\}^*, \text{ язык состоит из цепочек четной длины, из 0 и 1}\}$	Машина Тьюринга
1	Контекстно-зависимая (КЗ) $\alpha \rightarrow \beta, \alpha \leq \beta $	$S_0 \rightarrow 0B0$ $B \rightarrow 1 0BC$ $C0 \rightarrow 100$ $C1 \rightarrow 1C$	$\{0^n 1^n 0^n n \geq 1\}$	Машина Тьюринга с конечной лентой
2	Контекстно-свободная (КС) $A \rightarrow \alpha$	$S_0 \rightarrow AS_0B AB$ $A \rightarrow 0$ $B \rightarrow 1$ LL(k) грамматика LR(k) грамматика	$\{0^n 1^n n \geq 1\}$	с магазинной памятью (МП-автомат: LL-разбор, РМП-автомат: LR-разбор) Распознаватели: LL(1) LR(1)
3	Регулярная $A \rightarrow bV$ $A \rightarrow a$	$S_0 \rightarrow 0 0A$ $A \rightarrow 0B$ $B \rightarrow 1 1A$	$\{0(01)^n n \geq 0\}$	Конечный автомат (КА)

Грамматики классифицируются по виду правил вывода P. В правилах вывода P большими латинскими буквами обозначают нетерминальные символы, маленькими латинскими буквами - терминальные.

Автомат эквивалентен грамматике, если он воспринимает весь порожденный грамматикой язык и только этот язык.

Классификация Хомского дает представление об общности грамматик, и об эквивалентных им формальных автоматах.

Глава 2. Автоматные грамматики и конечные автоматы

2.1. Определение автоматных грамматик и конечных автоматов

Определение 3. Грамматика типа 3 или автоматная (регулярная) A -грамматика – это грамматика $G = (T, V, P, S)$, у которой правила порождения P имеют вид по классификации Хомского $A \rightarrow bV$ (праволинейное правило) или $A \rightarrow a$ (заключительное правило), где $A, V \in V, a, b \in T$. Каждое правило такой грамматики содержит единственный нетерминал в левой части, всегда один терминал в правой части, за которым может следовать один нетерминал. Такую грамматику также называют *праволинейной*.

Грамматика $G = (T, V, P, S)$ называется *леволинейной*, если каждое правило из P имеет вид по классификации Хомского $A \rightarrow Bb$ либо $A \rightarrow a$, где $A, B \in V, a, b \in T$.

Таким образом, грамматику типа 3 можно определить как праволинейную либо как леволинейную.

Выбор определения не влияет на множество языков, порождаемых грамматиками этого класса, поскольку доказано, что множество языков, порождаемых праволинейными грамматиками, совпадает с множеством языков, порождаемых леволинейными грамматиками.

Автоматной грамматике эквивалентен простейший распознаватель – недетерминированный конечный автомат. Автомат представляет собой устройство, у которого отсутствует вспомогательная память.

Определение 4. Конечный автомат (КА) – это пятерка объектов

$KA = (Q, \Sigma, \delta, q_0, F)$, где

Q - конечное множество состояний;

Σ - конечный алфавит входных символов;

δ - функция переходов, задаваемая отображением

$$\delta: Q \times \Sigma \rightarrow KA(Q),$$

где $KA(Q)$ конечное множество подмножеств множества Q ;

q_0 - начальное состояние автомата, $q_0 \in Q$;

$F \subseteq Q$ - множество заключительных состояний.

В каждый момент времени КА находится в некотором состоянии $q \in Q$ и читает поэлементно последовательность символов $a_k \in \Sigma$, записанную на конечной ленте. При этом либо читающая головка машины движется в одном направлении (слева направо), либо лента перемещается (справа налево) рис.2.1. при неподвижной читающей головке.



Рис. 2.1. Схема конечного автомата
Входная лента: Входные символы $a_i \in \Sigma$

Если автомат в состоянии q_i читает символ a_k и определена функция перехода $\delta(q_i, a_k) = q_j$, то автомат воспринимает символ a_k и переходит в состояние q_j для обработки следующего символа.

Определение 5. Конфигурация КА – это пара множества $(q, \omega) \in Q \times \Sigma^*$, где $q \in Q$, $\omega \in \Sigma^*$. Конфигурация (q_0, ω) называется *начальной*, а (q, ε) , где $q \in F$, - *заключительной*.

Определим бинарное отношение \vdash на конфигурациях, соответствующее одному такту работы КА. Если $q' \in \delta(q, a)$, то $(q, a\omega) \vdash (q', \omega)$ для всех $\omega \in \Sigma^*$.

Пусть $\{C\}$ – множество конфигураций.

1. $C \vdash^0 C'$ означает, что $C = C'$
2. $C_0 \vdash^k C_k$, если существует последовательность конфигураций C_1, C_2, \dots, C_{k-1} , для $k \geq 1$, в которых $C_i \vdash C_{i+1}$ для $0 \leq i < k$.
3. $C \vdash^+ C'$ означает, что $C \vdash^k C_k$ для некоторого $k \geq 1$, а $C \vdash C'$ означает, что $C \vdash^k C'$ для $k \geq 1$.

Конечный автомат $KA = (Q, \Sigma, \delta, q_0, F)$ распознает входную цепочку $\omega \in \Sigma^*$, если $(q_0, \omega) \vdash^k (q, \varepsilon)$ для $q \in F$.

Языком $L(KA)$, распознаваемым КА называется множество входных цепочек,

$$L(KA) = \{\omega \in \Sigma^* \mid (q_0, \omega) \vdash^k (q, \varepsilon), \text{ где } q \in F\}$$

КА называется *недетерминированным*, если для каждой конфигурации существует конечное множество всевозможных следующих шагов, любой из которых КА может сделать, исходя из этой конфигурации.

КА называется *детерминированным*, если для каждой конфигурации существует не более одного следующего шага.

Для любого конечного недетерминированного автомата можно построить ему эквивалентный детерминированный автомат.

Класс языков, распознаваемый конечными автоматными, совпадает с классом языков, порождаемых автоматными грамматиками и наоборот.

Утверждение 1. Пусть задана автоматная грамматика $G = (T, V, P, S_0)$, тогда существует такой (недетерминированный) конечный автомат $KA = (Q, \Sigma, \delta, q_0, F)$, что $L(KA) = L(G)$. КА строится следующим образом:

1. Входные алфавиты конечного автомата Σ и автоматной грамматики T совпадают, $\Sigma = T$.

2. $Q = V \cup \{q_f\}$, где q_f – заключительное состояние конечного автомата.

3. $q_0 = S_0$.

4. Если $S_0 \rightarrow \varepsilon \in P$, то $F = \{S_0, q_f\}$, в противном случае $F = \{q_f\}$.

5. Функция переходов КА определяется следующим образом:

- $q_f \in \delta(B, a)$, если $B \rightarrow a \in P$, $B \in V$, $a \in \Sigma$;
- если $B \rightarrow aC \in P$, то $C \in \delta(B, a)$;
- $\delta(q_f, a) = \emptyset$ для всех $a \in \Sigma$.

Пример. Пусть задан регулярный язык $L = \{0(10)^n \mid n \geq 0\}$. Построить автоматную грамматику $G = (T, V, P, S_0)$ для заданного языка L и привести пример вывода строки. Используя грамматику G , построить КА $= (Q, \Sigma, \delta, q_0, F)$ и привести пример конфигурации КА.

1. Построение грамматики. $L = \{0(10)^n \mid n \geq 0\}$, то 0, 010, 01010 и т.д. – этот язык порожден регулярной грамматикой $G = (T, V, P, S_0)$, где $T = \{0, 1\}$, $V = \{S_0, A, B\}$, $P = \{S_0 \rightarrow 0, S_0 \rightarrow 0A, A \rightarrow 1B, B \rightarrow 0, B \rightarrow 0A\}$. Пример вывода цепочки

$S_0 \Rightarrow 0A \Rightarrow 01B \Rightarrow 010A \Rightarrow 0101B \Rightarrow 01010$.

2. Построение КА. Воспользуемся утверждением 1, тогда $КА = (\{S_0, A, B, q_f\}, \{0, 1\}, \delta, S_0, q_f)$

$\delta(S_0, 0) = \{A, q_f\}$;

$\delta(A, 1) = \{B\}$;

$\delta(B, 0) = \{A, q_f\}$;

Пример конфигурации КА: $S_0 01010 \mid A 1010 \mid B 010 \mid A 10 \mid B 0 \mid q_f$

2.2. Способы задания конечных автоматов

На рис. 1.4. приведена диаграмма переходов КА для рассматриваемого примера.

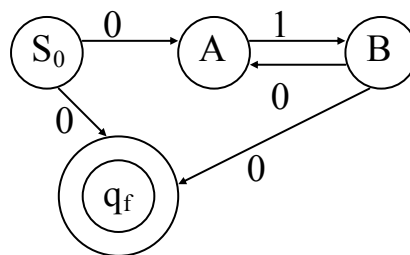


Рис. 1.4. Диаграмма переходов КА

Конечный автомат можно задать в виде таблицы переходов и диаграммы (графа) переходов.

В таблице переходов двум аргументам ставится в соответствие одно значение. В таб. 1.2 приведена таблица переходов КА для рассматриваемого примера.

Таб. 1.2. Таблица переходов для КА

	0	1
S_0	$\{A, q_f\}$	
A		$\{B\}$
B	$\{A, q_f\}$	
q_f		

Диаграммой переходов КА называется неупорядоченный граф, удовлетворяющий условиям:

- каждому состоянию q соответствует некоторая вершина, отмеченная его именем;
- диаграмма переходов содержит дугу из состояния q_k в состояние q_n , отмеченную символом a , если $q_k \in \delta(q_n, a)$. Дуга может быть помечена множеством символов, переводящих автомат из состояния q_k в состояние q_n ;
- вершины, соответствующие заключительным состояниям $q_f \in F$, отмечаются двойным кружком.

2.3. Свойства регулярных языков: лемма о накачке

Теорема, называемая “лемма о накачке” утверждает, что все достаточно длинные слова регулярного языка можно *накачать*, то есть **повторить** внутреннюю часть слова сколько угодно раз, производя новое слово, также принадлежащее языку. Лемма описывает существенное свойство всех регулярных языков и служит инструментом для доказательства нерегулярности некоторых языков. Она является одной из нескольких лемм о накачке.

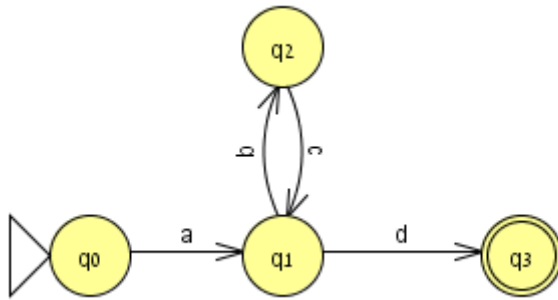
Два способа:

1. построить распознаватель для заданного языка - конечный автомат;
2. выделить цепочку y^i для всех $i \geq 0$, $xy^iz \in L$, на x и z ограничений не накладывается (то есть, нет цепочки символов y^i , то не регулярный язык).

Свойство замкнутости регулярных языков, позволяет минимизировать построение автомата:

1. строить распознаватели для одних языков, построенных из других с помощью операций;
2. определить, что два различных автомата определяют один язык.

Пример 1. Пусть КА (см. рис.) распознает строку **abcbcd**. Поскольку длина её превышает число состояний, существуют повторяющиеся состояния: q_1 и q_2 .



Поскольку подстрока **bcbc** строки **abcabcd** проводит автомат по переходам из состояния q_1 обратно в q_1 , эту строку можно повторять сколько угодно раз, и КА всё равно будет её принимать, например строки **abcbcbcabcd** и **ad**.

В терминах леммы о накачке, строка **abcabcd** разбивается на часть $x = a$, часть $y = bc$ и часть $z = bcd$.

Заметим, что можно разбить её различными способами, например $x = a$, $y = bcbc$, $z = d$, и все условия будут выполнены, кроме $|xy| \leq p$, где p – длина накачки.

Длина накачки p принимается более длины самого длинного слова языка.

Неформальное утверждение: слово w языка L длины по меньшей мере p , (где p константа, называемая длиной накачки, зависит лишь от L) можно разделить на три подцепочки, $w = xuz$, так что среднюю часть, u (непустую), можно повторить произвольное число раз (включая ноль, **то есть удалить**) и получить строку из L . Этот процесс повторения называется «накачкой».

Причем, длина xu не превысит p , ограничивая способы разделения строки w .

Конечные языки удовлетворяют требованиям леммы о накачке тривиально определяя m длиной максимальной строки из языка плюс один.

Пример 2. Рассмотрим язык $L_{01} = \{0^n 1^n \mid n \geq 1\}$, состоящий из всех цепочек вида 01, 0011, 000111 и так далее, содержащий один или несколько нулей, за которыми следует такое же количество единиц.

Утверждается, что язык L_{01} нерегулярен.

Если бы L_{01} был регулярным языком, то допускался некоторым ДКА, имеющим какое-то число состояний k . Пусть на вход ДКА поступает k нулей. Он находится в некотором состоянии после чтения каждого из $k+1$ префиксов входной цепочки, т.е. $\varepsilon, 0, 00, \dots, 0^k$. Поскольку есть только k различных состояний, например, $0^i, 0^j$, автомат должен находиться в одном и том же состоянии.

Прочитав i или j нулей ДКА получает на вход 1. По прочтении i единиц он должен допустить вход, если ранее получил i нулей, и отвергнуть его, получив j нулей. Но в момент поступления 1, автомат не способен вспомнить какое число нулей i, j было принято. Следовательно, он может работать неправильно.

Формальное утверждение.

Пусть L регулярный язык. Тогда существует целое $p \geq 1$ зависящее только от L , такое что цепочка w из L длины по меньшей мере p может быть записана как $w = xyz$, где y – это подцепочка, которую можно накачать (удалить или повторить произвольное число раз, так что результат останется в L), при этом:

1. $|y| \geq 1$, цикл y должен быть накачан хотя бы длиной 1;
 2. $|xy| \leq p$, цикл должен быть в пределах первых p символов;
 3. для всех $i \geq 0$, $xy^iz \in L$, на x и z ограничений не накладывается.
- $$\forall L \subseteq \Sigma^* (\text{regular}(L) \Rightarrow (\exists p \geq 1 (\forall w \in L (|w| \geq p) \Rightarrow (\exists x, y, z \in \Sigma^* (w = xyz \Rightarrow (|y| \geq 1 \wedge |xy| \leq p \wedge \forall i \geq 0 (xy^iz \in L))))))))))$$

Пример 3. Нерегулярность языка $L = \{a^n b^n \mid n \geq 0\}$ над алфавитом $\Sigma = \{a, b\}$ можно показать следующим образом.

Пусть w, x, y, z, p , и i заданы соответственно формулировке леммы выше. Пусть w из L задаётся как $w = a^p b^p$. По лемме о накачке, существует разбиение $w = xyz$, где $|xy| \leq p$, $|y| \geq 1$, такое что xy^iz принадлежит L для любого $i \geq 0$. Если допустить, что $|xy|=p$, а $|z|=p$, то xy — это первая часть w , состоящая из p последовательных экземпляров символа a .

Поскольку $|y| \geq 1$, она содержит по меньшей мере одну букву a , а xy^2z содержит больше букв a чем b . Следовательно, xy^2z не в языке L (заметим, что любое значение $i \neq 1$ даст противоречие). Достигнуто противоречие, поскольку в этом случае накачанное слово не принадлежит языку L . Предположение о регулярности L неверно и L – не регулярный язык.

Доказательство леммы о накачке

Для каждого регулярного языка существует конечный автомат (КА), распознающий этот язык.

Если язык конечен, то результат можно получить немедленно, задав длину накачки p более длины самого длинного слова языка: в этом случае нет ни одной цепочки в языке длиннее p , и утверждение леммы выполняется безусловно.

Если регулярный язык бесконечен, то существует минимальный КА, распознающий его. Число состояний этого КА и принимается за длину накачки p . Если длина цепочки превышает p , то хотя бы одно состояние при обработке цепочки повторяется (назовём его S). Переходы из состояния S и обратно соответствуют некоторой цепочки. Эту цепочку обозначим y по условиям леммы, и поскольку автомат примет строку как без части y , так и с повторяющейся частью y , условия леммы выполнены.

Доказательство. Пусть L – регулярный язык, тогда $L = L(\text{ДКА})$ для некоторого ДКА. Пусть ДКА имеет n состояний. Рассмотрим произвольную цепочку w длиной не менее n . и скажем, $w = a_1 a_2 a_3 \dots a_m$, где $m \geq n$ и каждый a_i есть входной символ. Для $i = 0, 1, 2, \dots, n$ определим состояние p_i как $\delta(q_0, a_1 a_2 a_3 \dots a_i)$, где δ – функция переходов автомата, q_0 – его начальное состояние.

Заметим, что $p_0 = q_0$.

Рассмотрим $n+1$ состояний p_i при $i = 0, 1, 2, \dots, n$. Поскольку автомат имеет n различных состояний, то найдутся два разных целых числа i и j ($0 \leq i < j \leq n$), при которых $p_i = p_j$. Теперь разобьем цепочку w на xuz .

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

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

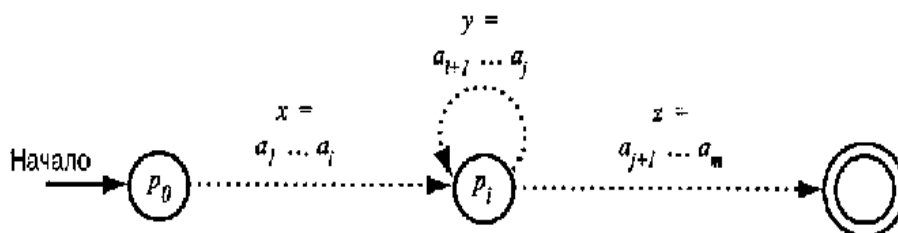


рис. 4.1. Каждая цепочка, длина которой больше числа состояний автомата, приводит к повторению некоторого состояния

На вход автомата поступает цепочка xu^kz для любого $k \geq 0$.

1. При $k = 0$ автомат переходит из начального состояния q_0 (которое есть также p_0) в p_i прочитав x . Поскольку $p_i = p_j$, то z переводит автомат из p_i в допускающее состояние (рис. 4.1.).

2. Если $k > 0$, то по x автомат переходит из q_0 в p_i , затем читая y^k , k раз циклически проходит через p_j , и, наконец, по z переходит в допускающие состояния.

3. Для любого $k \geq 0$ цепочка xu^kz также допускается автоматом, то есть принадлежит языку L .

2.4. Способы преобразования НКА в ДКА

1. Построение НКА из автоматов.

???????????????????????????????? см. лаб.

2. Детерминированный конечный автомат ДКА $= (Q, \Sigma, \delta, q_0, F)$, если множество $\delta(q, a)$ содержит не более одного состояния для любых $q \in Q$, $a \in \Sigma$. Рассмотрим алгоритм построения ДКА по недетерминированному КА (НКА).

Основная идея построения ДКА по НКА заключается в том, что после обработки отдельной входной цепочки состояние ДКА будет представлять собой множество всех состояний НКА, которые он может достичь из начальных состояний после применения данной цепочки.

Переходы ДКА можно получить из НКА, вычисляя множества состояний, которые могут следовать после данного множества при различных входных символах.

Допустимость цепочки определяется исходя из того, является ли последнее детерминированное состояние, которого достиг ДКА, множеством недетерминированных состояний, включающим хотя бы одно допускающее состояние.

Вход: НКА = $(Q, \Sigma, \delta, q_0, F)$

Выход: ДКА = $(Q', \Sigma, \delta', q_0', F')$, у которого $L(НКА) = L(ДКА)$.

1. Входные алфавиты НКА и ДКА совпадают.
2. Множество Q' содержит множество всех подмножеств множества Q (булеан множества Q). Если $|Q| = n$, то $|Q'| = 2^n$.
3. Для каждого множества $S \subseteq Q$ и каждого входного символа $a \in \Sigma$, выполняется $\delta'(S, a) = \bigcup_{p \in S} \delta(p, a)$, т.е. чтобы найти $\delta'(S, a)$ необходимо для каждого состояния p из S найти состояния, в которые можно попасть из p для входного $a \in \Sigma$. Затем взять объединение множеств всех состояний по всем состояниям p .
4. F' есть множество подмножеств S множества Q , $S \cap F \neq \emptyset$, то есть F' состоит из всех множеств состояний Q НКА, содержащего хотя бы одно заключительное состояние.
5. $q_0' = q_0$.
6. Не все состояния из Q' достижимы из начального состояния (состояние q достижимо из начального состояния q_0' , если имеется путь из состояния q_0' в состояние q). Недостижимые состояния надо исключить из множества Q' .
7. Определить ДКА.

Построим ДКА, допускающий язык, определяемый НКА для рассматриваемого примера рис. 1.4.

1. Пусть входные алфавиты НКА и ДКА совпадают.
2. Булеан множества без пустого подмножества $2^4 - 1 = 15$, $Q' = \{\{S\}, \{A\}, \{B\}, \{q_f\}, \{S,A\}, \{S,B\}, \{S,q_f\}, \{A,B\}, \{A,q_f\}, \{B,q_f\}, \{S,A,B\}, \{S,A,q_f\}, \{A,B,q_f\}, \{S,B,q_f\}, \{S,A,B,q_f\}\}$
3. Найти $\delta'(S, a)$:

$$\begin{aligned}
 \delta'(\{B\}, 0) &= \{A, q_f\} & \delta'(\{S\}, 0) &= \{A, q_f\} & \delta'(\{A\}, 1) &= \{B\} \\
 \delta'(\{S,B\}, 0) &= \{A, q_f\} & \delta'(\{S,A\}, 0) &= \{A, q_f\} & \delta'(\{S,A\}, 1) &= \{B\} \\
 \delta'(\{S,q_f\}, 0) &= \{A, q_f\} & \delta'(\{B,A\}, 0) &= \{A, q_f\} & \delta'(\{B,A\}, 1) &= \{B\} \\
 \delta'(\{B,q_f\}, 0) &= \{A, q_f\} & \delta'(\{A,q_f\}, 1) &= \{B\} & \delta'(\{S,B,A\}, 1) &= \{B\} \\
 \delta'(\{S,B,A\}, 0) &= \{A, q_f\} & \delta'(\{S,B,q_f\}, 0) &= \{A, q_f\} & \delta'(\{B,A,q_f\}, 0) &= \{A, q_f\} \\
 \delta'(\{B,A,q_f\}, 1) &= \{B\} & \delta'(\{S,A,q_f\}, 0) &= \{A, q_f\} & \delta'(\{S,A,q_f\}, 1) &= \{B\} \\
 \delta'(\{S,B,A,q_f\}, 0) &= \{A, q_f\} & \delta'(\{S,B,A,q_f\}, 1) &= \{B\}
 \end{aligned}$$

4. $F' = \{\{q_f\}, \{A,q_f\}, \{S,q_f\}, \{B,q_f\}, \{S,B,q_f\}, \{B,A,q_f\}, \{S,A,q_f\}, \{S,B,A,q_f\}\}$
Каждое из подмножеств F' является заключительным состоянием. При

определении достижимого заключительного состояния выбирается соответствующее подмножество.

5. $S = S_0$

6. Достижимыми состояниями в ДКА являются $\{S\}$, $\{B\}$ и $\{A, q_f\}$, остальные состояния удаляются.

Таб. 1.3. Таблица переходов для ДКА

	0	1
$p_0 = \{S\}$	p_1	
$p_1 = \{A, q_f\}$		p_2
$p_2 = \{B\}$	p_1	

7. ДКА = ($\{S, B, A, q_f\}$, $\{0, 1\}$, δ' , S , $\{A, q_f\}$), где

$$\delta'(\{B\}, 0) = \{A, q_f\} \quad \delta'(\{S\}, 0) = \{A, q_f\}$$

$$\delta'(\{A, q_f\}, 1) = \{B\}$$

ДКА представляется следующим графом:

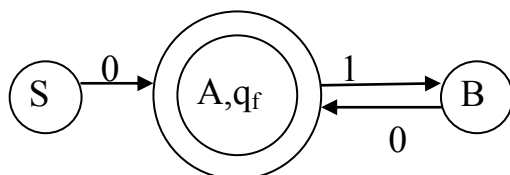


Рис. 1.5. Диаграмма переходов ДКА

Приведенный алгоритм можно использовать и для перехода от недетерминированной А-грамматики к детерминированной.

2.5. Способы реализации (по заданному регулярному выражению составные автоматы)

Реализовать на языке С# КА см. рис. 1.6. . Пользователь вводит цепочку символов в поле редактирования и получает ответ: “Да” – цепочка символов принадлежит языку, автомат распознал цепочку или “Нет” – цепочка символов не принадлежит заданному языку L.

Пример проектирования грамматики и распознавателя

1. Постановка задачи.

Задан язык $L = \{c\omega n \mid \omega \in \{+, d, -, k, f\}^*\}$, где ω обозначает множество всех цепочек, составленных из символов $\{+, d, -, k, f\}^*$. Цепочки начинаются с символа c, заканчиваются символом n.

Спроектировать грамматику $G = (T, V, P, S_0)$ для заданного языка L, привести пример вывода цепочки.

Используя грамматику G, построить КА $= (Q, \Sigma, \delta, q_0, F)$ и привести пример конфигурации КА. Построить диаграмму переходов КА. Определить свойства КА, если это НКА реализовать алгоритм преобразования НДКА в ДКА.

1.a. Спроектировать грамматику $G = (T, V, P, S_0)$, где

$T = \{c, d, k, f, n, +, -\}$ множество терминальных символов

$V = \{S_0, Q\}$ множество нетерминальных символов

$P = \{S_0 \rightarrow Qn, Q \rightarrow c \mid Q+ \mid Q- \mid Qd \mid Qk \mid Qf\}$

1.b. Определить свойства грамматики. Грамматика является левوليнейной, бесконечной. Пример вывода цепочки $S_0 \Rightarrow Qn \Rightarrow Qf-n \Rightarrow Qkf-n \Rightarrow Qdkf-n \Rightarrow Q-dkf-n \Rightarrow Qd-dkf-n \Rightarrow Q+d-dkf-n \Rightarrow c+d-dkf-n$.

1.c. Используя грамматику G построить КА.

КА = $(\{S_0, Q, q_f\}, \{c, d, k, f, n, +, -\}, \delta, S_0, q_f)$

$\delta(S_0, c) = \{Q\}$;

$\delta(Q, d) = \{Q\}$; $\delta(Q, k) = \{Q\}$; $\delta(Q, f) = \{Q\}$; $\delta(Q, +) = \{Q\}$; $\delta(Q, -) = \{Q\}$;

$\delta(Q, n) = \{q_f\}$;

Пример конфигурации КА: $S_0 c+d-dkf-n \vdash Q+d-dkf-n \vdash Qd-dkf-n \vdash Q-dkf-n \vdash Qdkf-n \vdash Qkf-n \vdash Qf-n \vdash Q-n \vdash Qn \vdash q_f$

1.d. Определить свойства конечного автомата. Конечный автомат является детерминированным.

1.e. Построить диаграмму переходов КА рис. 1.6.:

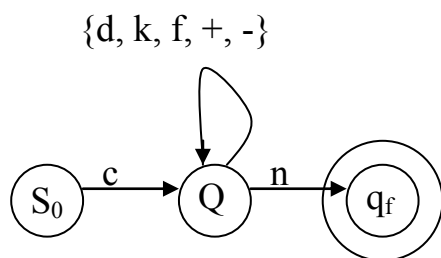


Рис. 1.6. Диаграмма переходов КА

Глава 3. Контекстно-свободные грамматики и МП-автоматы

3.1. Определение КС-грамматик и МП-автоматов

Из четырех типов грамматик иерархии Хомского класс контекстно-свободных грамматик наиболее важен с точки зрения приложений к языкам программирования и компиляции. С помощью этого типа грамматик определяется большая часть синтаксических структур языков программирования.

Определение 6. Контекстно-свободная грамматика типа 2:

Грамматика $G = (T, V, P, S_0)$ называется *контекстно-свободной* (КС) (или *бесконтекстной*), если каждое правило из P имеет вид $A \rightarrow \alpha$, где $A \in V$, $\alpha \in (T \cup V)^*$.

Заметим, что, согласно определению каждая праволинейная грамматика – КС. Языки, порождаемые КС-грамматиками, называются КС-языками.

Пример, КС-грамматики, порождающей скобочные арифметические выражения: $G_1 = (\{v, +, *, (,)\}, \{S, F, L\}, P, S)$, где $P = \{S \rightarrow S + F, S \rightarrow F, F \rightarrow F * L, F \rightarrow L, L \rightarrow v, L \rightarrow (S)\}$.

Пример вывода, заменяя самый левый нетерминал (левосторонний вывод) сентенциальной формы: $S \Rightarrow S + F \Rightarrow F + F \Rightarrow L + F \Rightarrow v + F \Rightarrow v + F * L \Rightarrow v + L * L \Rightarrow v + v * L \Rightarrow v + v * v$.

3.2. Преобразование КС-грамматик

Для построения синтаксических анализаторов необходимо, чтобы КС-грамматика была в *приведенной* форме.

Если применить к КС-грамматике алгоритмы:

- устранения бесполезных символов

1). *Непроизводящих* и

2). *Недостижимых* символов;

3). преобразования в грамматику без ϵ -правил;

4). устранения цепных правил, то получим *приведенную* КС-грамматику. Рассмотрим эти алгоритмы и их применение.

1. Символ $a \in T$ называется *недостижимым* в КС-грамматике G , если он не может появиться ни в одной из сентенциальных форм.

2. Нетерминальный символ $A \in V$ называется *производящим*, если из него можно вывести терминальную цепочку, т.е. если существует вывод $A \Rightarrow^+ \alpha$, где $\alpha \in T^+$. В противном случае символ называется *непроизводящим*.

Алгоритм 3.1. Определение множества *производящих* нетерминальных символов V_p .

Если все символы цепочки из правой части правила вывода являются производящими, то нетерминал в левой части правила вывода также должен быть производящим.

Вход: КС грамматика $G = (T, V, P, S)$.

Выход: $V_p = \{A \mid A \Rightarrow^+ \alpha, A \in V, \alpha \in T^+\}$.

Строим рекурсивно множества $V_p^0, V_p^1, \dots, V_p^i, \dots$

1. Положить $V_p^0 = \emptyset, i=1$.
2. Положить $V_p^i = V_p^{i-1} \cup \{A \mid (A \rightarrow \alpha) \in P, \alpha \in T^+\}$.
3. Если $V_p^i \neq V_p^{i-1}$, положить $i = i+1$ и перейти к шагу 2.
4. Положить $V_p = V_p^i$.

Поскольку $V_p \subseteq V$, то число повторений шага 2. $i+1$, где i – число нетерминальных символов грамматики G .

Алгоритм 3.2. Определение множества *достижимых* символов V_r . Если нетерминал в левой части правила грамматики является достижимым, то достижимы и все символы правой части этого правила.

Вход: КС грамматика $G = (T, V, P, S)$.

Выход: $V_r = \{X \mid S \Rightarrow^* \alpha X \beta, X \in V, \alpha, \beta \in (V \cup T)^*\}$.

Строим рекурсивно множества $V_r^0, V_r^1, \dots, V_r^i, \dots$

1. Положить $V_r^0 = \{S\}, i=1$.
2. Положить $V_r^i = V_r^{i-1} \cup \{X \mid A \rightarrow \alpha X \beta, X \in V, \alpha, \beta \in (V \cup T)^* \text{ и } A \in V_r^{i-1}\}$.
3. Если $V_r^i \neq V_r^{i-1}$, положить $i = i+1$ и перейти к шагу 2.
4. Положить $V_r = V_r^i$.

3. Символ $a \in T \cup V$ называется *бесполезным* в КС-грамматике G , если он *непроизводящий* или *недостижимый*.

Алгоритм 3.3. Устранение *бесполезных* символов. Вначале исключить *непроизводящие* нетерминалы, а затем *недостижимые* символы.

Вход: КС грамматика $G = (T, V, P, S)$, для которой $L(G) \neq \emptyset$.

