Теория формальных грамматик

1. Формальные грамматики

1.1. Определение формальной грамматики и способы ее описания

Рассмотрим некоторый алфавит X.

Словом в алфавите X называется последовательность символов алфавита. Набор допустимых слов составляет *лексику* языка.

Слова могут объединяться в более сложные конструкции – *предложения. Язык* есть множество предложений. Предложения строятся из слов и более простых предложения по правилам синтаксиса. *Синтаксис* языка представляет собой описание правильных предложений.

Алфавит, лексика и синтаксис полностью определяют набор допустимых конструкций языка и внутренние взаимоотношения между конструкциями.

Набор правил синтаксиса образует *грамматику* языка. Правила синтаксиса могу описывать либо процедуру получения правильных предложений, либо процедуру распознавания «правильности» предложений (т. е. их принадлежности данному языку). В первом случае грамматику называют *порождающей*, во втором – *распознающей.*

Пример. Грамматика G_0 задана следующим набором правил:

<предложение> → <подлежащее> <сказуемое>

<подлежащее> → <имя существительное>

<подлежащее> → <местоимение>

<имя существительное> → КОТ

<имя существительное> → ПЕС

<местоимение> → ОН

<сказуемое> → <глагольная форма>

<глагольная форма> → ИДЕТ

<глагольная форма> → ЛЕЖИТ

Знак → читается «это есть». Он делит каждое правило на две части: *правую* и *левую.*

Используя очевидные сокращения, грамматику G_0 можно записать короче:

<Пр> → <П> <С>

<П> → <ИС>

<П> → <М>

<ИС> → КОТ

<ИС> → ПЕС

<М> → ОН

<С> → <ГФ>

<ГФ> → ИДЕТ

<ГФ> → ЛЕЖИТ

Множество правил произвольной грамматики будем обозначать P. Символы, входящие в правила, образуют *словарь* V.

V = {<Пр>, <П>, <С>, <ИС>, <М>, <ГФ>, КОТ, ПЕС, ОН, ИДЕТ, ЛЕЖИТ}

Элементы словаря обычно обозначают прописными буквами.

Множество всех конечных последовательностей символов (строк) из словаря V, включая пустую строку, обозначим V^*. Элементы множества V^* обозначают строчными буквами.

Множество V делят на два подмножества:

– подмножество T символов, которые входят только в правые части правил P (T = { КОТ, ПЕС, ОН, ИДЕТ, ЛЕЖИТ});

– подмножество $N = V \setminus T$ (N = {<Пр>, <П>, <С>, <ИС>, <М>, <ГФ>}).

Символы подмножества T называются *терминальными* или *терминалами* (конечными). Символы подмножества N называются *нетерминальными* или *переменными*.

В грамматике также выделяется нетерминальный символ, называемый *начальным символом* грамматики (*аксиомой*) и обозначаемый A или S (A = {<Пр>}). Этот символ представляет определяемый язык.

Грамматику G, определяемую четверкой $G = \{V, T, P, A\}$, где элементы множества правил P имеют вид

$$x \to y, \quad (x \neq y, \quad x \in (V \setminus T)^*, \quad y \in V^*),$$

называют *порождающей*, причем правила вида $x \to y$ называют *порождающими правилами* или *правилами подстановки*.

Пример. Порождающая грамматика для арифметических выражений, использующих операции сложения и умножения, скобки и идентификаторы.

