МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Курганский государственный университет»

Кафедра информационных технологий
и методики преподавания информатики

ОСНОВЫ ТЕОРИИ ФОРМАЛЬНЫХ ЯЗЫКОВ

Методические рекомендации
для студентов направления 09.03.03 «Прикладная информатика»
(направленность «Прикладная информатика в дизайне»)

Курган 2018

Кафедра: «Информационные технологии и методика преподавания информатики».

Дисциплина: «Основы теории формальных языков» (направление 09.03.03).

Составитель: канд. пед. наук, доцент А.А. Медведев.

Утверждены на заседании кафедры «29» сентября 2017 г.

Рекомендованы методическим советом университета

«12» декабря 2016 г.

Практическое занятие №1. Языки и грамматики

Цель занятия – ввести понятия языка и грамматики, определить виды грамматик, способы их задания.

1 Фрагмент теории

Теория формальных языков изучает методы задания, распознавания и обработки языков. Здесь вводятся базовые понятия теории: это понятие языка и понятие грамматики как конечной модели задания языка.

Конечное множество объектов будем называть словарем. Элементы словаря будем называть символами. Цепочкой (символов) над словарем будем называть произвольную конечную последовательность символов словаря. Символы будем далее обозначать латинскими буквами начала словаря, цепочки — малыми греческими буквами.

ПРИМЕР 1.1. Пусть V — словарь, содержащий три символа: a, b, c. Над словарем V можно построить различные цепочки, например, $\alpha = abbca$, $\beta = cca$, $\gamma = b$.

Подобно тому, как ноль играет важную роль в построении числовой системы, в теории языков важную роль играет пустая цепочка, т. е. цепочка, вообще не содержащая символов. Пустую цепочку не всегда удобно обозначать пустой позицией, для ее обозначения введен специальный символ ε.

Пусть V — словарь. Множество всех возможных цепочек, составленных из символов словаря V, включая и пустую цепочку, обозначим V^*. Если словарь не пуст, то число возможных цепочек над ним бесконечно, точнее, их количество счетно. Обозначим a^n цепочку из n символов a, причем $a^0 = \varepsilon$.

Определение 1.1. Языком над словарем V называется произвольное множество цепочек над этим словарем.

Иными словами, язык над словарем V — это произвольное подмножество множества V^*. Будем обозначать язык через L.

ПРИМЕР 1.2. Рассмотрим несколько примеров, показывающих применимость введенного определения к различным языкам.

1 $V1 = \{a, b\}$; $L1 = \{aabb, baa, aaba\}$. Здесь язык составляют три цепочки. Все остальные цепочки множества V^* не принадлежат языку $L1$.

2 $V2 = \{a, b\}$; $L2 = \{a^n b^n \mid n > 0\}$. Язык $L2$ — это бесконечное множество таких цепочек, которые начинаются символами a, заканчиваются символами b, а количества символов a и символов b в цепочках одинаковы.

3 $V3 = \{a, b, c\}$; $L3 = \{a^n b c^m \mid n, m > 0\}$. Очевидно, $aaabcce \in L3$, $cbbaa \notin L3$. Количества символов a в начале цепочек языка и символов c в конце цепочек могут не совпадать.

4 $V4 = \{a, b\}$; $L4 = \{\alpha \mid$ в цепочке α количества вхождений a и b равны$\}$. Очевидно, $aabbab \in L4$, $aab \notin L4$.

5 $V5 = \{ (,) \}$; $L5$ — множество правильных скобочных выражений.

6 $V6 = \{a, b, c\}$; $L6 = \{a^n b^n c^n \mid n > 0\}$. Очевидно, $aabbcc \in L6$, $abca \notin L6$.

7 $V7 = \{(,),\ +,\ -,\ *,\ /,\ i\}$; $L7$ — арифметические выражения, в которых операнды обозначаются символом i. Очевидно, $i+i/(i-i*i) \in L7$, $i+*i))(i* \notin L7$.

8 $V8 = \{a,\ b,\ c\}$; $L8 = \{wcw \mid w \in \{a,\ b\}^*\}$. Цепочки языка $L8$ — это две идентичные копии произвольной цепочки из символов a и b, разделенные маркером c.

9 $V9 = \{a,\ b,\ c\}$; $L8 = \{w_1cw_2 \mid w_1, w_2 \in \{a,\ b\}^*;\ w_1 \neq w_2\}$. Цепочки языка Z9 состоят из двух несовпадающих цепочек над словарем $\{a,\ b\}$, разделенных маркером c.

Определение языка как подмножества множества всех возможных цепочек над конечным словарем является слишком общим и неконструктивным. Такое определение удобно только в случае конечного языка, например, азбуки Морзе, цепочки которой составлены из двух символов — точек и тире, а число цепочек языка конечно. Сложность работы с языками состоит в том, что язык — это обычно бесконечное множество, а бесконечное множество невозможно задать простым перечислением его элементов.

Лишь в некоторых случаях бесконечный язык можно задать с помощью условий (предикатов), определенных на цепочках языка, как в случаях 2, 3, 6 примера 1.2. Во всех остальных случаях этого примера языки задавались совершенно не годными для строгого формального анализа средствами. Так, для случая 5 нужно заранее знать, что такое «правильное скобочное выражение». Суть проблемы состоит в том, что невозможно задать новый язык, пользуясь определением 1.1.

Описанная выше ситуация отражает общую трудность задания бесконечного множества объектов (в нашем случае бесконечного множества цепочек). Для задания такого множества простым перечислением требуются бесконечные ресурсы. Нужно суметь задать бесконечное множество цепочек с помощью какого-либо **конечного** механизма (алгоритма, устройства, исчисления, набора правил и т. д.). Такие механизмы и называются **грамматиками**.

Определение 1.2. Любой конечный механизм задания языка называется грамматикой.

Существуют два пути задания языка. Первый — это конечное множество правил порождения за конечное число шагов правильных цепочек, причем эти правила не позволяют построить никакую цепочку, не принадлежащую языку. Другой подход — это задание механизма распознавания, который, получив в качестве аргумента любую конечную цепочку над словарем V, за конечное число шагов дает ответ, принадлежит ли эта цепочка определяемому языку или нет.

В соответствии с этими двумя подходами существует и два типа грамматик: порождающие и распознающие. Под **порождающей** грамматикой языка L понимается конечный набор правил, позволяющий строить все «правильные» предложения языка L, и применение которых не дает ни одного «неправильного» предложения. Распознающая грамматика задает критерий принадлежности произвольной цепочки данному языку. Это фактически

некоторый алгоритм, принимающий в качестве входа символ за символом произвольную цепочку над словарем V и дающий на выходе один из двух возможных ответов: либо «данная цепочка принадлежит языку L», либо «данная цепочка не принадлежит языку L»**.** Фактически этот алгоритм должен разделить все возможные входные цепочки на два класса: один класс — принадлежащие языку L цепочки, а другой — цепочки, не принадлежащие языку L.

Порождающая и распознающая грамматики играют несколько разные, но взаимодополняющие роли. Порождающая грамматика удобна и естественна для задания, спецификации языка. Порождающий механизм не является алгоритмом, он определяет правила построения предложений языка. Распознающий механизм, напротив, обычно представляется алгоритмом, он важен для анализа, трансляции цепочек языка в некоторый выход.

1.1 Грамматики Хомского

В 1956 г. американский лингвист Ноам Хомский предложил модель порождающей грамматики, которая весьма удобна для задания искусственных языков. Особенность этой модели состоит в том, что каждой порождаемой цепочке языка эта модель позволяет сопоставить ее структуру.

Определение 1.3. Порождающей грамматикой Хомского G называется четверка объектов: $G = (T, N, P, S)$, где

- T — конечное непустое множество (терминальный словарь). Элементы множества T будем называть **терминальными** символами;
- N — конечное непустое множество (нетерминальный словарь). Элементы множества N будем называть **нетерминальными** символами; множества N и T не пересекаются;
- R — конечное непустое множество правил (продукций), каждое из которых имеет вид $\alpha \to \beta$, где и α, и β — это цепочки над объединенным словарем $T \cup N$, т. е. составлены как из терминальных, так и из нетерминальных символов;
- S — выделенный элемент нетерминального словаря, $S \in N$, так называемый **начальный символ**.

ПРИМЕР 1.3. Рассмотрим следующую грамматику Хомского: $G_0 = (\{a, b, c\}, \{S, A, B\}, S, R)$, где

$R = \{S \to aSbAc,$
$aS \to bbAc,$
$bAc \to B,$
$SbA \to \varepsilon,$
$B \to b\}.$

Терминальный словарь грамматики G_0 содержит три терминальных символа или терминала: a, b и c. Нетерминальный словарь содержит три нетерминальных символа (нетерминала): A, B и начальный нетерминал S. Множество R содержит пять продукций (правил). Для того чтобы различать в

цепочках терминалы от нетерминалов, обычно (если не указано противное) терминалы обозначают строчными латинскими буквами, а нетерминалы — прописными (большими) латинскими буквами. Цепочки символов, как было сказано выше, обычно обозначаются греческими буквами α, β, … . При таком условии, если указать начальный нетерминал грамматики (по умолчанию будем считать начальным нетерминал *S),* то вся грамматика может быть задана перечислением конечного множества правил. Так, грамматика G_0 может быть задана просто набором правил:

$G_0 = 1\ S \to aSbAc,$

$2\ aS \to bbAc,$

$3\ bAc \to B,$

$4\ SbA \to \varepsilon,$

$5\ B \to b.$

Нумерация правил приведена здесь только для удобства ссылок на них. По такому заданию всегда могут быть восстановлены как терминальный, так и нетерминальный словари. Рассмотрим теперь каким образом можно, используя правила такой порождающей грамматики, получать цепочки языка. Во-первых, рассмотрим, как из одних цепочек с помощью правил грамматики могут быть порождены другие цепочки. Это делается с помощью операции подстановки.

Определение 1.4. Из цепочки α непосредственно выводима цепочка β в грамматике G (операция непосредственной выводимости обозначается $\to$), если:

- цепочку α можно представить как конкатенацию трех цепочек, $\alpha = \mu\tau\nu$ (некоторые из этих цепочек могут быть пустыми);
- цепочку β также можно представить как конкатенацию трех цепочек: $\beta = \mu\xi\nu$;
- в грамматике G есть продукция $\tau \to \xi$, «разрешающая» вместо цепочки τ подстановку цепочки ξ.