Выход: КС грамматика $G' = (T', V', P', S)$, у которой $L(G) \neq \emptyset$ и в $T' \cup V'$ нет бесполезных символов.

1. Построить множество V_p производящих нетерминалов грамматики G .
Алгоритм 3.1.
2. Положить $G_1 = (T, V_p, P_1, S)$, где P_1 состоит из правил множества P , содержащих только символы из $T \cup V_p$ (алгоритмы 3.1.)
3. Построить множество V_r (алгоритм 3.2.), достижимых символов грамматики G_1 .
4. Положить $G' = (T', V', P', S)$, где $T' = T \cap V_r, V' = V_r \cap V, P'$ состоит из правил множества P_1 , содержащих только символы из множества V_r .
Алгоритмы 3.1 и 3.2 следует употреблять в указанном порядке.

Алгоритм устранения ϵ -правил в КС-грамматике основан на использовании множества укорачивающих нетерминалов.

КС-грамматика называется *неукорачивающей* КС-грамматикой (НКС-грамматикой, КС-грамматикой без ϵ -правил) при условии, что P не содержит $S \rightarrow \epsilon$ и S не встречается в правах частях остальных правил.

Алгоритм 3.4. Построение множества V_ϵ укорачивающих нетерминалов.

Вход: КС грамматика $G = (T, V, P, S)$.

Выход: $V_\varepsilon = \{A \mid A \Rightarrow^+ \varepsilon, A \in V\}$.

Строим рекурсивно множества укорачивающих нетерминалов $V_\varepsilon^0, V_\varepsilon^1, \dots, V_\varepsilon^i, \dots$.

1. Положить $V_\varepsilon^0 = \{A \mid A \rightarrow \varepsilon, A \in V\}$, $i=1$.
2. Положить $V_\varepsilon^i = V_\varepsilon^{i-1} \cup \{A \mid (A \rightarrow \alpha) \in P \text{ и } \alpha \in (V_\varepsilon^{i-1})^+\}$.
3. Если $V_\varepsilon^i \neq V_\varepsilon^{i-1}$, положить $i = i+1$ и перейти к шагу 2.
4. Положить $V_\varepsilon = V_\varepsilon^i$.

Поскольку $V_\varepsilon \subseteq V$, то число повторений шага 2. $i+1$, где i – число нетерминальных символов грамматики G .

Алгоритм 3.5. Преобразование КС-грамматики с ε -правилами в эквивалентную НКС-грамматику.

Вход: КС грамматика $G = (T, V, P, S)$.

Выход: НКС грамматика $G' = (T, V', P', S')$, порождающая язык $L(G') = L(G)$.

1. Построить множество V_ε , укорачивающих нетерминалов (алгоритмы 3.4.).
2. Положить $P' = \emptyset$.
3. Если $(A \rightarrow \alpha_0 B_1 \alpha_1 B_j \alpha_i \dots B_k \alpha_k) \in P$, где $k \geq 0$, $B_j \in V_\varepsilon$ ($1 \leq j \leq k$) и ни один из символов цепочек $\alpha_i \in (V \cup T)^*$ ($0 \leq i \leq k$) не содержит символов из V_ε , то включить в P' , все правила вывода вида $A \rightarrow \alpha_0 X_1 \alpha_1 X_j \alpha_i \dots X_k \alpha_k$, где $X_j = B_j$ или $X = \varepsilon$.
Если все цепочки $\alpha_i = \varepsilon$, то правила $A \rightarrow \varepsilon$ не включать в P' .
4. Если $S \in V_\varepsilon$, то положить $V' = V \cup \{S'\}$ и добавить в P' два правила: $S' \rightarrow S$ и $S' \rightarrow \varepsilon$. В противном случае положить $V' = V$ и $S' = S$.
5. Положить $G' = (T, V', P', S')$.

В КС грамматиках часто встречаются правила, правая часть которых состоит из одного нетерминального символа. Правило вида $A \rightarrow B$ называется цепным правилом.

Алгоритм 3.6. Устранение цепных правил.

Вход: КС грамматика $G = (T, V, P, S)$.

Выход: Эквивалентная НКС грамматика $G' = (T, V, P', S)$ без цепных правил.

1. Для каждого $A \in V$, построить множество $V_A = \{B \mid A \Rightarrow^+ B\}$ следующим образом:
 - 1.1. Положить $V_A^0 = \{A\}$, $i=1$.
 - 1.2. Положить $V_A^i = V_A^{i-1} \cup \{C \mid (B \rightarrow C) \in P \text{ и } B \in V_A^{i-1}\}$.
 - 1.3. Если $V_A^i \neq V_A^{i-1}$, положить $V_A = V_A^i$.
2. Положить $P' = \emptyset$.
3. Если $B \rightarrow \alpha \in P$ и не является цепным правилом, то положить $P' = P' \cup \{A \rightarrow \alpha\}$ для всех таких A , что $B \in V_A$.
4. Положить $G' = (T, V, P', S)$.

Нетерминал КС-грамматики называется *рекурсивным*, если $A \Rightarrow^+ \alpha A \beta$, для некоторых α и β . Если $\alpha = \varepsilon$, то A называется *леворекурсивным*, если $\beta = \varepsilon$, то A называется *праворекурсивным*. Грамматика, имеющая хотя бы один

леворекурсивный нетерминал, называется *леворекурсивной*. Грамматика, имеющая хотя бы один праворекурсивный, нетерминал называется *праворекурсивной*.

При нисходящем синтаксическом анализе требуется, чтобы приведенная грамматика рассматриваемого языка не содержала **левой рекурсии**.

Для любой КС-грамматики существует эквивалентная грамматика без левой рекурсии. Рассмотрим алгоритм устранения левой рекурсии.

Алгоритм 3.7.

Вход: Приведенная КС грамматика $G = (T, V, P, S_0)$.

Выход: Эквивалентная КС грамматика G' .

1. Пусть $V = \{A_1, \dots, A_n\}$. Преобразуем G так, чтобы в правиле $A_i \rightarrow \alpha$, цепочка α начиналась либо с терминала, либо с такого A_j , что $j > i$. Пусть $i=1$.

2. Пусть множество A_i правил – это $A_i \rightarrow A_i\alpha_1 \mid \dots \mid A_i\alpha_m \mid \beta_1 \mid \dots \mid \beta_p$, где ни одна цепочка β_j не начинается с A_k , если $k \leq i$. Заменяем A_i - правила правилами:

$$A_i \rightarrow \beta_1 \mid \dots \mid \beta_p \mid \beta_1 A_i' \mid \dots \mid \beta_p A_i'$$

$$A_i \rightarrow \alpha_1 \mid \dots \mid \alpha_m \mid \alpha_1 A_i' \mid \dots \mid \alpha_m A_i'$$

где A_i' – новый терминал. Правые части всех A_i - правил начинаются теперь с терминала или с A_k для некоторого $k > i$.

3. Если $i = n$, то останов и получена грамматика G' , иначе $j = i$, $i = i + 1$.

4. Заменить каждое правило вида $A_i \rightarrow A_j\alpha$ правилами $A_i \rightarrow \beta_1\alpha \mid \dots \mid \beta_m\alpha$, где $A_j \rightarrow \beta_1 \mid \dots \mid \beta_m$ – все A_j - правила.

Так как правая часть каждого A_j – правила начинается уже с терминала или с A_k для $k > j$, то и правая часть каждого A_i – правила будет обладать этим же свойством.

5. Если $j = i - 1$, перейти к шагу 2, иначе $j = j + 1$ и перейти к шагу 4.

Частный случай не леворекурсивной грамматики – грамматика в нормальной форме Шейлы Грейбах.

Определение 7. КС грамматика $G = (T, V, P, S)$ называется грамматикой в нормальной форме Грейбах, если в ней нет ϵ -правил, т.е. правил вида $A \rightarrow \epsilon$, и каждое правло из P отличное от $S \rightarrow \epsilon$, имеет вид $A \rightarrow a\alpha$, где $a \in T$, $\alpha \in V^*$.

Также полезно представлять грамматику в нормальной форме Хомского, что позволяет упростить рассмотрение ее свойств.

Определение 8. КС грамматика $G = (T, V, P, S)$ называется грамматикой в нормальной форме Хомского, если каждое правло из P имеет один из следующих видов:

1. $A \rightarrow BC$, где $A, B, C \in V$;
2. $A \rightarrow a$, где $a \in T$;
3. $S \rightarrow \epsilon$, если $\epsilon \in L(G)$, причем S не встречается в правых частях правил.

3.3. Пример: приведенная форма КС-грамматики

Постановка задачи. Привести КС-грамматику к *приведенной* форме, исключить левую рекурсию.

А). Устранить из грамматики G бесполезные символы. Применить алгоритм 3.3. к грамматике $G = (T, V, P, S)$, где $V = \{A, B, C, S\}$, $T = \{a, b, c\}$, $P = \{S \rightarrow aC, S \rightarrow A, A \rightarrow cAB, B \rightarrow b, C \rightarrow a\}$.

Шаг 1. множество $V_p = \{B, C, S\}$ A – *непроизводящий символ*

Шаг 2. $G_1 = (\{B, C, S\}, \{a, b, c\}, P, S)$, где $P = \{S \rightarrow aC, B \rightarrow b, C \rightarrow a\}$,

Шаг 3. $V_r = \{C, S\}$ B – *недостижим, так как A – непроизводящий символ*

Шаг 4. $G' = (\{C, S\}, \{a\}, P, S)$, где $P = \{S \rightarrow aC, C \rightarrow a\}$, $L(G') = \{aa\}$

В). Устранить из грамматики G ϵ -правила, применить алгоритм 3.5. Преобразовать грамматику $G = (T, V, P, S)$, где $V = \{A, S\}$, $T = \{b, c\}$, $P = \{S \rightarrow cA, S \rightarrow \epsilon, A \rightarrow cA, A \rightarrow bA, A \rightarrow \epsilon\}$, в эквивалентную НКС-грамматику.

Шаг 1. Применяя алгоритм 3.4., получаем $V_\epsilon^0 = \{S, A\}$, $V_\epsilon^1 = \{S, A\}$, значит $V_\epsilon^0 = V_\epsilon^1 = \{S, A\}$.

Шаг 2. Положить $P' = \emptyset$.

Шаг 3. Алгоритм 3.5. Рассмотрим правило $S \rightarrow cA$.

Для него в новое множество правил грамматики P' добавляем правила $S \rightarrow cA$ и $S \rightarrow c$.

Для правила $A \rightarrow cA$, добавляем в P' правила $A \rightarrow cA$ и $A \rightarrow c$.

Для правила $A \rightarrow bA$, добавляем в P' $A \rightarrow bA$ и $A \rightarrow b$,

тогда $P' = \{S \rightarrow cA, S \rightarrow c, A \rightarrow cA, A \rightarrow c, A \rightarrow bA, A \rightarrow b\}$.

Шаг 4. $S \in V_\epsilon$, то в V' добавляем новый нетерминал S' , а в P' два правила $S' \rightarrow S$ и $S' \rightarrow \epsilon$.

Шаг 5. $G' = (\{b, c\}, \{S', S, A\}, \{S' \rightarrow S, S' \rightarrow \epsilon, S \rightarrow cA, S \rightarrow c, A \rightarrow cA, A \rightarrow c, A \rightarrow bA, A \rightarrow b\}, S')$.

С). Устранить из КС грамматики G цепные правила, применить алгоритм 3.6. $G = (\{S, F, L\}, \{v, +, *, ()\}, P, S)$, где P состоит из правил

$S \rightarrow S + F \mid F, F \rightarrow F * L \mid L, L \rightarrow v \mid (S)$

Шаг 1. Применяя алгоритм 3.6., получаем $V_S = \{S, F, L\}$, $V_F = \{F, L\}$, $V_L = \{L\}$.

Шаг 2. Положить $P' = \emptyset$.

Шаг 3. Выберем первый нетерминал S из множества V_S . Множество правил, левая часть которых – нетерминальный символ S , правые части – это правые части нецепных правил исходной грамматики, в левой части которых находятся символы из множества V_S , получаем: $\{S \rightarrow S + F \mid F * L \mid (S) \mid v\}$. Включаем эти правила в $P' = \{S \rightarrow S + F \mid F * L \mid (S) \mid v\}$. Таким же образом рассматриваем символы из V_F, V_L ,

Шаг 4. В результате $G' = (T, V, P', S)$, где $P' = \{S \rightarrow S + F \mid F * L \mid (S) \mid v, F \rightarrow F * L \mid (S) \mid v, L \rightarrow v \mid (S)\}$

Д). Исключить левую рекурсию из КС – грамматики G .

1. Устранить левую рекурсию в заданной КС-грамматике G , порождающей скобочные арифметические выражения.

$G = (\{v, +, *, (,)\}, \{S, F, L\}, P, S)$, где P состоит из правил

$S \rightarrow S + F \mid F$

$F \rightarrow F * L \mid L$

$L \rightarrow v \mid (S)$

Применить алгоритм 3.7. к грамматике G . Указаны шаги алгоритма 3.7.

Шаг 1. Пусть $A_1 = S, A_2 = F, A_3 = L$. $V = \{A_1, A_2, A_3\}$.

Шаг 2. Для $i = 1$ преобразуем правила: $S \rightarrow S + F \mid F$, $\alpha = + F$, $\beta = F$.
Заменим S - правила правилами: $S \rightarrow F \mid FS'$, и добавим в грамматику правило для нового нетерминала $S' \rightarrow + F \mid + FS'$.

Шаг 3. $i = 2$ и $j = 1$.

Шаг 4. Для $i = 2, j = 1$ правила вида $A_i \rightarrow A_j \alpha$ отсутствуют.

Шаг 5. Так как $j = i - 1$, то переходим к шагу 2.

Шаг 2. Для $i = 2$ преобразуем правила: $F \rightarrow F * L \mid L$, $\alpha = * L$, $\beta = L$.
Заменим F - правила правилами: $F \rightarrow L \mid LF'$, и добавим в грамматику правило для нового нетерминала $F' \rightarrow * L \mid LF'$.

Шаг 3. $i = 3, j = 1$.

Шаг 4. Для $i = 3, j = 1$ правила вида $A_i \rightarrow A_j \alpha$ отсутствуют.

Шаг 5. Так как $j \neq i - 1$, то $j = j + 1$ переходим к шагу 4.

Шаг 4. Для $i = 3, j = 2$ правила вида $A_i \rightarrow A_j \alpha$ отсутствуют.

Шаг 5. Так как $j = i - 1$, то переходим к шагу 2.

Шаг 2. Для $i = 3$ правила вида $A_i \rightarrow A_j \alpha$ отсутствуют.

Шаг 3. $i = n = 3$, преобразование завершено.

Получили правила новой грамматики G' :

$S \rightarrow F \mid FS'$, $S' \rightarrow + F \mid + FS'$, $F \rightarrow L \mid LF'$, $F' \rightarrow * L \mid LF'$, $L \rightarrow (S + F) \mid v \mid (S)$.

Г) Определить в какой форме (Грейбах, Хомского) находится КС-грамматика G' .

Е). G' – приведенная КС-грамматика.

3.4. Определение МП-автоматов

Автоматы с магазинной памятью (МП - автоматы) представляют собой модель распознавателей для языков, задаваемых КС-грамматиками. МП - автоматы имеют вспомогательную память, называемую магазином см. 1.10. В магазин можно поместить неограниченное количество символов. В каждый момент времени доступен только верхний символ магазина.

Верхний символом магазина будем считать самый левый символ цепочки.



Рис. 1.10. Модель МП - автомата

Определение 9. МП автомат – это семерка объектов

$МП = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

Q – конечное множество состояний устройства управления;

Σ – конечный алфавит входных символов;

Γ – конечный алфавит магазинных символов;

δ – функция переходов, отображает множества $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ в множество конечных подмножеств множества $Q \times \Gamma^*$;

q_0 – начальное состояние, $q_0 \in Q$;

z_0 – начальный символ магазина, $z_0 \in \Gamma$;

F – множество заключительных состояний, $F \subseteq Q$.

Определение 10. Конфигурацией МП-автомата называется тройка $(q, \omega, z) \in Q \times \Sigma^* \times \Gamma^*$, где

q – текущее состояние управляющего устройства;

ω – необработанная часть входной цепочки (первый символ цепочки ω находится под входной головкой; если $\omega = \epsilon$, то считается, что вся входная цепочка прочитана);

z – содержимое магазина (самый левый символ цепочки z считается верхним символом магазина; если $z = \epsilon$, то магазин считается пустым).

Такт МП-автомата будем описывать бинарным отношением \vdash , определенным на множестве конфигураций. Будем писать:

$(q, a\omega, z) \vdash (q', \omega, z\gamma)$, если $\delta(q, a, z) = (q', \gamma)$, где
 $q, q' \in Q, a \in \Sigma \cup \{\epsilon\}, \omega \in \Sigma^*, z \in \Gamma$ и $\gamma \in \Gamma^*$

Если $a \neq \epsilon$, то и входная цепочка прочитана не вся, то запись $(q, a\omega, z\gamma) \vdash (q', \omega, a\gamma)$ означает, что МП-автомат в состоянии q , обозревая символ во входной цепочки и имея символ z в верхушке магазина, может перейти в новое состояние q' , сдвинуть входную головку на один символ вправо и заменить верхний символ магазина z цепочкой магазинных символов γ .

Если $z = a$, то верхний символ удаляется из магазина.

Если $a = \epsilon$, то текущий входной символ в этом такте называется ϵ -тактом, не принимается во внимание и входная головка остается неподвижной.

ϵ -такты могут выполняться также в случае, когда вся входная цепочка прочитана, но если магазин пуст, то такт МП-автомата невозможен по определению.

Так же, как и для конечных автоматов, можно определить транзитивное \vdash^+ и рефлексивно-транзитивное \vdash^* замыкания (отношения \vdash).

Начальной конфигурацией МП-автомата называется конфигурация вида (q_0, ω, z_0) , где устройство управления находится в начальном состоянии, на входной ленте записана цепочка $\omega \in \Sigma^*$, которую необходимо распознать, а магазин содержит только начальный символ z_0 .

Заключительной конфигурацией МП-автомата называется конфигурация вида (q, ε, γ) , где $q \in F$ – одно из заключительных состояний устройства управления, входная цепочка прочитана до конца, а в магазине записана некоторая, заранее определенная цепочка $\gamma \in \Gamma^*$.

Есть два способа определить язык, допускаемый МП-автоматом:

1. множеством входных цепочек, для которых существует опустошающая магазин последовательность операций;
2. множеством входных цепочек, для которых существует последовательность операций, приводящая автомат в заключительное состояние.

Цепочка $\omega \in \Sigma^*$ допускается МП-автоматом $МП = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, если $(q_0, \omega, z_0) \vdash^* (q, \varepsilon, \gamma)$ для некоторых $q \in F$ и $\gamma \in \Gamma^*$.

Язык распознаваемый МП-автоматом, называется множество цепочек:

$$L(МП) = \{ \omega \mid \omega \in \Sigma^* \text{ и } (q_0, \omega, z_0) \vdash^* (q, \varepsilon, \gamma) \text{ для некоторых } q \in F \text{ и } \gamma \in \Gamma^* \}$$

Пример. Определим МП-автомат, допускающий язык $L = \{a^n b^n \mid n \geq 0\}$
Цепочка символов языка L : aaabbb

$$МП = (\{q_0, q_1, q_2, q_f\}, \{a, b\}, \{z_0, a\}, \delta, q_0, z_0, \{q_f\})$$

$$\delta(q_0, a, z_0) = \{(q_1, az_0)\}$$

$$\delta(q_1, a, a) = \{(q_1, aa)\}$$

$$\delta(q_1, b, a) = \{(q_2, \varepsilon)\}$$

$$\delta(q_2, b, a) = \{(q_2, \varepsilon)\}$$

$$\delta(q_2, \varepsilon, z_0) = \{(q_f, \varepsilon)\}$$

Последовательность тактов:

$$\begin{aligned} (q_0, aaabbb, z_0) &\vdash_1 (q_1, aabbb, az_0) \vdash_2 (q_1, abbb, aaz_0) \vdash_2 \\ (q_1, bbb, aaaz_0) &\vdash_3 (q_2, bb, aaz_0) \vdash_4 (q_2, b, az_0) \vdash_4 \\ (q_2, \varepsilon, z_0) &\vdash_5 (q_f, \varepsilon, \varepsilon) \end{aligned}$$

Алгоритм 3.8. По КС-грамматике $G = (T, V, P, S)$ можно построить МП автомат, $L(МП) = L(G)$. Пусть $МП = (\{q\}, \Sigma, \Sigma \cup V, \delta, q, S, \{q\})$, где δ определяется следующим образом:

1. Если $A \rightarrow \alpha$ - правило вывода грамматики G , то $\delta(q, \varepsilon, A) = (q, \alpha)$.
2. $\delta(q, a, a) = \{(q, \varepsilon)\}$ для всех $a \in \Sigma$.

Пример проектирования МП-автомата. Постановка задачи. Построить МП-автомат и расширенный МП-автомат по КС-грамматике $G = (T, V, P, S)$, без левой рекурсии, автомат распознает скобочные выражения. Написать

последовательность тактов для выделенной цепочки. Определить свойства автомата.

1. А). Построить МП-автомат, распознающий скобочные арифметические выражения заданные КС-грамматикой (не приведенная) $G_1 = (\{v, +, *, (,)\}, \{S, F, L\}, P, S)$, где P состоит из правил: $S \rightarrow S + F \mid F, F \rightarrow F * L \mid L, L \rightarrow v \mid (S)$.

Используя алгоритм 3.8. получим: $МП = (\{q\}, \{v, +, *, (,)\}, \{v, +, *, (,)\}, S, F, L\}, \delta, q_0, S, \{q\})$, в котором функция переходов δ определяется следующим образом:

1. $\delta(q_0, \varepsilon, S) = \{(q, S + F), (q, F)\};$
2. $\delta(q, \varepsilon, F) = \{(q, F * L), (q, L)\};$
3. $\delta(q, \varepsilon, L) = \{(q, v), (q, (S))\};$
4. $\delta(q, a, a) = \{(q, \varepsilon)\}$ для всех $a \in \Sigma = \{v, +, *, (,)\}$.

В). Последовательность тактов МП-автомата для цепочки $v^*(v+v)$:
 $(q_0, v * (v + v), S) \vdash_1 (q, v * (v + v), F) \vdash (q, v * (v + v), F * L) \vdash (q, v * (v + v), L * L) \vdash (q, v * (v + v), v * L) \vdash (q, * (v + v), * L) \vdash (q, (v + v), L) \vdash (q, (v + v), (S)) \vdash (q, v + v, S) \vdash (q, v + v, S + F) \vdash (q, v + v, F + F) \vdash (q, v + v, L + F) \vdash (q, v + v, v + F) \vdash (q, + v, + F) \vdash (q, v, F) \vdash (q, v, L) \vdash (q, v, v) \vdash (q, ,) \vdash (q, \varepsilon, \varepsilon).$

Последовательность тактов, которую выполняет МП-автомат, соответствует левому выводу цепочки $v * (v + v)$ в грамматике G_1 .

Тип синтаксических анализаторов, которые можно построить таким образом называют *нисходящим* (предсказывающим) анализатором.

Синтаксические анализаторы, построенные на основе расширенного МП-автомата, называют *восходящими* анализаторами, вывод строится снизу в верх.

Определение 11. Расширенным РМП-автоматом называется семерка объектов $РМП = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, где верхним элементом магазина является самый правый символ цепочки, δ -функция переходов, которая задает отображение множества $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ в множество конечных подмножеств множества $Q \times \Gamma^*$, а все остальные объекты такие же, как и у МП-автомата см. определение 9.

Конфигурация расширенного МП-автомата определяется так же, как и для МП-автомата.

Алгоритм 3.9. По КС-грамматике $G = (T, V, P, S)$ можно построить расширенный РМП автомат, $L(РМП) = L(G)$. Будем обозначать символом \perp - конец цепочки. Пусть $РМП = (\{q, r\}, \Sigma, \Sigma \cup V \cup \perp, \delta, q, \perp, \{r\})$, где δ определяется следующим образом:

1. $\delta(q, a, \varepsilon) = \{(q, a)\}$ для всех $a \in \Sigma$ (символы с входной ленты на этих тактах переносятся в магазин).
2. Если $A \rightarrow \alpha$ - правило вывода грамматики G , то $\delta(q, \varepsilon, \alpha) = (q, A)$.

$$3. \delta(q, \varepsilon, \perp S) = \{(r, \varepsilon)\}$$

РМП-автомат продолжает работу пока магазин не станет пустым.

На практике часто используются детерминированные МП-автоматы.

Пример построения расширенного РМП-автомата

А). Построение расширенного РМП-автомата $(Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, используем алгоритм 3.9, получим $\text{РМП} = (\{q, r\}, \{v, +, *, (,)\}, \{v, +, *, (,)\}, S, F, L, \perp, \delta, q, \perp, \{r\})$, где функция переходов δ определена следующим образом:

1. $\delta(q, a, \varepsilon) = \{(q, a)\}$ для всех $a \in \Sigma = \{v, +, *, (,)\}$;
2. $\delta(q, \varepsilon, S+F) = \{(q, S)\}$;
3. $\delta(q, \varepsilon, F) = \{(q, S)\}$;
4. $\delta(q, \varepsilon, F * L) = \{(q, F)\}$;
5. $\delta(q, \varepsilon, L) = \{(q, F)\}$;
6. $\delta(q, \varepsilon, v) = \{(q, L)\}$;
7. $\delta(q, \varepsilon, (S)) = \{(q, L)\}$;
8. $\delta(q, \varepsilon, \perp S) = \{(r, \varepsilon)\}$;

В). При анализе входной цепочки $v+v*v$ расширенный РМП-автомат выполнит следующую последовательность тактов:

$$\begin{aligned}
 &(q, v+v*v, \perp) \xrightarrow{1} (q, +v*v, \perp v) \\
 &\xrightarrow{6} (q, +v*v, \perp L) \\
 &\xrightarrow{5} (q, +v*v, \perp F) \\
 &\xrightarrow{3} (q, +v*v, \perp S) \\
 &\xrightarrow{1} (q, v*v, \perp S+) \\
 &\xrightarrow{1} (q, *v, \perp S+v) \\
 &\xrightarrow{6} (q, *v, \perp S+L) \\
 &\xrightarrow{5} (q, *v, \perp S+F) \\
 &\xrightarrow{1} (q, v, \perp S+F*) \\
 &\xrightarrow{1} (q, \varepsilon, \perp S+F*v) \\
 &\xrightarrow{6} (q, \varepsilon, \perp S+F*L) \\
 &\xrightarrow{4} (q, \varepsilon, \perp S+F) \\
 &\xrightarrow{2} (q, \varepsilon, \perp S) \\
 &\xrightarrow{8} (r, \varepsilon, \varepsilon)
 \end{aligned}$$

МП-автомат и расширенный РМП-автоматы – детерминированные автоматы.

Обычно синтаксический анализ выполняется путем моделирования МП-автомата, анализирующего входные цепочки.

МП-автомат отображает входные цепочки в соответствующие левые выводы (нисходящий анализ) или правые разборы (восходящий анализ).

Пусть задана КС-грамматика $G = (T, V, P, S)$, правила которой пронумерованы числами $1, 2, \dots, p$, и цепочка $\alpha \in (T \cup V)^*$,

- левым разбором цепочки α называется последовательность правил, примененных при ее левом выводе из S ;
- правым разбором цепочки α называется обращение последовательности правил, примененных при ее левом выводе из S .

Определение 12. Детерминированным МП-автоматом (ДМП-автоматом) с магазинной памятью называется МП-автомат $\text{ДМП} = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, у которого для каждого $q \in Q$ и $z \in \Gamma$ выполняется одно из условий:

1. $\delta(q, a, z)$ содержит не более одного элемента для каждого $a \in \Sigma$ и $\delta(q, \varepsilon, z) = \emptyset$;
2. $\delta(q, a, z) = \emptyset$ для всех $a \in \Sigma$, и $\delta(q, \varepsilon, z)$ содержит не более одного элемента.

Если каждое правило КС-грамматики начинается с терминального символа, причем альтернативы начинаются с различных символов, то достаточно просто построить *детерминированный* синтаксический анализатор.

3.5. Способы реализации синтаксических анализаторов

Рассмотрим три способа реализации синтаксических анализаторов для заданного языка $L(G)$. Каждый имеет свои достоинства и недостатки. Напомним, что G является приведенной.

1. МП-автомат строим для $L(G) = L(MP)$, правила автомата строятся в соответствии с рассмотренными алгоритмами.

2. Строятся диаграммы Вирта, для каждой грамматики, создается неявный стек при косвенной рекурсии, что не требует реализации.

3. Таблично-управляемый разбор (предварительно строится диаграмма Вирта), диаграмма реализуется в виде списочной структуры, над которой выполняется алгоритм.

Диаграммы Вирта

1. Постановка задачи. Реализовать синтаксический анализатор левым разбором, используя приведенную грамматику без левой рекурсии $G = (\{a, b, c, -, +, (,)\}, \{E, T, F, \text{op}\}, P, S)$, где $P = \{E \rightarrow T \text{ op } T, \text{op} \rightarrow + \mid -, T \rightarrow F \mid (E), F \rightarrow a \mid b \mid c\}$

2. Шаги реализации.

Выделить цепочку, принадлежащую языку задаваемому КС-грамматикой G , например: $L(G) = a + (b - c)$.

Привести вывод выделенной цепочки:

$$E \Rightarrow T \text{ op } T \Rightarrow T \text{ op } (E) \Rightarrow F \text{ op } (T \text{ op } T) \Rightarrow a \text{ op } (T \text{ op } T) \Rightarrow a + (T \text{ op } T) \\ \Rightarrow a + (F \text{ op } T) \Rightarrow a + (b \text{ op } T) \Rightarrow a + (b \text{ op } F) \Rightarrow a + (b \text{ op } c) \Rightarrow a + (b - c)$$

Пронумеровать правила грамматики и представить их, используя метасимволы $\{\dots\}$, := расширенной формы Бекуса-Наура:

$p_1: E := \{ T \text{ op } T \}, p_2: \text{op} := \{ +, - \}, p_3: T := \{ F, (E) \}, p_4: F := \{ a, b, c \}$

$p_4: T := \{\{a, b, c\}, (E)\}$

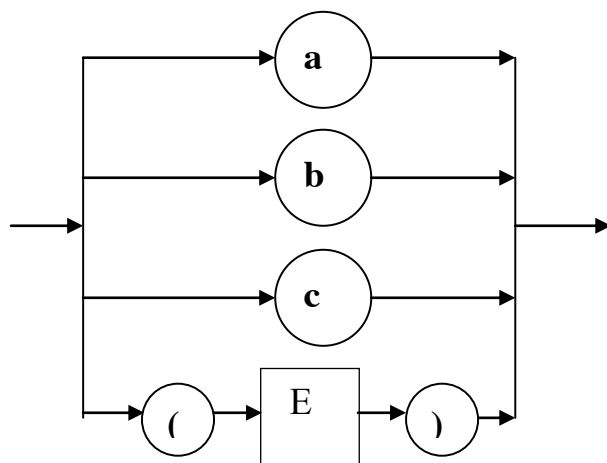


Рис 1.12. Преобразованные диаграммы Вирта.

Полученные правила продолжения на графе соответствуют нормальной форме Грейбах.

Преобразовать каждый синтаксический граф в описание процедуры в соответствии с правилами:

1. Каждому графу (диаграмме Вирта) ставится в соответствие процедура.
2. Элемент графа, обозначающий другой граф, переводится в оператор обращения к процедуре.
3. Последовательность элементов в последовательный вызов операторов. Выбор элементов в switch или условный оператор if.

Входные и выходные данные. Пользователь вводит строку в поле редактирования и получает ответ: Да – строка принадлежит языку, порождаемому данной грамматикой, и Нет – строка не принадлежит этому языку. Программа распознает язык $L(G)$ по заданной КС-грамматике G .

Таблично-управляемый разбор

1. Грамматика задается в виде данных, процедура разбора универсально для всех грамматик. Программа работает в строгом соответствии с методом простого нисходящего грамматического разбора и основывается на детерминированном синтаксическом графе (т.е. предложение должно анализироваться просмотром вперед на один символ без возврата).