<Выражение> → <Терм>

<Выражение> → <Выражение> + <Терм>

<Терм> → <Множитель>

<Терм> → <Терм> × <Множитель>

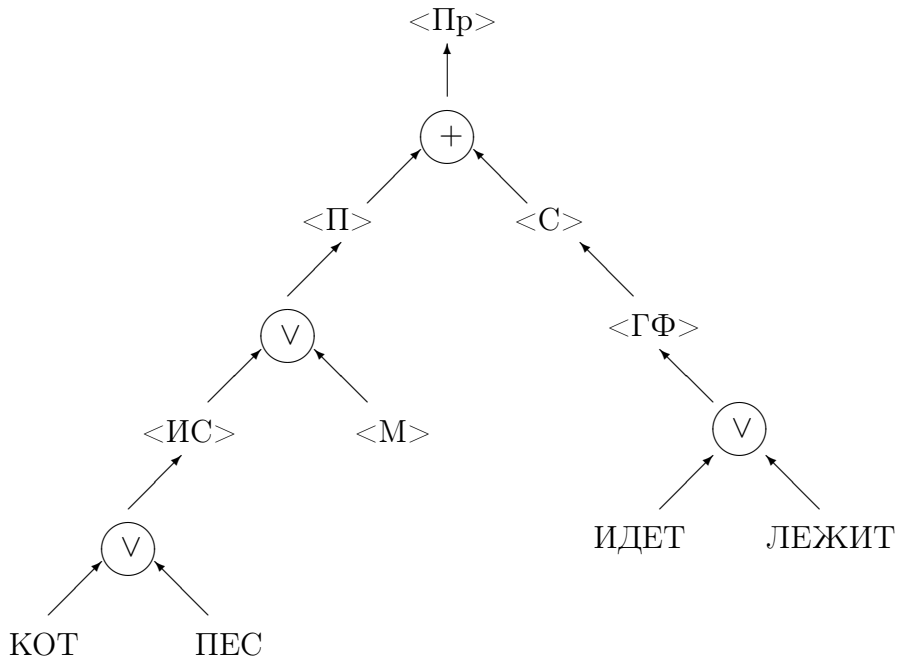
<Множитель> → <Идентификатор>

<Множитель> → (<Выражение>)

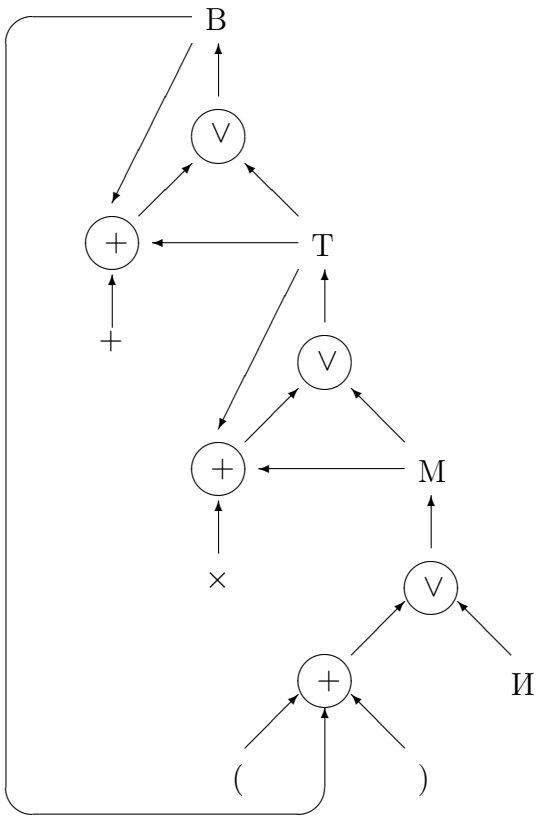
Для наглядного представления взаимосвязи между различными правилами и символами грамматику можно изобразить в виде ориентированного графа.

Символ левой части правила изображают выше символа правой части и соединяют их стрелкой. Если в правую часть входят несколько символов, то их объединяют узлом, отмеченным знаком «+». Узлы, помеченные одинаковыми терминалами, объединяются в один узел. Для изображения правил с одинаковыми левыми частями используют узел, отмеченный знаком «∨». В терминальные символы не входят никакие дуги графа, в нетерминальные входит по крайней мере одна дуга.

Пример. Рассмотрим граф грамматики G_0.



Граф грамматики G_1.



Граф грамматики G_0 ацикличен, а граф грамматики G_1 содержит циклы. Наличие циклов – следствие рекурсивности грамматики. Цикличность графа отражает существенные для анализа свойства грамматики.

1.2. Формальные языки

Порождающая грамматика G порождает язык $\mathcal{L}(G)$. Чтобы показать, как это происходит, введем дополнительные определения.

Строка $x \in V^*$ *прямо порождает* строку $y \in V^*$ (обозначается $x \Rightarrow y$), если:

$$x = pUq, \quad y = pzq,$$

где $p \in V^*$, $q \in V^*$, и существует правило $U \to z \in P$.

Иными словами, y – прямое порождение символа x, если y можно получить заменой в x нетерминального символа U строкой z в соответствии с правилом.

Пример. В грамматике G_0 строка ОН ИДЕТ – прямое порождение строки <М> ИДЕТ, полученное применением правила <М> → ОН.

Строка $x \in V^*$ *порождает* строку $y \in V^*$ (обозначается $x^* \Rightarrow y$), если существует последовательность строк $x = x_0, x_1, \ldots, x_n = y$, таких, что

$$x_i \Rightarrow x_{i+1}, \quad i = 0, 1, \ldots, n-1.$$

Такая последовательность строк называется *выводом*.