ПРИМЕР 1.4. Из цепочки *bAcaS* в грамматике G_0 можно непосредственно вывести несколько цепочек, в зависимости от того, какое правило подстановки использовать и какую подцепочку в исходной цепочке мы будем заменять. Например,

bAc*aS* $\to$ ***B****aS,* если по правилу 3 грамматики G_0 вхождение подцепочки *bAc* мы заменим на цепочку *B;*

*bAc****aS*** $\to$ *bAc****bbAc***, если по правилу 2 грамматики G_0 вхождение подцепочки *aS* заменяется на цепочку *bbAc;*

*bAca****S*** $\to$ *bAca****aSbAc***, если по правилу 1 грамматики G_0 заменяется вхождение подцепочки *S* на цепочку *aSbAc.*

Таким образом, грамматика Хомского — это просто конечное число правил подстановки одних цепочек вместо других. Правила имеют вид $\varphi \to \psi$, где φ и ψ — это цепочки терминальных и нетерминальных символов, а символ $\to$ можно интерпретировать, как «можно заменить на». С помощью этих правил можно преобразовывать цепочки символов: из одних цепочек можно получить

другие цепочки — объекты той же самой природы, что и исходные объекты. Правила грамматики — операции непосредственной подстановки — можно выполнять многократно, используя каждую следующую полученную цепочку как исходную для дальнейшего преобразования.

Определение 1.5. Из цепочки α выводима цепочка β в грамматике G (обозначается $\alpha => \beta$), если существует конечное множество цепочек $\pi_0, \pi_1,..., \pi_n, n \geq 0$, такое, что $\alpha = \pi_0, \beta = \pi_n$, и для всех $i = 1, ..., n$ выполняется $\pi_{i-1} \rightarrow \pi_i$.

Иными словами, цепочка β выводима из цепочки α в грамматике G, если β можно получить из α за конечное число шагов применения операции непосредственной выводимости. Символ «=>» означает рефлексивное транзитивное замыкание отношения «$\rightarrow$» т. е. 0 или любое конечное число шагов выполнения операции «$\rightarrow$».

Таким образом, грамматика Хомского представляет собой механизм порождения одних символьных цепочек из других символьных цепочек, причем из одной и той же цепочки можно породить несколько различных цепочек, часто бесконечное их количество.

Отметим, что порождающая грамматика Хомского не предписывает определенный единственный вывод, т. е. не задает **алгоритм** порождения. Здесь отсутствует важнейший элемент алгоритма — его детерминированность, определенность последовательности операций. Грамматика представляет только возможность — это инструмент, механизм порождения цепочек.

Определение 1.6. Языком, порождаемым грамматикой G, называется множество **терминальных** цепочек, выводимых из начального символа грамматики. Или, формально, $L(G) = \{ \alpha \in T^* \mid S => \alpha \}$.

Итак, любой вывод цепочек языка мы должны начинать только с начального нетерминального символа. Если после произвольного конечного числа подстановок, выполненных в соответствии с правилами грамматики, полученная в результате цепочка состоит только из терминалов, то это цепочка является цепочкой порождаемого данной грамматикой языка. Отсюда ясны названия **терминальные/нетерминальные** символы. Только терминальные, *окончательные* символы могут встретиться в цепочках языка, заканчивающих вывод. Нетерминальные — это дополнительные, вспомогательные символы, необходимые для задания языка конечным набором правил.

Например, цепочка *bbb* принадлежит языку, порождаемому грамматикой G_0. Действительно: $\boldsymbol{S} \rightarrow \boldsymbol{aS}bAc \rightarrow \boldsymbol{bbAc}bAc => BBbAc => BBB => bbb$ (здесь в промежуточных цепочках вывода выделены подцепочки, которые на следующем шаге заменяются в соответствии с одним из правил грамматики).

ПРИМЕР 1.5. Попробуем охарактеризовать весь язык, который порождается введенной нами грамматикой G_0. Начинаться любой вывод может только с начального нетерминала, и для его замены в G_0 есть только одно, первое правило: $S \rightarrow aSbAc$. Заменять S по первому правилу можно многократно, следовательно, мы в общем случае можем получить промежуточные цепочки вывода в этой грамматике вида: $S => a^nS(bAc)^n$.

Конечно, ни одна из этих цепочек не является цепочкой языка, порождаемого грамматикой G_0, они все включают нетерминальные символы. Для того чтобы получить любую цепочку языка, вывод после любого числа n первых шагов может пойти двумя путями: либо мы используем второе правило $aS \to bbAc$, для того чтобы заменить подцепочку aS на цепочку, не включающую S, либо по четвертому правилу подцепочка SbA будет заменена на пустую цепочку (поскольку мы используем кроме S и соседние символы, по крайней мере, один шаг в выводе должен быть сделан, т. е. $n > 0$).

В первом случае имеем: $S => a^nS(bAc)^n \to a^{n-1}aS(bAc)^n \to a^{n-1}bbAc(bAc)^n => => a^{n-1}bB^{n+1} => a^{n-1}b^{n+2}$.

Во втором: $S => a^nS(bAc)^n \to a^nSbAc(bAc)^{n-1} \to a^nc(bAc)^{n-1} => a^ncB^{n-1} => => a^ncb^{n-1}$.

Легко видеть, что никакие другие выводы из S в G_0 невозможны. Окончательно, $L(G_0) = \{ a^{n-1}b^{n+2}, a^ncb^{n-1} | n > 0\}$.

1.2 Примеры грамматик

Порождающие грамматики Хомского служат для формального задания языков, и если грамматика задана, можно получать различные цепочки языка, порождаемого этой грамматикой. На практике встречается обратная задача: построить грамматику языка на основе некоторого числа примеров «правильных» цепочек языка и интуитивных представлений о правильности некоторых языковых конструкций. Очевидно, что перечислить ВСЕ возможные цепочки языка нельзя — их бесконечное количество. Поэтому разработка грамматики — это неформальный акт, которому можно научиться на примерах.

ПРИМЕР 1.6. Рассмотрим задачу построения грамматик для некоторых языков, перечисленных в примере 1.2.

1 $V1 = \{a, b\}; L1 = \{aabb, baa, aaba\}$.

Порождающие грамматики разработаны для задания бесконечных языков. Для простейшего частного случая конечных языков построение порождающих грамматик должно быть весьма простым. Действительно, для этого языка грамматика G_1 имеет вид:

$G_1: S \to aabb \mid baa \mid aaba.$

Применение любого из трех правил сразу завершит вывод с получением соответствующей цепочки языка из начального символа.

2 $V2 = \{a, b\}; L2 = \{a^nb^n \mid n > 0\}$.

Любая цепочка языка $L2$, порождаемая искомой грамматикой, имеет вид AB, где A — цепочка, состоящая только из a, а B — цепочка из b, причем количества символов в группах A и B равны. При порождении удобно одновременно порождать и a, и b, тогда это требование будет выполнено автоматически. Если одним из правил грамматики будет $S \to aSb$, то многократным его применением можно построить:

$S \to aSb \to aaSbb => ... => aa...aSb...bb => ...$.

Все промежуточные цепочки вывода не являются цепочками языка, порождаемого G_2, поскольку эти цепочки состоят не только из терминальных символов. Для получения терминальных цепочек требуемого вида достаточно в этих промежуточных цепочках убрать S, что может быть сделано при наличии в грамматике правила $S \to \varepsilon$. Окончательно:

G_2: $S \to aSb \mid \varepsilon$.

3 $V3 = \{a, b, c\}$; $L3 = \{a^n bc^m \mid n, m > 0\}$.

Структура любой цепочки языка $L3$ — ABC (A — цепочки из a, C — цепочки из c, средняя часть B любой цепочки порождаемого языка состоит из единственного терминального символа b). Поскольку количества символов a в начале цепочек языка и c в конце цепочек могут не совпадать, отдельные части структуры цепочек можно генерировать независимо. Искомая грамматика:

G_3: $S \to ABC$,

$A \to aA \mid a$,

$B \to b$,

$C \to Cc \mid c$.

4 $V4 = \{a, b\}$; $L4 = \{\alpha \mid$ в цепочке α количества вхождений a и b равны$\}$.

Можно придумать различные грамматики, порождающие этот язык. Очевидно, что каждая грамматика отражает некоторую глубинную идею порождения цепочек с указанными свойствами. Например, одной из идей может быть такая: любая цепочка с указанным свойством есть перестановка символов уже известного нам языка $a^n b^n$, который мы генерировать умеем. Поэтому можно использовать грамматику G_2 для порождения цепочек $a^n b^n$ и добавить в нее правило, разрешающее произвольную перестановку терминальных символов:

G_4: $S \to aSb \mid \varepsilon$,

$ab \to ba$.

Вывод цепочки $aabbab$: $S \to aSb \to aaSbb \to aaaSbbb \to aaabbb \to$
$\to aababb \to aabbab$.

Попытаемся по-другому построить грамматику, порождающую этот же язык. Если в любой цепочке с совпадающим числом символов a и b первый символ — a, то во всей остальной части цепочки число символов b должно быть на единицу больше числа символов a. Пусть B — нетерминал, из которого порождаются все цепочки с этим свойством. Если первый символ такой цепочки b, то в оставшейся части цепочки количества символов a и b одинаковы, а любая такая цепочка, очевидно, порождается из S. Если первый символ такой цепочки тоже a, то это значит, что в оставшейся части цепочки число символов b уже на два больше, чем символов a и, следовательно, этот остаток всегда можно представить как две стоящие рядом подцепочки, каждая из которых содержит на одно вхождение символа b больше, чем символа a. Окончательно,

G_5: $S \to aB \mid bA \mid \varepsilon$,

$B \to bS \mid aBB$,

$A \to aS \mid bAA$.

5 $V5 = \{ (,) \}$; $L5$ — множество правильных скобочных выражений.