2. Естественный способ представить граф в виде структуры данных, а не программ – это ввести узел для каждого символа и связать эти узлы с помощью ссылок. Выделим два типа узлов для терминальных (идентифицируется терминальным символом) и нетерминальных символов (ссылка на данные нетерминального символа). Тогда структура узла графически может быть представлена как:

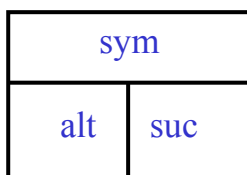


Рис. 1. 14. Структура узла Node

Где “suc” - последователь, “alt” – альтернатива. Пустую последовательность обозначим “empty”. Конструируется класс Node (Узел) для данной структуры.

3. Правила преобразования графов, в структуру данных: получить как можно меньшее число графов с помощью подстановок.

4. Последовательность элементов преобразовать в список узлов (горизонтальный по “suc”), список альтернатив в список узлов (горизонтальный по “alt”), цикл в узел с указателем “suc” на себя, а “alt” на узел с “empty”, где “alt” = NULL, а “suc” выход из цикла.

5. Построим реализацию в виде структуры данных для рассмотренных в п.3 диаграмм.

6. Программа, реализующая структуру на рис. 1.15 приведена ниже. Булевская переменная b управляет алгоритмом разбора. Переменная state определяет состояние цепочки символов.

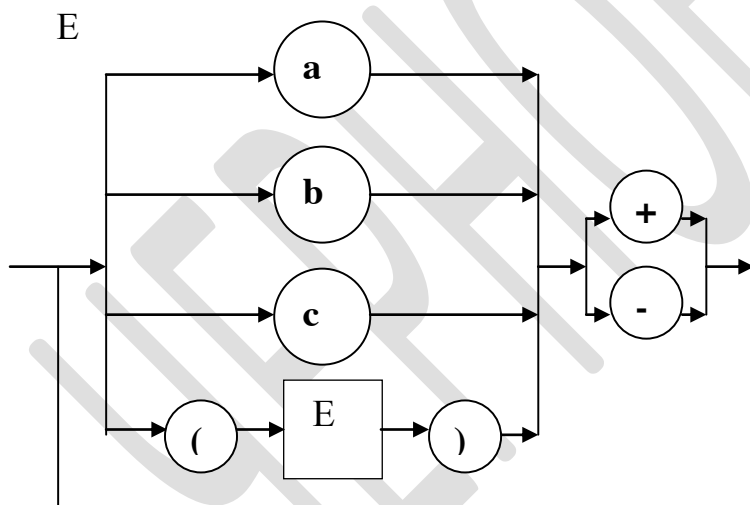


Рис 1.15. Диаграмма Вирта для КС-грамматики G.

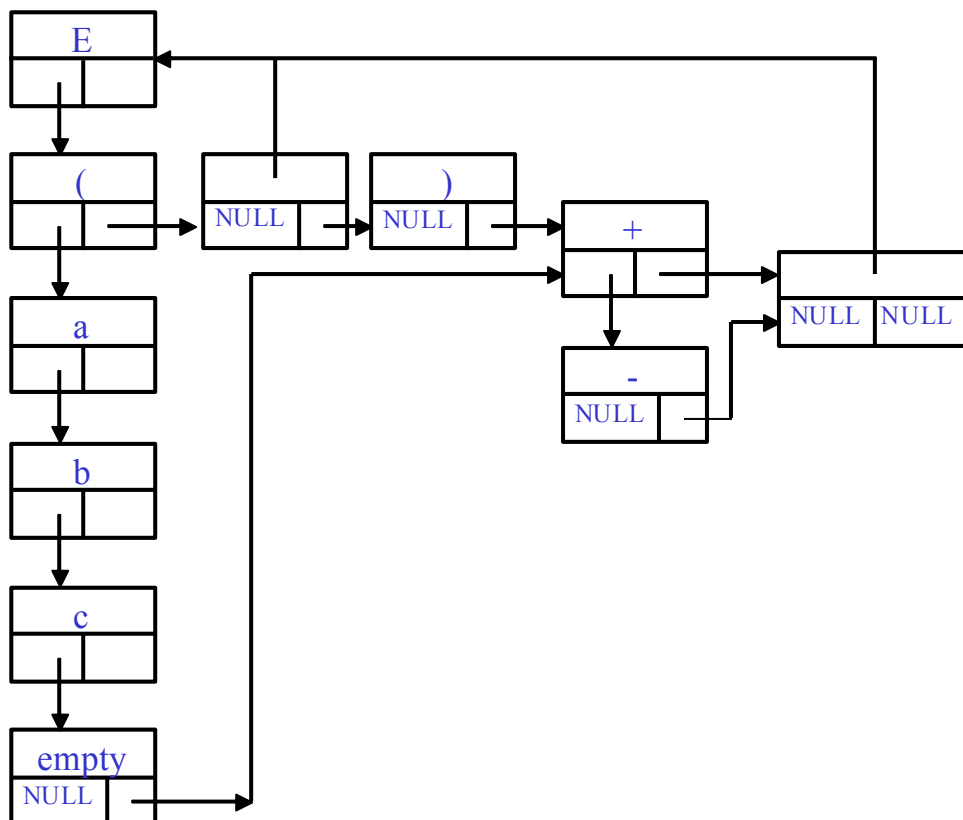


Рис. 1.16. Структура данных для диаграмм п.5

На рис. 1.16. E – начальная вершина Node.

3.6. LL(k)-грамматики

Синтаксический LL-анализатор - анализирует цепочку символов входного алфавита на ленте слева (L) направо, и строит левый (L) вывод грамматики.

Определение 13. КС-грамматика $G = (T, V, P, S)$ без ϵ -правил называется простой LL(1) грамматикой (s-грамматикой, разделенной грамматикой), если для каждого $v \in V$ все его альтернативы начинаются различными терминальными символами. Единица в названии алгоритма означает, что при чтении анализируемой цепочки, находящейся на входной ленте, входная головка может заглядывать вперед на один символ.

$FIRST(A)$ – это множество первых терминальных символов, которыми начинаются цепочки, выводимые из нетерминала $A \in V$:

$$FIRST(A) = \{a \in T \mid A \Rightarrow^+ a\beta, \text{ где } \beta \in (T \cup V)^*\}$$

$FOLLOW(A)$ – это множество следующих терминальных символов, которые могут встретиться непосредственно справа от нетерминала в некоторой сентенциальной форме:

$$FOLLOW(A) = \{a \in T \mid S \Rightarrow^* aA\gamma \text{ и } a \in FIRST(\gamma)\}$$

Магазин содержит цепочку $Xa\perp$ (см. рис. 1.17), где Xa – цепочка магазинных символов (X - верхний символ магазина), а символ (\perp) – специальный символ, называемый *маркером дна* магазина. Если верхним

символом магазина является *маркер дна*, то магазин пуст. Выходная лента содержит цепочку номеров правил π , представляющую собой текущее состояние левого разбора.

Входная лента

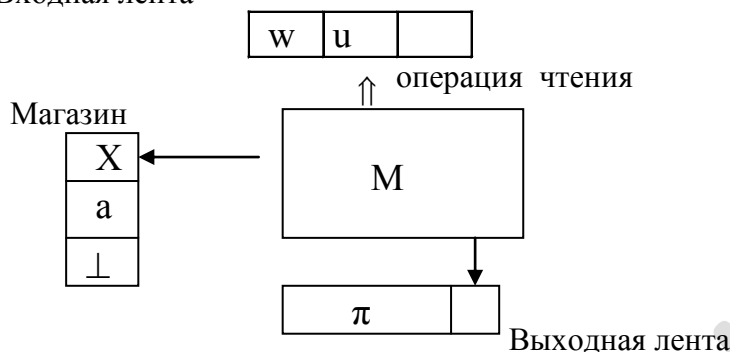


Рис. 1.18. LL(k)-анализатор

Конфигурацию “1-предсказывающего” алгоритма разбора будем представлять в виде $(x, Xa\perp, \pi)$, где x – неиспользуемая часть входной цепочки, Xa – цепочка в магазине, а π – цепочка на выходной ленте, отражающая состояние алгоритма.

Обозначим алфавит магазинных символов (без символа (\perp)) как V_p .

M – управляющая таблица управляет работой алгоритма. M задает отображение множества $(V_p \cup \{\perp\}) \times (T \cup \{\epsilon\})$ в множество, состоящее из следующих элементов:

1. (β, i) , где βV_p^* правая часть правила вывода с номером i .
2. ВЫБРОС.
3. ДОПУСК.
4. ОШИБКА.

Работа алгоритма в зависимости от элемента управляющей таблицы $M(X, a) = (\beta, i)$ следующая:

1. $(x, Xa, \pi) \vdash (x, \beta a, \pi i)$, если $M(X, a) = (\beta, i)$. В этом случае верхний символ магазина X заменяется на цепочку βV_p^* , и в выходную цепочку дописывается номер правила i . Выходная головка при этом не сдвигается.

2. $(ax, aa, \pi) \vdash (x, a, \pi)$, если $M(a, a) = \text{ВЫБРОС}$. Это означает, что, если верхний символ магазина совпадает с текущим входным символом, он выбрасывается из магазина, и входная головка сдвигается на один символ вправо.

3. Если алгоритм достигает конфигурации (ϵ, \perp, π) , что соответствует элементу управляющей таблицы $M(\perp, \epsilon) = \text{ДОПУСК}$, то его работа прекращается, и выходная цепочка π является левым разбором входной цепочки.

4. Если алгоритм достигает конфигурации (x, Xa, π) и $M(X, a) = \text{ОШИБКА}$, то разбор прекращается и выдается сообщение об ошибке.

Конфигурация $(\omega, S\perp, \epsilon)$, где $S \in V_p$ – начальный символ магазина (начальный символ грамматики), называется *начальной конфигурацией*.

Если $(\omega, S\perp, \epsilon) \vdash^+ (\epsilon, \perp, \pi)$, то π называется выходом алгоритма для входа ω .

Алгоритм 3.10. Построение управляющей таблицы M для $LL(1)$ -грамматики

Вход: $LL(1)$ -грамматика $G = (T, V, P, S)$

Выход: Управляющая таблица M для грамматики G .

Таблица M определяется на множестве $(V \cup T \cup \{\perp\}) \times (T \cup \{\epsilon\})$ по правилам:

1. Если $A \rightarrow \beta$ – правило вывода грамматики с номером i , то $M(A, a) = (\beta, i)$ для всех $a \neq \epsilon$, принадлежащих множеству $FIRST(\beta)$.

Если $\epsilon \in FIRST(\beta)$, то $M(A, b) = (\beta, i)$ для всех $b \in FOLLOW(A)$.

2. $M(a, a) = \text{ВЫБРОС}$ для всех $a \in T$.

3. $M(\perp, \epsilon) = \text{ДОПУСК}$.

4. В остальных случаях $M(X, a) = \text{ОШИБКА}$ для $X(V \cup T \cup \{\perp\})$ и $a \in T \cup \{\epsilon\}$

Естественным обобщением $LL(1)$ -грамматик являются $LL(k)$ -грамматики. Для КС-грамматики $G = (T, V, P, S)$ определим множество:

$$FIRST_k(\alpha) = \{x \mid \alpha \Rightarrow_i^* x\beta \text{ и } |x| = k \text{ или } \alpha \Rightarrow^* x \text{ и } |x| < k\}$$

Применение алгоритма 3.10. требует использование аванцепочек длиной до k символов, что существенно увеличивает размеры управляющей таблицы.

Кроме того, для некоторых $LL(k)$ -грамматик ($k > 1$) верхний символ магазина и аванцепочка длиной k (или меньше) символов не всегда однозначно определяют правило, которое должно быть определено при разборе.

Если в КС-грамматике $G = (T, V, P, S)$ для двух различных A -правил $A \rightarrow \beta$ и $A \rightarrow \gamma$ выполняется:

$$FIRST_k(\beta \cdot FOLLOW_k(A)) \cap FIRST_k(\gamma \cdot FOLLOW_k(A)) = \emptyset, \text{ то такая}$$

КС-грамматика называется *сильно* $LL(k)$ -грамматикой.

Сложность построения синтаксических анализаторов и их неэффективность не позволяют практически использовать $LL(k)$ -грамматики.

Проектирование $LL(k)$ -анализатора.

А) Построить управляющую таблицу M для грамматики $G = (T, V, P, S)$, где $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$. $p_1: S \rightarrow TE'$, $p_2: E' \rightarrow +TE'$, $p_3: E' \rightarrow \epsilon$, $p_4: T \rightarrow PT'$, $p_5: T' \rightarrow *PT'$, $p_6: T' \rightarrow \epsilon$, $p_7: P \rightarrow (S)$, $p_8: P \rightarrow i$

В) Рассмотреть работу алгоритма для выделенной цепочки.

С) Определить является ли $LL(k)$ -грамматика *сильно* $LL(k)$ -грамматикой.

А) Используем алгоритм 3.10. Управляющая таблица должна содержать 11 строк, помеченных символами из множества $(VT \cup \{\perp\})$, и 6 столбцов, помеченных символами из множества $(T \cup \{\epsilon\})$.

	i	()	+	*	ϵ
S	TE', 1	TE', 1				
E'			$\epsilon, 3$	+TE', 2		$\epsilon, 3$
T	PT', 4	PT', 4				
T'			$\epsilon, 6$	$\epsilon, 6$	*PT', 5	$\epsilon, 6$
P	i, 8	(S), 7				
i	ВЫБРОС					
(ВЫБРОС				
)			ВЫБРОС			
+				ВЫБРОС		
*					ВЫБРОС	
\perp						ДОПУСК

Шаг 1. Строим таблицу построчно. Последовательно рассмотрим все нетерминальные символы.

1. Нетерминалу S соответствует правило вывода грамматики $p_1: S \rightarrow TE'$. Так как $FIRST(TE') = \{ (, i \}$, два терминальных символа, то:

$$M(S, () = M(S, i) = M(TE', 1)$$

2. Для нетерминала E' в грамматике имеются два правила вывода p_2 и p_3 :

$p_2: E' \rightarrow +TE'$ множество $FIRST(+TE') = \{ + \}$ и, следовательно, $M(E', +) = M(+TE', 2)$;

$p_3: E' \rightarrow \epsilon$ имеет простую правую часть, вычислим множество символов, следующих за нетерминалом E' в сентенциальных формах. Построив левый вывод $S \Rightarrow TE' \Rightarrow PT'E' \Rightarrow (S)T'E' \Rightarrow (TE')TE' \dots$, имеем $FOLLOW(E') = \{), \epsilon \}$.

Таким образом, $M(E',)) = M(E', \epsilon) = (\epsilon, 3)$.

Выполняя шаг 1 алгоритма для нетерминалов T, T' и P получим:

Правило грамматики	Множество	Значение M
$p_4: T \rightarrow PT'$ $p_5: T' \rightarrow *PT'$ $p_6: T' \rightarrow \epsilon$	$FIRST(PT') = \{ (, i \}$ $FIRST(*PT') = \{ * \}$ $FOLLOW(T') = \{ +,), \epsilon \}$	$M(T', () = M(T', i) = M(TE', 4)$ $M(T', *) = (*PT', 5)$ $M(T', +) = M(T',)) = M(T', \epsilon) = (\epsilon, 6)$ $M(P, () = ((S), 7)$ $M(P, i) = (i, 8)$
$p_7: P \rightarrow (S)$ $p_8: P \rightarrow i$	$FIRST((S)) = \{ (\}$ $FIRST(i) = \{ i \}$	

Шаг 2. Далее всем элементам таблицы, находящимся на пересечении строки и столбца, отмеченных одним и тем же терминальным символом, присвоим значение ВЫБРОС.

Шаг 3. Элементу таблицы $M(\perp, \epsilon)$ присвоим значение ДОПУСК.

Шаг 4. Остальным элементам таблицы присвоим значение ОШИБКА и представим результат в виде таблицы.

Начальное содержимое магазина - $S\perp$

В) Рассмотрим работу алгоритма для цепочки $i+i*i$.

Шаг 1. Алгоритм находится в начальной конфигурации $(i+i*i, S\perp, \varepsilon)$. Значение управляющей таблицы $M(S, i) = M(TE', 1)$, при этом выполняются следующие действия:

- заменить верхний символ магазина S цепочкой TE' ;
- не сдвигать читающую головку;
- на выходную ленту поместить номер использованного правила 1.

Шаг 2. Выполняя действия, аналогичные описанным для шага 1, получим следующие конфигурации:

Текущая конфигурация	Значение М
$(i+i*i, TE'\perp, 1) \vdash$	$M(T, i) = (PT', 4)$
$(i+i*i, PT'E'\perp, 14) \vdash$	$M(P, i) = (i, 8)$
$(i+i*i, iT'E'\perp, 148) \vdash$	$M(i, i) = \text{ВЫБРОС}$

Шаг 3. Алгоритм находится в конфигурации $(i+i*i, iT'E'\perp, 148)$. $M(i, i) = \text{ВЫБРОС}$. Выполняем следующие действия:

- удаляем верхний символ магазина;
- сдвигаем читающую головку на один символ вправо;
- при этом выходная лента не изменяется.

Алгоритм переходит в конфигурацию $(+i*i, T'E'\perp, 148)$.

Выполняя шаги 1 и 2, алгоритм произведет следующие действия:

Текущая конфигурация	Значение М
$(+i*i, T'E'\perp, 148) \vdash$	$M(T', +) = (\varepsilon, 6)$
$(+i*i, E'\perp, 1486) \vdash$	$M(E', +) = (+TE', 2)$
$(+i*i, +TE'\perp, 14862) \vdash$	$M(+, +) = \text{ВЫБРОС}$
$(i*i, TE'\perp, 14862) \vdash$	$M(T, i) = (PT', 4)$
$(i*i, PT'E'\perp, 148624) \vdash$	$M(P, i) = (i, 8)$
$(i*i, iT'E'\perp, 1486248) \vdash$	$M(i, i) = \text{ВЫБРОС}$
$(*i, T'E'\perp, 1486248) \vdash$	$M(T', *) = (*PT', 5)$
$(*i, *PT'E'\perp, 14862485) \vdash$	$M(*, *) = \text{ВЫБРОС}$
$(i, PT'E'\perp, 14862485) \vdash$	$M(P, i) = (i, 8)$
$(i, iT'E'\perp, 148624858) \vdash$	$M(i, i) = \text{ВЫБРОС}$
$(\varepsilon, T'E'\perp, 148624858) \vdash$	$M(T', \varepsilon) = (\varepsilon, 6)$
$(\varepsilon, E'\perp, 1486248586) \vdash$	$M(E', \varepsilon) = (\varepsilon, 3)$
$(\varepsilon, \perp, 14862485863) \vdash$	

Шаг 5. Алгоритм находится в конфигурации $(\varepsilon, \perp, 14862485863)$

Так как значение $M(\perp, \varepsilon) = \text{ДОПУСК}$, то цепочка $i+i*i$ принадлежит языку и последовательность номеров правил 14862485863 на выходной ленте – это ее разбор.

С) Пусть задана LL(2)-грамматика G с правилами $\{S \rightarrow aAaa \mid bAba, A \rightarrow b \mid \varepsilon\}$ множество $FOLLOW_2(A) = \{aa, ba\}$, а значит, для A -правила:

FIRST₂($b \cdot FOLLOW_2(A)$) \cap FIRST₂($\gamma \cdot FOLLOW_2(A)$) = $\{ba\}$, и это не *сильно* LL(2)-грамматика.

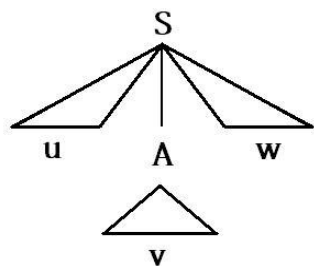
3.7. LR(k)-грамматики

Синтаксический LR-анализатор - анализирует входную цепочку слева направо (L), и строит правый (R) вывод грамматики.

Грамматики, для которых можно построить детерминированный восходящий анализатор, получили название *LR(k)-грамматик* (входная цепочка читается слева (Left) направо, выходом анализатора является правый (Right) разбор, k-число символов входной цепочки, на которое можно “заглянуть” вперед для выделения основы).

Наиболее наглядно *LR(k)-грамматику* можно определить в терминах деревьев вывода.

Грамматика $G = (T, V, P, S)$ является *LR(k)-грамматикой*, если просмотрев только часть кроны дерева вывода в этой грамматике, расположенной слева от данной внутренней вершины, часть кроны, выведенную из нее, и следующие k символов входной цепочки, можно установить правило вывода, которое было применено к этой вершине при порождении входной цепочки.



Дерево вывода цепочки uvw.

В определении *LR(k)-грамматики* используется

1. множество $FIRST_k(\gamma)$, состоящее из префиксов длины k терминальных цепочек, выводимых из γ .

Если из γ выводятся терминальные цепочки, длина которых меньше k, то эти цепочки также включаются в множество $FIRST_k(\gamma)$. Формально:

$$FIRST_k(\gamma) = \{x \mid \gamma \Rightarrow^*_1 xw \text{ и } |x| = k \text{ или } \gamma \Rightarrow^*_1 x \text{ и } |x| < k\}$$

2. Пополненной грамматикой, полученной из КС-грамматики $G = (T, V, P, S)$, называется грамматика $G' = (V \cup \{S'\}, T, P \cup \{S' \rightarrow S\}, S')$. Если правила грамматики G' пронумерованы числами 1, 2, ..., p то, будем считать, что $S' \rightarrow S$ – нулевое правило грамматики G' , а нумерация остальных правил такая же, как в грамматике G.

Определение 14. КС-грамматика $G = (T, V, P, S)$ называется *LR(k)-грамматикой* для $k \geq 0$, если из существования двух правых выводов для *пополненной* грамматики $G' = (T, V', P', S')$ полученной из G:

$$\begin{aligned} S' &\Rightarrow_r^* \alpha A w \Rightarrow_r \alpha \beta w, \\ S' &\Rightarrow_r^* \gamma B x \Rightarrow_r \alpha \beta y, \end{aligned}$$

для которых $FIRST_k(w) = FIRST_k(y)$ следует, что $\alpha Ay = \gamma Bx$.

То есть, если $\alpha\beta w$ и $\alpha\beta y$ – правовыводимы цепочки пополненной грамматики G' , у которых $FIRST_k(w) = FIRST_k(y)$ и $A \rightarrow \beta$ – последнее правило, использованное в правом выводе цепочки $\alpha\beta w$, то правило $A \rightarrow \beta$ должно использоваться также в правом разборе при свертке $\alpha\beta y$ к αAy .

Поскольку правило грамматики $A \rightarrow \beta$ не зависит от w , то из определения LR(k)-грамматики следует, что во множестве $FIRST_k(w)$ содержится информация достаточная для определения **основы**: кодируемая цепочка символов в верхней части магазина.

Для любой LR(k)-грамматики $G = (T, V, P, S)$ можно построить детерминированный анализатор, который выдает правый разбор входной цепочки.

Анализатор состоит из магазина, входной ленты, выходной ленты и управляющего устройства (пара функций f и g) см. рис. 1.18.

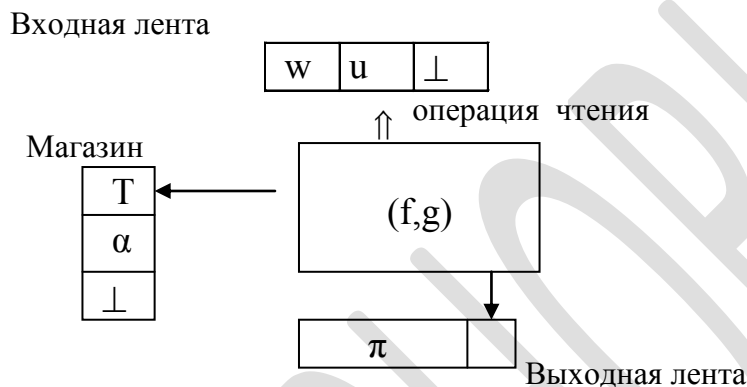


Рис. 1.18. LR(k)-анализатор

Магазинный алфавит V_p представляет собой множество специальных символов, соответствующих грамматическим вхождениям или их множествам.

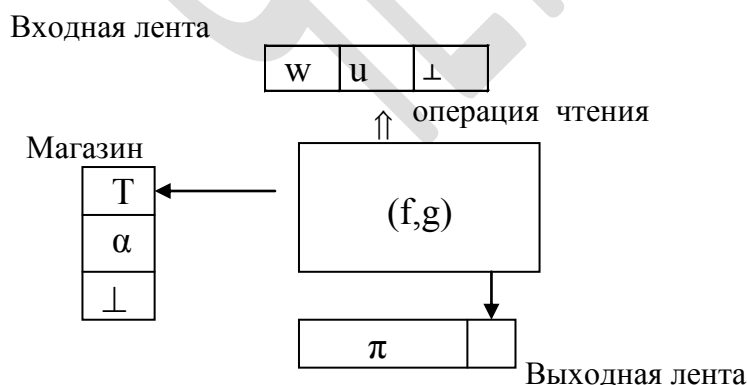


Рис. 1.18. LR(k)-анализатор

Рассмотрим алгоритм “перенос-свертка”, выполняемый детерминированным восходящим синтаксическим анализатором.

Алгоритм состоит в переносе входных символов в магазин до тех пор, пока в его верхней части не встретится основа.

В этот момент производится свертка, в результате которой основа заменяется левой частью основывающего правила. Этот процесс продолжается до тех пор, пока не будет прочитана вся входная цепочка, а в магазине не останется начальный символ грамматики или алгоритм не выдаст сообщение об ошибке.

Пусть в КС-грамматике G имеется правовыводимая цепочка $S \Rightarrow_r^* \alpha x$, такая, что цепочка α заканчивается нетерминальным символом (или это пустая цепочка).

Назовем цепочку α открытой частью цепочки αx , а x – замкнутой частью цепочки. Граница между открытой и замкнутой частью цепочки называется рубежом.

Алгоритм “перенос-свертка” можно рассматривать как программу работы расширенного детерминированного МП-преобразователя.

Рассмотрим правый вывод $S \Rightarrow_r \alpha_0 \Rightarrow_r \alpha_1 \Rightarrow_r \dots \Rightarrow_r \alpha_m = w$. Допустим, что $\alpha_{i-1} = \gamma Bz$ и на i -том шаге вывода $\alpha_{i-1} \Rightarrow_r \alpha_i$ было применено правило $B \rightarrow \beta u$ грамматики. Пусть $\gamma B = \alpha A$ и $uz = x$. Тогда цепочка $\alpha_i = \gamma \beta uz = \alpha Ax$. При этом преобразователь моделирует обращение вывода $\alpha_{i-1} \Leftarrow \alpha_i$.

В начале моделирования i -го шага состояние ДМП-преобразователя определяется цепочкой α_i : в магазине находится αA , а на входной ленте – x . Выполняя разбор, преобразователь будет переносить символы из цепочки x в магазин, пока не определит правый конец основы (возможно, ни одного переноса не потребуется). К этому моменту в магазин будет перенесена цепочка u и в нем будет находиться цепочка $\alpha Au = \gamma Bu$.

Преобразователь выполнит свертку, используя правило $B \rightarrow \beta u$, в магазине появится цепочка γB – открытая часть правовыводимой цепочки α_{i-1} , а на входной ленте останется цепочка z – замкнутая часть α_{i-1} .

Моделирование шага вывода завершено.

Два способа построения LR(k) анализаторов

Два способа:

1. Определения активных префиксов - использование расширенного магазинного алфавита (алгоритм перенос-свертка)
2. На основе грамматического вхождения.

Использование расширенного магазинного алфавита: алгоритм перенос-свертка

Шаг 1. Использование расширенного магазинного алфавита. Анализируя верхний символ магазина определяется наличие в нем основы и правила грамматики для свертки.

- | | |
|-------------------------|--------------------------|
| 1. $S \rightarrow (AS)$ | 3. $A \rightarrow (SaA)$ |
| 2. $S \rightarrow (b)$ | 4. $A \rightarrow (a)$ |

Каждый магазинный символ алгоритма содержит сведения о символе грамматики «кодируемой цепочки» (состояние алгоритма). Пример кодирования магазинных символов для рассмотренной ранее грамматики приведен в таблице 3.12.1. Например, магазинный символ « \rangle_3 », находящийся на вершине магазина, означает что:

1. В верхушке магазина находится терминальный символ грамматики « \rangle »;
2. В верхней части магазина находится кодируемая этим символом цепочка символов « (SaA) », то есть *основа* - правая часть правила, 3.

Таблица 3.12.1. Кодирование магазинных символов.

Символ грамматики	Магазинный символ	Кодируемая цепочка
S	S_1	(AS
	S_2	(S
	S_3	$\perp S$
A	A_1	(A
	A_2	(SaA
a	a_1	(Sa
	a_2	(a
b	b_1	(b
($(_1$	(
)	$)_1$	(AS)
	$)_2$	(b)
	$)_3$	(SaA)
	$)_4$	(a)
–	\perp	\perp

Магазинный алфавит построен таким образом, что для каждого магазинного символа (за исключением S_3 и \perp), кодируемая им цепочка является *префиксом правой части* некоторого правила грамматики. И наоборот, каждый непустой префикс правой части правила кодируется некоторым магазинным символом.

Например, правая часть правила грамматики:

$$(1) S \rightarrow (AS)$$

имеет четыре непустых префикса: « $($ », « $(A$ », « $(AS$ », « (AS) », которые кодируются четырьмя магазинными символами: « $(_1$ », « A_1 », « S_1 », « \rangle_1 » соответственно.

Шаг 2. Символ переносится в магазин только в том случае, если он кодирует цепочку, «совместимую» с цепочкой, которая будет находиться в магазине после переноса.

Цепочка, кодируемая данным магазинным символом, *совместима* с цепочкой в магазине, если она является суффиксом магазинной цепочки после переноса данного символа.

Например, если в магазине находится цепочка $\perp_1(S_2a_1$, то алгоритм может втолкнуть в магазин только символ A_2 , так как после этого в магазине будет находиться цепочка $\perp_1(S_2a_1A_2$, суффиксом которой является цепочка, кодируемая символом A_2 , а именно $(SaA$.

Управление алгоритмом осуществляется при помощи двух функций, приведенных в таблице 8.2, следующим образом:

- Используя значение верхнего символа магазина и входного символа, алгоритм определяет значение функции действия: *ПЕРЕНОС* или *СВЕРТКА*;
- При выполнении переноса определяется значение функции переходов, равное магазинному символу, который нужно втолкнуть в магазин;
- Значение функции действия, равное $СВЕРТКА(i)$, однозначно определяет этот шаг.

Таблица 8.2. Функции управления алгоритмом.

	функция действий f(u)					функции переходов g(X)					
	(a	b)	⊥	S	A	a	b	()
S ₁	П	П	П	П						(₁) ₁
S ₂	П	П	П	П				a ₁		(₁	
S ₃	П	П	П	П	ДОП					(₁	
A ₁	П	П	П	П		S ₁				(₁) ₃
A ₂	П	П	П	П						(₁	
a ₁	П	П	П	П			A ₂		S ₃	(₁	
a ₂	П	П	П	П						(₁) ₄
b ₁	П	П	П	П						(₁) ₂
(₁	П	П	П	П		S ₂	A ₁	a ₂	b ₁	(₁	
) ₁	C(1)	C(1)	C(1)	C(1)	C(1)						
) ₂	C(2)	C(2)	C(2)	C(2)	C(2)						
) ₃	C(3)	C(3)	C(3)	C(3)	C(3)						
) ₄	C(4)	C(4)	C(4)	C(4)	C(4)						
⊥	П	П	П	П		S ₃				(₁	

Пример 3.12.1

Рассмотрим работу алгоритма при разборе цепочки $((b)a(a))(a(b))$. Эта цепочка принадлежит языку, определяемому рассматриваемой грамматикой, так как ее правый вывод:

$$S \Rightarrow_{(1)} (AS) \Rightarrow_{(1)} (A(AS)) \Rightarrow_{(2)} (A(A(b))) \Rightarrow_{(4)} (A((a)(b))) \Rightarrow_{(3)} ((SaA)((a)(b))) \Rightarrow_{(4)} ((Sa(a))((a)(b))) \Rightarrow_{(2)} (((b)a(a))((a)(b)))$$

Шаг 1. Начальная конфигурация алгоритма $(\perp, (((b)a(a))((a)(b))), \varepsilon)$.