Пример. В грамматике G_0 строка ОН ИДЕТ – порождение строки <Пр>:

<Пр> ⇒ <П> <С> ⇒ <М> <С> ⇒ ОН <С> ⇒ ОН <ГФ> ⇒ ОН ИДЕТ

Строки, порожденные символом A и состоящие только из терминальных символов, составляют *язык* $\mathcal{L}(G)$ (их называют *предложениями* языка).

Пример. $\mathcal{L}(G_0)$ = {КОТ ИДЕТ, ПЕС ИДЕТ, ОН ИДЕТ, КОТ ЛЕЖИТ, ПЕС ЛЕЖИТ, ОН ЛЕЖИТ }

В языке $\mathcal{L}(G_0)$ только шесть правильных предложений, в языке $\mathcal{L}(G_1)$ их число бесконечно. Это – следствие рекурсивности грамматики G_1.

Каждому предложению соответствует по крайней мере один вывод. Однако одному предложению может соответствовать несколько выводов.

Пример. В грамматике G_0 строка ОН ИДЕТ – порождение строки <Пр>:

<Пр> ⇒ <П> <С> ⇒ <П> <ГФ> ⇒ <П> ИДЕТ ⇒ <М> ИДЕТ ⇒ ОН ИДЕТ

Вывод называется *левым*, если на каждом шаге происходит замена левой переменной, и *правым*, если происходит замена правой переменной.

Пример.

Левый вывод:

<Пр> ⇒ <П> <С> ⇒ <М> <С> ⇒ ОН <С> ⇒ ОН <ГФ> ⇒ ОН ИДЕТ

Правый вывод:

<Пр> ⇒ <П> <С> ⇒ <П> <ГФ> ⇒ <П> ИДЕТ ⇒ <М> ИДЕТ ⇒ ОН ИДЕТ

Вывод можно представить *синтаксическим деревом*, которое иначе называется *деревом вывода* или *деревом разбора*. Если хотя бы одно предложение имеет более одного синтаксического дерева, грамматику называют *неоднозначной*.

Обратная к задаче выбора – это *задача разбора*. Преобразование строки, обратное порождению, называют *приведением* (*редукцией*) строки.

Строка $y \in V^*$ *прямо приводима* к строке $x \in V^*$, если x прямо порождает y. Строка $y \in V^*$ *приводима* к строке $x \in V^*$, если x порождает y.

Основная задача синтаксического анализа состоит в отыскании разбора (вывода) для заданного предложения языка. Если разбор (вывод) существует, то

предложение синтаксически правильное. Разбор дает его структуру (синтаксическое дерево). Алгоритм, решающий задачу разбора, называют *распознавателем.*

1.3. Классификация языков по Хомскому

В 1959 году американский ученый-лингвист Н. Хомский предложил классифицировать формальные языки по типу правил порождающей грамматики.

Класс 0. Правила имеют форму

$$a \to b$$

без каких-либо ограничений на строки a и b. Языки этого класса могут служить моделью естественных языков. Разбор предложения такого языка можно выполнить с помощью *машины Тьюринга.*

Класс 1. Все правила имеют вид

$$pUq \to puq,$$

где строки p и q могут быть пустыми, u – непустая строка, U – нетерминальный символ. Порождающая грамматика с такими правилами называется грамматикой *непосредственно составляющих.* Языки, порождаемые грамматиками этого класса, называют *контекстно-зависимыми*, поскольку подстановка $U \to u$ допустима только в контексте p, q. Разбор предложения такого языка можно выполнить с помощью *линейно ограниченной машины.* В современной практике такие языки большого значения не имеют.

Класс 2. Все порождающие правила имеют вид

$$U \to u,$$

где U – нетерминальный символ, u – непустая строка. Такие грамматики и порождаемые ими языки называют *контекстно-свободными*, поскольку подстановка $U \to u$ возможна независимо от контекста.

Грамматики класса 2 являются хорошей моделью для языков программирования. Разбор предложения языка класса 2 можно выполнить, используя *автомат с магазинной (стековой) памятью.*

Класс 3. Все правила относятся к одной из двух форм

$$U \to a, \quad U \to aU' \ (U \to U'a),$$

где U и U' – нетерминальные символы, a – терминальный символ. Языки класса 3 называют *автоматными* языками. Разбор предложения языка класса 3 можно выполнить, используя *конечный автомат.*

В реальных языках программирования отдельные подмножества также можно отнести с третьему классу. Например, конечные автоматы применяются для обработки лексем.