Грамматика, порождающая $L5$, очень проста. Пусть S — начальный символ грамматики, из него порождаются все цепочки языка $L5$. Грамматика порождает язык, описывая структуру правильных цепочек языка. Структура скобочных выражений имеет один из двух видов: либо вложенное в скобки правильное скобочное выражение, либо пара скобочных выражений, стоящих рядом. «Крайним частным случаем» является просто отсутствие скобок. В соответствии с этим представлением структуры грамматику этого языка можно задать тремя правилами, каждое — для конкретного типа структуры цепочки языка:

G_5: $S \to (S) \mid SS \mid \varepsilon$.

2 Домашнее задание

1 Построить вывод данной цепочки α в грамматике:

а) $S \to T \mid T+S \mid T-S$,
$T \to F \mid F*T$,
$F \to a \mid b$,
α = a-b*a+b.

б) $S \to aSBC \mid abC$,
$CB \to BC$,
$bB \to bb$,
$bC \to bc$,
$cC \to cc$.
α = aaabbbccc.

Практическое занятие №2. Классификация грамматик по Хомскому

Цель занятия – познакомить с классификацией грамматик, научиться определять тип грамматики.

1 Фрагмент теории

Ноам Хомский в своих работах выделил четыре основных типа грамматик, в зависимости от их условной сложности. Чем меньше «номер» типа, тем грамматика считается более сложной. Для отнесения грамматики к тому или иному типу необходимо соответствие всех ее правил вывода некоторым схемам.

1.1 Неограниченные грамматики (тип 0)

К неограниченным грамматикам (или грамматикам с фазовой структурой) относятся все формальные грамматики без исключения.

Определение 2.1. Формальная грамматика $G = (V, N, R, S)$ называется **неограниченной** или **грамматикой типа 0**, если все ее правила имеют вид $\alpha \to \beta$, где $\alpha \in (T \cup N)^+$ и содержит хотя бы один нетерминальный символ, $\beta \in (T \cup N)^*$.

В силу своей сложности, этот тип грамматик представляет исключительно теоретический интерес, на практике они не применяются. К типу 0 обычно относят естественные языки.

В грамматиках типа 0 на левую и правую части продукций не накладывается никаких ограничений. Цепочки α и β в правилах грамматики $\alpha \to \beta$ могут быть любыми. Единственное ограничение — левая часть α не может быть пустой цепочкой. Иными словами, запрещается порождать *нечто* из *ничего* в любом месте промежуточной цепочки вывода. Обычно α включает по крайней мере один нетерминал, однако Хомский не исключает и возможность продукций, в которых терминальная цепочка заменяется на другую терминальную, как, например, $ab \to ba$ в грамматике G_4. Очевидно, что все рассмотренные нами грамматики принадлежат этому общему классу — типу 0 грамматик Хомского. Как уже говорилось, общего алгоритма распознавания для этого типа грамматик не существует.

1.2 Контекстно-зависимые грамматики (КЗ-грамматики) (тип 1)

К типу 1 относят контекстно-зависимые и неукорачивающие грамматики.

Определение 2.2. Неукорачивающей грамматикой называется формальная грамматика $G = (T, N, P, S)$, все правила вывода которой имеют вид $\alpha \to \beta$, где $\alpha, \beta \in (T \cup N)^+$ и $|\alpha| \le |\beta|$ (т. е. каждая последующая цепочка вывода этой грамматики будет иметь такую же или бóльшую длину).

Определение 2.3. Контекстно-зависимой грамматикой называется формальная грамматика $G = (T, N, P, S)$, все правила которой имеют вид $\xi_1 A \xi_2 \to \xi_1 \beta \xi_2$, где $\xi_1, \xi_2 \in (T \cup N)^*$, $\beta \in (T \cup N)^+$.

Доказано, что множество языков, порождаемых неукорачивающими грамматиками, совпадает со множеством языков, порождаемым контекстно-зависимыми грамматиками, поэтому все грамматики типа 1 можно называть контекстно-зависимыми. Действительно, если внимательно посмотреть на правила вывода любой контекстно-зависимой грамматики, можно сказать, что интерпретация любого нетерминального символа в выводе будет зависеть от его контекста, т. е. окружающих его цепочек.

Иногда контекстно-зависимые грамматики используются для анализа текста на естественных языках, но в силу своей сложности при трансляции они не применяются.

1.3 Контекстно-свободные грамматики (КС-грамматики) (тип 2)

Определение 2.4. Контекстно-свободной грамматикой называется формальная грамматика $G = (T, N, P, S)$, все правила которой имеют вид $A \to \beta$, где $\beta \in (T \cup N)^+$.

Определение 2.5. Укорачивающей контекстно-свободной называется формальная грамматика $G = (T, N, P, S)$, все правила вывода которой имеют вид $A \to \beta$, где $\beta \in (T \cup N)^*$.

Укорачивающие контекстно-свободные грамматики почти эквивалентны контекстно-свободным.

Практически для всех языков программирования можно построить порождающие их контекстно-свободные грамматики. Именно поэтому стандартный способ описания синтаксиса языков программирования, данных, протоколов (например, в документах RFC или ISO) – форма Бэкуса-Наура (или BNF), на самом деле является своеобразной формой контекстно-свободных грамматик.

В грамматиках типа 2, или контекстно-свободных грамматиках, вид продукций $A \to \beta$. Левая часть продукций состоит из единственного нетерминала, и замена нетерминала на цепочку может осуществляться в любом контексте: контекстные ограничения в этих правилах отсутствуют. Такие продукции наиболее просты и естественны, они показывают, как каждая конструкция языка может быть построена из других конструкций и символов языка. В отличие от грамматик типа 1, ограничение, состоящее в невозможности замены нетерминала пустой цепочкой, в продукциях грамматик типа 2 обычно снимается.

Для КС-грамматик существуют сравнительно эффективные алгоритмы синтаксического анализа, применимые для распознавания цепочек языков, порождаемых любой грамматикой этого класса. Для цепочек КС-языков вывод их из начального символа удобно представляется деревом, называемым деревом вывода, или синтаксическим деревом. Именно это дерево явно и определенно отражает структуру конкретной цепочки языка, а именно то, как строится каждая конструкция данной цепочки из других конструкций в соответствии с одним из разрешенных правил построения.

КС-грамматики представляют наибольший интерес в информатике. Контекстно-свободными продукциями может быть представлено большинство правил языков программирования высокого уровня, хотя во многих таких языках есть правила, требующие более сложных механизмов определения. Например, согласованность типов в операторах присваивания требует контекстно-зависимой подстановки выражения правой части после того, как тип объекта левой части определен. Другими подобными правилами являются требование согласованности формальных и фактических операторов процедур, запрет использования неописанных переменных и т. п. Несмотря на наличие подобных контекстно-зависимых требований, все языки программирования задаются именно КС-грамматиками, а контекстно-зависимые правила и ограничения задаются на естественном языке как дополнительные ограничения. При трансляции их выполнение проверяется специальными семантическими программами, что не нарушает общей идеи рассмотрения синтаксиса языков программирования как заданного контекстно-свободными грамматиками.

1.4 Регулярные грамматики (тип 3). Регулярные выражения

Определение 2.6. *Праволинейной* называется формальная грамматика $G = (T, N, P, S)$, все правила вывода которой имеют вид $A \to tB$, где $A, B \in N, t \in T$.

Определение 2.7 *Леволинейной* называется формальная грамматика $G = = (T, N, P, S)$, все правила вывода которой имеют вид $A \to Bt$, $A, B \in N$, $t \in T$.

Праволинейные и леволинейные грамматики называются *регулярными* или *грамматиками типа 3*. Доказано, что они эквивалентны.

В грамматиках этого типа ограничения накладываются на правую часть продукций. Эти ограничения приводят к тому, что порождаемые языки этого типа являются автоматными, и распознающее их автоматическое устройство — это конечный автомат. Отсюда следует, что для языков типа 3 существует очень эффективный (линейный по сложности) алгоритм синтаксического анализа, описывающий работу конечного автомата. Однако языки типа 3 имеют сравнительно узкое применение в информатике из-за их ограниченных порождающих возможностей. Такими языками, например, нельзя описать даже скобочные структуры арифметических выражений и, конечно, любые вложенные конструкции.

Регулярные языки (т. е. языки, порождаемые регулярными грамматиками) могут быть описаны и другим способом – с помощью так называемых *регулярных выражений*.

Определим понятие регулярного выражения рекурсивно:

1 ε – регулярное выражение, порождающее пустой регулярный язык $\{\varepsilon\}$;

2 a – регулярное выражение, порождающее регулярный язык $\{a\}$.

3 Пусть P и Q – регулярные языки, которые порождены регулярными выражениями p и q соответственно. Тогда:

a) $(p|q)$ – регулярное выражение, порождающее регулярный язык $P \cup Q$;

b) (pq) – регулярное выражение, порождающее регулярный язык PQ;

c) (p^*) – регулярное выражение, порождающее регулярный язык P^*.

4 Других регулярных выражений нет.

В регулярных выражениях лишние скобки обычно опускаются, при этом действует соглашение о приоритете операций: наивысшим приоритетом обладает операция итерации, затем идет операция конкатенации, и, наконец, операция объединения обладает самым низким приоритетом.

Кроме стандартной операции итерации, введем еще одно обозначение p^+, которое будет эквивалентно регулярному выражению pp^*.

Примеры регулярных выражений:

- $a(a|b)$ – множество цепочек, состоящих из a и b и начинающихся с a;
- $((0|1)(0|1)(0|1))^+$ – множество всех цепочек, состоящих из нулей и единиц и имеющих длину, кратную 3.

Определение 2.8. Регулярные выражения называются *эквивалентными*, если они порождают один и тот же язык. Отношение эквивалентности будем обозначать знаком равенства (=).

Пусть p, q и r – регулярные выражения. Тогда справедливы следующие соотношения:

1) $p|q = q|p$; 2) $p|(q|r) = (p|q)|r$; 3) $p(qr) = (pq)r$;

4) $p(q|r) = pq|pr$; 5) $(p|q)r = pr|qr$; 6) $p\varepsilon = \varepsilon p = p$;

7) $p^* = p|p^*$; 8) $(p^*)^* = p^*$; 9) $p|p = p$.

1.5 Соотношения между типами грамматик и языков

Множества языков, порождаемых грамматиками различных типов, вложены друг в друга. Если обозначить множество всех грамматик типа 0 как *Г0*, типа 1 – как *КЗ*, типа 2 – как *КС*, а типа 3 – как *Р*, то получим следующее соотношение: $Р \subseteq КС \subseteq КЗ \subseteq Г0$.