Функция действий $f(u)$ определяется верхним символом магазина и входным символом. Для параметров начальной конфигурации (дно магазина « \perp » и первый входной символ « $($ ») функция действия равна ПЕРЕНОС, $f(u)=\Pi$.

Определяем магазинный символ, который должен быть помещен в магазин. Так как значение функции переносов для тех же параметров « \perp » и « $($ » равно магазинному символу « $(_1$ », то алгоритм переходит в конфигурацию:

$$(\perp_1, ((b)a(a))((a)(b))), \varepsilon).$$

Шаг 2. Выполняя аналогичные действия алгоритм проходит следующие конфигурации:

$$\begin{aligned} (\perp, (((b)a(a))((a)(b))), \varepsilon) &\vdash_s (\perp_1, ((b)a(a))((a)(b))), \varepsilon) \\ &\vdash_s (\perp_1(\perp_1, (b)a(a))((a)(b))), \varepsilon) \\ &\vdash_s (\perp_1(\perp_1(\perp_1, b)a(a))((a)(b))), \varepsilon) \\ &\vdash_s (\perp_1(\perp_1(\perp_1 b_1,)a(a))((a)(b))), \varepsilon) \\ &\vdash_s (\perp_1(\perp_1(\perp_1 b_1)_2, a(a))((a)(b))), \varepsilon) \end{aligned}$$

Шаг 3. Алгоритм находится в конфигурации $(\perp_1(\perp_1(\perp_1 b_1)_2, a(a))((a)(b))), \varepsilon)$.

Значение функции действий для аргументов « $)_2$ » и символа « a » равно СВЕРТКА(2). Для выполнения свертки по второму правилу грамматики мы должны вычеркнуть из магазина 3 верхних символа и втолкнуть в него некоторый магазинный символ.

Для нахождения значения вталкиваемого символа нужно определить значение функции переходов для следующих аргументов: левой части правила, указанного в значении функции действий и входного символа.

В данном случае это левая часть второго правила грамматики S и входного символа « a », следовательно, получаем значение S_2 , которое помещаем в верхушку магазина.

Шаг 4. Продолжая работу, алгоритм пройдет следующие конфигурации:

$$\begin{aligned} (\perp_1(\perp_1(\perp_1 b_1)_2, a(a))((a)(b))), \varepsilon) &\vdash_{r(2)} (\perp_1(\perp_1 S_2, a(a))((a)(b))), \perp, 2) \\ &\vdash_s (\perp_1(\perp_1 S_2 a_1, (a))((a)(b))), \perp, 2) \\ &\vdash_s (\perp_1(\perp_1 S_2 a_1(\perp_1, a))((a)(b))), \perp, 2) \\ &\vdash_s (\perp_1(\perp_1 S_2 a_1(\perp_1 a_2,))((a)(b))), \perp, 2) \\ &\vdash_s (\perp_1(\perp_1 S_2 a_1(\perp_1 a_2)_4,))((a)(b))), \perp, 2) \\ &\vdash_{r(4)} (\perp_1(\perp_1 S_2 a_1 A_2,))((a)(b))), \perp, 24) \\ &\vdash_s (\perp_1(\perp_1 S_2 a_1 A_2)_3, ((a)(b))), \perp, 24) \\ &\vdash_{r(3)} (\perp_1 A_1, ((a)(b))), \perp, 243) \\ &\vdash_s (\perp_1 A_1(\perp_1, (a)(b))), \perp, 243) \\ &\vdash_s (\perp_1 A_1(\perp_1(\perp_1, a)(b))), \perp, 243) \\ &\vdash_s (\perp_1 A_1(\perp_1(\perp_1 a_2,)(b))), \perp, 243) \\ &\vdash_s (\perp_1 A_1(\perp_1(\perp_1 a_2)_4, (b))), \perp, 243) \\ &\vdash_{r(4)} (\perp_1 A_1(\perp_1 A_1, (b))), \perp, 2434) \\ &\vdash_s (\perp_1 A_1(\perp_1 A_1(\perp_1, b))), \perp, 2434) \\ &\vdash_s (\perp_1 A_1(\perp_1 A_1(\perp_1 b_1,))), \perp, 2434) \\ &\vdash_s (\perp_1 A_1(\perp_1 A_1(\perp_1 b_1)_2,))), \perp, 2434) \\ &\vdash_{r(2)} (\perp_1 A_1(\perp_1 A_1 S_1,))), \perp, 24342) \end{aligned}$$

$$\begin{aligned}
&\vdash_s (\perp({}_1A_1({}_1A_1S_1)_1), \perp, 24342) \\
&\vdash_{r(1)} (\perp({}_1A_1S_1), \perp, 243421) \\
&\vdash_s (\perp({}_1A_1S_1)_1, \perp, 243421) \\
&\vdash_{r(1)} (\perp S_3, \perp, 2434211) \\
&\vdash_s \text{ДОПУСК}.
\end{aligned}$$

Шаг 5. Алгоритм успешно завершил работу и выдал *правый разбор* входной цепочки $((((b)a(a))((a)(b))))$, равный 2434211.

Алгоритм 3.12. LR(k)-алгоритм разбора

Вход: Анализируемая цепочка $z = t_1t_2\dots t_j\dots t_n \in T^*$, где j - номер текущего символа входной цепочки, находящегося под читающей головкой. Управляющая таблица M (множество LR(k)-таблиц) для LR(k)-грамматики $G = (T, V, P, S)$

Выход: Если $z \in L(G)$, то *правый разбор* цепочки z , в противном случае – сигнал об ошибке.

Алгоритм.

1. $j := 0$.
2. $j := j+1$. Если $j > n$, то выдать сообщение об ошибке и перейти к шагу 5.
3. Определить цепочку u следующим образом:
 - если $k = 0$, то $u = t_j$;
 - если $k \geq 1$ и $j + k - 1 \leq n$, то $u = t_jt_{j+1}\dots t_{j+k-1}$ – первые k -символов цепочки $t_jt_{j+1}\dots t_n$;
 - если $k \geq 1$ и $j + k - 1 > n$, то $u = t_jt_{j+1}\dots t_n$ – остаток входной цепочки.
4. Применить функцию действия f из строки таблицы M , отмеченной верхним символом магазина T , к цепочке u :
 - $f(u) = \text{ПЕРЕНОС (П)}$. Определить функцию перехода $g(t_j)$ из строки таблицы M , отмеченной символом T из верхушки магазина. Если $g(t_j) = T'$ и $T'V_p\{\perp\}$, то записать T' в магазин и перейти к шагу 2. Если $g(t_j) = \text{ОШИБКА}$, то выдать сигнал об ошибке и перейти к шагу 5;
 - $f(u) = (\text{СВЕРТКА}, i) \text{ (С)}$ и $A \rightarrow \alpha$ – правило вывода с номером i грамматики G . Удалить из верхней части магазина $|\alpha|$ символов, в результате чего в верхушке магазина окажется символ $T'V_p\{\perp\}$, и выдать номер правила i на входную ленту. Определить символ $T'' = g(A)$ из строки таблицы M , отмеченной символом T' , записать его в магазин и перейти к шагу 3.
 - $f(u) = \text{ОШИБКА (О)}$. Выдать сообщение об ошибке и перейти к шагу 5.
 - $f(u) = \text{ДОПУСК (Д)}$. Объявить цепочку, записанную на входной ленте, *правым разбором* входной цепочки z .
5. Останов.

Грамматическое вхождение

Грамматическое вхождение – это символы полного словаря грамматики, снабженные двумя индексами. Первый индекс задает номер i правила грамматики, в правую часть которого входит данный символ, а второй индекс j – номер позиции символа в этой правой части.

Пример: грамматическое вхождение $A_{5,2} = ?$

(1): $S \rightarrow aAb$

(2): $S \rightarrow c$

(3): $A \rightarrow bS$

(4): $A \rightarrow bB$

(5): $B \rightarrow aA$

Существуют несколько способов построения детерминированного конечного автомата, распознающего активные префиксы [2, 3, 28, 37]. Рассмотрим вначале достаточно простой способ построения канонической системы LR(0)-ситуаций, приведенный в [2], для чего определим две функции: CLOSURE и GOTO.

Пример 8.4.

Так как LR(0) - ситуация - это просто правило грамматики с точкой в некоторой позиции правой части, то для правила $P \rightarrow (E)$ можно получить 4 ситуации:

$[P \rightarrow \bullet (E)]$

$[P \rightarrow (\bullet E)]$

$[P \rightarrow (E \bullet)]$

$[P \rightarrow (E) \bullet]$.

Замечание.

LR(0)-ситуация не содержит аванцепочку ϵ , поэтому при ее записи можно опускать квадратные скобки.

Из определения LR(k)-ситуации следует, что для каждого активного префикса существует непустое множество ситуаций. Основная идея построения простого LR(0)-анализатора состоит в том, чтобы вначале построить на базе КС-грамматики детерминированный конечный автомат для распознавания активных префиксов. Для этого ситуации группируются в множества, которые приводят к состояниям анализатора.

Пусть I – множество LR(0)-ситуаций КС-грамматики G . Тогда назовем замыканием множества I множество ситуаций CLOSURE(I), построенное по следующим правилам:

1. Включить в CLOSURE(I) все ситуации из I .

2. Если ситуация $A \rightarrow \alpha \bullet B\beta$ уже включена в $CLOSURE(I)$ и $B \rightarrow \gamma$ - правило грамматики, то добавить в множество $CLOSURE(I)$ ситуацию $B \rightarrow \bullet \gamma$ при условии, что там ее еще нет.
3. Повторять правило 2, до тех пор, пока в $CLOSURE(I)$ нельзя будет включить новую ситуацию.

Второе правило построения можно пояснить следующим образом:

- Наличие ситуации $A \rightarrow \alpha \bullet B\beta$ в множестве $CLOSURE(I)$ говорит о том, что в некоторый момент разбора мы можем встретить в входном потоке анализатора подстроку, выводимую из $B\beta$.
- Если в грамматике имеется правило $B \rightarrow \gamma$, то мы также можем встретить во входном потоке анализатора подстроку, выводимую из γ , следовательно, в $CLOSURE(I)$ нужно включить ситуацию $B \rightarrow \bullet \gamma$.

Рассмотрим расширенную грамматику G'_0 , имеющую следующие правила:

- | | |
|---------------------------|-------------------------|
| (0) $E' \rightarrow E$ | |
| (1) $E \rightarrow E + T$ | (4) $T \rightarrow P$ |
| (2) $E \rightarrow T$ | (5) $P \rightarrow i$ |
| (3) $T \rightarrow T * P$ | (6) $P \rightarrow (E)$ |

Пусть вначале множество ситуаций $CLOSURE(I)$ содержит одну ситуацию $E' \rightarrow \bullet E$, т.е. $CLOSURE(I) = \{E' \rightarrow \bullet E\}$

Ситуация $E' \rightarrow \bullet E$ содержит точку перед символом E , поэтому по второму правилу в $CLOSURE(I)$ необходимо включить E -правила с точкой слева:

$T \rightarrow \bullet T * P, T \rightarrow \bullet P, P \rightarrow \bullet i, P \rightarrow \bullet (E)$

Окончательно получим:

$CLOSURE(I) = \{E' \rightarrow \bullet E, E \rightarrow \bullet E + T, E \rightarrow \bullet T, T \rightarrow \bullet T * P, T \rightarrow \bullet P, P \rightarrow \bullet i, P \rightarrow \bullet (E)\}$.

Алгоритм вычисления множества ситуаций $CLOSURE(I)$ приведен в [2].

Неформально можно сказать, что если I -множество ситуаций, допустимых для некоторого активного префикса γ , то $GOTO(I, X)$ – это множество ситуаций, допустимых для активного префикса γX .

Аргументами функции $GOTO(I, X)$ являются множество ситуаций I и символ грамматики X .

Определение. Функция $GOTO(I, X)$ определяется как замыкание множества всех ситуаций $[A \rightarrow \alpha X \bullet \beta]$, таких что $[A \rightarrow \alpha \bullet X \beta] \in I$

Пусть для грамматики G'_0 множество $I = \{[E' \rightarrow \bullet E], [E \rightarrow \bullet E + T]\}$.

Для вычисления значения $GOTO(I, +)$ необходимо рассмотреть ситуации, в которых сразу за точкой идет символ $(+)$. Это ситуация $[E \rightarrow \bullet + T]$.

Перемещая точку за символ (+), построим множество ситуаций $\{[E \rightarrow E + \cdot T]\}$ и замыкание этого множества:

$$\{T \rightarrow \cdot T * P, T \rightarrow \cdot P, P \rightarrow \cdot i, P \rightarrow \cdot (E)\}$$

Тогда $GOTO(I, +) = \{[E \rightarrow E \cdot + T], [T \rightarrow \cdot T * P], [T \rightarrow \cdot P], [P \rightarrow \cdot i], [P \rightarrow \cdot (E)]\}$

В [2] и [3] приведены алгоритмы. Позволяющие построить каноническую систему множеств $LR(0)$ – ситуаций φ . Процесс построения канонической системы множеств $LR(0)$ – ситуаций можно описать с помощью следующих действий:

1. $\varphi = \emptyset$.
2. Включить в φ множество $A_0 = \text{CLOSURE}([S' \rightarrow \cdot S])$, которое в начале «не отмечено».
3. Если множество ситуаций A , входящее в систему φ , «не отмечено», то:
 - отметить множество A ;
 - вычислить для каждого символа $X \in (N \cup \Sigma)$ значение $A' = \text{GOTO}(A, X)$;
 - если множество $A' \neq \emptyset$ и еще не включено в φ , то включить его в систему множеств как φ «неотмеченное» множество.
4. Повторять шаг 3, пока все множества ситуаций системы φ не будут отмечены.

Пример 8.5. стр.296.

Определим каноническую систему множеств $LR(0)$ – ситуаций для пополненной грамматики G_0 с правилами:

- (0) $E' \rightarrow E$
- (1) $E \rightarrow E + T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow T * P$
- (4) $T \rightarrow P$
- (5) $P \rightarrow i$
- (6) $P \rightarrow (E)$

В начале построения система множеств $\varphi = \emptyset$.

Определив множество $A_0 = \text{CLOSURE}(\{[E' \rightarrow \cdot E]\}) = \{E' \rightarrow \cdot E, E \rightarrow \cdot E + T, E \rightarrow \cdot T, T \rightarrow \cdot T * P, T \rightarrow \cdot P, P \rightarrow \cdot i, P \rightarrow \cdot (E)\}$, включим его в систему φ в качестве «неотмеченного» множества:

A ₀	E' → • E
	E → • E + T
	E → • T
	T → • T * P
	T → • P
	P → • (E)
	P → • i

Теперь мы должны отметить множество A₀ и определить множества GOTO(A₀, X) для всех символов грамматики G₀:

$$A_1 = \text{GOTO}(A_0, E) = \text{GOTO}(\{E' \rightarrow \bullet E, E \rightarrow \bullet E + T\}, E) =$$

$$\text{CLOSURE}(\{E' \rightarrow E \bullet, E \rightarrow E \bullet + T\}) = \{E' \rightarrow E \bullet, E \rightarrow E \bullet + T\}.$$

Множество A₁ непустое и отсутствует в системе φ. Включим A₁ в φ как «неотмеченное» множество:

A ₀	E' → • E
	E → • E + T
	E → • T
	T → • T * P
	T → • P
	P → • (E)
	P → • i

A ₁	E → E •
	E → E • + T

Продолжая строить GOTO(A₀, X), получаем:

$$A_2 = \text{GOTO}(A_0, T) = \text{GOTO}(\{E \rightarrow \bullet T, T \rightarrow \bullet T * P\}, T) =$$

$$\text{CLOSURE}(\{E \rightarrow T \bullet, T \rightarrow T \bullet * P\}) = \{E \rightarrow T \bullet, T \rightarrow T \bullet * P\}.$$

Включаем A₂ в φ.

$$A_3 = \text{GOTO}(A_0, P) = \text{GOTO}(\{T \rightarrow \bullet P\}, P) = \{T \rightarrow P \bullet\}.$$

Включаем A₃ в φ.

$$A_4 = \text{GOTO}(A_0, () = \text{GOTO}(\{P \rightarrow \bullet (E)\}, () = \{P \rightarrow (\bullet E), E \rightarrow \bullet E + T, \\ E \rightarrow \bullet T, T \rightarrow \bullet T * P, T \rightarrow \bullet P, P \rightarrow \bullet i, P \rightarrow \bullet (E)\}.$$

Включаем A₅ в φ.

Остальные множества GOTO(A₀, X), где X ∈ {), +, *}, пусты, поэтому система φ принимает вид:

A ₀	E' → • E
	E → • E + T
	E → • T
	T → • T * P
	T → • P
	P → • (E)
	P → • i
A ₁	E' → E •
	E → E • + T

A ₂	$E \rightarrow T \bullet$ $T \rightarrow T \bullet * P$
A ₃	$T \rightarrow P \bullet$
A ₄	$P \rightarrow (\bullet E)$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet T * P$ $T \rightarrow \bullet P$ $P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$
A ₅	$P \rightarrow i \bullet$

Продолжая выполнять шаг 3 построения, «отмечаем» множество A₁ и строим множества A₃ = GOTO(A₁, X).

Кроме множества:

$$\begin{aligned}
 A_6 &= \text{GOTO}(A_1, +) = \text{GOTO}(\{E \rightarrow E \bullet + T\}, +) = \\
 &= \text{CLOSURE}(\{E \rightarrow E + \bullet T\}) = \\
 &= \{E \rightarrow E + \bullet T, T \rightarrow \bullet T * P, T \rightarrow \bullet P, P \rightarrow \bullet i, P \rightarrow \bullet (E)\},
 \end{aligned}$$

Остальные множества пусты. Включаем A₆ в φ и получаем:

A ₀	$E' \rightarrow \bullet E$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet T * P$ $T \rightarrow \bullet P$ $P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$
A ₁	$E' \rightarrow E \bullet$ $E \rightarrow E \bullet + T$
A ₂	$E \rightarrow T \bullet$ $T \rightarrow T \bullet * P$
A ₃	$T \rightarrow P \bullet$
A ₄	$P \rightarrow (\bullet E)$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet T * P$ $T \rightarrow \bullet P$

	$P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$
A_5	$P \rightarrow i \bullet$
A_6	$E \rightarrow E + \bullet T$ $T \rightarrow \bullet T * P$ $T \rightarrow \bullet P$ $P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$

Продолжая выполнять шаг 3, получаем:

$$A_7 = \text{GOTO}(A_2, *) = \text{GOTO}(\{E \rightarrow E \bullet * P\}, *) = \text{CLOSURE}(\{T \rightarrow T * \bullet P\}) = \{T \rightarrow T * \bullet P, P \rightarrow \bullet i, P \rightarrow \bullet (E)\},$$

Включаем A_7 в Φ .

$$A_8 = \text{GOTO}(A_4, E) = \text{GOTO}(\{P \rightarrow (\bullet E), E \rightarrow \bullet E + T\}, E) = \text{CLOSURE}(\{P \rightarrow (E \bullet), E \rightarrow E \bullet + T\}) = \{P \rightarrow (E \bullet), E \rightarrow E \bullet + T\}.$$

Включаем A_8 в Φ .

Вычисляем множество $\text{GOTO}(A_4, T)$.

$$A_8 = \text{GOTO}(A_4, T) = \text{GOTO}(\{E \rightarrow \bullet T, T \rightarrow \bullet T * P\}, T) = \text{CLOSURE}(\{E \rightarrow T \bullet, T \rightarrow T \bullet * P\}) = \{E' \rightarrow T \bullet, T \rightarrow T \bullet * P\}.$$

Такое множество (множество A_2) уже есть в Φ .

Множества $\text{GOTO}(A_4, P)$, $\text{GOTO}(A_4, ())$ и $\text{GOTO}(A_4, i)$ также уже помещены в систему Φ .

Множество $\text{GOTO}(A_5, X) = \emptyset$ для всех $X \in (N \cup \Sigma)$.

Продолжая аналогичным образом, включаем в систему Φ множества:

$$A_9 = \text{GOTO}(A_6, T) = \text{GOTO}(\{E \rightarrow E + \bullet T, T \rightarrow \bullet T * P\}, T) = \text{CLOSURE}(\{E \rightarrow E + T \bullet, T \rightarrow T \bullet * P\}) = \{E \rightarrow E + T \bullet, T \rightarrow T \bullet * P\},$$

$$A_{10} = \text{GOTO}(A_7, P) = \text{GOTO}(\{T \rightarrow T * \bullet P\}, P) = \text{CLOSURE}(\{T \rightarrow T * P \bullet\}) = \{T \rightarrow T * P \bullet\},$$

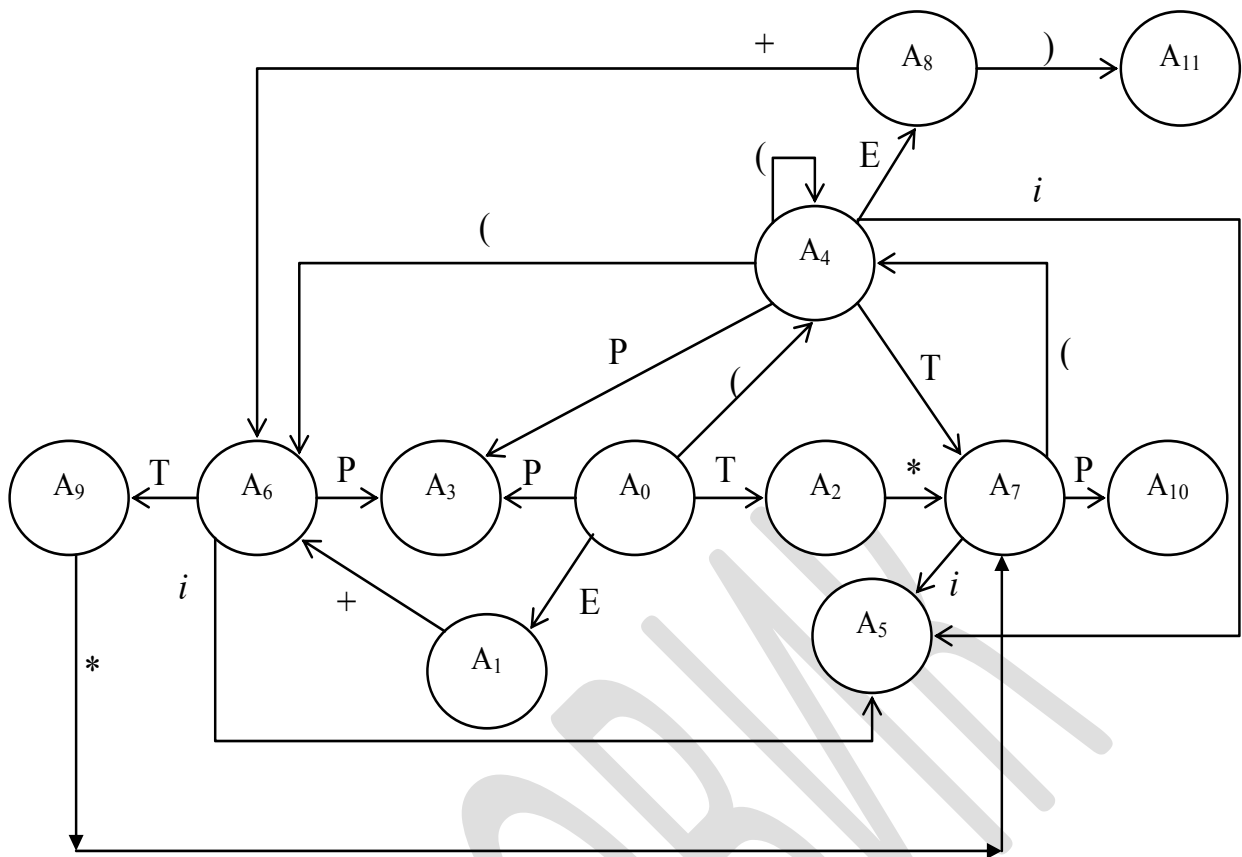
$$A_{11} = \text{GOTO}(A_8, ()) = \text{GOTO}(\{E \rightarrow (\bullet E)\}, ()) = \text{CLOSURE}(\{E \rightarrow (E) \bullet\}) = \{E \rightarrow (E) \bullet\}.$$

В результате получаем окончательную систему $LR(0)$ – множеств Φ :

A_0	$E' \rightarrow \bullet E$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet T * P$ $T \rightarrow \bullet P$ $P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$
A_1	$E' \rightarrow E \bullet$ $E \rightarrow E \bullet + T$

A ₂	$E \rightarrow T \bullet$ $T \rightarrow T \bullet * P$
A ₃	$T \rightarrow P \bullet$
A ₄	$P \rightarrow (\bullet E)$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet T * P$ $T \rightarrow \bullet P$ $P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$
A ₅	$P \rightarrow i \bullet$
A ₆	$E \rightarrow E + \bullet T$ $T \rightarrow \bullet T * P$ $T \rightarrow \bullet P$ $P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$
A ₇	$T \rightarrow T * \bullet P$ $P \rightarrow \bullet (E)$ $P \rightarrow \bullet i$
A ₈	$E \rightarrow E \bullet + T$ $P \rightarrow (E \bullet)$
A ₉	$E \rightarrow E + T \bullet$ $T \rightarrow T \bullet * P$
A ₁₀	$T \rightarrow T * P \bullet$
A ₁₁	$P \rightarrow (E) \bullet$

Диаграмма переходов ДКА для активных префиксов грамматики G_0 .



Используя каноническую систему LR(0) – множеств, можно представить функцию GOTO в виде диаграммы детерминированного конечного автомата.

+++++

Практический интерес представляют LR(k)-грамматики, для которых $k=0$ или $k=1$, в случаях, когда $k > 1$ требуется построение большого числа вспомогательных множеств, поэтому применение LR(k)-анализатора не оправдано.

Алгоритм построения управляющей таблицы M для LR(0)-грамматик основывается на рассмотрении пар грамматических вхождений, которые могут быть представлены соседними магазинными символами в процессе разбора допустимых цепочек.

Пример построения LR(k)-анализатора.

1. Построить управляющую таблицу M для заданной LR(k) грамматики $G = (T, V, P, S)$, где $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, $p_1) S \rightarrow aAb$, $p_2) S \rightarrow c$, $p_3) A \rightarrow bS$, $p_4) A \rightarrow Bb$, $p_5) B \rightarrow aA$, $p_6) B \rightarrow c$

2. Описать работу алгоритма.

1. Для построения управляющей таблицы М воспользуемся алгоритмом 1.4.12. В табл. 3.12.1. приведен результат выполнения алгоритма.

Управляющая таблица М для этой грамматики приведена в табл. 1.3.

Т	f(u) функция действия				g(X)					
	A	B	C	\perp	A	B	C	S	A	B
S ₀				Д						
a ₁	П	П	П		a ₅	b ₃	c ₆		A ₁	B ₄
A ₁	П	П	П			b ₁				
b ₁	C,1	C,1	C,1	C,1						
c ₂	C,2	C,2	C,2	C,2						
b ₃	П	П	П		a ₁		c ₂	S ₃		
S ₃	C,3	C,3	C,3	C,3						
B ₄	П	П	П			b ₄				
b ₄	C,4	C,4	C,4	C,4						
a ₅	П	П	П	П	a ₅	b ₃	c ₆		A ₅	B ₄
A ₅	C,5	C,5	C,5	C,5						
c ₆	C,6	C,6	C,6	C,6						
\perp	П	П	П		a ₁		c ₂	S ₀		

Рассмотрим, например, вторую строку таблицы М, отмеченную грамматическим вхождением a₁, которое представляет собой префикс правой части правила p₁. Пусть b – текущий входной символ. Согласно правилу p₁ за a₁ может следовать грамматическое вхождение нетерминала A₁ или символ, представляющий подцепочку, порождаемую вхождением A₁. Цепочка порождаемая вхождением A₁, может начинаться либо вхождением b₃ (правило p₃), либо вхождением B₄ (правило p₄), которое в свою очередь порождает цепочку, начинающуюся с a₁ или c₆. Следовательно, для входного символа b в магазин можно втолкнуть только грамматическое вхождение b₃, которое является кодированным представлением префикса правой части правила p₃.

Строка таблицы, отмеченная маркером для магазина (\perp), соответствует начальной конфигурации алгоритма. В первый момент времени в магазин можно записать только грамматическое вхождение a₁ или c₂, которые представляют собой префиксы цепочек, выводимых из начального вхождения S₀ (грамматического вхождения начального символа грамматики S, входящего в правую часть нулевого правила вывода пополненной грамматики). Функция действия f(u) не зависит от текущего входного символа, а определяется только верхним символом магазина, то есть для выбора нужного действия (переноса или свертки) алгоритм не должен заглядывать вперед на символы входной цепочки (k=0).

2. Работа алгоритма 3.11. описывается в терминах конфигураций, представляющих собой тройки вида (αT , ax, π), где αT – цепочка магазинных символов (T-верхний символ магазина), ax – необработанная часть входной цепочки, π – выход, построенный к настоящему моменту времени.

Рассмотрим последовательность тактов LR(k)-алгоритма при анализе входной цепочки $abcb$.

Шаг 1. Начальная конфигурация алгоритма – $(\perp, abcb\perp, \epsilon)$ (магазина находится маркер для магазина, а текущим входным символом является символ a).

Для строки управляющей таблицы, отмеченной символом \perp , $f(a)=\text{ПЕРЕНОС}$ и $g(a) = a_1$, поэтому:

- в магазин записывается символ a_1 (грамматическое вхождение символа a в правую часть первого правила);
- входная головка сдвигается на один символ справа.

Алгоритм приходит в конфигурацию $(\perp a_1, bcb\perp, \epsilon)$.

Шаг 2. Для строки таблицы, отмеченной символом a_1 , $f(b)=\text{ПЕРЕНОС}$ и $g(b) = b_3$, следовательно, алгоритм перейдет в конфигурацию $(\perp a_1 b_3, cb\perp, \epsilon)$. Аналогично для магазинного символа b_3 и текущего символа входной цепочки магазин перейдет в конфигурацию $(\perp a_1 b_3 c_2, b\perp, \epsilon)$

Шаг 3. Рассмотрим строку таблицы M , помеченную грамматическим вхождением c_2 . В этом случае $f(b)=(c, 2)$, значит, необходимо выполнить свертку с использованием правила $p_2) S \rightarrow c$.

Правая часть этого правила содержит только один символ, поэтому:

- удаляем из магазина символ c_2 ;
- определяем значение функции переходов для символа S из левой части правила p_2 в строке таблицы M , отмеченной символом b_3 , который стал верхним символом магазина. Значение функции переходов $g(S) = S_3$.

Алгоритм переходит в конфигурацию $(\perp a_1 b_3 S_3, b\perp, 2)$ и в выходную цепочку записывается число 2 (номер использованного варианта).

Продолжая, получим следующую последовательность тактов:

$(\perp a_1 b_3 S_3, b\perp, 2) \vdash (\perp a_1 A_1, b\perp, 23) \vdash (\perp a_1 A_1 b_1, \perp, 23) \vdash (\perp a_1 S_0, \perp, 231)$

Заметим, что конфигурация $(\perp a_1 S_0, \perp, 231)$ является заключительной, а цепочка 231 – правым разбором цепочки $abcb$.

Пусть анализируется цепочка $aabc$, содержащая синтаксическую ошибку, тогда последовательность тактов следующая:

$(\perp, aabc\perp, \epsilon) \vdash (\perp, aabc\perp, \epsilon) \vdash (\perp a_1 abc\perp, \epsilon) \vdash (\perp a_1 a_5, bc\perp, \epsilon) \vdash (\perp a_1 a_5 b_3, c\perp, \epsilon) \vdash (\perp a_1 a_5 b_3 c_2, \perp, \epsilon) \vdash (\perp a_1 a_5 b_3 S_3, \perp, 2) \vdash (\perp a_1 a_5 A_5, \perp, 23) \vdash (\perp a_1 B_4, \perp, 235) \vdash \text{ОШИБКА}$.