Пример 1. Грамматику G_0 можно представить эквивалентной грамматикой класса 3:

<Пр> → КОТ <С>
<Пр> → ПЕС <С>
<Пр> → ОН <С>
<С> → ИДЕТ
<С> → ЛЕЖИТ

Пример 2. Грамматику G_1 нельзя заменить эквивалентной грамматикой третьего класса, это грамматика второго класса.

1.4. Стратегии синтаксического анализа

Рассмотрим контекстно-свободную грамматику (в их число входят и автоматные грамматики). Существуют две стратегии синтаксического анализа:

– *нисходящая* (сверху вниз), когда для данного слова, исходя из начального символа грамматики, строят вывод;

– *восходящая* (снизу вверх), когда для данного слова, исходя из символов самого слова, строят разбор.

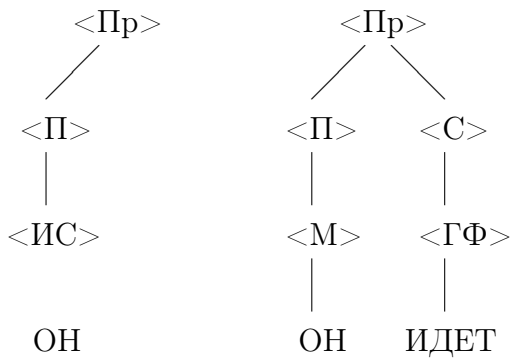
Основная черта нисходящего анализа – *целенаправленность*. На каждом шаге анализа нисходящий распознаватель формирует цель – найти вывод, начинающийся с некоторого нетерминального символа о порождающий часть входного слова. Распознаватель пытается достичь этой цели путем направленного перебора различных возможностей.

Начиная процесс анализа слова $S_1S_2\dots S_n$, распознаватель исходит из предположения, что это слово принадлежит входному языку. Цель анализа – найти вывод $A \Rightarrow S_1S_2\dots S_n$, где A – начальный символ грамматики. Если существует такой вывод, то имеется порождающее правило $A \to U_1U_2\dots U_m$ и вывод $U_1U_2\dots U_m \Rightarrow S_1S_2\dots S_n$. Среди символов $U_1U_2\dots U_m$ могут быть терминальные U_T и нетерминальные U_N. Поскольку входное слово состоит лишь из терминальных символов и в левую часть любого порождающего правила терминальные символы не входят, то каждый терминальный символ U_T олжен совпадать с одним из символов входной строки S_j, а для каждого нетерминального символа U_N должен существовать вывод $U_N \Rightarrow S_iS_{i+1}\dots S_{i+j}$. Соответственно возникает новая цель – найти все такие выводы, и т. д.

Заметим, что для каждого нетерминального символа U_N в грамматике может быть несколько правил с различными правыми частями. Какое именно правило нужно выбрать, заранее неизвестно, и при неудачном выборе правила вспомогательная цель может оказаться недостижимой. В этом случае нужно вернуться и выбрать другое правило. Может оказаться, что для данного нетерминального символа все правила приводят к неудаче. Это означает, что неправильно выбрано правило более высокого уровня, которое породило вспомогательную цель U_N. В этом случае нужно вернуться уже на два шага назад.

Процесс прекращается, когда либо найден вывод, либо перебраны все возможности и установлено, что вывода не существует.

Пример. Пусть требуется найти вывод для предложения «ОН ИДЕТ» в грамматике G_0. Слева – вспомогательная цель <ИС> недостижима, справа разбор получен.



Обычно нисходящий распознаватель просматривает символы входного слова и символы правой части слева направо. Такой распознаватель недопустим, если грамматика леворекурсивна.

Пример. Правило грамматики G_1

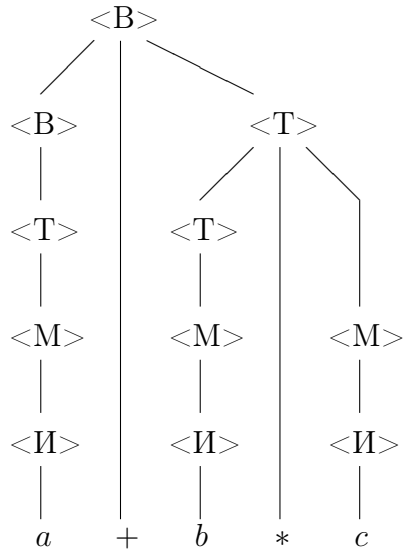
<Выражение> → <Выражение> + <Терм>

требует, чтобы для отыскания цели <Выражение> формировалась вспомогательная цель <Выражение>, и т. д., что приведет к зацикливанию программы. В этом случае нужно заменить грамматику G_1 эквивалентной грамматикой без левой рекурсии.