В заключение приведем итоговую таблицу по разобранным типам грамматик.

Таблица 1 – Подклассы грамматик иерархии Хомского

Тип	Вид правил	Название грамматик	Распознающие абстрактные устройства
0	$\alpha \to \beta$	Неограниченные грамматики Свободные (free) грамматики	Машины Тьюринга
1	$\xi_1 A \xi_2 \to \xi_1 \beta \xi_2$	Контекстно-зависимые грамматики Неукорачивающие грамматики	Линейно-ограниченные автоматы
2	$A \to \beta$	Контекстно-свободные грамматики	Автоматы с магазинной памятью
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики А-грамматики Регулярные грамматики	Конечные автоматы

2 Задания для решения в аудитории

Задание 2.1. Построить все сентенциальные формы для грамматики:

$S \to A+B \mid B-A$,

$A \to a$,

$B \to b$.

Задание 2.2. К какому типу по Хомскому относится данная грамматика? Какой язык она порождает?

а) $S \to 0A1 \mid 01$,
$0A \to 00A1$,
$A \to 01$.

б) $S \to AB$,
$AB \to BA$,
$A \to a$,
$B \to b$.

в) $S \to A \mid B$,
$A \to aAb \mid 0$,
$B \to aBbb \mid 1$.

г) $S \to 0A \mid 1S$,
$A \to 0A \mid 1B$,
$B \to 0B \mid 1B \mid \perp$.

Задание 2.3. Какой язык генерируется следующими грамматиками?

$S \to 0S1 \mid 01$,

$S \to aSbS \mid bSaS \mid \varepsilon$.

3 Домашнее задание

1 К какому типу по Хомскому относится данная грамматика? Какой язык она порождает?

а) $S \to APA$,

б) $S \to aQb \mid \varepsilon$,

$P \rightarrow + \mid -$, $Q \rightarrow cSc$.

$A \rightarrow a \mid b$.

2 Какой язык генерируется следующими грамматиками?

$S \rightarrow +SS \mid -SS \mid a$,

$S \rightarrow S(S)S \mid \varepsilon$.

3 Построить грамматику для языка, представляющего собой строки из нулей и единиц, в которых за каждым нулем следует, по крайней мере, одна единица.

Практическое занятие №3. Разбор цепочек. Дерево вывода

Цель занятия – рассмотреть различные способы вывода и представления цепочек.

1 Фрагмент теории

С точки зрения формальных грамматик, текст программы представляет собой цепочку. Как известно, цепочка принадлежит языку, порождаемому грамматикой, тогда и только тогда, когда существует ее вывод из цели этой грамматики. Такой процесс называется разбором. Как было указано выше, подавляющее большинство языков программирования может быть описано с помощью КС-грамматик, для которых были разработаны алгоритмы решения задачи разбора.

Рассмотрим основные понятия, связанные с разбором цепочек.

Определение 3.1. Вывод цепочки $\beta, \beta \in T$ из цели грамматики $G = (T, N, P, S)$ называется **левым (левосторонним)**, если в этом выводе каждая следующая сентенциальная форма получается из предыдущей путем замены самого левого нетерминала.

Определение 3.2. Вывод цепочки $\beta, \beta \in T$ из цели грамматики $G = (T, N, P, S)$ называется **правым (правосторонним)**, если в этом выводе каждая следующая сентенциальная форма получается из предыдущей путем замены самого правого нетерминала.

Для одной и той же цепочки можно построить несколько выводов, которые будут отличаться порядком применения правил грамматики. Такие выводы называются **эквивалентными**.

Рассмотрим пример. В грамматике $G = (\{a, b\}, \{S, T\}, \{S \rightarrow T + S;\ T \rightarrow a \mid b\}, S)$ для цепочки $a+b+a$ можно построить следующие выводы:

1 $S \rightarrow T+S \rightarrow T+T+S \rightarrow T+T+T \rightarrow a+T+T \rightarrow a+b+T \rightarrow a+b+a$;

2 $S \rightarrow T+S \rightarrow a+S \rightarrow a+T+S \rightarrow a+b+S \rightarrow a+b+T \rightarrow a+b+a$;

3 $S \rightarrow T+S \rightarrow T+T+S \rightarrow T+T+T \rightarrow T+T+a \rightarrow T+b+a \rightarrow a+b+a$.

Все эти выводы эквивалентны, при этом вывод 2 является левосторонним, а 3 – правосторонним.

Для КС-грамматик вывод можно представить в графической форме, которая называется **деревом вывода**. Для эквивалентных выводов их деревья будут совпадать.

Определение 3.3. Дерево называется **деревом вывода** в КС-грамматике $G = (T, N, P, S)$, если для него выполнены следующие условия.

1 Каждая вершина дерева помечена символом из множества $T \cup N \cup \{\varepsilon\}$, причем корень помечен нетерминалом S, а листья – терминальными символами или ε.

2 Если промежуточная вершина дерева помечена нетерминалом A, а ее потомки – символами $a_1, a_2, ..., a_n$; $a_i \in (T \cup N)$, в грамматике G существует правило вывода $A \to a_1, a_2, ..., a_n$.

3 Если единственный потомок вершины, помеченной нетерминалом A – это вершина, помеченная символом ε, то правило $A \to \varepsilon$ есть в грамматике G.

Пример дерева вывода для цепочки $a+b+a$ в грамматике из предыдущего примера (рисунок 1).

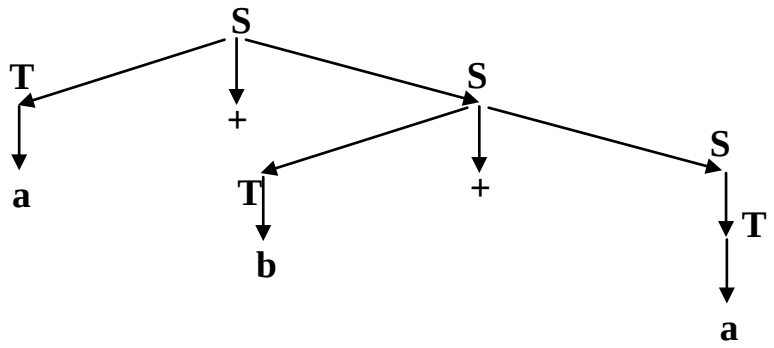


Рисунок 1 – Дерево вывода цепочки a + b+ a

Определение 3.4. КС-грамматика называется **неоднозначной**, если существует хотя бы одна цепочка, для которой можно построить два или больше различных деревьев вывода. В противном случае грамматика называется **однозначной**.

Можно также сказать, что в неоднозначной грамматике существует хотя бы одна цепочка, для которой можно построить два различных правосторонних или левосторонних вывода.

Дерево вывода можно строить либо восходящим, либо нисходящим способом.

При нисходящем способе дерево строится от корня к листьям: на каждом шаге вершины очередного яруса помечаются таким образом, чтобы правила вывода, которые проектируются на вершины следующего яруса, в итоге образовали разбираемую цепочку. При восходящем способе разбираемую цепочку пытаются свернуть к цели грамматики, ища на каждом шаге такую подцепочку, чтобы она совпала с правой частью одного из правил вывода. В этом случае для всех вершин подцепочки строится родительская вершина, помечаемая левой частью этого правила.

Если грамматика однозначная, то независимо от выбора способа построения в итоге должно получиться одно и то же дерево.

Ниже приведен пример неоднозначной грамматики:

G = ({if, then, else, a, b}, P, S);

P = {S → if b then S else S | if b then S | a}.

В этой грамматике для цепочки *if b then if b then a else a* можно построить два различных дерева вывода (проверьте это самостоятельно).

Заметим, что определенная однозначность – это свойство не языка, а грамматики, и для некоторых неоднозначных грамматик, существуют однозначные грамматики, эквивалентные им.

В приведенном выше примере конструкция *else* в двух различных деревьях разбора будет относиться к различным *then*, и может по-разному трактоваться, что для языка программирования недопустимо. Но если договориться о том, что *else* всегда будет соответствовать последнему *then*, то можно преобразовать грамматику из примера таким образом, что она станет однозначной (как?).

Определение 3.5. Язык называется **неоднозначным**, если он не может быть порожден однозначной грамматикой. **Все языки программирования однозначны**.

Проблема, порождает ли данная КС-грамматика однозначный язык, является алгоритмически неразрешимой.

2 Задания для решения в аудитории

Задание 3.1. Построить восходящим и нисходящим методами дерево вывода для цепочки α в данной грамматике:

S→ S0 | S1 | D0 | D1,

D→ H.,

H→ 0 | 1 | H0 | H1,

α = 10.1001 .

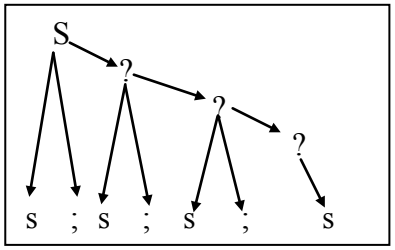
Задание 3.2. Написать КС-грамматику для данного языка, построить вывод цепочки α в этой грамматике:

а) L = $\{a^{2n}b^{m}c^{2k} \mid m = n + k, m > 1; n, k > 0\}$ α = aabbbcccc;

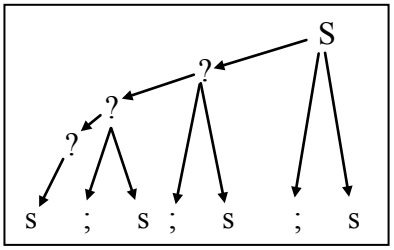
б) L = $\{1^{n}0^{m}1^{p} \mid n + p > m; n, p, m > 0\}$; α = 110000111.

Задание 3.3. Язык L, заданный регулярным выражением R = (s;)*s, состоит из цепочек вида s;s;s;s;s, т. е. последовательности символов s, разделенных точками с запятой. Построить две грамматики, в которых синтаксические деревья для таких цепочек имеют вид, представленный, на рисунках ниже.

а)



б)



Задание 3.4. Построить грамматику для каждого из следующих языков:

а) строки из 0 и 1, в которых отсутствует подстрока 011;

b) $L = \{ab^nc \mid n \geq 1\}$.