$f(\epsilon) = \text{ОШИБКА}$. Алгоритм должен выдать сообщение об ошибке и закончит работу.

3.8. Грамматики предшествования

Существует подкласс LR(k)-грамматик, который называется грамматиками предшествования для них легко строится алгоритм типа “перенос-свертка”. При восходящих методах синтаксического анализа в текущей сентенциальной форме выполняется поиск основы α , которая в соответствии с правилом грамматики $A \rightarrow \alpha$ сворачивается к нетерминальному символу A .

Основная проблема восходящего синтаксического анализа заключается в поиске основы и нетерминального символа, к которому ее нужно приводить (сворачивать).

Алгоритм 1.4.13. Пусть $G = (T, V, P, S)$ - КС-грамматика, правила которой пронумерованы числами от 1 до p .

Алгоритм типа “перенос-свертка” для грамматики G назовем парой функций $\lambda = (f, g)$, где f – называется функцией переноса, а g – функцией свертки. Функции f и g определяются следующим образом:

1. f отображает $(V \cup T \cup \{\perp\})^* \times (T \cup \{\perp\})^*$ в множество {ПЕРЕНОС, СВЕРТКА, ОШИБКА, ДОПУСК}.

2. g отображает $(V \cup T \cup \{\perp\})^* \times (T \cup \{\perp\})^*$ в множество $\{1, \dots, \text{ОШИБКА}\}$. Если $g(\alpha, w) = i$, то правая часть i -го правила является суффиксом цепочки α .

Пусть имеется сентенциальная форма $\alpha X_1 X_2 \beta$, где $X_1, X_2 \in V \cup T$. На некотором этапе разбора либо символ X_1 , либо символ X_2 , либо они вместе должны войти в основу. Рассмотрим три следующих случая:

X_1 - часть основы, символ X_2 не входит в основу. Символ X_1 будет свернут раньше X_2 . В этом случае говорят, что символ X_1 предшествует X_2 , и записывают как это как $X_1 \cdot > X_2$. X_1 – последний символ основы (хвост основы), т.е. последний символ правой части некоторого правила, а символ X_2 – терминальный символ.

X_1 и X_2 входят в основу, т.е. сворачиваются одновременно, записывают как $X_1 \cdot = X_2$. Символы X_1 и X_2 входят в правую часть некоторого правила грамматики.

X_2 - часть основы, а символ X_1 не входит в основу. Символ X_2 – должен быть свернут раньше X_1 , записывают как $X_1 < \cdot X_2$. X_1 – это первый символ основы (головы основы), т.е. первый символ правой части некоторого правила.

Отношения $(\cdot >)$, $(\cdot =)$ и $(< \cdot)$ называются отношениями предшествования.

Работу алгоритма типа “перенос-свертка” удобно описывать в терминах конфигураций вида $(\perp X_1 \dots X_m, a_1 \dots a_n, p_1 \dots p_r)$, где :

$\perp X_1 \dots X_m$ – содержимое магазина, X_m – верхний символ магазина и $X_i \in V \cup T$, символ (\perp) для магазина;

$a_1 \dots a_n$ – оставшаяся непрочитанная часть входной цепочки, a_1 – текущий входной символ;

$p_1 \dots p_r$ – разбор, полученный к данному моменту времени.

Один шаг алгоритма λ можно описать с помощью двух отношений: (\vdash^S) (перенос) и (\vdash^I) (свертка), определенных на конфигурациях следующим образом:

1. Если $f(\alpha, aw) = \text{ПЕРЕНОС (П)}$, то входной символ переносится в верхушку магазина и читающая головка сдвигается на один символ вправо. В терминах конфигураций этот процесс описывается так:

$(\alpha, aw, \pi) \vdash^S (\alpha a, w, \pi)$ для $\alpha \in (V \cup T \cup \{\perp\})^*$, $w \in (T \cup \{\epsilon\})^*$ и $\pi \in \{1, \dots, p\}^*$.

2. Если $f(\alpha\beta, w) = \text{СВЕРТКА (C)}$, $g(\alpha\beta, w) = i$ и $A \rightarrow \beta$ – правило грамматики с номером i , то цепочка β заменяется правой частью правила с номером i , а его номер помещается на выходную ленту, т.е. $(\alpha, aw, \pi) \vdash^r (\alpha A, w, \pi i)$.

3. Если $f(\alpha, w) = \text{ДОПУСК}$, то $(\alpha, w, \pi) \vdash^s \text{ДОПУСК (D)}$.

В остальных случаях $(\alpha, w, \pi) \vdash^s \text{ОШИБКА}$ (пустое значение в таблице).

Отношение (\vdash) определяется как объединение отношений (\vdash^s) и (\vdash^r) , а транзитивное замыкание отношений (\vdash^+) и (\vdash^*) определяется как обычно.

Для $w \in T^*$ будем записывать $\lambda(w) = \pi$, если $(\perp, w, \varepsilon) \vdash^* (\perp S, \varepsilon, \pi) \vdash^s \text{ДОПУСК}$, и $\lambda(w) = \text{ОШИБКА}$, в противном случае.

Для практического применения алгоритм мало пригоден, так как для определения функции переноса он требует анализа всей цепочки в магазине и всей необработанной части входной цепочки. Аналогичный недостаток имеет алгоритм и при определении функции свертки. В литературе описано несколько вариантов грамматик предшествования.

Для всех классов грамматик предшествования общим является способ поиска правого конца основы (*хвоста* основы). При разборе алгоритмом “перенос-свертка”, всякий раз, когда между верхним символом магазина и первым из необработанных входных символов выполняется отношение \triangleright , принимается решение о свертке, в противном случае делается перенос.

Таким образом, с помощью отношения \triangleright локализуется правый конец основы правывыводимой цепочки. Определение левого конца основы (*головой* основы) и нужной свертки осуществляется в зависимости от используемого типа отношения предшествования.

3.9. Пример алгоритм “перенос-свертка”

1. Применить алгоритм типа “перенос-свертка” для заданной грамматики $G = (T, V, P, S)$, где $P = \{p_1, p_2, p_3, p_4\}$, $p_1) S \rightarrow bAb$, $p_2) A \rightarrow cB$, $p_3) A \rightarrow a$, $p_4) B \rightarrow Aad$

2. Описать работу алгоритма.

Функция переноса

f	Ax	bх	сх	dx	ε
αA	П	П			
αB	С	С			
Aa	С	С		П	
Ab	П		П		С
Ac	П		П		
Ad	С	С			
\perp		П			
$\perp S$					Д

Функция свертки

g	ax	bх	х	ε
$\perp bAb$				1
αAad		4		
αcB	2	2		

Аа			3	
----	--	--	---	--

2. Разберем с помощью построенного алгоритма λ входную цепочку bcaadb.

Начальная конфигурация алгоритма $(\perp, bcaadb, \varepsilon)$.

Шаг 1. Выполнение алгоритма определяется значением $f(\perp, bcaadb)$, которое, как видно из определения функции f , имеет значение ПЕРЕНОС. Алгоритм переходит в конфигурацию $(\perp b, caadb, \varepsilon)$, выполняя шаг:

$(\perp, bcaadb, \varepsilon) \vdash^S (\perp b, caadb, \varepsilon)$ и следующие шаги:

$(\perp b, caadb, \varepsilon) \vdash^S (\perp bc, aadb, \varepsilon) \vdash^S (\perp bca, adb, \varepsilon)$.

Шаг 2. Очередной шаг выполнения алгоритма определяется значением $f(\perp bca, adb, \varepsilon)$, которое имеет значение СВЕРТКА. Правило, используемое при свертке, определяется значением функции $g(\perp bca, adb) = 3$ и переходит в следующую конфигурацию: $(\perp bca, adb, \varepsilon) \vdash^r (\perp bcA, adb, 3)$, следующие шаги: $(\perp bcA, adb, 3) \vdash^S (\perp bcAa, db, 3) \vdash^S (\perp bcAad, b, 3) \vdash^r (\perp bcB, b, 34)$

$\vdash^r (\perp bA, b, 342) \vdash^S (\perp bAb, \varepsilon, 342) \vdash^r (\perp S, \varepsilon, 3421) \vdash^S$ ДОПУСК.

Таким образом, $\lambda(bcaadb) = 3421$, правый вывод цепочки bcaadb должен быть равен 1243. Для проверки построим цепочку по заданному правому выводу:

$S \Rightarrow (p_1) bAb \Rightarrow (p_2) bcBb \Rightarrow (p_4) bcAdb \Rightarrow (p_3) bcaadb$

Глава 4. Грамматики общего вида и машины Тьюринга

4.1. Грамматики общего вида и машина Тьюринга

Определение 15. Грамматика $G = (T, V, P, S)$ называется *грамматикой типа 0*, если на правила вывода не накладывается никаких ограничений (кроме тех, которые указаны в определении грамматики).

Граматики типа 0 - это самые общие грамматики.

Языки высокого уровня не являются рекурсивными. Проблема выявления принадлежности выражения w языку, порожденному грамматикой типа 0, где $w \in T^*$, в общем случае неразрешима.

Определение 16. Машина Тьюринга (служит для проверки разрешимости алгоритма), в иерархии Хомского занимает самый верхний уровень:

$$M = (Q, \Sigma, R, L, B, \delta, q_0)$$

Q - множество состояний

Σ - входной словарь

R, L - правые и левые символы, не входящие в $Q \cup \Sigma$

B - пробел ($B \in \Sigma$)

δ - множество правил переписывания

q_0 - исходное состояние

$\delta = q_i a_i \rightarrow q_k a_l$, a_i - входные символы $q_i a_i \rightarrow q_k R$,

$\delta = Q \times \Sigma \rightarrow Q \times (\Sigma \cup \{0, L\})$ $q_i a_j \rightarrow q_k L$

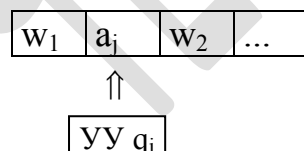
Машина Тьюринга - детерминированный преобразователь.

Конфигурация машины - множество элементов следующего вида:

$\Sigma^* \times Q \times \Sigma^+$, то есть последовательности $w_1 q_i a_j w_2$, где w_1 и $w_2 \in \Sigma^*$, $a_j \in \Sigma$, $q_i \in Q$.

Правила из δ задают последовательные переходы от одной конфигурации (до тех пор, пока не будет достигнута заключительная конфигурация).

Конфигурация $w_1 q_i a_j w_2$ называется заключительной, если к ней неприменимо ни одно из правил δ (т.е. правил с заголовком $q_i a_j$).



Работа машины сводится к следующему, управляющее устройство (УУ):

1. Читает содержимое ячейки, обозреваемой головкой в текущий момент времени.
2. Печатает в обозреваемой ячейке соответствующий символ из входного словаря.
3. Перемещается в соседнюю ячейку, находящуюся слева или справа от обозреваемой ячейки.

Находясь в q_i читает символ a_i и переходит в состояние q_k и либо печатает в этой ячейке символ a_i , либо перемещает головку влево или вправо.

Если к каждой фразе применить правило, то получится новая конфигурация. Воспринимаемый машиной Тьюринга язык $L = \{w \in \Sigma^* \mid q_0 \text{ и } w \Rightarrow^* C_f, \text{ где } C_f - \text{заключительная конфигурация.}\}$

Если w -фраза языка, то машина за конечный период времени находит заключительную конфигурацию C_f . Если же $w \notin L$, то машина не останавливается, что приводит к неразрешимости проблемы распознавания принадлежности данного выражения языку.

4.2. Контекстно-зависимые грамматики и ленточные автоматы

Определение 17. Грамматика $G = (T, V, P, S)$ называется *неукорачивающей грамматикой*, если каждое правило из P имеет вид $\alpha \rightarrow \beta$, где $\alpha \in (T \cup V)^+$, $\beta \in (T \cup V)^+$ и $|\alpha| \leq |\beta|$.

Определение 18. Грамматика $G = (T, V, P, S)$ называется *контекстно-зависимой (КЗ)*, если каждое правило из P имеет вид $\alpha \rightarrow \beta$, где $\alpha = \xi_1 A \xi_2$; $\beta = \xi_1 \gamma \xi_2$; $A \in V$; $\gamma \in (T \cup V)^+$; $\xi_1, \xi_2 \in (T \cup V)^*$.

Грамматику типа 1 можно определить как неукорачивающую либо как контекстно-зависимую.

Выбор определения не влияет на множество языков, порождаемых грамматиками этого класса, поскольку доказано, что множество языков, порождаемых неукорачивающими грамматиками, совпадает с множеством языков, порождаемых КЗ-грамматиками.

4.3. Соотношения между грамматиками и языками

Соотношения между типами грамматик:

- 1) любая регулярная грамматика является КС-грамматикой;
- 2) любая регулярная грамматика является укорачивающей КС-грамматикой (УКС). Отметим, что УКС-грамматика, содержащая правила вида $A \rightarrow \varepsilon$, не является КЗ-грамматикой и не является неукорачивающей грамматикой;
- 3) любая (приведенная) КС-грамматика является КЗ-грамматикой;
- 4) любая (приведенная) КС-грамматика является неукорачивающей грамматикой;
- 5) любая КЗ-грамматика является грамматикой типа 0.
- 6) любая неукорачивающая грамматика является грамматикой типа 0.

Определение 19. Язык $L(G)$ является *языком типа k* , если его можно описать грамматикой типа k .

Соотношения между типами языков:

- 1) каждый регулярный язык является КС-языком, но существуют КС-языки, которые не являются регулярными (например, $L = \{a^n b^n \mid n > 0\}$);
- 2) каждый КС-язык является КЗ-языком, но существуют КЗ-языки, которые не являются КС-языками (например, $L = \{a^n b^n c^n \mid n > 0\}$);

3) каждый КЗ-язык является языком типа 0. УКС-язык, содержащий пустую цепочку, не является КЗ-языком. Если язык задан грамматикой типа k , то это не значит, что не существует грамматики типа k' ($k' > k$), описывающей тот же язык. Поэтому, когда говорят о языке типа k , обычно имеют в виду максимально возможный номер k .

Например, КЗ-грамматика $G_1 = (\{0,1\}, \{A,S\}, P_1, S)$ и КС-грамматика $G_2 = (\{0,1\}, \{S\}, P_2, S)$, где $P_1 = \{S \rightarrow 0A1, 0A \rightarrow 00A1, A \rightarrow \varepsilon\}$ и $P_2 = \{S \rightarrow 0S1 \mid 01\}$

описывают один и тот же язык $L = L(G_1) = L(G_2) = \{0^n 1^n \mid n > 0\}$. Язык L называют КС-языком, т.к. существует КС-грамматика, его описывающая. Но он не является регулярным языком, т.к. не существует регулярной грамматики, описывающей этот язык.

Глава 5. Динамические модели и грамматические системы

В главе рассматриваются динамические модели: клеточные автоматы, динамические автоматы и L-системы (Линденмайера) [4].

5.1. Клеточные автоматы

Джон фон Нейман определил клеточные автоматы (КА) в середине 1940-х [“Теория самовоспроизводящихся автоматов” в 1971 году]. Его определение во многом пересекается с К. Цузе и С. Вольфрамом [“A new kind of science”, (2002)].

Клеточным автоматом называется дискретная динамическая система, определяемая некоторой периодической многомерной решеткой, из неограниченного итерированного, или мозаичного, массива конечных автоматов (клеток), каждый из которых взаимодействует со своими соседями. Состояние i -й клетки в момент времени k характеризуется некоторой переменной, которая может быть целым, действительным или комплексным числом, либо представлять собой набор из нескольких чисел или объектом, в зависимости от решаемой задачи.

Для КА задается правило перехода состояния клетки из текущего в следующее. Правило перехода определяется локально: состоянием клетки и/или состояниями соседних клеток. Правила не меняются со временем.

Состоянием решётки называется совокупность состояний всех клеток решётки. Каждое изменение состояния решётки называется итерацией, а каждая итерация соответствует некоторому дискретному интервалу времени. Состояния клеток изменяются синхронным образом через дискретные интервалы времени в соответствии с правилами.

Глобальное поведение КА полностью определяется в терминах локальных зависимостей между клетками, что позволяет моделировать создание сложного глобального поведения из локальных взаимодействий в пространстве достижимых состояний. Проектирование правил перехода, задающих требуемое глобальное поведение, является сложной задачей.

КА характеризуется параллельной архитектурой получившей название “не-фон-неймановской”, так как созданная им ранее последовательная архитектура уже была названа “фон-неймановской”. Клеточные автоматы моделируют параллельные вычисления подобно тому, как машины Тьюринга моделируют последовательные вычисления.

Сложность КА возрастает с количеством клеток и/или взаимодействий в правилах переходов, что требует увеличения количества памяти и вычислений.

Самый простой линейный КА имеет одномерную решетку, заполненную клетками, которые могут принимать значение ноль или единица. Такой КА также называют булевым КА.

Определение 1. $KA = \langle \text{Grid}, N, Q, \delta, \text{Grid}_0, \rangle$ – одномерный или линейный КА, где

$\text{Grid} = [c_1, c_2, \dots, c_N]$ – решетка, состоящая из N позиций. Каждая позиция i содержит клетку c_i , $i = 1, 2, \dots, N$.

Q – конечное множество состояний клетки c_i , при $Q = \{0,1\}$ клетка c_i может находиться в состоянии нуль или единица.

$neib_i$ – конечное множество локальных клеток, влияющих на значение данной, за исключением её самой c_i , называется окрестностью (или соседями) клетки, которую задают вводя метрику, поэтому о решётке говорят, как о дискретном метрическом пространстве. Соседями $neib_i$ для клетки c_i является множество локальных позиций решетки Grid с дистанцией от c_i цепочки не больше заданного радиуса r .

Для одномерной решетки каждая клетка c_i имеет ровно два соседа, причем c_N сосед c_1 , то есть первая позиция решетки связывается с последней, образуя тор.

Для одномерного КА количество соседей всегда равно $2r+1 = |neib_i|$. Например, при $r = 3$ и $N = 129$, соседями для c_2 $neib_2 = \{c_{128}, c_{129}, c_1, c_2, c_3, c_4, c_5\}$, $|neib_i| = 7$.

$\delta: c_i^k \xrightarrow{neib_i} c_i^{k+1}$ – локальная функция перехода состояния клетки c_i^k на шаге работы k , $k = 0, 1, \dots$ в новое состояние c_i^{k+1} на шаге $k+1$ в соответствии с состояниями соседей $neib_i$. При $Q = \{0,1\}$ количество возможных состояний $neib_i = 2^{2r+1}$.

Состояния клеток КА могут меняться все одновременно (синхронный КА) или неодновременно (асинхронный КА). Состояния клетки c_i на такте m выполнения КА записывается как c_i^m , тогда $Grid_k = [c_1^m, c_2^m, \dots, c_N^m]$.

В синхронном КА правило перехода δ имеет вид $\delta: [c_1^k, c_2^k, \dots, c_N^k] \xrightarrow{neib} [c_1^{k+1}, c_2^{k+1}, \dots, c_N^{k+1}]$, то есть $\delta(Grid_k)$.

$Grid_0$ – начальная конфигурация автомата, $Grid_0 = [c_1^0, c_2^0, \dots, c_N^0]$.

Определим бинарное отношение \vdash на конфигурациях, соответствующее одному такту k работы КА $\delta(Grid_{k-1}) \vdash^m \delta(Grid_k)$.

Пусть $\{Grid\}$ – множество конфигураций КА.

1. $Grid \vdash^0 Grid'$ означает, что $Grid = Grid'$
2. $Grid_0 \vdash^k Grid_k$, если существует последовательность конфигураций $Grid_1, Grid_2, \dots, Grid_{k-1}$, для $k \geq 1$, в которых $Grid_m \vdash Grid_{m+1}$ для $0 \leq m < k$, то :
 $Grid_0 \vdash \dots \vdash Grid_m = [c_1^m, c_2^m, \dots, c_N^m] \vdash^{m+1} Grid_{m+1} = [\delta(neib_1^m), \dots, \delta(neib_N^m)] \vdash \dots \vdash^k Grid_k$.

Пример. Пусть задан КА $\langle Grid, N = 29, Q = \{0,1\}, c_{n+1} = \delta(c_{n-1}^n \mid c_n^n \mid c_{n+1}^n) \rangle$, $Grid_0 = [c_1 = \{0\}, \dots, c_{14} = \{1\}, c_{15} = \{0\}, \dots, c_{29} = \{0\}]$ >

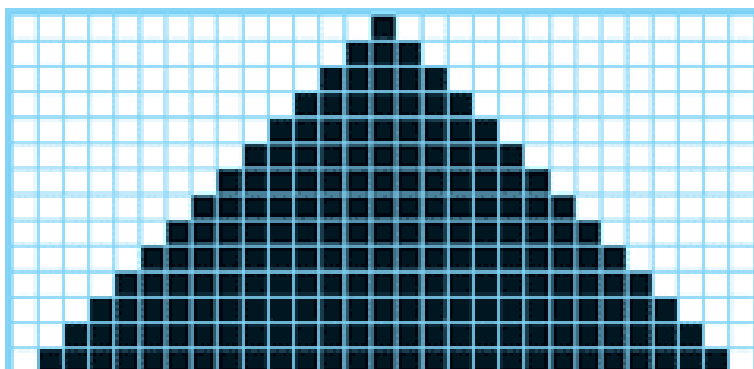


Рис. 5.1. Конфигурации булевого КА

Множество конфигураций булевого КА порождает побитовое (из нулей и единиц) изображение фрактального множества (треугольника Серпинского), которое иллюстрирует фрагмент на рис. 5.1.

Классификация клеточных автоматов

Клеточные автоматы классифицируют:

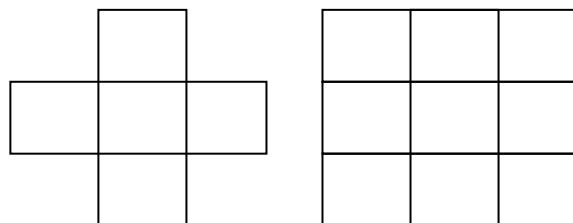
1. по размерности: линейные или одномерные КА, двумерные КА и многомерные КА;
2. по отношениям между решеткой и клеткой: подвижные и неподвижные, однородные и неоднородные. Если позиция i клетки в решетке во время выполнения КА изменяется, то такой КА называется подвижным, если нет, то неподвижным КА.
3. по свойствам правил переходов: с памятью (новое состояние клетки зависит от текущего состояния клетки) и без памяти, детерминированные и вероятностные ВКА, с периодическими паттернами, с хаотическими аperiodическими паттернами, со сложными временными паттернами [Wolfram A986]. Если в правиле перехода δ состояние клетки определяется на основе некоторых вероятностей, то КА называют вероятностным (ВКА), иначе детерминированным. Правило перехода δ задает вероятностные изменения состояния клетки: по сгенерированному случайному числу p , $0 < p < 1$, определяется возможное состояние клетки. Состояния ее ближайших соседей в k -й момент времени детерминированно. Это позволяет моделировать флуктуации.
4. по множеству состояний, которые может принимать клетка: булевы и многозначные. Клетки многозначных («цветных») автоматов могут находиться более чем в двух состояниях. Каждой клетке соответствует целочисленная переменная, кодирующая ее состояние. Например, RGB-кодирование ставит в соответствие цвету число из трех последовательных байтов (дескриптор цвета), показывающих насыщенность красного (R), зеленого (G) и синего цветов (B).

Многомерные клеточные автоматы

Двумерный (плоскостной) КА – это решетка с позициями $s(i,j)$, реализуемая двумерным массивом. Границы решетки соединены таким

образом, что первая клетка в строке решетки $c(1,j)$, соединена с последней клеткой $c(w,j)$, а первая клетка в колонке $c(i,1)$. Такая топология известна как торус или тороидная форма.

В двумерных КА обычно используют два типа соседства: по фон Нейману рис.2.а. и по Муру рис.2.б.



а. по фон Нейману б. по Муру

Рис. 2. Два типа соседства для двухмерного КА

Размер области соседей может быть расширен, для увеличения радиуса и размерности; в терминологии дистанций каждый среди соседей имеет путь к центральной клетке, который меньше или равен радиусу r .

В КА с d размерностью c_1, c_2, \dots, c_d клетки $a(i_1, \dots, i_d)$ и $b(j_1, \dots, j_d)$ являются соседними:

по фон Нейману, если $\sum_{k=1}^d \min(|i_k - j_k|, c_k - |i_k - j_k|) = 1$

по Муру, если $\min(|i_k - j_k|, c_k - |i_k - j_k|) \leq 1$, где $1 \leq k \leq d$

Для одномерного КА соседи по фон Нейману соответствуют соседям по Муру.

Правила переходов определяются аналогично одномерному КА: каждый бит в последовательности соответствует одной входящей клетке с соседями. Сопряженные клетки пронумерованы от 1 до N рекурсивно по всей размерности d . Это означает, что центральная клетка будет всегда иметь номер $(N+1)/2$.

Номера сопряженных клеток растут очень быстро, если увеличиваются r и d . Таб. 1. Иллюстрирует, что количество соседей по Муру растет быстрее количества соседей по фон Нейману.

Таб. 1. Количество клеток соседей в многомерном КА.

$S^N(d,r)$ определен для d - мерного соседства по фон Нейману, с

$S^M(d,r)$ определен для d - мерного соседства по фон Муру, где r – радиус.

	r- радиус						
	0	1	2	3	4	5	6
$S^N(1,r)$	1	3	5	7	9	11	13
$S^N(2,r)$	1	5	13	25	41	61	85
$S^N(3,r)$	1	7	25	63	129	231	377
$S^N(4,r)$	1	9	41	129	321	681	1289
$S^N(5,r)$	1	11	61	231	681	1683	3653

$S^N(6,r)$	1	13	85	377	1289	3653	8989
$S^M(1,r)$	1	3	5	7	9	11	13
$S^M(2,r)$	1	9	25	49	81	121	169
$S^M(3,r)$	1	27	125	343	729	1331	2197
$S^M(4,r)$	1	81	625	2401	6561	14641	28561
$S^M(5,r)$	1	243	3125	16807	59049	161051	371293
$S^M(6,r)$	1	729	15625	117649	531441	1771561	4826809

В Таб. 1. каждая клетка имеет восемь соседей. Для устранения краевых эффектов решетка «заворачивается» в тор. Иногда удобнее рассматривать не всех «соседей» клетки, а только «главных» (лишь те, которые имеют с ней общую сторону), в таком случае мы получаем окрестность из четырех клеток.

Правило переходов δ для двумерного КА с окрестностью из четырех клеток:

$$c'_{ij} = \delta(c_{ij}, c_{i-1j}, c_{ij+1}, c_{i+1j}, c_{ij-1})$$

Для окрестности из восьми клеток:

$$c'_{ij} = \delta(c_{ij}, c_{i-1j}, c_{i-1j+1}, c_{ij+1}, c_{i+1j+1}, c_{i+1j}, c_{i+1j-1}, c_{ij-1}, c_{i-1j-1})$$

В КА с окрестностью из 8 клеток самовоспроизведение имеет место после любого числа шагов, являющегося степенью двух, начиная со значения, определяемого соотношением:

$$k = 2^{\lceil \log_2(\min(a;b)) \rceil},$$

где a и b – соответственно ширина и высота, если $a=3$, $b=5$, то $k = 8$ и самовоспроизведение будет наблюдаться через 8, 16, 32... и т.д. шагов, в вследствие суммы по модулю два.

2d-КА моделирует игру Д. Х. Конвея «Жизнь». Клетки могут находиться в двух состояниях. Правило переходов клеток реализует следующие условия:

- если данная клетка находится в состоянии 0, то она перейдет в состояние 1 при условии, что у нее имеются три соседа с 1;
- если данная клетка с 1, то она останется с 1 только при условии, что у нее есть два или три соседа с 1, в противном же случае она перейдет в состояние 0.

Трехмерный КА определяется трехмерной решеткой, которая моделирует пространственные процессы.

Садом Эдема называется конфигурация клеток КА, не имеющая предыдущей конфигурации. Для игры "жизнь" можно доказать существование сада Эдема, но построить конфигурацию гораздо сложнее.

Метод реализации

1. Определяются два массива для хранения состояний клеток. Первый содержит текущее состояние каждой клетки. Второй – новое (для каждой клетки).

2. Определяется функция переходов клетки решетки. Она задается в виде булевой формулы и вычисляет новое состояние для конкретной клетки, используя в качестве входных параметров текущие значения состояний соседних клеток, возможно, включая текущее значение состояния этой клетки;

3. На нулевом шаге производится заполнение первого массива начальными данными, то есть заполнение решетки.

4. Для вычисления новых состояний вводится цикл. На очередной итерации этого цикла для каждой клетки с помощью правила переходов вычисляется её новое состояние, в качестве переменных используются соответствующие элементы первого массива (старые значения), а ее новое состояние помещается во второй массив.

5. После завершения итерации (заполнения второго массива) значения из всех элементов второго массива переносятся в первый массив.

6. Осуществляется визуализация содержимого решетки: для одномерной (линейной) решетки после каждой итерации выводится соответствующая ей строка; для двумерной (плоскостной) решетки после каждой итерации выводится матрица значений клеток решетки.

Применение клеточных автоматов

В конце 1980-х посредством клеточных автоматов исследовались модели формальных принципов организации биологической жизни и эволюции жизнеподобных структур – “организмов” (искусственных кибернетических сущностей). Результаты исследований сформировали направление “искусственная жизнь”, Д.Холланд, С.Кауфман и К.Лангтон [37,38]. К.Лангтон сформулировал основной принцип: “логическая форма” организма может быть отделена от материальной основы его конструкции”.

Клеточные автоматы позволяют аппроксимировать и моделировать много естественных систем.

При решении дифференциальных уравнений в частных производных пользуются дискретной решеткой, заполненной клетками, а правила переходов соответствуют разностным аналогам дифференциальных уравнений. Динамика исследуемой системы описывается локальными правилами КА, в отличие от обыкновенных дифференциальных уравнений (ОДУ). Для КА достаточно знать законы развития системы (правила переходов) на микро- или мезоуровне в небольших пространственных областях (клетках), из которых состоит макросистема. Важно, что эти локальные правила одинаковы для всех клеток.

В КА в отличие от дифференциальных уравнений (ДУ) могут использоваться не только дискретные, но и целочисленные переменные. Дискретность переменных позволяет рассматривать большой класс разрывных недифференцируемых функций. Дискретные свойства КА заметно уменьшаются при работе с большими значениями переменных. Всегда существует минимальный дискретный шаг изменения переменной.

При численном решении ОДУ или ДУ в частных производных можно уменьшать шаг дискретности до сколь угодно малых величин.

Вероятностные КА применяют для решения задач типа "реакция – диффузия – конвекция" с применением процедуры Монте-Карло.

ЦЕРНОВИЖ

5.2. Динамические автоматы распознающие КС-языки

Классическая пыль Кантора – это самоподобный фрактал. Согласно Бенуа Мандельброту [9], фракталы должны определяться в терминах фрактальной (дробной) размерности. Связь размерности и самоподобия состоит в том, что с помощью самоподобия можно сконструировать множество размерности в пространстве наиболее простым образом.

Определение 1. Размерность фракталов определяется соотношением:

$$Nr^d = 1, \text{ где}$$

N – число равных подобъектов,

r – коэффициент подобия.

Величину d называют фрактальной размерностью или размерностью подобия. Величина d находится логарифмированием обеих частей:

Классическое множество Кантора строится методом “выброшенный средний отрезок” и является самоподобным фракталом, который может быть определен следующим образом.

Определение 2. Классической пылью Кантора называется предельное множество C , которое представляет собой пересечение множеств C_n , $n = 0, 1, 2, \dots$, для которого выполняется соотношение: при $N = 2$, $r = 1/3$, размерность фрактала: $d = \log(2) / \log(3) \approx 0,6309$

Построение пыли Кантора начинается с выбрасывания средней части исходного множества C_0 , являющегося отрезком $[0, 1]$.

На первом шаге см. рис. П.1 удаляют открытый интервал $(1/3, 2/3)$. На следующих шагах выкидывают среднюю часть всех отрезков текущего уровня. Таким образом, получается последовательность множеств:

$$C_0 = [0, 1]$$

$$C_1 = [0, 1/3] \cup [2/3, 1]$$

$$C_2 = [0, 1/9] \cup [2/9, 1/3] \cup [2/3, 7/9] \cup [8/9, 1]$$

$$\vdots$$

$$C$$

Автомат Мура

Автомат Мура (предложен 1998 году), вычисляется на метрическом пространстве X , обращаясь к точному отображению $f_i: X \rightarrow X$ для каждого символа i . Автомат должен быть в указанной точке на X , когда он начинает читать строку. Если он останавливается в указанной конечной области после обработки каждого символа строки, то строку считают частью его языка.

Мур использует множество Кантора, чтобы сконструировать автомат в одномерном пространстве \mathfrak{X} для любого КС-языка:

Таб.П.1. $push_i$ и pop_i функции Мура

Имя	Функция
-----	---------

$push_i$	$\alpha x + (1-\alpha) \frac{i}{m}$
pop_i	$\frac{1}{\alpha} (x - (1-\alpha) \frac{i}{m}) = push_i^{-1}(x)$

Здесь, $1 \leq i \leq m$ и $0 < \alpha \leq 1/(3m+1)$, где m – число символов в алфавите стека. Начальное состояние $x_0 = 1/2$. Заключительная область $[0, 1]$. Эти $2m$ функции используются для моделирования стека с m символами. Функции $push_i$ и pop_i означают соответственно "втолкнуть символ i в стек" и "вытолкнуть символ i из стека". Стековые автоматы генерируют те же самые языки, как и их подмножество, в которых не имеется никаких управлений изменением состояния [5]. То есть все КС языки могут быть сгенерированы устройством, состоящим из стека и правил его управления. Поэтому, при соответствующей композиции функций $push_i$ и pop_i можно построить механизм генерации любого КС языка.

Необходимо предусмотреть два условия:

- обнаружение неправильной pop_i при $pop_j \otimes push_i$, где $i \neq j$. При любом выполнении неправильной формы $pop_j \otimes push_i$, $|x| \geq 2$, а также $|x| \geq 2$, если цепочка в целом была обработана. При всех правильных выполнениях $|x| < 1$. Для записи информации автоматом о граничном состоянии $|x|$ введена переменная y , с начальным значением 0. y обновляется согласно $y \leftarrow f(x, y) = \max(|x|, |y|)$, если $y \in [0, 1]$ в конце обработки строки, то все шаги выполнили стековые операции автомата;
- обеспечение уникальной связи точки в \mathfrak{X} с каждой последовательностью pop_j и $push_i$. Выбор $x_0 = 1/2$ гарантирует, что каждое состояние стека (каждая последовательность символов стека) размещает распознаватель в уникальной точке \mathfrak{X} .

При метрическом кодировании КС-языка с уровнем вложенности и с увеличением мощности алфавита стека, так как $\alpha \leq 1/(3m+1)$, требуемая точность растет по экспоненте. Использование многомерности может устранить этот рост. Следующий пример иллюстрирует применение метода Мура к треугольнику Серпинского.

Автомат на основе треугольника Серпинского

Автомат на основе треугольника Серпинского рассмотрен в работе [10]. Пусть задана КС грамматика G , генерирующая строки стандартным способом, где ε – пустая строка [5].

КС грамматика G :

$$\begin{array}{llll} G \rightarrow A B C D & A \rightarrow a G & B \rightarrow b G & C \rightarrow c G \quad D \rightarrow d G \\ G \rightarrow \varepsilon & & C \rightarrow a G & \end{array}$$

Грамматика S генерирует, например, строки:

1. $a b c d$
2. $a b c d a b c d$

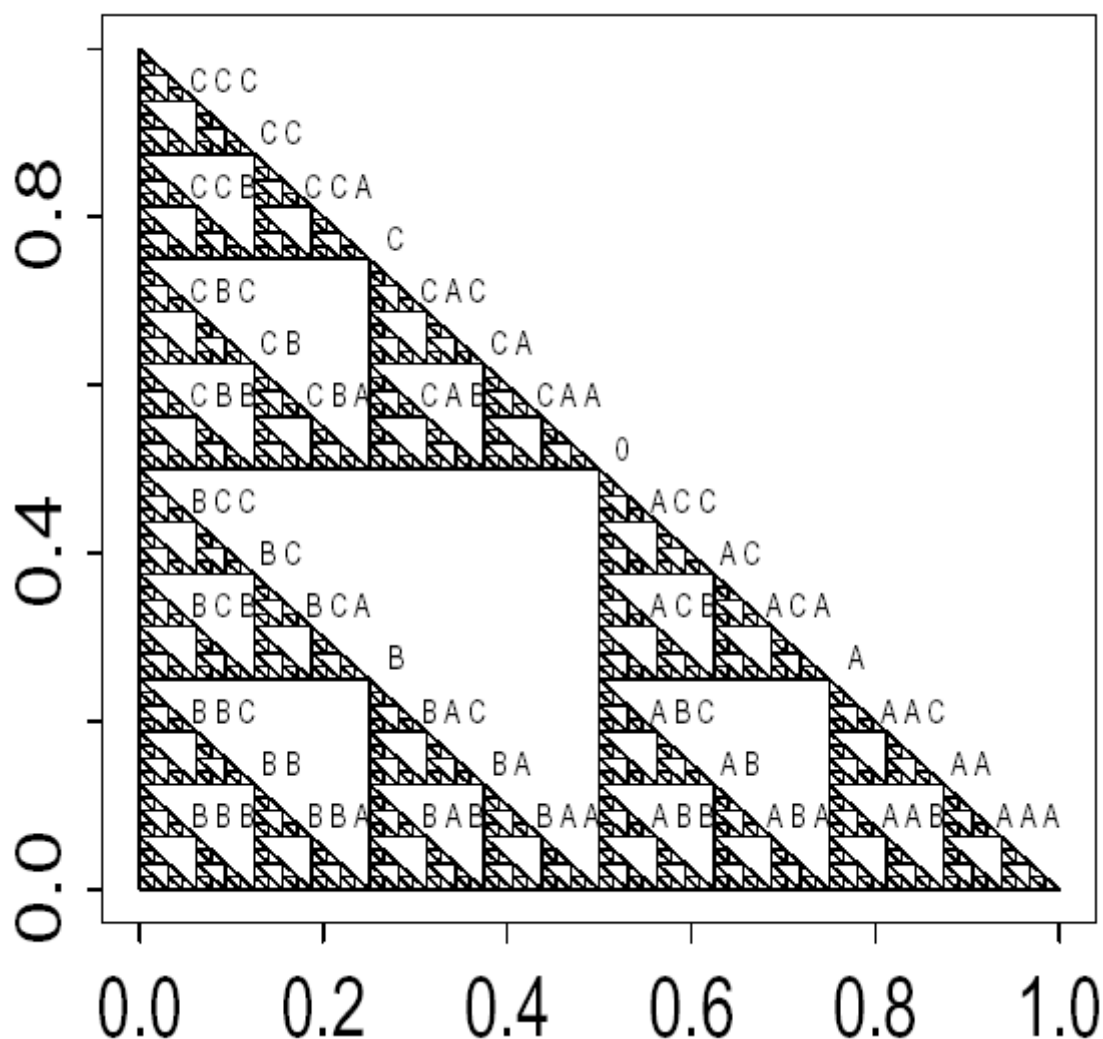
3. $a b c a a b c d b c d d$ - иллюстрирует вложенность: $G \Rightarrow ABCD \Rightarrow ABcGD \Rightarrow ABcABCDD \Rightarrow ABcaGBCDD \Rightarrow ABcaABCDBCDD \Rightarrow a b c a a b c d b c d d$

Автомат должен сохранять след каждой цепочки "abcd", которая была начата, но не закончена. Сохраняется последний символ любой частично завершённой цепочки стека. Например, автомат сохраняет символ "A" для вложения "a", "B" для вложения "b" и "C" для вложения "c". Состояниями стека должны быть члены $\{A, B, C\}^*$.

Треугольник Серпинского, своего рода вид множества Кантора, является пределом метода последовательного выкидывания "средней части" треугольника, воспроизводя три новых треугольника. На треугольнике Серпинского сохраняются состояния траекторий грамматики G.

Рассмотрим схему индексации выбранных точек рис. 2. Точки являются аналогами состояний стека в МП автомате. Метка заносится в список перед добавленными символами слева. Каждый член $\{A, B, C\}^*$ является меткой некоторой средней точки гипотенуз подтреугольников. Например, метке "CB" соответствует точка $(1/8, 5/8)$. На рис. II.2 дана разметка мощности ≤ 3 .

Рис. II.2. Схема индексации точек на треугольнике Серпинского.



Автомат на основе треугольника Серпинского для распознавания языка порождаемого грамматикой S по входному отображению показан в таб. II.2.

Таблица II.2. Автомат S.

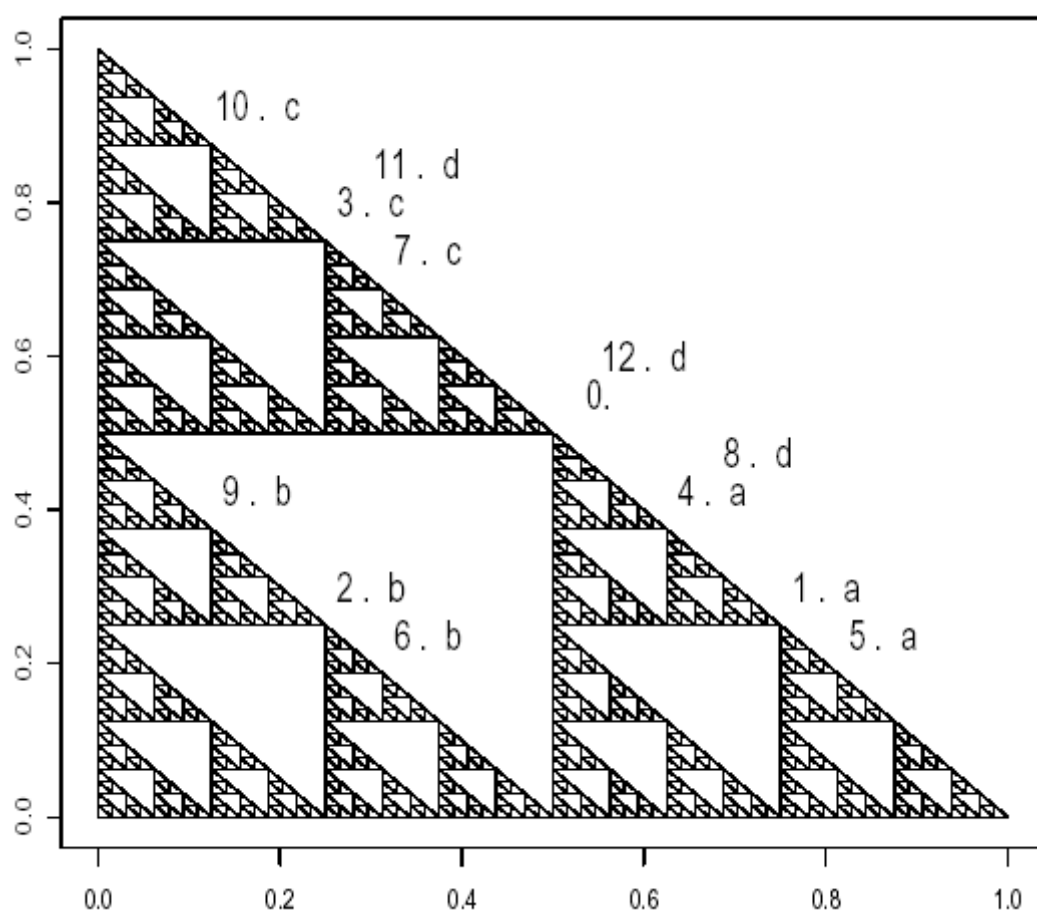
Сегмент	Вход	Изменение состояния
$z_1 > 1/2$ and $z_2 < 1/2$ переход из A	b	$\vec{z} \leftarrow \vec{z} - \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}$
$z_1 < 1/2$ and $z_2 < 1/2$ переход из B	c a	$\vec{z} \leftarrow \vec{z} + \begin{pmatrix} 0 \\ 1/2 \end{pmatrix}$
$z_1 < 1/2$ and $z_2 > 1/2$ переход из C	d	$\vec{z} \leftarrow 2 \left(\vec{z} - \begin{pmatrix} 0 \\ 1/2 \end{pmatrix} \right)$
Любой	a	$\vec{z} \leftarrow \frac{1}{2} \vec{z} + \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}$

Вектор z с двумя элементами z_1 и z_2 соответствует позиции на треугольнике Серпинского. Функции, изменения состояния автомата G, следующие: когда z находится в подмножестве указанной в колонке "сегмент", возможные входные символы – показаны в колонке "вход". Учитывая сегмент и разрешенный вход для сегмента, z изменяется в

результате чтение входного символа, как показано в колонке "Изменение состояния". Автомат начинает выполнение с $z_0 = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$, изменения состояния согласовано с правилами таб. II.2. Символы читаются из входной строки и возвращаются в $z = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$ (заключительный сегмент), когда последний символ прочитан.

Любая подпоследовательность формы "a b c d" вызывает идентичное отображение z . Автомат эквивалентен вложенной машине конечных состояний для грамматики G . На рис. II.3 показана траектория для строки "a b c a a b c d b c d d" (1. a - позиция после первого символа, "a" был обработан; 2. b – позиция после второго символа, "b" был обработан, и т.д.)

Рис. II.3. Траектория помеченной строки автомата G



Выводы к разделу. Свойства автомата, распознающего грамматику на основе фрактальных множеств:

- функции автомата ограничиваются сегментами;
- разные лексические классы соответствуют разным сегментам пространства. Три лексических класса A, B, и C (см. пример) соответствуют трем разным областям пространства (каждый класс имеет только одного члена). Метка "d" не нуждается в собственном классе, так как после ее обработки автомат переходит в состояние одного из других трех классов;
- входной символ расширяет функцию, если автомат находится в соответствующем сегменте;

- автомат согласован с поступлением информации. Строка, не принадлежащая языку, всегда обнаруживается при поступлении информации, но может быть, и не обнаружена, пока не будет обработан заключительный символ;

- вычисления подобны империческим наблюдаемым вычислениям рекуррентных нейро-сетей;

- при обработки фраз с небольшими искажениями, рекуррентные нейро-сети дифференцируют последовательные уровни вложения. Единственная фраза G связана с характерной (треугольной) траекторией, а небольшие смещения также дифференцируют последовательные уровни вложения.

Класс динамических автоматов

Д-автоматы ведут себя подобно МП-автоматам (МП – магазинная память). При работе Д-автоматы используют вместо стека фрактальные множества подобные пыли Кантора, так как они достаточно просто могут быть построены на основе СИО.

Для однозначного определения Д-автоматов подмножества фрактала должны быть изолированы друг от друга, каждому подмножеству поставим в соответствие символ стека. Вершина стека - всегда символ, соответствующий текущему подмножеству. Тогда Д-автомат кодирует точки фрактала, идентифицируя вершину стека и определяет, какое подмножество активно.

Точки траектории x соответствуют один-к-одному состоянию стека с алфавитом $\Sigma = \{1, \dots, N\}$. Текущая траектория x фрактала соответствует текущему состоянию стека. Аналогами функций $push_i$ и pop_i являются: для $push_i$ символ $a_j \in \Sigma$ должен изменить состояние автомата с $r_i(x)$ на $r_{i+1}(x)$; для pop_i символ a_j может быть выполнен, только когда текущее подмножество фрактала – это a_j (такое условие может быть предписано правилами переписывания) и оно состоит из изменения состояния с $r_i(x)$ на $r_{i-1}(x)$.

Введем определения, необходимые для построения Д-автоматов. Докажем теорему, которая помогает выбрать начальные условия для Д-автоматов, распознающих КС-языки. Даются определение класса взаимнооднозначных ФВ-автоматов (без правил переписывания) и класса Д-автоматов.

Определение 8. Строка $\sigma \in \Sigma$, где Σ символьное пространство, называется x_0 -адресом точки x из S , если каждой точке $x \in S$ сопоставлена последовательность целых чисел $\sigma = \sigma_1 \sigma_2 \dots \sigma_k$, σ_i – есть строки в $\{1, \dots, N\}^*$ и точка $x = r_\sigma(x_0) = r_{\sigma_1} r_{\sigma_2} \dots r_{\sigma_k}(x_0)$ задается последовательностью итераций r , действующей на значение $x_0 \in X$. Множество точек из X , которое может быть достигнуто от начальной точки x_0 конечной последовательностью применения r , называется r -орбитой x_0 .

Лемма 1. Если каждая точка орбиты $x = r_{\sigma_1} r_{\sigma_2} \dots r_{\sigma_k}(x_0)$ из СИО S , $x_0 \in X$ имеет уникальный x_0 -адрес, то x_0 – уникально обратимая точка. Орбита $r^{-1}_{\sigma_k}(x_0)$ – обратимая орбита x_0 .

Доказательство. Пусть $\rho_{\sigma k}^{-1}(x_0)$ не является обратной орбитой $x_0 \in X$, тогда x_0 -адрес не уникален, а $S = \{X \mid \rho_1, \dots, \rho_N\}$ не является уникально обратимой СИО, тогда не существует уникально обратимых операторов $\rho_{\sigma k}^{-1}(x_0)$, что противоречит определению 7. Следовательно, x_0 - уникально обратимая точка.

Теорема 3. Уникально обратимая точка x_0 с уникальным x_0 -адресом является начальной уникальной обратимой точкой, если $\rho_{\sigma k}(x_0) \neq <<_1$ и $x_0 \notin X_1$, где X_K – самоподобное множество, полученное оператором разбиения $X = X \div N^1, \dots, N^m = X_1 \cup X_2 \cup \dots \cup X_K$, где $K = \{1, 2, \dots, N\}$, $n = \{1, 2, \dots, m\}$ – размерность пространства \mathfrak{X} .

Доказательство. Пусть $x_0 \in X_1$ и $\rho_{\sigma k}(x_0) = <<_1$ для композиции $<<_1 X \div N^1, \dots, N^m$, тогда точка x_0 не будет уникально обратимой точкой, так как после выполнения оператора сдвига $x_0 \notin X$. Следовательно, x_0 -адрес является уникально обратимым по лемме 1 и $\rho_{\sigma k}(x_0)$ по определению 7, если $\rho_{\sigma k}(x_0) \neq <<_1$ и $x_0 \notin X_1$.

Теорема 4. Обратимая орбита начальной точки $x = \rho_{\sigma}(x_0) = \rho_{\sigma 1} \rho_{\sigma 2} \dots \rho_{\sigma k}(x_0)$ эквивалентна состоянию МП-автомата.

Доказательство: Пусть существует взаимнооднозначное соответствие между операторами уникально обратимой системы $S = \{X \mid \rho_1, \dots, \rho_N\}$ и МП-автоматом:

$$\begin{aligned} \rho_i(x) &\equiv \text{push}_i(x), \\ \rho_i^{-1}(x) &\equiv \text{push}_i^{-1}(x) \end{aligned}$$

Пусть заданы начальные условия для МП-автомата $Z_0 = \varepsilon$ - стек пуст и x_0 -- уникальная начальная точка $x_0 \in S$. Тогда композиция функций МП-автомата $\text{push}_i(x) \otimes \text{push}_i^{-1}(x)$ и состояние его стека соответствует композиции $\rho_i(x) \otimes \rho_i^{-1}(x)$ и состоянию обратной орбиты $\rho_{\sigma}(x_0)$, для каждого i шага выполнения МП-автомата.

Следствие 1. Каждая точка обратной орбиты $\rho^{-1}(x)$ при $\rho_{\sigma}(x_0) = \rho_{\sigma 1} \rho_{\sigma 2} \dots \rho_{\sigma k}(x_0)$ может быть отображена уникальным состоянием стека.

Следствие 2. Все КС-языки могут быть распознаны Д-автоматом, построенным по аналогии с МП-автоматом.

Утверждение 1. Так как любой КС язык может быть распознан МП-автоматом, то можно построить механизм распознавания любого КС языка Д-автоматом.

Рассмотрим язык $L = 0^n 1^n$, распознаваемый автоматом Мура см. раздел 1. Построим Ф-автомат, распознающий язык L . Для построения используем теорему 4. Построение фрактального множества однозначно свяжем с композицией $\rho_i(x) \otimes \rho_i^{-1}(x)$ используем его как память Ф-автомата. Составим таб. П.3. Орбита $\rho_{\sigma}(x_0) = \rho_{\sigma 1} \rho_{\sigma 2} \dots \rho_{\sigma k}(x_0)$ изменяется в зависимости от выполнения операторов $\rho_i(x) \otimes \rho_i^{-1}(x)$. Выполнением управляет прочитанный из строки символ. Если после изменения орбиты x находится в указанной области, то строку считают частью языка автомата. Заключительная область $[0, 1]$, означает, что строка распознана без искажений, во всех остальных случаях с искажениями.

Таб. II.3. Правила Ф-автомата для распознавания языка $0^n 1^n$.

Символ	Имя	Функция
0	$X_i = \llcorner_K X_{i-1} \div N$	$X_i = \llcorner_K X_{i-1} \div N$
1	\llcorner_K^{-1}	$X_i = \llcorner_K^{-1} X_{i+1}$
x_0	Без искажений	$[0,1]$

Рассмотрим два случая работы автомата при распознавании строки L языка, когда строка задана без искажения и с искажением.

1. Распознавание без искажений. Пусть задана строка 0011 и требуется распознать принадлежит ли она языку $0^n 1^n$. Ф-автомат работает следующим образом. Считывается первый символ 0, выполняется оператор $\rho_{\sigma 1}(x)$, т.е. строится фрактальное множество Кантора. Считывается первый символ 1, выполняется оператор $\rho_{\tau 1}(x)$, восстанавливается множество, полученное на предыдущем шаге, считывается второй символ 1, выполняется оператор $\rho_{\sigma 1}^{-1}(x)$. Если все символы считаны и $x_0 = 1/2$. Строка принадлежит распознаваемому языку L .

2. Распознавание с искажением. Пусть задана строка 00011 и требуется распознать принадлежит ли она языку $0^n 1^n$. Выполним все шаги аналогично случаю 1. После того как все символы считаны, метка $x_0 \neq 1/2$. Строка принадлежит распознаваемому языку $C \subset [0,1]$, с искажением.

Воспользуемся теоремой 1. о сжимающем отображении и введем следующее определение.

Определение 9. Искажение, полученное после завершения работы Д-автомата определяется коэффициентом сжатия $s > 0$. Чем меньше s , тем меньше язык L подобен распознаваемому языку.

Для рассматриваемого примера $s = 1/6$, можно считать, что строка принадлежит языку. Если, например, $s = 1/20$, то нет. Могут быть предложены правила для принятия решения по имеющимся искажениям, в зависимости от их диапазона.

Определим специальный класс взаимнооднозначных ФВ-автоматов, которые двигаются вокруг уникально обратимых орбит при распознавании строк КС-языков.

Определение 10. ФВ-автомат $M_{\text{ФВ}} = \langle \Sigma, S, x_0, F_x \rangle$, где

$S = \{X \mid \rho_{\sigma 1}, \dots, \rho_{\sigma N}\}$ – уникально обратимая система итерированных операторов (СИО) с точками на метрическом пространстве $x_{\sigma 1}, x_{\sigma 2}, \dots, x_{\sigma N} \in X$, где $N \in \{1, 2, \dots\}$ и каждому входному символу соответствует один из операторов:

$$\rho_{\sigma i}(x) = \llcorner_K X_{i-1} \div N,$$

$$\rho_{\tau i}^{-1}(x) = \llcorner_K^{-1} X_{i+1}, \text{ где } \tau, \sigma \in \Sigma \text{ конечному входному словарию};$$

Операторы изменяют орбиту уникальной начальной точки $x_0 \in X$;

Выполнение заканчивается при $x_0 \in F_x$ - конечное подмножество $F_x \subset X$.

Следующее определение делает возможным контролировать орбиту $x = \rho_{\sigma i}(x_0)$ с помощью правил переписывания σ .

Определение 11. Пусть $S = \{ X \mid \rho_{\sigma 1}, \dots, \rho_{\sigma N} \}$ - уникально обратимая СИО. Пусть $\rho_{\sigma 1}(x), \rho_{\tau_2}(x) \in S$ - орбиты $x \in X$. Тогда правило переписывания $\sigma : \rho_{\sigma 1}(x)\tau_j \rightarrow \rho_{\tau_2} \rho_{\sigma 1}(x)$ такое, что адрес $x = x_2x_1 = \rho_{\tau_2} \rho_{\sigma 1}(x)$, где x_2 - адрес $\rho_{\tau_2}(x)$, x_1 - адрес $\rho_{\sigma 1}(x)$, $\tau_j \in \Sigma$. σ - уникально обратима для всех $\rho_{\tau_2}(x)$, так как уникально обратимы орбиты $\rho_{\sigma N}(x)$, то есть существует σ^{-1} .

Треугольник Серпинского по сравнению с ковром Серпинского [1] имеет ряд свойств, упрощающих его построение: исходное множество X занимает половину пространства \mathbb{R}^2 , а после разбиения X на равные части сразу получается средняя часть гипотенузы, это начальная точка x_0 . Отсутствие смежных точек между самоподобными подмножествами после разбиения, повышает надежность фрактального множества для кодирования грамматики КС-языка. Однако для построения треугольника надо вводить дополнительные операторы. В рассматриваемом примере оператор разбиения и точки пересечения разбиения соответствуют точкам на треугольнике Серпинского.

Введем формальное определение Д-автомата:

Определение 12. Динамический автомат M_Φ - это кортеж:

$M_\Phi = \langle X, \Sigma, S, \sigma, x_0, F_x \rangle$, где

X - метрическое пространство с метрикой d , (X, d) ;

Σ - конечный входной словарь;

$S = \{ X \mid \rho_{\sigma 1}, \dots, \rho_{\sigma N} \}$ - уникально обратимая система итерированных операторов (СИО) с точками на метрическом пространстве $x_{\sigma 1}, x_{\sigma 2}, \dots, x_{\sigma N} \in X$, где $N \in \{1, 2, \dots\}$. $\rho_{\sigma i}(x)$ - операторы, изменяющие орбиту точки $x \in X$.

x_0 - начальный адрес точки x с которого Д-автомат начинает выполнение, $x_0 \in X$;

F_x - конечное подмножество $F_x \subset X$. Если вся входная цепочка распознана, то осуществляется возврат в конечное подмножество F_x ;

δ - множество правил переписывания, управляющие орбитами $\rho_{\sigma i}(x)$, имеют следующий вид:

$\rho_{\sigma i}(x)\tau_j \rightarrow \rho_{\tau_i} \rho_{\sigma i}(x_0)$, где $\tau_j \in \Sigma$ - входной символ, $\rho_{\tau_i}, \rho_{\sigma i} \in S$.

Конфигурация C имеет вид $(S \times \Sigma \times S)$. Для $C_m = \rho_{\sigma i}(x)\tau_j \omega \rho_{\sigma i}(x_0)$, где ω - выражение из Σ^* (последовательность входных слов), $\rho_{\tau_i}(x), \rho_{\sigma i}(x) \in S$. Процесс переписывания определяется отношениями \Rightarrow и $*\Rightarrow$ между конфигурациями:

$\rho_{\sigma i}(x)\tau_j \omega \rho_{\sigma i}(x_0) \Rightarrow \rho_{\tau_{i+1}} \omega \rho_{\tau_{i+1}} \rho_{\sigma i}(x_0)$, если правило $\rho_{\sigma i}(x)\tau_j \rightarrow \rho_{\tau_{i+1}} \rho_{\sigma i}(x_0) \in \delta$, где i - итерация разбиения. Оператор разбиения выполняется каждый раз при новой итерации.

Конфигурация $C_1 \xRightarrow{*} C_n$ имеет место, если существует последовательность конфигураций $\bigcup_{m=1}^n C_m$ для $C_m \xRightarrow{*} C_{m+1}$, где $m = 1, 2, \dots, n-1$.

Пример. Определим M_Φ автомат для грамматики G из раздела 1.3.

В качестве X взята плоскость $C_0 = [0,1] \times [0,1]$ в пространстве \mathbb{R}^2 .

$$\Sigma^* = \{a, b, c, d, \varepsilon\},$$

$$S = \{X \mid \rho_a, \rho_b, \rho_c, \rho_{\Sigma_{i-1}}, \rho_{\Sigma_i}\}, \text{ где}$$

$\rho_{\Sigma_{i-1}}^{-1}$ -- оператор обратного сдвига выполняется до первого элемента с индексом $i-1$. Индекс Σ означает все возможные символы множества Σ .

Для определения итерированных операторов сдвига компактной точки выполним оператор разбиения $X \div 2,2$, где $N^n=4$, тогда $X = X_1 \cup X_2 \cup X_3 \cup X_4$ рис. 5.7.а. Зададим адрес $x_0 \in F_x \subset X$, $x_0 = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$. Выполним следующее

разбиение каждого из полученных подмножеств на $N^n=4$, $(X \div N) \div 2,2$. Получим четыре точки в результате разбиения рис. II.7.б. Для того чтобы точку x_0 сдвинуть в точку a, b, c надо выполнить соответствующие операторы: сдвиг влево $>>$, сдвиг вправо $<<$, в верх \uparrow , вниз \downarrow . Сдвиг осуществляется на единицу разбиения, относительно x_0 , тогда уникально обратимые операторы :

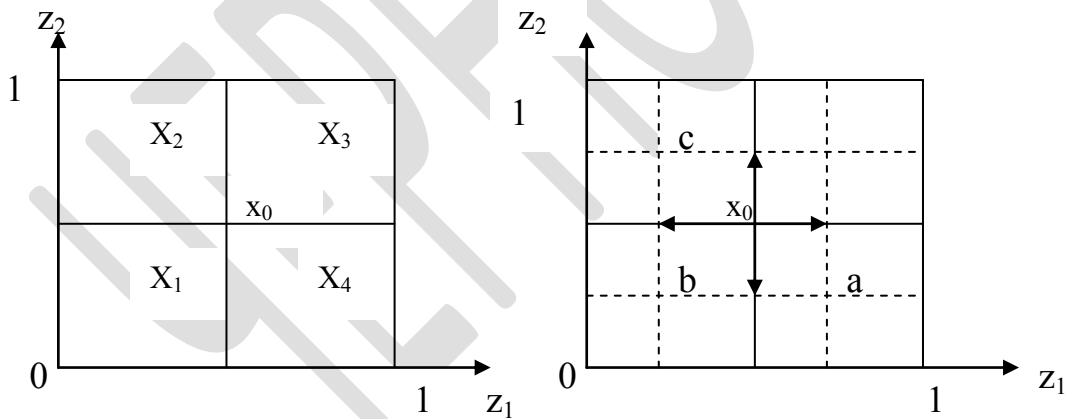
$$\rho_{ai}(x_0) = x_0 >> \downarrow$$

$$\rho_{bi}(x_0) = x_0 << \downarrow$$

$$\rho_{ci}(x_0) = x_0 >> \uparrow$$

Оператор разбиения выполняется всякий раз, когда увеличивается индекс i , например: $\rho_{\Sigma_{i+1}}$. Оператор разбиения действует только на то подмножество, для которого он указан, например:

$\rho_{a1+1}(x_0)$ осуществляет разбиение подмножества, в котором находится $\rho_{a1}(x_0)$.