Задание 3.5. Построить КС-грамматику, эквивалентную грамматике с продукциями: $\{S \to aAb;\ aA \to aaAb;\ A \to \varepsilon\}$.

3 Домашнее задание

1 Построить восходящим и нисходящим методами дерево вывода для цепочки α в данной грамматике:

$S \to$ if B then S | B = E,

$E \to$ B | B+E,

$B \to$ a | b,

α = if a then b = a+b+b.

2 Написать КС-грамматику для данного языка, построить вывод цепочки α в этой грамматике:

$L = \{a^ncb^mca^n \mid n,m > 0\}$; α = aacbbbcaa.

3 Построить грамматику для языка, представляющего собой строки из нулей и единиц с одинаковым числом вхождений нулей и единиц.

4 Построить КС-грамматику для следующего языка: $L = \{a^mb^nc \mid m, n \geq 1\}$.

Практические занятия №4-5. Преобразования КС-грамматик

Цель занятий – познакомить с различными способыми преобразования грамматик.

1 Фрагмент теории

Прежде всего сделаем одно важное замечание по поводу пустой строки ε. Наличие в грамматике пустой строки сильно усложняет рассуждения относительно такой грамматики, поэтому будем предполагать, что рассматриваемые нами КС-грамматики не содержат ε-продукций, т. е. продукций вида $A \to \varepsilon$. Это не сузит наши возможности в изучении свойств КС-грамматик и КС-языков. Действительно, пусть L – контекстно-свободный язык и пусть $G = (T, N, P, S)$ – КС-грамматика, порождающая язык $L\backslash\{\varepsilon\}$. Тогда, добавив к N новый нетерминальный символ S' в качестве начального символа и расширив R двумя новыми продукциями: $S' \to S \mid \varepsilon$, мы получим новую грамматику G', порождающую язык L. Поэтому любое нетривиальное заключение по поводу свойств языка $L\backslash\{\varepsilon\}$ может быть перенесено и на язык L. Кроме того, существует метод получения из любой КС-грамматики G некоторой другой КС-грамматики G_ε такой, что $L(G_\varepsilon) = L(G)\backslash\{\varepsilon\}$. Следовательно, для всех приложений можно ограничиться случаем, когда контекстно-свободный язык не содержит ε, что мы и будем предполагать в дальнейшем.

Изучение преобразований КС-грамматик начнем с так называемых **правил подстановки**.

Определение 4.1. Продукции вида $A \to \alpha$, где A – нетерминальный символ, отличный от ε, будем называть ***A*-продукциями**.

Теорема 4.1. Пусть $G = (T, N, P, S)$ – КС-грамматика. Предположим, что P содержит продукции вида $A \to \alpha_1 B \alpha_2$ (1), где A и B – различные символы из N. Допустим, что $B \to \beta_1|\beta_2|...|\beta_n$ (2) множество всех B-продукций в P. Пусть $G = (T, N, P', S)$ – грамматика, в которой P' получено удалением из P продукции (1) и добавлением новых продукций $A \to \alpha_1\beta_1\alpha_2|\alpha_1\beta_2\alpha_2|...|\alpha_1\beta_n\alpha_2$. Тогда $L(G') = L(G)$.

ПРИМЕР 4.1. Рассмотрим КС-грамматику $G = (\{B,S\}, \{a, b\}, P, S)$ с продукциями $S \to a \mid aaS \mid abBc$; $B \to abbS \mid b$. Производя подстановку вместо символа B в правой части продукции $S \to abBc$ на правые части B-продукций, получаем новую грамматику G' с множеством продукций $S \to a \mid aaS \mid ababbSc \mid abbc$; $B \to abbS \mid b$. При этом G' эквивалентна G. В частности, строка $aaabbc$ выводима как в G: $S \Rightarrow aaS \Rightarrow aaabBc \Rightarrow aaabbc$, так и в G': $S \Rightarrow aaS \Rightarrow \Rightarrow aaabbc$.

В теореме 4.1 мы рассмотрели случай замены продукции $A \to \alpha_1 B \alpha_2$, когда $A \neq B$. В следующей теореме рассматривается случай $A = B$.

Определение 4.2. Продукции вида $A \to A\alpha$, где $A \in N$, $\alpha \in (N \cup T)^*$, назовем **леворекурсивными продукциями**.

Теорема 4.2. Пусть дана КС-грамматика $G = (T, N, P, S)$. Допустим, что $A \to A\alpha_1 \mid A\alpha_2 \mid ... \mid A\alpha_n$ (3), где $A \in N$, $\alpha_i \in (N \cup T)^*$, $i = 1, 2, 3, ..., n$ – все ее леворекурсивные A-продукции, а $A \to \beta_1 \mid \beta_2 \mid ... \mid \beta_m$ (4), где $\beta_i \in (N \cup T)^*$, i = 1, 2, 3, ..., m – все остальные ее A-продукции.

Рассмотрим грамматику $G' = (T, N \cup \{ZA\}, P', S)$, где $Z_A \notin N$, а P' получен заменой всех продукций вида (3) и (4) на следующую совокупность продукций: $A \to \beta_i \mid \beta_i Z_A$, $i = 1, 2, ..., m$; $Z_A \to \alpha_i \mid \alpha_i Z_A$, $i = 1, 2, ..., n$.

Тогда $L(G) = L(G')$.

Мы получили способ избавиться от леворекурсивных продукций в КС-грамматиках. Эти продукции являются частным случаем левой рекурсии; в общем случае грамматика G называется **леворекурсивной**, если существует нетерминал A, для которого возможен вывод $A \Rightarrow A\alpha$. Это свойство чаще всего является нежелательным, и теорема 4.2 позволяет систематически исключать леворекурсивные продукции, которые вначале могут быть полезны при построении грамматики.

ПРИМЕР 4.2. Пусть G_0 – грамматика с продукциями: $E \to E + T \mid T$; $T \to T \times F \mid F$; $F \to (E) \mid a$. Применяя к ней процедуру из теоремы 4.2, получим эквивалентную ей грамматику со следующим множеством продукций:

$E \to T \mid TZ_E$; $Z_E \to +T \mid +TZ_E$; $T \to F \mid FZ_T$; $Z_T \to \times F \mid \times FZ_T$; $F \to (E) \mid a$.

Следующий тип продукций, который мы хотели научиться исключать из грамматики, – это так называемые **бесполезные продукции**, т. е. те, которые никогда не участвуют в выводе никакой строки терминалов. Например, в грамматике с продукциями $S \to aSb \mid \varepsilon \mid A$; $A \to aA$ последняя продукция является бесполезной, так как символ A никогда не может быть преобразован в строку терминалов. Если при выводе из S символ A окажется входящим в сентенциальную форму, то она никогда не сможет быть трансформирована в сентенцию. Значит, удаление продукции $A \to aA$ не окажет влияния на язык,

порождаемый грамматикой, а лишь приведет к упрощению множества продукций.

Определение 4.3. В КС-грамматике $G = (N, T, S, P)$ нетерминал $A \in N$ называется **полезным**, если существует хотя бы одна строка $\varphi \in L(G)$ такая, что $S => \alpha A \beta => \varphi, \alpha, \beta \in (N \cup T)^*$.

Нетерминал, не являющийся полезным в грамматике G, называется **бесполезным**.

Продукция в грамматике G является **бесполезной**, если в нее входит хотя бы один бесполезный нетерминал.

Нетерминальный символ может быть бесполезным по двум причинам: он недостижим из начального символа грамматики или из него невыводима строка терминалов. Рассмотренный выше пример относится ко второму случаю. Возьмем другую грамматику с начальным символом S и следующими продукциями:

$S \to A$,

$A \to aA \mid \varepsilon$,

$B \to bA$.

Здесь как символ B, так и продукция $B \to bA$ являются бесполезными и, хотя из B можно вывести терминальную строку, но не существует ни одного вывода вида $S => \alpha B \beta$. Для исключения нетерминалов, недостижимых из начального символа грамматики, полезен **граф зависимости нетерминалов**. Он наглядно показывает сложные связи между нетерминалами по выводимости и находит много применений.

Определение 4.4. Для КС-грамматики граф зависимости – это граф, вершины которого помечены нетерминалами и в котором дуга между двумя вершинами A и B существует тогда и только тогда, когда в грамматике есть продукция вида $A \to \alpha B \beta$.

Нетрудно заметить, что нетерминал X полезен только тогда, когда в графе зависимости существует путь из вершины S в вершину X.

ПРИМЕР 4.3. Пусть дана КС-грамматика $G = (N, T, P, S)$, где $N = \{S, A, B, C\}$, $T = \{a, b\}$, а P состоит из продукций: $\{S \to aS \mid A \mid C;\ A \to a;\ B \to aa;\ C \to aCb\}$. Требуется удалить из G все бесполезные символы и продукции.

Вначале определим множество терминалов, которые могут породить терминальные строки. Так как $A \to a$ и $B \to aa$, то A и B принадлежат этому множеству. То же самое справедливо для S, так как $S => A => a$. Нетерминал C не войдет в это множество, т. е. он является бесполезным символом так же, как и те продукции, которые его содержат. Исключая нетерминал C вместе с соответствующими продукциями, получаем грамматику $G' = (N', T, P', S)$, где $N' = \{S, A, B\}$, а P' включает продукции: $\{S \to aS \mid A;\ A \to a;\ B \to aa\}$.

Теперь наша задача – удалить из G' недостижимые нетерминалы, т. е. символы из N', недостижимые с помощью вывода из S.

Воспользуемся для этого графом зависимости для N', изображенным на рисунке 2.

На основании вида этого графа делаем вывод, что терминал B недостижим, а, следовательно, и бесполезен. Удаляя его и соответствующую продукцию, окончательно получаем грамматику: $G'' = (\{S, A\}, \{a\}, P'', S)$, где P'' состоит из следующих продукций: $\{S \to aS \mid A;\ A \to a\}$.

Очевидно, что при этом $L(G'') = L(G)$.

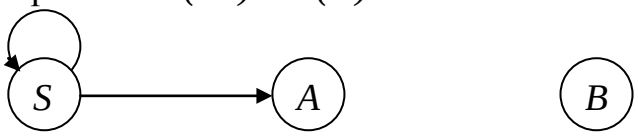


Рисунок 2 – Граф зависимости

Обобщим наши рассуждения в следующей теореме.