а. Начальные значения и подмножества

б. Определение операторов сдвига компактной точки

Рис. 5.7. Динамический автомат для грамматики G

$$\sigma = \{ \rho_{\Sigma_i}(x) a \rightarrow \rho_{a i+1}(x)$$

$$\rho_{ai}(x) b \rightarrow \rho_{bi}(x)$$

$$\rho_{bi}(x) c \rightarrow \rho_{ci}(x)$$

$$\rho_{ci}(x) a \rightarrow \rho_{ai}(x)$$

$$\rho_{\Sigma_i}(x) d \rightarrow \rho_{\Sigma_{i-1}}^{-1}(x)$$

}

Пример 1. $\omega = a b c d$, то конфигурации :

$$C_1 = \rho_{\varepsilon 0}(x) a b c d x_0$$

$$C_2 = \rho_{a1}(x) b c d \rho_{a1}(x_0)$$

$$C_3 = \rho_{b1}(x) c d \rho_{b1}\rho_{a1}(x_0)$$

$$C_4 = \rho_{c1}(x) d \rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_5 = x_0$$

Пример 2. Пусть $\omega = a b c a a b c d b c d d$, то конфигурации :

$$C_1 = \rho_{\varepsilon 0}(x) a b c a a b c d b c d d x_0$$

$$C_2 = \rho_{a1}(x) b c a a b c d b c d d \rho_{a1}(x_0)$$

$$C_3 = \rho_{b1}(x) c a a b c d b c d d \rho_{b1}\rho_{a1}(x_0)$$

$$C_4 = \rho_{c1}(x) a a b c d b c d d \rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_5 = \rho_{a2}(x) a b c d b c d d \rho_{a2}\rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_6 = \rho_{a3}(x) b c d b c d d \rho_{a3}\rho_{a2}\rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_7 = \rho_{b3}(x) c d b c d d \rho_{b3}\rho_{a3}\rho_{a2}\rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_8 = \rho_{c3}(x) d b c d d \rho_{c3}\rho_{b3}\rho_{a3}\rho_{a2}\rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_9 = \rho_{a2}(x) b c d d \rho_{a2}\rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_{10} = \rho_{b2}(x) c d d \rho_{b2}\rho_{a2}\rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_{11} = \rho_{c2}(x) d d \rho_{c2}\rho_{b2}\rho_{a2}\rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_{12} = \rho_{a1}(x) d \rho_{c1}\rho_{b1}\rho_{a1}(x_0)$$

$$C_{13} = x_0$$

Рис. II.8. иллюстрирует конфигурацию C_5 .

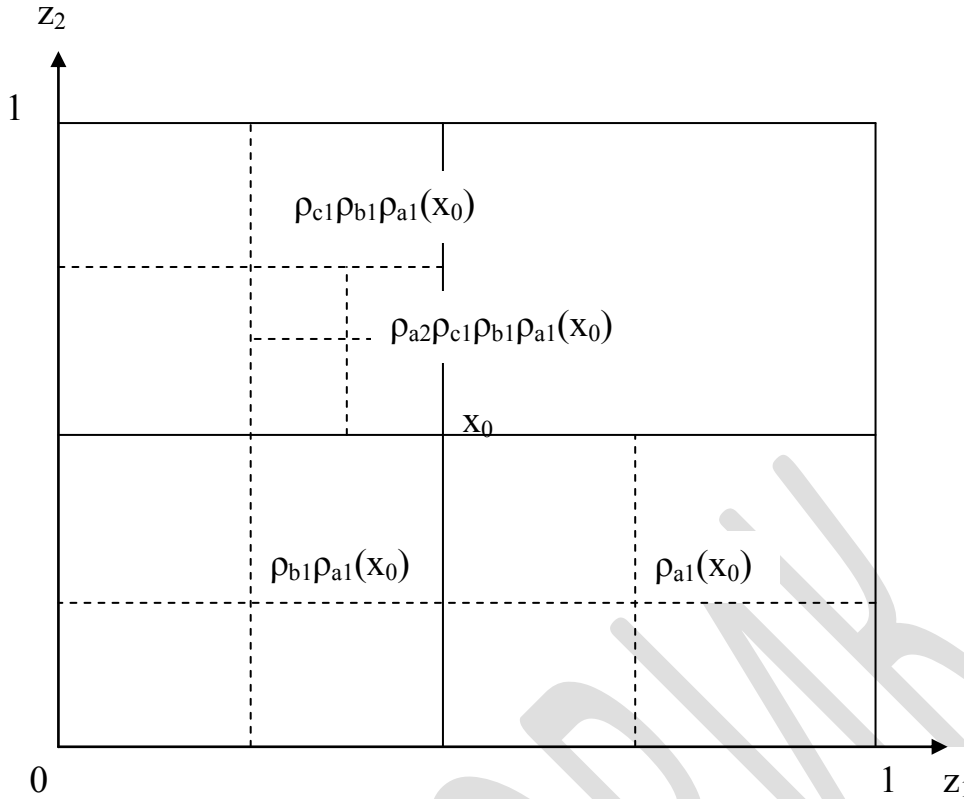


Рис. 5.8. Конфигурация C_5

Выполнение автомата заключается в построении фрактального множества по мере поступления символов строки на вход и выполнения правил переписывания автомата.

Теорема 5. Множество языков L , распознаваемых Д-автоматами (M_Φ), эквивалентно множеству языков L , распознаваемых МП-автоматами ($M_{МП}$): $L(M_\Phi) = L(M_{МП})$ и наоборот.

Доказательство. Дано: $M_{МП} = (Q, \Sigma_{МП}, \Gamma_{МП}, \delta_{МП}, q_0, Z_0, F)$ и $M_\Phi = (X, \Sigma_\Phi, S, \delta_\Phi, x_0, F_x)$.

Из равенства распознаваемых языков L следует, что входные словари равны $\Sigma_{МП} = \Sigma_\Phi = \Sigma$. Пусть внутренние словари $\Gamma_{МП} = \Gamma_\Phi$.

Утверждение 1. Если конфигурации $M_{МП}$ для распознавания языка L эквивалентна конфигурациям M_Φ для распознавания этого же языка L , то языки распознаваемые $M_{МП}$ и M_Φ - эквивалентны.

Пусть конфигурация $M_{МП}$ и M_Φ будет C_i . Если существует последовательность конфигураций $\bigcup_{i=1}^K C_i$, где $i \in \{1, \dots, K\}$, то $C_{i-1}^* \Rightarrow C_i$. Пусть

индексы в конфигурациях $M_{МП}$ и M_Φ соответствуют, а адрес x M_Φ представлен в $M_{МП}$ положительными целыми числами из Γ .

Представим компоненты $M_{МП}$:

$Q = \{q_i \mid i \in \{1, \dots, K\}\}$, где K положительное целое число. Q - конечное множество состояний $M_{МП}$ помеченных q_1, \dots, q_K .

$\Gamma = \{1, \dots, N\}$, где N положительное целое число.

Определим язык воспринимаемый $M_{МП}$, как язык воспринимаемый заключительным состоянием $q_i \in F$:

$L = \{\omega \in \Sigma^* \mid q_0 \omega Z_0 \xRightarrow{*} q_i \varepsilon \gamma, \text{ где } q_i \in F \text{ и } Z_0, \gamma \in \Gamma^*\}, Z_0 = \varepsilon$ – начальный символ стека.

Конфигурация $M_{МП}, C_{i-1} \xRightarrow{*} C_i$, имеет вид $q_{i-1} \omega Z_k \gamma \Rightarrow q_i \omega \gamma_m \gamma$, если правило $q_{i-1} a_j Z_k \rightarrow q_i \gamma_m \in \sigma$, где $a_j \in \Sigma$, входной символ. Если $\gamma_m = \varepsilon'$, то содержимое стека уменьшается.

Представим компоненты M_Φ :

$S = \{X \mid \rho_{\sigma_1}, \dots, \rho_{\sigma_K}\}$ – есть ассоциированная СИО с точками $x_{\sigma_1}, x_{\sigma_2}, \dots, x_{\sigma_K} \in X$, где $K \in \{1, 2, \dots\}$ им соответствуют конфигурации $\bigcup_{i=1}^K C_i$, причем индексы K

для M_Φ и $M_{МП}$ эквивалентны. M_Φ объединяет непересекающиеся уникальные орбиты $\rho_{\sigma_1}, \dots, \rho_{\sigma_K}$.

Определим язык воспринимаемый M_Φ , как язык с орбитой в заключительной точке $\rho_0(x_0) = x_0 \in F_x$:

$L = \{\omega \in \Sigma^* \mid \rho_{\sigma_i}(x) \tau_j \omega \rho_{\sigma_i}(x_0) \Rightarrow \rho_{\tau_{i+1}}(x) \omega \rho_{\tau_{i+1}} \rho_{\sigma_i}(x_0) \text{ и } \rho_{\tau_{i+1}}, \rho_{\sigma_i}(x) \in S = \{X \mid \rho_{\sigma_1}, \dots, \rho_{\sigma_K}\}\}$, где x_0 – уникальная начальная точка, S – уникально обратимая система итерированных операторов.

Конфигурация $M_\Phi, C_{i-1} \xRightarrow{*} C_i$, имеет вид $\rho_{\sigma_i}(x) \tau_j \omega \rho_{\sigma_i}(x_0) \Rightarrow \rho_{\tau_{i+1}}(x) \omega \rho_{\tau_{i+1}} \rho_{\sigma_i}(x_0)$, если $\rho_{\sigma_i}(x) \tau_j \rightarrow \rho_{\tau_{i+1}} \rho_{\sigma_i}(x_0) \in \delta$, где $\tau_j \in \Sigma$, входной символ. Точка $x \in \bigcup_{i=1}^K C_i$ – адрес точки x из S , которому сопоставлена последовательность

целых чисел $\sigma = \sigma_1 \sigma_2 \dots \sigma_K$, σ_i – есть строки в $\{\varepsilon, 1, \dots, N\}^*$ и точка $x = \rho_\sigma(x_0)$, строка $\sigma \in \Sigma$. Индекс N из M_Φ эквивалентен индексу $M_{МП}$ для Γ .

Утверждение 2. Если правило переписывания $\rho_{\sigma_i}(x) a_j \rightarrow \rho_{\alpha_i} \rho_{\sigma_i}(x_0)$ по j , где $a_j \in \Sigma$ эквивалентно правилу $q_{i-1} a_j Z_k \rightarrow q_i \gamma_m$, то конфигурации M_Φ и $M_{МП}$ эквивалентны.

Допустим без потери общности, что:

1. $q_0 = q_1$, для любого $i \in \{1, \dots, K\}$, если $q_i \in F$ то $x_0 \in F_x$;
2. $q_i \equiv \rho_{\sigma_i}(x)$ и $\sigma \in \Sigma$
3. $Z_0 \equiv x_0$.

Тогда конфигурации, приведенные в таб. II.4 для правил $\delta_{МП} \equiv \delta_\Phi$ эквивалентны при распознавании строки ω , где Z_k – символ из стека, γ_m – последовательность символов из стека на которую заменяется Z_k , γ – последовательность символов из стека, ε' – устранение символа из стека.

Таб. II.4. Эквивалентные конфигурации M_Φ и $M_{МП}$ автоматов

Конфигурации автоматов $a_j \in \Sigma$	$C_{МП} = Q \times \Sigma \times \Gamma^*$	$C_\Phi = S \times \Sigma \times S$
1. Начальная конфигурация C_1	$C_1 = q_1 \varepsilon Z_0$, где $Z_0 = \varepsilon$, $q_0 = q_1$	$C_1 = \rho_{\varepsilon 0}(x) \varepsilon x_0$, где $\rho_{\varepsilon 0}(x) = x_0$, $Z_0 \equiv x_0$
2. Заключительная конфигурация C_n	$q_i \omega \varepsilon \Rightarrow q_j \varepsilon \varepsilon'$, где $q_j \in F$	$\rho_{\sigma_1}(x) a_j \rho_{\sigma_1} \dots \rho_{\sigma_n}(x_0) \Rightarrow x_0$, где $x_0 \in F_x$
3. Отношения между конфигурациями	$q_i a_j \omega Z_k \gamma \Rightarrow q_i \omega \gamma_m \gamma$ $Z_k, \gamma, \gamma_m \in N$,	$\rho_{\sigma_i}(x) a_j \omega \rho_{\sigma_i}(x_0) \Rightarrow \rho_{\alpha_i}(x) \omega \rho_{\alpha_i} \rho_{\sigma_i}(x_0)$

$C_i \Rightarrow C_l$	$q_i, q_l \in K$	$\rho_{\sigma_i}(x) \equiv q_i, \sigma_i \equiv \gamma, \alpha_i \equiv \gamma_m, \rho_{\alpha_i}(x) \equiv q_l$
4. Конфигурация для функции pop_i (вытолкнуть символ)	$q_i a_j \omega \Rightarrow q_i \varepsilon \omega \gamma$ $q_i \in K$	$\rho_{\Sigma_i}(x) a_j \omega \rho_{\sigma_i} \dots \rho_{\tau_{i-1}}(x_0) \Rightarrow$ $\rho_{\tau_{i-1}}(x) \omega \rho_{\tau_{i-1}}(x_0)$ $\rho_{\Sigma_i} \equiv q_i, \tau_{i-1} \equiv \gamma, \varepsilon \equiv \rho_{\sigma_i} \dots \rho_{\sigma_{i-1}}(x_0)$
5. Конфигурация для функции push_i (втолкнуть символ)	$q_i a_j \omega Z_k \gamma \Rightarrow$ $q_{i+1} \gamma_m \gamma$ $Z_k \in N, q_i, q_{i+1} \in K$	$\rho_{\sigma_i}(x) a_j \omega \rho_{\sigma_i}(x_0) \Rightarrow$ $\rho_{\alpha_{i+1}}(x) \omega \rho_{\alpha_{i+1}} \rho_{\sigma_i}(x_0)$ $\rho_{\sigma_i}(x) \equiv q_i, \rho_{\alpha_{i+1}}(x) \equiv q_{i+1}, \alpha_{i+1} \equiv \gamma_m,$ $\sigma_i \equiv \gamma$

В каждом случае конфигурации четко определены и эквивалентны. Если $C_1 \Rightarrow C_j$, то $M_{МП}$ и M_{Φ} автоматы распознают один и тот же язык L , $L(M_{МП}) = L(M_{\Phi})$.

Заключение

Динамические автоматы организует вычисления на основе фрактальных множеств и операторов разбиения и сдвига. Класс распознает КС языки, позволяет избежать сложности в хранении траектории и состояния автомата, интерпретирует значения метрического пространства в виде состояний и энтропии.

Д-автоматы ведут себя подобно МП-автоматам и имеют подобные сходства с нейро-сетями. *Д-автоматы* отображают пространственные отношения между автоматами на метрическом пространстве.

Лабораторная работа 10.

1. Построить алгоритм для определения уникальных точек получаемых в результате применения оператора разбиения $X = X \div N^1, \dots, N^m = X_1 \cup X_2 \cup \dots \cup X_K$, где $K = \{1, 2, \dots, N\}$, $m = \{1, 2, \dots, j\}$ – размерность пространства \mathbb{R} .

2. Реализовать работу Ф-автомата. Ф-автомат строит фрактальное множество подобное множеству Кантора и управляет орбитой x_0 в соответствии с заданными правилами переписывания.

5.4. L-системы

Свойство – “подобия” используется в качестве определяющего во многих работах по фракталам. Слово "фрактал" [1] было введено Бенуа Мандельбротом в конце 1970'х. Часть фрактальной структуры подобна целому. Не имеет значения как сделано разбиение малой части, часть содержит не меньше деталей, чем целое. Отсюда применение термина фрактал, как производного от латинского прилагательного fractus – дробный и глагола frangere – ломать.

Понятие L-системы, тесно связано с самоподобными фракталами, и появилось только в 1968 году благодаря А. Линденмайеру [11]. Изначально

L-системы были введены при изучении формальных языков, а также использовались в биологических моделях, селекции. С их помощью можно строить многие известные самоподобные фракталы, включая снежинку Коха и ковер Серпинского, кривые Пеано (работы Пеано, Гильберта, Серпинского) и т.д.

L-системы подобны грамматикам с одной лишь принципиальной разницей; если в грамматиках на каждом шаге вывода заменяется единственное вхождение нетерминала, то в L-системах все нетерминалы замещаются параллельно.

В качестве подсистемы вывода, для графической реализации L-систем, используется *тертл*-графика (turtle – черепаха). Черепашка, двигаясь по экрану дискретными шагами как правило прочерчивает свой след, но при необходимости может перемещаться без рисования. Для ее движения задаются три параметра (x, y, a) : (x, y) – координаты черепашки, a – направление движения.

Алгоритм работы L-системы заключается в следующем. По правилам порождения строится цепочка символов или кодовое слово, символы представляют собой команды для черепашки и определяют траекторию ее движения. Черепашка выполняет последовательность команд, задаваемых словом. Символы слова читаются слева направо и кодируют следующие команды:

- F – переместиться вперед на один шаг, прорисовывая след.
- b – переместиться вперед на один шаг, не прорисовывая след.
- [– открыть ветвь (подробнее см. ниже)
-] – закрыть ветвь (подробнее см. ниже)
- + – увеличить угол a на величину q
- – уменьшить угол a на величину q

Размер шага и величина приращения по углу q задаются заранее и остаются неизменными для всех перемещений черепашки.

Команды ветвления (обозначаются], [) используются для построения деревьев растений, а вспомогательные переменные (обозначаются X, Y и т.д.) существенно облегчают построение некоторых L-систем.

Формально, детерминированная L-система состоит из *алфавита*, слова инициализации, называемого *аксиомой* или *инициатором*, и набора *порождающих правил*, указывающих, как следует преобразовывать слово при переходе от уровня к уровню (от итерации к итерации). Символы +, -,], [при подстановки правил порождения не заменяются, а просто остаются на тех местах, где они встретились.

Определение 1. Детерминированной L-системой называется кортеж из пяти элементов $\mathcal{L} = \langle \Theta, t, axiom, P, n \rangle$, где

Θ – алфавит, конечное множество символов операций $\Theta = \{F, +, -\}$, – множество всех слов над Θ , – множество не пустых слов над Θ ;

$t = \{a, \theta\}$ – параметры для черепашки. a – начальное направление движения, то есть угол, отсчитываемый от положительного направления оси

X , если не указано, то полагаем a равным нулю. θ - величина приращения по углу. a, θ остаются неизменными для всех перемещений черепашки.

$axiom$ – не пустое слово называемое аксиомой, $axiom$;

$n = 1, 2, \dots, N$ – шаг итерации;

\mathcal{P} - конечное множество продукций или правил, продукция $(p, \mathcal{A}) \in \mathcal{P}$ записывается как $p \rightarrow \mathcal{A}$, где p , а \mathcal{A} . Начиная с аксиомы $axiom$, продукции применяются параллельно ко всем символам слова. Этот итеративный процесс продолжается для заданного количества итераций n .

Алгоритм L-системы формирует кодируемое слово и может быть расширен при введении новых символов в Θ .

Алгоритм 1. L-система: реализует правило $F = p_1, b = p_2$.

Вход:

$axiom$ (слово инициализации)

p_1 (порождающее правило)

n (число итераций)

Выход:

word - слово результат

Инициализация:

$W = axiom$

$k = \text{length}(W)$

$T = \{\}$ (пустое множество)

```
while n > 0
  for j = 1 to k
    if W(j) = +, T = {T +}, end if
    if W(j) = -, T = {T -}, end if
    if W(j) = F, T = {T p1}, end if
    if W(j) = b, T = {T p2}, end if
  end for
  W = T
  n = n - 1
end while
```

word = W, где W(j) – j-ая буква в слове W, {T +} – строка T, к которой присоединен знак +.

Пример 1. Для снежинки Коха на рис. 1.3. L-система задается следующим образом:

$\mathcal{L} = \langle \Theta = \{F, +, -\}, t = \{\theta = \pi/3\}, axiom = F++F++F, p_1 = F-F++F-F, n=4 \rangle$,

Графическое представление аксиомы $F++F++F$ – равносторонний треугольник. Черепашка делает один шаг вперед, затем угол a увеличивается на $2\pi/3$ и черепашка делает еще один шаг.

Шаг 1. Каждый символ F в аксиоме $F++F++F$ заменяется при помощи порождающего правила $p_1 = F-F++F-F$:

$(F-F++F-F)++(F-F++F-F)++(F-F++F-F)$

Шаг 2.

$((F-F++F-F)-(F-F++F-F)++(F-F++F-F)-(F-F++F-F))++$

$((F-F++F-F)-(F-F++F-F)++(F-F++F-F)-(F-F++F-F))++$

$((F-F++F-F)-(F-F++F-F)++(F-F++F-F)-(F-F++F-F))$

и т.д.

Каждая треть снежинки строится итеративно, начиная с одной из сторон равностороннего треугольника.

Алгоритм 1. может построить еще некоторые фракталы.

Пример 2. Кривая Пиано заполняющая плоскость:

$\mathcal{L} = \langle \Theta = \{F, +, -\}, t = \{a = \pi/4, \theta = \pi/3\}, axiom = F, p_1 = F-F+F+F-F-F-F+F, n = 4 \rangle,$

В Алгоритме 2. интерпретируются команды ветвления. Для операции ветвления [– открыть ветвь положение и направление черепашки (x, y, a) запоминается в стеке, при закрытии ветки команда] значения выталкиваются из стека и считываются.

Алгоритм 2. Реализует команды для графики, рассмотренные выше.

Вход:

$word$ (результат работы L-системы)

θ (приращение по углу)

a (начальное направление)

Выход:

Графическое представление слова $word$

Инициализация:

$W = word$

$x_0 = 0$

$y_0 = 0$

$k = \text{length}(W)$

$stack = \{\}$ (пустое множество)

Шаги

for $j = 1$ to k

if $W(j) = +$, $a = a + \theta$, end if

if $W(j) = -$, $a = a - \theta$, end if

if $W(j) = F$, $x = x_0 + \cos(a)$, $y = y_0 + \sin(a)$,

соединить линией (x_0, y_0) с (x, y) ,

$x_0 = x$, $y_0 = y$

end if

if $W(j) = b$, $x_0 = x_0 + \cos(a)$, $y_0 = y_0 + \sin(a)$, end if

if $W(j) = [$,


```

    l = length (stack),
    stack(l+1,1) = x0
    stack(l+1,2) = y0
    stack(l+1,3) = a
end if
if W(j) = ],
    l = length (stack),
    x0 = stack(l,1)
    y0 = stack(l,2)
    a = stack(l,3)
    вытолкнуть l-ую запись из стека
end if
end for

```

Пример 3. Снежинка см. рис. 5.4.

$\mathcal{L} = \langle \Theta = \{F, +, -, [,]\}, t = \{\theta = \pi/3\}, axiom = [F]+[F]+[F]+[F]+[F]+[F], p_1 = F[++F] [-FF]FF[+F][-F]FF, n = 4 \rangle$

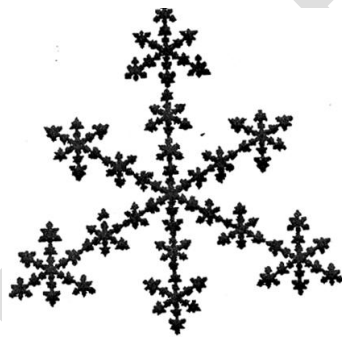


Рис.5.4. Стадия построения снежинки.

Приложение А. Порядок выполнения лабораторных работ

Практические занятия по курсу состоят из тематических работ, включающих одну или несколько лабораторных работ.

Лабораторная работа №1-3

Спроектировать автоматную грамматику по заданному языку L , построить конечный автомат.

1. Изучить классификацию Хомского (см. раздел 1.1., 1.2., 1.3.). Ответьте на вопрос, какие грамматики называются автоматными. Какие есть виды автоматных грамматик.

2. Спроектировать по заданному языку L автоматную грамматику и конечный автомат. Используйте пример, и последовательность выполнения работы из раздела 2.3. “Практическая работа 1”.

1. Постановка задачи.
2. Входные и выходные данные.
3. Спроектировать грамматику (Лаб 1).
4. Определить свойства грамматики.

5. Спроектировать конечный автомат, составить диаграмму переходов КА и реализовать на C# (Лаб 2.).

5.1. Создать проект – консольное приложение:

5.2. Используйте следующие фрагменты программ для реализации КА, распознающего заданный язык.

Пример 1. Создайте проекта простого консольного приложения.

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;

namespace ConsoleApplication {
    class Program {
        static void Main(string[] args) {
            ArrayList Q = new ArrayList();
            string str;
            Console.WriteLine("Hello");
            str = Console.ReadLine();
            Console.WriteLine("is= "+str);
            Console.ReadLine();
        }
    }
}
```

```
}
```

Пример 2. Разбор входной строки по символам.

Пример 3. Проектирование и реализация правил.

```
    Delta delta = null;
// 1. transition
    delta = new Delta("S0", "0", new ArrayList() {"A","qf"});
    DeltaList.Add(delta);
// 2. transition
    delta = new Delta("A", "1", new ArrayList() {"B"});
    DeltaList.Add(delta);
// 3. transition
    delta = new Delta("B", "0", new ArrayList() {"A","qf"});
    DeltaList.Add(delta);

class Delta { // структура Delta правил переписывания
    private string LeftNoTerm = null;
    private string LeftTerm = null;
    private ArrayList Right = null;
    public string leftNoTerm { get { return LeftNoTerm; } set { LeftNoTerm = value; } }
    public string leftTerm { get { return LeftTerm; } set { LeftTerm = value; } }
    public ArrayList right { get { return Right; } set { Right = value; } }

    // модель правила
    // delta( A, 1) = {qf}
    // LeftNoTerm LeftTerm Right
    public Delta(string LeftNoTerm, string LeftTerm, ArrayList Right) {
        this.LeftNoTerm = LeftNoTerm;
        this.LeftTerm = LeftTerm;
        this.Right = Right;
    }

    public void DebugDeltaRight() {
        int i = 0;
        for (; i < this.right.Count; i++) {
            if (i == 0)
                System.Console.Write("(" + this.right[i]);
            else
                System.Console.Write(", " + this.right[i]);
        }
        if (i == 0)
```

```

        System.Console.WriteLine();
    else
        System.Console.WriteLine(" ");
    }
} // end class Delta

```

Пример 4. Определение автомата как объекта и тестирование.

```

class Automate { // NDKA (Q,Sigma,deltaList,q0,F)
    ArrayList Q      = null;    // множество всех состояний
    ArrayList Sigma  = null;    // конечный алфавит входных символов
    ArrayList DeltaList = new ArrayList(); // множество всех правил
    string Q0        = null;    // начальное состояние
    ArrayList F       = null;    // заключительные состояния
    // атрибутное программирование на C#
    public ArrayList q { get { return Q; } set { Q = value; } }
    public ArrayList sigma { get { return Sigma; } set { Sigma = value; } }
    public ArrayList deltaList { get { return DeltaList; }
                                set { DeltaList = value; } }
    public string q0 { get { return Q0; } set { Q0 = value; } }
    public ArrayList f { get { return F; } set { F = value; } }

    public Automate() {
        this.Q      = new ArrayList();
        this.Sigma  = new ArrayList();
        this.DeltaList = new ArrayList();
        this.F       = new ArrayList();
    }

    public Automate(string aname) {
        System.Console.WriteLine(aname);
        // сделать диалог, инициализирующий NDKA
        // альтернативные состояния переходов хранить в массиве см. Test
        //init();
        Test();
    }

    public void Test() { // задание правил для тестирования
        Q = new ArrayList() { "S0", "A", "B", "qf" }; // "C" для отладки
        Sigma = new ArrayList() { "0", "1" };
        q0 = "S0";
        F = new ArrayList() { "qf" };
        ... задать правила
    } // end test
}

```

Пример 5. Считывание символа и выбор правила:

```
foreach (Delta d in this.DeltaList) {  
    if (d.leftNoTerm == q && d.leftTerm == chain.Substring(i,1) ) {  
        ...  
    }  
    ...  
}
```

6. Определить свойства КА. Реализовать алгоритм преобразования НДКА в ДКА (Лаб 3.).

Пример 6. Построение побитового кода булеана для множества мощности n :

Для построения Булеана вначале строим все побитовые комбинации от 0 до $2^n - 1$, где n – мощность множества. Затем побитовая комбинация преобразуется в символы. Например, бинарная последовательность для множества для множества из символов {SAB}, $n = 3$ приведена в таблице. "1" означает, что символ в подмножестве, "0" – его отсутствие:

Пример 7. Построение по побитовому коду дельта правила:

Реализация алгоритма преобразования на C#:

```
// NDKA (Q,S,Delta,q0,F)  
namespace NDKA2DKA {  
    class Program {  
        static void Main(string[] args) {  
            Automate NDKA = new Automate("NDKA");  
            NDKA.Debug();  
            Converter converter = new Converter();  
            Automate DKA = converter.convert(NDKA);  
            DKA.Debug();  
            DKA.recognize(DKA.q0, "01010", 0);  
        }  
    }  
  
    //  
    class Converter {  
        Automate DKA = null;  
        // множество всех правил deltaListAll  
        ArrayList deltaListAll = null; // all transitions  
        // подмножества, которые содержат все заключительные  
        // состояния qf, то есть F'  
        ArrayList FAll = null;  
  
        public Converter() {}  
    }  
}
```

```

public Automate convert(Automate NDKA) {
    // инициализировать данные для каждого вызова convert
    this.DKA = new Automate();
    this.deltaListAll = new ArrayList();
    this.FAll = new ArrayList();

    // Шаг 1. Init q0 & sigma
    DKA.q0 = NDKA.q0;
    DKA.sigma = NDKA.sigma;

    //2. Создать множество всех подмножеств по Q (булеан) и
    //3. Создать множество всех правил DeltaList
    // 4 и 5, подмножества, которые содержат заключительные
    // состояния qf, то есть F'
    BuildDeltaList(NDKA);
    // Шаг 6. Определить достижимые состояния,
    // исключить недостижимые состояния из множества Q'
    // 1. Берем начальное состояние и определяем правило дельта,
    Reachability(DKA.q0);
    return DKA;
}

void BuildDeltaList(Automate NDKA) {
    ArrayList right = null; // для нового правила
    int count = NDKA.q.Count;
    // Time Complexity:  $O(n^2 \cdot n)$ , Space Complexity:  $O(1)$ 
    // 1. set size of power set of a set with set size n is  $(2^{*}n)$ 
    int sizeOfPowerSet = (int)Math.Pow(2, count);
    string leftNoTerm = null; // is subset
    string[] noTerm = null; // для split
    // 2. Run from counter 000..0 to 111..1
    Console.WriteLine("Boolean _____");
    for (int counter = 0; counter < sizeOfPowerSet; counter++) {
        leftNoTerm = null;
        for (int j = 0; j < count; j++) {
            // Check if j-th bit in the counter is set If set then build set, use comma
            // Console.WriteLine("! 0x{0:x8}", 1 << j);
            if ((counter & 1 << j) != 0) {
                // System.Console.WriteLine("NDKA.q[j] = " + NDKA.q[j]);
                if (leftNoTerm != null) leftNoTerm = leftNoTerm + ',' + NDKA.q[j];
                else leftNoTerm = "" + NDKA.q[j];
            }
        }
        if (leftNoTerm != null) { //  $2^{*}n - 1$  без пустых подмножеств
            // Найти delta'(S, a)

```

```

noTerm = leftNoTerm.Split(',');
// Шаг 4. построить subset F
BuildFall(leftNoTerm, NDKA.f);
foreach (string leftTerm in NDKA.sigma){
// по deltaListNDKA посмотреть имеющиеся правила для данного, одно
    foreach (string n in noTerm) { // ищем правило
        right = findTransition(n, leftTerm, NDKA.deltaList);
        if (right != null) {
deltaListAll.Add(new Delta(leftNoTerm, leftTerm, new ArrayList(right)));
            break;
        }
    }
}
System.Console.WriteLine("    "+leftNoTerm); // булеан
}
} // end for
DebugDeltaList(deltaListAll);
DebugF(Fall);
} // BuildDeltaList

// найти переход
public ArrayList findTransition (string leftNoTerm, string leftTerm,
                                ArrayList NDKAdeltaList) {
    foreach (Delta d in NDKAdeltaList) { // найдено правило в ArrayList
        if (d.leftNoTerm == leftNoTerm && d.leftTerm == leftTerm)
            return d.right;
    }
    return null;
}

void BuildFall(string leftNoTerm, ArrayList qf) {
// если в подмножестве noTerm есть заключительное состояние NDKA.f
    string[] noTerm = leftNoTerm.Split(',');
    foreach (string n in noTerm) { // ищем правило
        foreach (string f in qf) {
            if (n == f) {
                Fall.Add(leftNoTerm);
// System.Console.WriteLine(" Fall.Add = " + leftNoTerm);
                return;
            }
        }
    }
} // end BuildFall

void Reachability(string q) {

```

```

// 1. Берем состояние по правилу дельта и определяем следующее дельта
string right = null;
foreach (Delta d in deltaListAll) {
    if (d.leftNoTerm == q) {
        // преобразовать в метку подмножество из right
        d.right = markSubset(d.right);
        DKA.deltaList.Add(d);
        DKA.q.Add(q);
        // всегда один элемент, так как markSubset
        right = d.right[0].ToString();
        break;
    }
}
if (right == null) return; // нет достижимых состояний
if (DKA.q.Contains(right)) { // это состояние уже было, останов
    // заключительное состояние должно быть F'
    if (F.all.Contains(right))
        // в F' оставить последнее достижимое состояние
        DKA.f.Add(right);
    else {
        System.Console.WriteLine(" Reachability error " + 01);
        return;
    }
}
else Reachability(right);
} // end Reachability

ArrayList markSubset(ArrayList right) {
    string r = null;
    foreach (string s in right) {
        if (r != null) r = r + ',' + s;
        else r = s;
    }
    return new ArrayList(){r};
}

void DebugF(ArrayList F) {
    System.Console.WriteLine(" F all: _ ");
    for (int i = 0; i < F.Count; i++) {
        System.Console.WriteLine("      "+F[i]);
    }
}

public void DebugDeltaList(ArrayList deltaList) {
    System.Console.WriteLine("deltaList all: _ ");
}

```



```

foreach (Delta d in deltaList) {
    Console.WriteLine("      (" + d.leftNoTerm + ")," + d.leftTerm + " = ");
    d.DebugDeltaRight();
}
}
} // end class Convertor

```

7. Оформить работу согласно указанным шагам.

Лабораторная работа №4-6

Привести заданную КС-грамматику $G = (T, V, P, S)$ к приведенной форме.

1. Изучить алгоритмы приведения КС-грамматик к приведенной форме. (см. раздел 3.4.). Ответьте на вопрос, какие КС-грамматики называются грамматиками в приведенной форме. Лаб. 4 А,В. Лаб. 5 С,Д., Лаб. 6 F,Е.

2. Использовать алгоритмы преобразования КС-грамматик..

А). Устранить из грамматики G бесполезные символы. Применить алгоритм 3.3. к грамматике G .

В). Устранить из грамматики G ϵ -правила, применить алгоритм 3.5.

С). Устранить из КС грамматики G цепные правила, применить алгоритм 3.6.

Д). Устранить левую рекурсию в заданной КС-грамматике G_1 , порождающей скобочные арифметические выражения. Применить алгоритм 3.7. к грамматике G .

Ф). Определить в какой форме (Грейбах, Хомского) находится КС-грамматика G' .

Е). G' – приведенная КС-грамматика.

3. Оформить работу согласно шагам.

Лабораторная работа №7-8

Построить МП-автомат P и расширенный МП-автомат по КС-грамматике $G = (T, V, P, S)$, без левой рекурсии. Написать последовательность тактов автоматов для выделенной цепочки. Определить свойства автоматов. Лаб. 7. -2.1. Лаб. 8. -2.2.

1. Изучить алгоритмы построения МП-автомат P и расширенного МП-автомата по заданной КС-грамматике (см. раздел 3.4.). Ответьте на вопрос, чем отличается МП-автомат P от расширенного МП-автомата.

2. Выполнить построение согласно алгоритмам. Смотрите пример и последовательность выполнения работы из раздела 3.4. Практическая работа 4.

2.1. А). Построить МП-автомат по КС-грамматике G , используя алгоритм 3.8. Для моделирования магазина используйте Stack из библиотеки C#.