Теорема 4.3. Для любой КС-грамматики $G = (N, T, P, S)$ существует эквивалентная грамматика $G^{\oplus} = (T^{\oplus}, N^{\oplus}, P^{\oplus}, S)$, не содержащая ни бесполезных нетерминальных символов, ни бесполезных продукций.

Доказательство. Построение $G^{\oplus}$ проведем в два этапа. Сначала постоим грамматику $G' = (N' , T , P', S)$ такую, что N содержит только такие нетерминалы A, для которых в этой грамматике возможен вывод $A \Rightarrow \varphi$, $\varphi \in T^*$. Это построение проведем с помощью следующего алгоритма:

Шаг 1. Полагаем N' равным $\varnothing$.

Шаг 2. Выполнять цикл до тех пор, пока больше не будет добавлен новый символ в N':

для каждого $A \in N$, для которого в P есть продукция вида: $A \to \alpha_1\alpha_2...\alpha_n$, где $\alpha_i \in T^* \cup N'$, $i = 1, 2, ..., n$, добавить A в N'.

Шаг 3. Поместить в P' все продукции из P, образованные из символов, входящих в множество $N' \cup T \cup \{\varepsilon\}$.

Нетрудно видеть, что этот алгоритм, начав работать на G, через конечное число шагов останавливается. Ясно также, что если $A \in N'$, то в G' возможен вывод $A \Rightarrow \varphi$, $\varphi \in T^*$.

Теперь переходим ко второму этапу построения грамматики $G^{\oplus}$. Построим граф зависимости для $G^{\oplus}$ и с помощью него найдем все недостижимые нетерминальные символы. Исключим их из G' вместе с содержащими эти символы продукциями. Исключим также те терминалы, которые не входят в какую-нибудь полезную продукцию. В результате получим грамматику $G^{\oplus} = (T^{\oplus}, N^{\oplus}, P^{\oplus}, S)$.

По построению, $G^{\oplus}$ не содержит бесполезных символов или продукций.

Пусть для произвольной строки $\varphi \in L(G)$ существует вывод в G:
$S \Rightarrow \alpha A \beta \Rightarrow \varphi$.

Так как по построению грамматика $G^{\oplus}$ сохранила A и все связанные с ним полезные продукции, то у нас есть все необходимое, чтобы построить соответствующий вывод в $G^{\oplus}$: $S \Rightarrow \alpha A \beta \Rightarrow \varphi$.

Грамматика $G^{\oplus}$ получилась из G удалением части продукций, значит, $P^{\oplus} \subseteq P$ и $L(G^{\oplus}) \subseteq L(G)$.

В итоге получаем равенство: $L(G^{\oplus}) = L(G)$, т.е. грамматики G и $G^{\oplus}$ эквивалентны.

Следующий тип продукций, которые в ряде случаев нежелательны, – это **продукции с пустой строкой в правой части.**

Определение 4.5. Продукция КС-грамматики, имеющая вид $A \to \varepsilon$, называется **ε-продукцией**. Нетерминал A, для которого в программе возможен вывод $A \Rightarrow \varepsilon$, называется **обнуляемым**.

Заметим, что грамматика может порождать язык, не содержащий ε, имея при этом ε-продукции или обнуляемые нетерминалы. В этом случае ε-продукции могут быть исключены из грамматики.

Теорема 4.4. Пусть КС-грамматика G порождает язык $L(G)$, не содержащий ε. Тогда существует эквивалентная ей грамматика G', не содержащая ε-продукций.

Доказательство. Вначале построим множество N_0 всех обнуляемых нетерминалов в G, используя следующую процедуру.

Шаг 1. Для всех продукций $A \to \varepsilon$ помещаем A в N_0.

Шаг 2. Повторяем следующий цикл до тех пор, пока возможно добавление нетерминалов в N_0:

для всех продукций $B \to A_1A_2...A_n$, где $A_1, A_2, ..., A_n \in N_0$, поместить B в N_0.

После окончания формирования множества N_0 можем перейти к построению P'. Рассмотрим все продукции из P вида $A \to X_1X_2...X_m$, $m \geq 1$, для которых $X_i \in N \cup T$. Для каждой такой продукции из P мы помещаем в P' как саму эту продукцию, так и все производные от нее продукции, получающиеся стиранием любого количества обнуляемых символов в правой части исходной продукции, за одним исключением: если все X_i обнуляемые, то продукция $A \to \varepsilon$ в P' не добавляется.

ПРИМЕР 4.4. Найти КС-грамматику без ε-продукций, эквивалентную грамматике с продукциями: $S \to AbaC$; $A \to BC$; $B \to b \mid \varepsilon$; $C \to D \mid \varepsilon$; $D \to d$.

Находим множество обнуляемых нетерминалов: $\{A, B, C\}$. Затем видоизменяем совокупность продукций; получаем:

$S \to ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a$,

$A \to B \mid C \mid BC$,

$B \to b$,

$C \to D$,

$D \to d$.

Последний класс «нежелательных» продукций, который мы рассмотрим, – это продукции, у которых **левая и правая части представляют собой изолированные нетерминальные символы**.

Определение 4.6. Продукция КС-грамматики, имеющая вид $A \to B$, где A, B – нетерминальные символы, называется **цепной продукцией**.

Теорема 4.5. Любая КС-грамматика $G = (T, N, P, S)$ без ε-продукций может быть преобразована к эквивалентной грамматике $G' = (T', N', P', S)$, не имеющей цепных продукций.

Доказательство. Очевидно, любая цепная продукция вида $A \to A$ может быть удалена из грамматики без всякого ущерба для порождаемого языка, поэтому будем рассматривать продукции вида $A \to B$, где A и B различные нетерминалы.

Прежде всего, для каждого $A \in N$ найдем множество $D(A) = \{B \mid A \Rightarrow B\}$. Это можно сделать с помощью графа зависимости для G: он будет содержать дугу (C, D), если в грамматике есть продукция $C \to D$.

Тогда $A \Rightarrow B$ имеет место всякий раз, когда существует путь из A в B.

Построение новой грамматики G' начинаем с помещения в P' всех продукций из P, не являющихся цепными. Пусть, далее, $B \in D(A)$. Тогда добавляем в P' новые продукции: $A \to \beta_1|\beta_2|...|\beta_n$, где $B \to \beta_1|\beta_2|...|\beta_n$ – множество всех B-продукций в P' (не в P!). Заметим, что, так как продукции $B \to \beta_i$ берутся из P', ни одна из строк β_i не может быть единственным нетерминалом, следовательно, ни одной цепной продукции в P' добавлено не будет.

Нетрудно показать, что получившаяся грамматика G' эквивалентна G.

ПРИМЕР 4.5. Исключить все цепные продукции из грамматики $\{S \to Aa \mid B; B \to A \mid bb; A \to a \mid bc \mid B\}$.

Строим граф зависимости, учитывая только цепные продукции (рисунок 3):

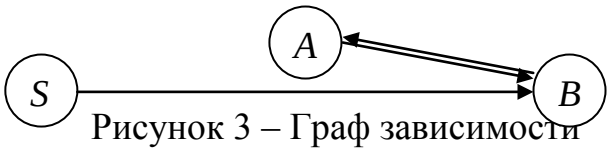


Рисунок 3 – Граф зависимости

Видим, что $D(S) = \{A, B\}$, $D(A) = \{B\}$, $D(B) = A$. Исключаем из грамматики все цепные продукции $C \to B$, $B \to A$ и $A \to B$ и добавляем к оставшимся новые продукции; получаем:

$S \to Aa$ $A \to a\|bc$ $B \to bb$ } «старые» продукции;	$S \to a\|bc\|bb$ $A \to bb$ $B \to a\|bc$ } «новые» продукции.

Объединяя полученные результаты, можно показать, что КС-грамматики могут быть «очищены» от бесполезных продукций, ε-продукций и цепных продукций.

Существует, однако, опасность, что при удалении одного типа продукций могут возникнуть продукции другого, ранее удаленного, типа. Заметим, что процедура удаления из грамматики цепных продукций не порождает ε-продукций, а процедура удаления бесполезных продукций не создает ни ε-продукций, ни цепных продукций. Следовательно, можно удалить все нежелательные продукции в следующем порядке:

1) удаляем ε-продукции;
2) удаляем цепные продукции;
3) удаляем бесполезные продукции.

2 Задания для решения в аудитории

Задание 4.1. Преобразовать следующие грамматики, выбросив все бесполезные продукции. Какой язык порождает каждая из этих грамматик?

a) $S \to aSbBc$,
$B \to cD$,
$S \to BdD$,
$C \to bCc$,
$A \to BcS$,
$C \to cDA$,
$A \to acb$,
$D \to cSD$,
$B \to bE$,
$E \to ce$.

b) $S \to SbAc$,
$B \to Ec$,
$S \to BdA$,
$C \to bCc$,
$A \to BcS$,
$C \to cDA$,
$A \to acb$,
$D \to cAD$,
$B \to b$,
$E \to cB$.

Задание 4.2. Преобразовать следующие грамматики, выбросив все ε-продукции. Какой язык порождает каждая из этих грамматик?

a) $S \to AaAb$,
$S \to BbBa$,
$A \to \varepsilon$,
$B \to \varepsilon$.

b) $S \to aSbS$,
$S \to bSaS$,
$S \to \varepsilon$.

Задание 4.3. Преобразовать следующую грамматику, удалив из нее все цепные продукции:

$S \to BA;\ A \to C \mid ac;\ B \to b;\ C \to A$.

Задание 4.4. Построить КС-грамматику, эквивалентную грамматике с правилами:

a) $S \to aAb$,
$aA \to aaAb$,
$A \to \varepsilon$.

b) $S \to AB \mid ABS$,
$AB \to BA$,
$BA \to AB$,
$A \to a$,
$B \to b$.

Задание 4.5. Доказать, что грамматика с правилами: $\{S \to aSBC \mid abC;\ CB \to BC;\ bB \to bb;\ bC \to bc;\ cC \to cc\}$ порождает язык $L = \{a^n b^n c^n \mid n \geq 1\}$. Для этого показать, что в данной грамматике:

- выводится любая цепочка вида $a^n b^n c^n$ $(n \geq 1)$;
- не выводятся никакие другие цепочки.