```
Stack <string> stack = new Stack <string> ();
```

```
stack.Push("c");
stack.Push("d");
```

```
Console.WriteLine(" simbol: " + stack.Pop());
```

В). Определить последовательность тактов МП-автомата для выделенной цепочк.

2.2. А). Построить расширенный МП-автомат, используя алгоритм 3.9.

В). Определить последовательность тактов расширенного МП-автомата при анализе входной выделенной цепочки Р.

3. *Определить свойства построенный МП-автоматов.*

4. *Оформить работу согласно шагам.*

Лабораторная работа №9-10

Разработать контекстно-свободную грамматику по заданной строке (см. раздел 3.3.). Алгоритм разбора реализуется в виде процедур, каждой, из которой соответствует диаграмма.

1. *Повторить классификацию Хомского* (см. раздел 1.1.). Ответьте на вопрос, какие грамматики называются контекстно-свободными. Какие есть виды контекстно-свободных грамматик.

2. *Спроектировать по заданной строке* контекстно-свободную грамматику и автомат с магазинной памятью. Используйте пример и последовательность выполнения работы из раздела 3.3. Обратите внимание, что в качестве неявного стека могут выступать: рекурсивная процедура и древовидная структура объектов. Лаб. 9 1-4. Лаб. 10 5-7.

1. Спроектировать контекстно-свободную грамматику.

2. Записать вывод заданной строки по грамматики.

3. Определить свойства грамматики.

4. Устранить левую рекурсию и записать вывод заданной строки по грамматики.

5. Оптимизировать грамматику метасимволами {...} и записать вывод заданной строки по грамматики.

6. Составить синтаксический граф для грамматики.

7. Преобразовать граф в программу.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
```

```
class State { // абстрактный класс с чисто виртуальной функцией
public:
    virtual void parse(char c)=0;
};
//подкласс для объекта с состоянием правильного разбора
class OK:public State {
```

```

    public:
        OK () {}
        virtual void parse(char c){cout << "OK" << endl;}
};
//подкласс для объекта с состоянием не правильного разбора
class ERROR:public State {
    public:
        ERROR() {}
        virtual void parse(char c){cout << "ERROR string"<< endl;}
};
//класс для автомата агрегация по ссылке объектов классов OK,
ERROR
class Automate {
    public:
        static State* state; // полиморфная переменная
        static ERROR* error;
        static OK* ok;

        Automate(string String){
            i=0;
            this->String = String;
        }
        char getNextChar(){// получить следующий символ из строки
            if (i<(int)String.length()){char ch=String[i];i++;return ch;}
            else {return ' ';}
            //при окончании строки возвращается пробел
        }
        virtual void parse(){ // начать разбор
            while ( (state !=error)&&(state !=ok) ){
                state->parse(getNextChar());
            }
            state->parse(getNextChar());// принцип подстановки
        }
    private:
        string String;
        int i;
}; // end class
// присвоение начальных значений переменным автомата
State* Automate::state = NULL;
ERROR* Automate::error = new ERROR;
OK* Automate::ok = new OK;

// определение подкласса для объекта автомата, анализирующего
// контекстно-свободную грамматику
class AutomateCF: public Automate {
    public:
        AutomateCF(string String):Automate(String) {c = ' ';}
        virtual void parse(){// замещение функции parse в объекте
            // класса Automate
            c = getNextChar(); // вызов функции класса автомат
            E(); // вызов метода
            state->parse(getNextChar());
        }
};

```

```

void T () { // реализация функции по диаграмме
    cout << "step1 T=" << c << endl;
    if ( (c == 'a' || c == 'b' || c == 'c')) {
        c = getNextChar();
        cout << "step2 T=" << c << endl;
    }
    if (c == ')') {c=getNextChar();}
    else if (c == '(') {c=getNextChar(); E();}
}

void E () { // реализация функции по диаграмме
    cout << "step0 E=" << c << endl;
    T(); // вызов функции
    cout << "step1 E=" << c << endl;
    while ( (c == '+' || c == '-') ) {
        c = getNextChar(); T(); // вызов функции
        cout << "step2 E=" << c << endl;
    }
    if (c == ' ') {state=ok;}
    else {state=error;}
}
private:
    char c;
};
// программа ввода строки с клавиатуры
string getString () {
    //string String = "c+d-dkf-n";
    string String = "a+(b-c)";
    // string String = "a+b";
    char c;
    int N = (int)String.length();
    cout << "Enter string " << N << " char " << endl;
    for (int i=0; i < N; i++) {
        cin >> c;
        String[i] = c;
    }
    return String;
}

int main() {
    //создание объекта автомат
    Automate * a = new AutomateCF(getString());
    //инициализация начальных значений
    a->state = Automate::ok;
    a->parse(); //синтаксический анализ
    return 0;
}

```

Обратите внимание, что в переменную “с”, записывается следующий символ, который доступен из функций T() и E(). В функциях указаны трассировочные шаги выполнения, которые необходимо сохранить в программе. Состояния OK и ERROR автомата используются для идентификации выполнения разбора.

3. Оформить работу согласно шагам.

Лабораторная работа №11-10

Реализовать контекстно-свободную грамматику, полученную в работе 5 на основе, таблично-управляемой программы грамматического разбора. Изучить контекстно-зависимые грамматики (см. раздел 1.4.).

1. Повторить классификацию Хомского (см. раздел 1.1.). Ответьте на вопрос, какие грамматики называются контекстно – зависимыми, неограниченными. Какие свойства контекстно – зависимых и неограниченных грамматик, в чем их отличие. Чем они отличаются от изученных ранее грамматик ?

2. Разработать объектно-ориентированную реализацию для таблично-управляемой программы грамматического разбора.

1. Представить граф грамматики в виде структуры данных (см. раздел 3.3.).

2. Классифицировать типы вершин.

3. Выполнить подстановку графов и получить как можно меньшее число графов.

4. Последовательность вершин графа преобразовать в структуру узлов.

5. Реализовать программу разбора по структуре узлов.

В алгоритме разбора, результат каждого шага разбора выводится на экран, чтобы можно было видеть, как происходит разбор.

```
#include "stdafx.h"
#using <mscorlib.dll>

using namespace System;
using namespace std;
class State { // абстрактный класс с чисто виртуальной функцией
public:
    virtual void parse(char c)=0;
};
// класс для объектов вершин, агрегация по указателю
class Node {
public:
    Node (char c){ // задание символа вершинам
        this->alt = NULL;
        this->suc = NULL;
        this->sym = NULL;
        this->c = c;
    }
    // метод для соединения вершин
    void link (Node *alt, Node *suc, Node *sym){
        this->alt = alt;
        this->suc = suc;
        this->sym = sym;
    }
};
```

```

    }

    Node * sym; // sym != NULL
    Node * suc; // terminal empty !=' ' && sym == NULL
    Node * alt;
    char c;      // empty=' ' && sym == NULL
};

//подкласс для объекта с состоянием правильного разбора
class OK:public State {
public:
    OK () {}
    virtual void parse(char c){cout << "OK" << endl;}
};
//подкласс для объекта с состоянием не правильного разбора
class ERROR:public State {
public:
    ERROR () {}
    virtual void parse(char c){cout << "ERROR string"<< endl;}
};

//класс для автомата агрегация по ссылке объектов классов OK,
// ERROR
class Automate {
public:
    static State* state; // полиморфная переменная
    static ERROR* error;
    static OK* ok;

    Automate(string String){
        i=0;
        this->String = String;
    }
    char getNextChar(){// получить следующий символ из строки
if (i<(int)String.length()){char ch=String[i]; i++; return ch;}
    else {return ' ';}
    //при окончании строки возвращается пробел
    }
    virtual void parse(){// начать разбор
        while ( (state !=error)&&(state !=ok) ){
            state->parse(getNextChar()); // принцип подстановки
        }
        state->parse(getNextChar());
    }
private:
    string String;
    int i;
}; // end class

// присвоение начальных значений переменным автомата
State* Automate::state = NULL;
ERROR* Automate::error = new ERROR;
OK* Automate::ok = new OK;

```

```

// определение подкласса для объекта автомата, анализирующего
// контекстно-свободную грамматику
class AutomateCF: public Automate {
public:
    AutomateCF(string String, Node &node):Automate(String) {
        this->node = &node; // передача начальной вершины
        c = ' ';
    }

    void parse(Node * nd, bool b){
        Node *p = nd->alt; // Node head
        do {
            if (p->sym == NULL){ // терминал или пусто "empty"
                if (p->c == c){
                    cout<< "step01 c=" << c<< ": p->c ="<<p->c<<endl;
                    b=true; c=getNextChar();
                    state=ok;
                    // конец строки
                }
                if (c == ' ') {cout<< "step012 c="<<endl; return;}
                } else if (p->c == ' ') { // "empty"
                    cout<< "step02 if c=" << c<< ": p->c ="<<p->c<<endl;
                    b=true;
                } else { b=false; state=error;
                    cout<< "step03 c=" << c<< ": p->c ="<<p->c<<endl;
                }
            }
            else if (p->sym != NULL){ //- nil
                cout<< "step04 c=" << c<< "p->sym ="<<p->sym<<endl;
                parse(p->sym, b);
            }
            if (b) {
                cout<< "step05 ok: next c=" << c<< endl;
                p=p->suc;
            }
            else {p=p->alt;}
        } while (p != NULL);
    } // end parse

    virtual void parse(){ // замещение функции parse в объекте
        c = getNextChar();
        parse(this->node, true);
        state->parse(getNextChar());
    }
private:
    Node * node;
    char c;
};

// программа ввода строки с клавиатуры
string getString (){ // различные варианты для тестирования..
    // string String = "c+d-dkf-n";
    // string String = "a+(b-cba)-c";

```

```

// string  String = "a+b";
// string  String = "a+b-c-c+a";
// string  String = "a+gfg+fgfg"; // error
    string  String = "a+(b-c)";
char c;
int N = (int)String.length();
cout << "Enter string "<<N<<" char "<< endl;
for (int i=0; i < N;i++){
    cin>>c;
    String[i] = c;
}
return String;
}

int main(){
    // Построение графа
    // 1. Объявление вершин
    Node E = Node(' ');
    Node braceL = Node('(');
    Node nil = Node(' ');
    Node braceR = Node(')');
    Node c_a = Node('a');
    Node c_b = Node('b');
    Node c_c = Node('c');
    Node emptyR = Node(' ');
    Node plus = Node('+');
    Node minus = Node('-');
    Node nil_1 = Node(' ');

    // 2. Соединение вершин в дерево методом link
    // void link (Node * alt, Node * suc, Node * sym)
    E.link(&braceL,NULL,NULL);
    braceL.link(&c_a,&nil,NULL);
    nil.link(NULL,&braceR,&E);
    braceR.link(NULL,&plus,NULL);
    c_a.link(&c_b,NULL,NULL);
    c_b.link(&c_c,NULL,NULL);
    c_c.link(&emptyR,NULL,NULL);
    emptyR.link(NULL,&plus,NULL);
    plus.link(&minus,&nil_1,NULL);
    minus.link(NULL,&nil_1,NULL);
    nil_1.link(NULL,NULL,&E);

    //создание объекта автомат
    Automate * a = new AutomateCF(getString(),E);
    //инициализация начальных значений
    a->state = Automate::ok;
    a->parse(); //синтаксический анализ

    return 0;
}

```

6. Оформить работу согласно шагам.

Лабораторная работа №12-13.

Построить управляющую таблицу М для LL(k)-грамматики, написать правило вывода, определить является ли G грамматика *сильно* LL(k)-грамматикой (см. раздел 3.4.).

1. Изучить раздел 3.4. Построение управляющей таблицы М для грамматики $G = (T, V, P, S)$, работу алгоритма для определенной цепочки и определение *сильно* LL(k)-грамматики.

Пример построения управляющей таблицы.

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;

namespace lab {
    class Program {
        static void Main(string[] args) {
            MTable mTable = new MTable();
            // 1 row
            Row row = new Row();
            row.addItem("TE", 1);
            row.addItem("TE", 1);
            row.addItem("");
            row.addItem("");
            row.addItem("");
            row.addItem("");
            mTable.setRow(row);
            row = new Row();
            row.addItem("");
            row.addItem("");
            row.addItem("E, 3");
            row.addItem("+TE", 2);
            row.addItem("");
            row.addItem("E, 3");
            mTable.setRow(row);
            //      row.DebugRow();
            mTable.DebugTable();
        }
    }
} //конец для namespace

class MTable {
    ArrayList M = new ArrayList(); //
```

```

        public void setRow(Row row) { M.Add(row); }
        public void DebugTable() {
            //foreach (row r in M) {
                for(int i=0;i<M.Count;i++){
                    ((Row) M[i]).DebugRow();
                }
            //Console.WriteLine(r);
        }
    }

    class Row { // строка таблицы любого размера из
string - это колонка
        ArrayList row = new ArrayList(); //
        public void addItem (string item) {
            row.Add(item);
        }
        // public ArrayList getRow() { return row; }
        public void DebugRow() {
            string[] str = null;
            foreach (string r in row) {
                Console.WriteLine(r);

                str = r.Split(',');
                foreach (string l in str) {
                    Console.WriteLine(l);
                }
            }
        }
    }
}

```

2. Определить размеры управляющей таблицы M. В соответствии с шагами алгоритм 3.10 построить управляющую таблицу M (см. раздел 3.3. “Практическая работа 7.”)

3. Выделить цепочку, принадлежащую языку порождаемому грамматикой G и написать правила вывода для цепочки.

4. Для LL(k)-грамматики G определить является ли она *сильно* LL(k)-грамматикой.

5. *Оформить работу согласно шагам.*

Лабораторная работа №14-15

Построить управляющую таблицу M для LR(k)-грамматики, написать правило вывода выделенной строки (см. раздел 3.4.). Описать работу алгоритма LR(k) анализатора.

1. Изучить раздел 3.4. Построение управляющей таблицы M для LR(k)-грамматики $G = (T, V, P, S)$.

2. В соответствии с шагами алгоритм 3.12 построить управляющую таблицу М (см. раздел 1.4. “Практическая работа 8.”)

3. Выделить цепочку, принадлежащую языку порождаемому грамматикой G и написать правила вывода для цепочки.

4. Для выделенной цепочки показать работу LR(k)-анализатора в соответствии с шагами алгоритм 3.11.

5. *Оформить работу согласно указанным пунктам и используемым шагам алгоритмов.*

Лабораторная работа №16

Применить алгоритм типа “перенос-свертка” для заданной грамматики $G=(T, V, P, S)$. Описать работу алгоритма.

1. Изучить алгоритм типа “перенос-свертка”

2. В соответствии с шагами алгоритм 3.13 по шагам рассмотреть работу алгоритма (см. раздел 3.4. “Практическая работа 8.”)

3. Выделить цепочку, принадлежащую языку порождаемому грамматикой G и написать правила вывода для цепочки.

4. Для выделенной цепочки показать работу LR(k)-анализатора в соответствии с шагами алгоритм 3.13 .

5. *Оформить работу согласно указанным пунктам и используемым шагам алгоритма.*

Приложение В. Задания к лабораторным работам

Для лабораторных работ 1-3 определены варианты автоматных языков:

1. $L=\{0\omega_1+(01)^* \mid \omega_1 \in \{0,1\}^*\}$
2. $L=\{01-(10)^*+\omega_1 01 \mid \omega_1 \in \{0,1\}^*\}$
3. $L=\{0(00)^*+01\omega_1 \mid \omega_1 \in (0,1,2)^*\}$
4. $L=\{\omega_1\omega_2 1 \mid \omega_1 \in \{1,0\}^+, \omega_2 \in \{1,0\}^+\}$
5. $L=\{1\omega_1 1-(00)^* \mid \omega_1 \in \{1,0\}^*\}$
6. $L=\{\omega_1-0\omega_2-0+\omega_3 \mid \omega_1 \in \{0,1\}^+, \omega_2 \in \{0,1\}^+, \omega_3 \in \{0,1\}^+\}$
7. $L=\{(0+1)(01)^*+\omega_1 \mid \omega_1 \in \{0,1\}^+\}$
8. $L=\{(0+1)^*\omega_1\omega_2 \mid \omega_1 \in \{0,1,2\}^+, \omega_2 \in \{0,1\}^+\}$
9. $L=\{(01)^*-1-(01)^*+\omega_1 \mid \omega_1 \in \{0,1\}^+\}$
10. $L=\{1(01)^* 0-1+\omega_1 \mid \omega_1 \in \{0,1\}^+\}$
11. $L=\{(0+1)+\omega_1+(01)^* 0 \mid \omega_1 \in \{0,1\}^+\}$
12. $L=\{0(000)^*(0+1)\omega_1 \mid \omega_1 \in \{0,1\}^*\}$
13. $L=\{1(01)^*(01)\omega_1 \mid \omega_1 \in \{0,1\}^*\}$
14. $L=\{00\omega_1+(1)^* \mid \omega_1 \in \{0,1\}^*\}$
15. $L=\{0(01)^*+1\omega_1 0 \mid \omega_1 \in \{1,0\}^+\}$
16. $L=\{1\omega_1 1\omega_2 1 \mid \omega_1 \in \{0,1\}^+, \omega_2 \in \{0,1\}^+\}$
17. $L=\{011\omega_1 1(0)^* \mid \omega_1 \in (0,1)^+\}$
18. $L=\{\omega_1 0-\omega_2 \mid \omega_1 \in \{0,1\}^+, \omega_2 \in \{0,1\}^+\}$
19. $L=\{\omega_1(1)^* 0\omega_2 \mid \omega_1 \in \{1,2\}^*, \omega_2 \in \{0,1\}^*\}$

20. $L = \{10 + \omega_1(10)^* \omega_2 \mid \omega_1 \in \{1, 2\}^*, \omega_2 \in \{0, 1\}^*\}$
 21. $L = \{10\omega_1 0 - 1\omega_2 \mid \omega_1 \in \{0, 1\}^+, \omega_2 \in \{1, 2\}^*\}$
 22. $L = \{1 - 1\omega_1 01\omega_2 \mid \omega_1 \in \{1, 2\}^+, \omega_2 \in \{1, 2\}^*\}$

Лабораторные работы 3-6

А). Устранить из грамматики $G = (T, V, P, S)$ бесполезные символы, где

1. $P = \{S \rightarrow b, S \rightarrow F, S \rightarrow cFB, A \rightarrow Ab, A \rightarrow c, B \rightarrow cB, F \rightarrow Ca, C \rightarrow d\}$
2. $P = \{S \rightarrow b, S \rightarrow cAB, A \rightarrow Ab, A \rightarrow c, B \rightarrow cB, C \rightarrow Ca, F \rightarrow d\}$
3. $P = \{S \rightarrow b, S \rightarrow BA, A \rightarrow Ab, A \rightarrow c, B \rightarrow cB, F \rightarrow Ca, C \rightarrow d\}$
4. $P = \{S \rightarrow cB, B \rightarrow cB, B \rightarrow cA, A \rightarrow Ab, C \rightarrow Ca, F \rightarrow d\}$
5. $P = \{S \rightarrow c, F \rightarrow A, S \rightarrow cAB, A \rightarrow Ab, A \rightarrow c, B \rightarrow cB, C \rightarrow Ca, C \rightarrow d\}$
6. $P = \{S \rightarrow cFCB, A \rightarrow ACb, A \rightarrow cC, B \rightarrow cB, C \rightarrow Ca, F \rightarrow d\}$
7. $P = \{S \rightarrow b, S \rightarrow CF, S \rightarrow cCB, A \rightarrow Ab, A \rightarrow c, B \rightarrow cB, C \rightarrow Ca, F \rightarrow d\}$

В). Устранить из грамматики $G = (T, V, P, S)$ ϵ -правила, где

1. $P = \{S \rightarrow AB, A \rightarrow SA, A \rightarrow BB, A \rightarrow bB, A \rightarrow c, B \rightarrow c, B \rightarrow \epsilon\}$
2. $P = \{S \rightarrow b, S \rightarrow cAB, A \rightarrow Ab, A \rightarrow c, B \rightarrow cB, B \rightarrow \epsilon\}$
3. $P = \{S \rightarrow bA, S \rightarrow bA, A \rightarrow Ab, A \rightarrow \epsilon, B \rightarrow cB, B \rightarrow \epsilon\}$
4. $P = \{S \rightarrow cB, B \rightarrow cB, B \rightarrow cA, A \rightarrow ACb, A \rightarrow \epsilon, C \rightarrow Ca, C \rightarrow \epsilon\}$
5. $P = \{S \rightarrow c, S \rightarrow cAB, S \rightarrow \epsilon, A \rightarrow c, B \rightarrow cB, B \rightarrow \epsilon\}$
6. $P = \{S \rightarrow cFCB, A \rightarrow ACb, A \rightarrow cC, B \rightarrow cB, B \rightarrow \epsilon, C \rightarrow Ca, F \rightarrow \epsilon\}$
7. $P = \{S \rightarrow b, S \rightarrow C, S \rightarrow cCB, A \rightarrow Ab, A \rightarrow c, B \rightarrow cB, C \rightarrow Ca, C \rightarrow \epsilon\}$

С). Устранить из КС грамматики G цепные правила, где

1. $P = \{S \rightarrow AB, A \rightarrow S, A \rightarrow B, A \rightarrow bB, A \rightarrow c, B \rightarrow c\}$
2. $P = \{S \rightarrow b, S \rightarrow cAB, A \rightarrow Ab, A \rightarrow B, B \rightarrow cB, B \rightarrow b\}$
3. $P = \{S \rightarrow bA, S \rightarrow bA, A \rightarrow Ab, A \rightarrow S, B \rightarrow cB, B \rightarrow c\}$
4. $P = \{S \rightarrow cB, B \rightarrow cB, B \rightarrow cA, A \rightarrow C, A \rightarrow aB, C \rightarrow Ca, C \rightarrow cf\}$
5. $P = \{S \rightarrow c, S \rightarrow cAB, S \rightarrow a, A \rightarrow B, B \rightarrow cB, B \rightarrow f\}$
6. $P = \{S \rightarrow cFCB, A \rightarrow ACb, A \rightarrow C, B \rightarrow cB, B \rightarrow b, C \rightarrow Ca, F \rightarrow c\}$
7. $P = \{S \rightarrow b, S \rightarrow C, S \rightarrow cCB, A \rightarrow Ab, A \rightarrow C, B \rightarrow cB, C \rightarrow Ca, C \rightarrow b\}$

Д). Исключить левую рекурсию из КС – грамматики G , где

1. $P = \{S \rightarrow Bc, S \rightarrow Ad, A \rightarrow Sa, A \rightarrow AbB, A \rightarrow c, B \rightarrow Sc, B \rightarrow b\}$
2. $P = \{S \rightarrow FA, S \rightarrow c, A \rightarrow FS, A \rightarrow Sa, B \rightarrow SB, B \rightarrow b, F \rightarrow f\}$
3. $P = \{S \rightarrow Bb, B \rightarrow Sa, B \rightarrow cB, B \rightarrow Ac, A \rightarrow cSB, A \rightarrow a\}$
4. $P = \{S \rightarrow Ba, S \rightarrow Ab, A \rightarrow Sa, A \rightarrow AAb, A \rightarrow c, B \rightarrow Sb, B \rightarrow b\}$
5. $P = \{S \rightarrow ScB, S \rightarrow cAB, S \rightarrow c, A \rightarrow AbB, B \rightarrow b, B \rightarrow aA\}$
6. $P = \{S \rightarrow cFCB, A \rightarrow ACb, B \rightarrow cB, B \rightarrow b, C \rightarrow Ca, C \rightarrow c, F \rightarrow f\}$
7. $P = \{S \rightarrow AB, S \rightarrow SC, A \rightarrow BB, A \rightarrow Ab, A \rightarrow a, B \rightarrow b, C \rightarrow Ca, C \rightarrow b\}$

Лабораторные работы 3-8 задана КС-грамматика $G = (T, V, P, S)$, где

1. $T = \{i, =, *, (,)\}, V = \{S, F, L\}, P = \{S \rightarrow F = L, S \rightarrow L, F \rightarrow (* L), F \rightarrow i, L \rightarrow F\}$
2. $T = \{i, \&, *, (,)\}, V = \{S, F, L\}, P = \{S \rightarrow F \& L, S \rightarrow (S), F \rightarrow * L, F \rightarrow i, L \rightarrow F\}$
3. $T = \{i, ^, -, (,)\}, V = \{S, F, L\}, P = \{S \rightarrow (F \wedge L), F \rightarrow - L, F \rightarrow i, L \rightarrow F\}$

4. $T = \{i, *, -, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F)^* L, F \rightarrow - L, F \rightarrow i, L \rightarrow F\}$
5. $T = \{i, +, -, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F)+(L), F \rightarrow - L, F \rightarrow i, L \rightarrow F\}$
6. $T = \{i, +, -, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F)+(L), F \rightarrow - L, F \rightarrow i, L \rightarrow F\}$
7. $T = \{i, +, -, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F+L), S \rightarrow (F), F \rightarrow - L, F \rightarrow i, L \rightarrow F\}$
8. $T = \{i, \&, ^, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow F^{\wedge} L, S \rightarrow (F), F \rightarrow \&L, F \rightarrow i, L \rightarrow F\}$
9. $T = \{i, \&, ^, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow F^{\wedge} L, S \rightarrow (S), F \rightarrow \&L, F \rightarrow i, L \rightarrow F\}$
10. $T = \{i, \&, ^, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F^{\wedge} L), S \rightarrow (S), F \rightarrow \&L, F \rightarrow i, L \rightarrow F\}$
11. $T = \{i, \&, ^, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F^{\wedge} L), F \rightarrow \&L, F \rightarrow i, L \rightarrow F\}$
12. $T = \{i, \&, ^, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow \&F^{\wedge}, S \rightarrow (L), F \rightarrow L, F \rightarrow i, L \rightarrow F\}$
13. $T = \{i, *, :, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow F: L, S \rightarrow (L), F \rightarrow L^*, F \rightarrow i, L \rightarrow F\}$
14. $T = \{i, *, :, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F: L), F \rightarrow L^*, F \rightarrow i, L \rightarrow F\}$
15. $T = \{i, *, :, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F: L), S \rightarrow (F), F \rightarrow L^*, F \rightarrow i, L \rightarrow F\}$
16. $T = \{i, *, +, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F+L), F \rightarrow L^*, F \rightarrow i, L \rightarrow F\}$
17. $T = \{i, *, +, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F+L), F \rightarrow (L^*), F \rightarrow i, L \rightarrow F\}$
18. $T = \{i, *, +, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow F+L, F \rightarrow (L^*), F \rightarrow i, L \rightarrow F\}$
19. $T = \{i, *, +, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow F+L, S \rightarrow (S), F \rightarrow L^*, F \rightarrow i, L \rightarrow F\}$
20. $T = \{i, @, \&, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow (F@L), S \rightarrow (F\&L), F \rightarrow i, L \rightarrow F\}$
21. $T = \{i, +, -, ()\}$, $V = \{S, F, L\}$, $P = \{S \rightarrow F+L, S \rightarrow (S), S \rightarrow (L-), F \rightarrow i, L \rightarrow F\}$

Лабораторные работы 14-16, правила G грамматики рассмотреть как правила LL(k) грамматики, а для работы 7 как правила LR(k).

Заключение

Представленный теоретический материал служит основой для практического проектирования грамматик по заданным языкам и эквивалентным грамматикам автоматов, для реализации синтаксических анализаторов на основе объектно-ориентированного подхода.

Лабораторные работы могут быть использованы для формирования курсовых и дипломных работ, а также исследовательских работ по теме теория автоматов и языков.

Библиографический список

1. Ахо А., Ульман Д. Теория синтаксического анализа перевода и компиляции // Пер. с англ. - М.: Мир, 1978.
2. Р.Хантер. Проектирование и конструирование компиляторов. Пер. с англ. - М.: Финансы и статистика, 1984.
3. Н.Вирт. Алгоритмы + структуры данных = программы. Пер. с англ. - М.: МИР, 1985.
4. С.Мооре. Dynamical Recognizers: Real-time Language Recognition by Analog Computers. Theoretical Computer Science **201**, 1998, pp. 99-136.
5. W.Tabor. Fractal Encoding of Context Free Grammars in Connectionist Networks. University of Connecticut Expert Systems 17(1), 2000, pp. 41-56
6. А.С.Семенов. Построение класса фрактальных систем по шаблону на примере дерева Фибоначчи // Изв.РАН. Информационные технологии и Вычислительные системы. - М.: N2, 2005 стр.10-17.

7. *А.С.Семенов* Класс фрактальных автоматов для распознавания КС-языков // Программирование (рукопись).

ЦЕРНОВИЖ