Задание 4.6. Определить тип грамматики. Описать язык, порождаемый этой грамматикой. Написать для этого языка КС-грамматику.

$S \to P \perp$,
$P \to 1P1 \mid 0P0 \mid T$,
$T \to 021 \mid 120R$,
$R1 \to 0R$,
$R0 \to 1$,
$R \perp \to 1 \perp$

Задание 4.7. Построить регулярную грамматику, порождающую цепочки в алфавите {a, b}, в которых символ a не встречается два раза подряд.

Задание 4.8. Написать КС-грамматику для языка L, построить дерево вывода и левосторонний вывод для цепочки aabbbcccc.

$L = \{a^{2n}b^{m}c^{2k} \mid m=n+k,\ m>1\}$.

Задание 4.9. Написать КС-грамматику, порождающую язык L, и вывод для цепочки aacbbbcaa в этой грамматике.

$L = \{a^{n}cb^{m}ca^{n} \mid n,\ m>0\}$.

Задание 4.10. Написать КС-грамматику, порождающую язык L, и вывод для цепочки 110000111 в этой грамматике.

$L = \{1^{n}0^{m}1^{p} \mid n+p>m;\ n, p, m>0\}$.

3 Домашнее задание

1 Преобразовать грамматику, выбросив все бесполезные продукции. Какой язык порождает эта грамматика?

$S \to aC;\ B \to cAb;\ S \to BdA;\ C \to DB;\ A \to cS;\ C \to cA;$

$A \to a;\ D \to cCAD;\ B \to SbB;\ E \to BcAe$.

2 Преобразовать грамматику, выбросив все ε-продукции. Какой язык порождает эта грамматика?

a) $S \to aSbS;\ S \to BaB;\ B \to \varepsilon;\ B \to bS$;

b) $S \to AB;\ A \to aAA \mid \varepsilon;\ B \to bBB \mid \varepsilon$.

3 Преобразовать грамматику, выбросив все цепные продукции:

$S \to B \mid a;\ B \to C \mid b;\ C \to DD \mid c$.

4 Дан язык $L = \{1^{3n+2}0^{n} \mid n\geq 0\}$. Определить его тип, написать грамматику, порождающую L. Построить левосторонний и правосторонний выводы, дерево разбора для цепочки 1111111100.

Практическое занятие №6. Нормальные формы грамматик

Цель занятия – познакомить с нормальными формами грамматик, рассмотреть алгоритмы их получения.

1 Фрагмент теории

Рассмотрим преобразования грамматик в нормальную форму Хомского и Грейбах. Одна из этих двух форм имеет жесткое ограничение на длину правой части продукций грамматики: допускается использование не более двух символов. Она представляет собой нормальную форму Хомского.

Определение 6.1. КС-грамматика $G = (T, N, P, S)$ имеет **нормальную форму Хомского**, если любая ее продукция имеет вид либо $A \to BC$, либо $A \to \to a$, где $A, B, C \in N$, $a \in T$.

Теорема 6.1. Любая КС-грамматика $G = (T, N, P, S)$ такая, что $\varepsilon \notin L(G)$, может быть преобразована в эквивалентную ей грамматику $G_1 = (T, N_1, P_1, S)$, имеющую нормальную форму Хомского.

Доказательство. Будем считать, что G не содержит ни ε-продукций, ни цепных продукций.

Построение грамматики G_1 осуществим за два шага.

Шаг 1. Построим грамматику $G' = (T, N', P', S)$.

Рассмотрим все продукции из P вида: $A \to X_1X_2...X_n$ (*), где каждый символ X_i либо терминал из T, либо нетерминал из N. Если n = 1, тогда X_1 должен быть терминалом, так как в P нет цепных продукций. В этом случае помещаем эту продукцию в P'.

Если n ≥ 2, то вводим новые нетерминальные символы Z_a для каждого $X = a$, где $a \in T$. Каждой продукции из P, имеющей вид (*), ставим в соответствие продукцию: $A \to B_1B_2...B_n$, где

$$B_i = \begin{cases} X_i, \text{ если } X_i \in N, \\ Z_a, \text{ если } X_i = a \text{ и } a \in T. \end{cases}$$

Для каждого нового нетерминального символа Z_a мы формируем продукцию $Z_a \to a$ и помещаем ее в P'.

На этом шаге удаляются все терминалы из продукций, правая часть которых имеет длину больше 1, а вместо них на их места вводятся новые нетерминалы. В результате после завершения данного шага получаем грамматику $G' = (T, N', P', S)$, все продукции которой имеют вид: $A \to a$ (**), $a \in T$, либо $A \to B_1B_2...B_n$ (***), где $B_i \in N$, $i = 1,2,...n$.

Шаг 2. Вводим вспомогательные нетерминалы для того, чтобы сократить длину правых частей до 2, если она превышает эту границу.

Сначала помещаем все продукции вида (**) в P_1, а также помещаем в P_1 все продукции вида (***), правая часть которых имеет длину 2 (т. е. когда n = 2). Если n ≥ 2, вводим новые нетерминалы B_1' , B_2' , ... и добавляем в P' новые продукции: $A \to B_1B_1'$; $B_1' \to B_2B_2'$; ...; $B_{n-2}' \to B_{n-1}B_n$.

Очевидно, полученная в результате этого шага грамматика $G_1 = (T, N_1, P_1, S)$ будет иметь нормальную форму Хомского, причем $L(G) = L(G_1)$.

ПРИМЕР 6.1. Преобразовать грамматику G с продукциями $\{S \to Aba; A \to \to aab; B \to Ac\}$ к нормальной форме Хомского.

Прежде всего убеждаемся, что G не содержит ни цепных, ни ε-продукций. На шаге 1 вводим новые нетерминалы Z_a, Z_b, Z_c и изменяем множество продукций: $\{S \to ABZ_a; A \to Z_aZ_aZ_b; B \to A\,Z_c; Z_a \to a; Z_b \to b; Z_c \to c\}$.

На втором шаге вводим новые нетерминалы и редуцируем правые части продукций (первой и второй); в результате получаем грамматику в нормальной форме Хомского с множеством продукций: $\{S \to AB'; B' \to BZ_a; A \to Z_aZ_a'; Z_a' \to Z_aZ_b; B \to A\,Z_c; Z_a \to a; Z_b \to b; Z_c \to c\}$.

Другой полезной грамматической формой является **нормальная форма Грейбах**. В этом случае ограничение накладывается не на длину правых частей, а на порядок, в котором терминалы и нетерминалы входят в правую часть продукции. Эта форма имеет много теоретических и практических применений,

однако процедура преобразования к ней сложнее, чем в случае нормальной формы Хомского, а эквивалентность исходной грамматике не столь очевидна.

Будем говорить, что КС-грамматика представлена в **нормальной форме Грейбах**, если она ε-свободна и все продукции ее имеют вид: $A \to a\gamma$, где a — терминал, а γ — произвольная цепочка нетерминалов, возможно пустая.

Существуют простые правила перевода КС-грамматики в нормальную форму Грейбах. Приведем пример такого перевода.

ПРИМЕР 6.2. Приведем к нормальной форме Грейбах грамматику: $\{S \to BA;\ A \to BS;\ A \to x;\ B \to zAy\}$.

Во-первых, в продукциях 1 и 2 заменим первый нетерминал B правой частью последней продукции: $\{S \to zAyA;\ A \to zAyS;\ A \to x;\ B \to zAy\}$.

Далее, введем новый нетерминал Y, и заменим во всех продукциях теминал y этим нетерминалом с добавлением новой продукции $Y \to y$ к множеству продукций грамматики. Окончательно грамматика в форме Грейбах, эквивалентная исходной грамматике, имеет вид: $\{S \to zAYA;\ A \to zAyS;\ A \to x;\ B \to zAY,\ Y \to y\}$.

Без доказательства приведем следующую теорему.

Теорема 6.2. Для каждой КС-грамматики G, такой, что $\varepsilon \notin L(G)$, существует эквивалентная грамматика в нормальной форме Грейбах.

2 Задания для решения в аудитории

Задание 6.1. Рассмотрите следующую грамматику:

a) $S \to ASB \mid \varepsilon,$
$A \to aAS \mid a,$
$B \to SbS \mid A \mid bb.$

b) $S \to 0A0 \mid 1B1 \mid BB,$
$A \to C,$
$B \to S \mid A,$
$C \to S \mid \varepsilon.$

c) $S \to AAA \mid B,$
$A \to aA \mid B,$
$B \to \varepsilon.$

Удалите ε-продукции. Удалите цепные продукции. Есть ли здесь бесполезные символы? Если да, удалите их. Приведите грамматику к нормальной форме Хомского. Приведите грамматику к нормальной форме Грейбах.

3 Домашнее задание

1 Рассмотрите следующую грамматику: $\{S \to aAa \mid bBb \mid \varepsilon;\ A \to C \mid a;\ B \to C \mid b;\ C \to CDE \mid \varepsilon;\ D \to A \mid B \mid ab\}$.

Удалите ε-продукции. Удалите цепные продукции. Есть ли здесь бесполезные символы? Если да, удалите их. Приведите грамматику к нормальной форме Хомского. Приведите грамматику к нормальной форме Грейбах.

2 Найдите грамматику, не содержащую бесполезных символов и эквивалентрую следующей грамматике: $\{S \to AB \mid CA;\ A \to a;\ B \to BC \mid AB;\ C \to aB \mid b\}$.

Практические занятия №7-8. Конечные автоматы. Детерминированные конечные автоматы (ДКА)

Цель занятия – дать понятие конечного автомата, разобрать формы представления таких автоматов, примеры использования.

Определение 7.1/ Детерминированный конечный автомат (распознаватель) (ДКА) – это пятерка $M = (Q, \Sigma, \theta, q_0, F)$, где:

Q – конечное множество *внутренних состояний*;

Σ – конечное множество символов, называемое *входным алфавитом*;

θ: $Q \times \Sigma \to Q$ – всюду определенная функция, называемая *функцией переходов*;

$q_0 \in Q$ – *начальное состояние*;

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

В дальнейшем под ДКА мы будем понимать только детерминированный конечный распознаватель. ДКА функционирует следующим образом. В начальный момент времени, находясь в состоянии q_0, он с помощью читающей головки обозревает самый левый символ входной строки. Головка читает строку слева направо, без возвратов, сдвигаясь вправо на один символ за каждый шаг работы автомата. В соответствии с заданной функцией переходов автомат изменяет свои внутренние состояния в процессе чтения входной строки. Прочитав ее до конца, автомат оказывается либо в одном из заключительных состояний (и в этом случае строка будет допущена или распознана автоматом), либо в каком-то другом, не заключительном (и в этом случае строка отвергается).

Для наглядного представления функционирования автомата используются **диаграммы переходов**. Это ориентированные конечные графы, в которых вершины соответствуют внутренним состояниям автомата (и помечаются соответствующими символами).

Если $M = (Q, \Sigma, \theta, q_0, F)$ – ДКА, тогда его диаграмма переходов – это граф D_M, имеющий ровно $|Q|$ вершин, каждая из которых помечена некоторым $q_i \in Q$. Для каждого правила перехода $\theta(q_i, a) = q_j$ в D_M существует дуга (q_i, q_j), помеченная символом a. Вершина, помеченная q_0, называется **начальной**, а те вершины, метки которых $q_f \in F$, называются **финальными** (или **заключительными**). Финальные вершины будем изображать двойгыми кружочками, а нефинальные – одинарными.

Нетрудно видеть, что определения ДКА с помощью пятерки $(Q, \Sigma, \theta, q_0, F)$ и посредством диаграммы переходов D_M равносильны.

Полезно ввести обобщенную функцию переходов θ^*: $Q \times \Sigma^* \to Q$.

Вторым аргументом у θ^* является не отдельный символ, а целая строка, а значение функции указывает на то внутреннее состояние, в котором окажется автомат после серии переходов в результате чтения этой строки.

ПРИМЕР 7.1. Рассмотрим ДКА $M = (\{q_0, q_1, q_2\}, \{a, b\}, \theta, q_0, \{q_1\})$, где $\theta(q_0, a) = q_0$, $\theta(q_0, b) = q_1$, $\theta(q_1, a) = q_0$, $\theta(q_1, b) = q_2$, $\theta(q_2, a) = q_2$, $\theta(q_2, b) = q_1$.

Ему соответствует диаграмма D_M (рисунок 4).

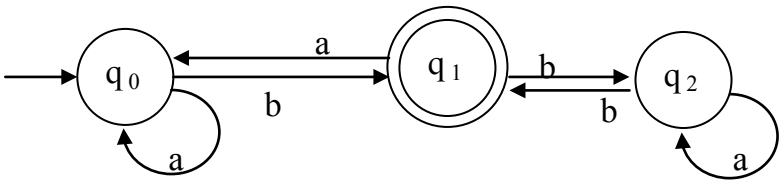


Рисунок 4 – ДКА из примера 7.1

Этот автомат распознает строки ab, bab, abbb, bbaab, но отвергает baa или bbaa.

Дадим теперь строгое определение языка, связанного с конкретным ДКА.

Определение 7.2. Язык, допустимый ДКА $M = (Q, \Sigma, \theta, q_0, F)$, – это множество всех строк из Σ^*, допустимых автоматом M, т.е. $L(M) = \{\alpha \mid \alpha \in \Sigma^* \text{ и } \theta^*(q_0, \alpha) \in F\}$.

Так как функции θ и θ^* являются всюду определенными и каждый шаг работы автомата определяется единственным образом, то любая строка $\alpha \in \Sigma^*$ после прочтения ее либо допускается (распознается) автоматом, либо отвергается им.

ПРИМЕР 7.2.

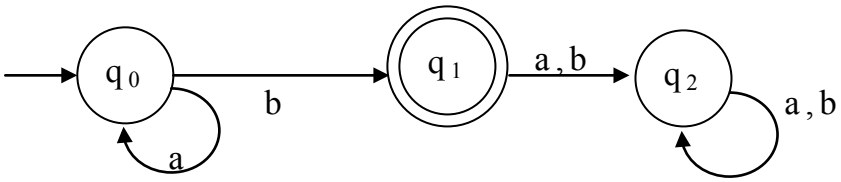


Рисунок 5 – ДКА из примера 7.2

Этой диаграмме (рисунок 5) соответствует ДКА, допускающий язык $L = \{a^n b \mid n \geq 0\}$.

Функция θ может быть задана таблицей:

	a_1	a_2	…	a_k	…	a_n
q_0						
q_1						
…						
q_i				q_j		
…						
q_m						

Здесь на пересечении i-й строки и k-го столбца находится q_j тогда и только тогда, когда $\theta(q_i, a_k) = q_j$.

Если функция θ не всюду определенная, то ее можно доопределить так, что соответствующий автомату язык не изменится.

Для любой пары *(q, a)*, для которой $\theta(q, a)$ не определено, положим $\theta(q,a) = \Delta$, где $\Delta \notin Q \cup \Sigma$. Кроме того, для любого $a \in \Sigma$ положим $\theta(\Delta, a) = \Delta$.

При этом функция θ становится всюду определенной.

ПРИМЕР 7.3. У приведенного ниже автомата (рисунок 6) функция переходов не является всюду определенной. Вводя новое состояние Δ, являющееся «ловушкой», мы доопределяем эту функцию (достройка диаграммы показана штрихами).

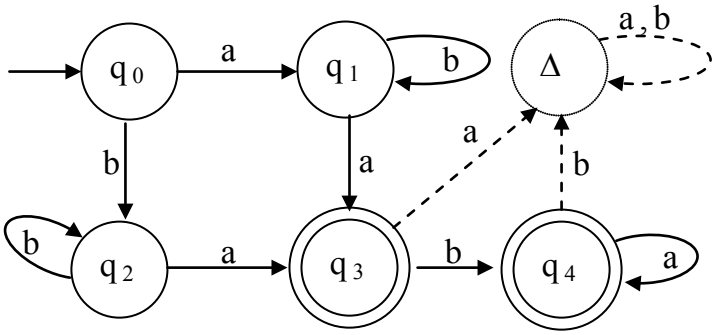


Рисунок 6 – ДКА из примера 7.3

Определение 7.2. Язык L называется **автоматным**, если существует ДКА M такой, что L = L(M).

Таким образом, семейство всех ДКА определяет класс **автоматных языков**.

Рассмотрим пример составления ДКА.

ПРИМЕР 7.4. Опишите ДКА, распознающий язык L, состоящий из строк, составленных из символов a и b, причем эти строки начинаются на aa и содержат нечетное количество символов b.

Для выделения слов, начинающихся на aa, создадим начальное состояние q_0, которое первый символ a будет переводить в состояние q_1, а второй символ a будет переводить q_1 в состояние q_2. Ясно, что все слова, которые начинаются на ab, ba, bb, сами не входят в язык L и все их продолжения также ошибочны. Заведем для них «ошибочное» состояние q_4. Остальные слова разбиваются на два класса: те, в которых четное число символов b, и те, в которых число таких символов нечетно (они и принадлежат L).

Так как после получения aa число b четно, то для представления слов первого класса будем использовать состояние q_2, а для представления слов второго – создадим состояние q_3, которое и будет заключительным. В результате получаем автомат, диаграмма которого представлена на рисунке 7.

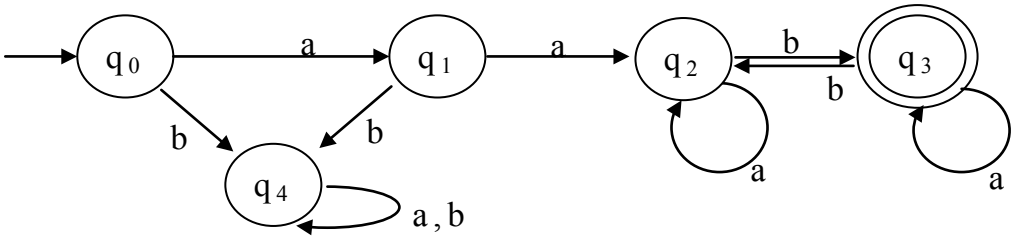


Рисунок 7 – ДКА из примера 7.4

2 Задания для решения в аудитории

Задание 7.1. Опишите ДКА, которые допускают следующие языки над алфавитом {0, 1}:

a) множество всех цепочек, содержащих три нуля подряд;
b) множество всех цепочек, в которых всякая подцепочка из пяти последовательных символов содержит хотя бы два 0;
c) множество всех цепочек, у которых на десятой позиции справа стоит 1;
d) множество цепочек, которые начинаются или оканчиваются (или и то, и другое) последовательностью 01.

Задание 7.2. Составить ДКА для поиска в строке из символов a и b подстроки abbab.

Задание 7.3. Составить ДКА для определения, является ли строка, составленная из символов a и b, чередующейся (строки abababa, babab удовлетворяют условию, а строка ababba – нет).

3 Домашнее задание

1 Опишите ДКА, которые допускают следующие языки над алфавитом {0, 1}:

a) множество цепочек, содержащих в качестве подцепочки 011;
b) множество цепочек, в которых число нулей делится на пять, а число единиц— на три.

2 Составить ДКА для определения, имеет ли строка, составленная из символов a и b, вид, подобный следующим: aabbaabbaabb или bbaabbaabb.

Список литературы

1 Соколов В. А. Формальные языки и грамматики: учебное пособие. – Ярославль : Яросл. гос. ун-т, 1998. – 152 с.

2 Карпов Ю. Г. Основы построения трансляторов. – Санкт-Петербург : БХВ-Петербург, 2005. – 272 с.

3 Ишакова Е. Н. Теория формальных языков, грамматик и автоматов. Методические указания к лабораторному практикуму – Оренбург : ГОУ ОГУ, 2005. – 54 с.

Медведев Аркадий Андреевич

ОСНОВЫ ТЕОРИИ ФОРМАЛЬНЫХ ЯЗЫКОВ

Методические рекомендации
для студентов направления 09.03.03 «Прикладная информатика»
(направленность «Прикладная информатика в дизайне»)

Редактор И. Н. Погребняк

……

Подписано в печать 13.11.18	Формат 60х84 1/16	Бумага 65 г/м2
Печать цифровая	Усл. печ. л. 2,0	Уч.-изд. л. 2,0
Заказ № 202	Тираж 25	Не для продажи

……

БИЦ Курганского государственного университета.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.