Общая идея восходящего анализа состоит в следующем. Распознаватель на каждом шаге отыскивает часть строки, которую можно привести к нетерминальному символу. Такую часть стрки называют *фразой*. Фраза, прямо приводимая к нетерминальному символу, называется *непосредственно приводимой*. В большинстве восходящих распознавателей отыскивается самая левая непосредственно приводивая фраза, называемая *основой*. Основа заменяется нетерминальным символом. Во вновь полученной строке опять отыскивается основа, которая также заменяется нетерминальным символом, и т. д.

Процесс продолжается либо до получения начального символа, либо до установления невозможности приведения строки к начальному символу.

Пример. Выполним анализ строки $a + b * c$ в грамматике G_1.



Восходящий анализ также может приводить к ситуации «тупика» и возврата далеко назад. Например, терминалы b и c можно было привести к нетерминалу <B> и зайти в «тупик». Чтобы этого не происходило, грамматика должна обладать определенными свойствами. Иногда применение конкретного метода возможно только после существенного изменения грамматики, при этом нужно следить, чтобы исходная и измененная грамматики были эквивалентны.

2. Автоматные языки

2.1. Конечные распознаватели

Конечный распознаватель – это пятерка объектов $A =< X, Q, \psi, q_0, F >$, где
X – входной алфавит;
Q – множество состояний;
$\psi : \ X \times Q \to Q$ – функция переходов;
$q_0 \in Q$ – начальное состояние;
$F \subseteq Q$ – множество финальных состояний.

Иными словами, конечный распознаватель – это конечный автомат без выхода, в которов выделены начальное состояние и множество финальных состояний. Как и ранее, распространим функцию переходов на множество X^*.

Конечный распознаватель $A =< X, Q, \psi, q_0, F >$ допускает слово $\alpha \in X^*$, если α переводит автомат из начального в одно из заключительных состояний, т. е. если $\psi(\alpha, q_0) \in F$. Множество всех слов, допускаемых автоматом A, образует язык, допускаемый A.

Можно дать другое определение автоматного языка – это язык, для которого существует распознающий его конечный автомат-распознаватель.

Все конечные языки являются автоматными.

Распознаватель для автоматного языка можно представить в виде таблицы переходов-выходов, столбцы которой соответствуют нетерминальным символам (состояниям), а строки – терминальным (входам). Исходное состояние – начальный символ грамматики. В клетках таблицы записано следующее состояние для пары «вход, состояние», пустые клетки соответствуют ошибочным предложениям языка. Первое состояние – начальное, финальные состояния в таблице не указываются (из них нет переходов).

Чтобы проверить, принадлежит ли слово α автоматному языку, достаточно выяснить, существует ли на диаграмме переходов-выходов путь, помеченный символами слова α и соединяющий начальную вершину с конечной. Это можно выяснить и по таблице переходов-выходов. Процедура анализа слов по диаграмме или таблице представляет собой восходящий анализ.

Пример 1.

	<Пр>	<С>
КОТ	<С>	
ПЕС	<С>	
ОН	<С>	
ИДЕТ		Разбор получен
ЛЕЖИТ		Разбор получен

Рассмотрим предложение «КОТ ИДЕТ»: по символу «КОТ» из начального состояния <Пр> мы переходим в состояние <С>, из состояния <С> по символу «ИДЕТ» – в состояние <Разбор получен>, значит, предложение принадлежит языку. Предложение «ОН КОТ» не принадлежит языку, т. к. по символу «ОН» из начального состояния <Пр> мы переходим в состояние <С>, из состояния <С> по символу «КОТ» – в пустую клетку таблицы, соответствующую ошибке.

Пример 2. Конечный автомат для анализа вещественной константы.

	1	2	3	4	5	6	7
Цифра	2	2	4	4	7	7	7
Точка	3	4	О 1	О 1	О 1	О 1	О 1
e	О 7	5	О 2	5	О 3	О 3	О 3
+, –	2	О 4	О 4	О 4	6	О 4	О 4
Конец числа		Выход	О 6	Выход	О 5	О 5	Выход

Матрица переходов выявляет ошибки в записи константы:
О 1 – лишняя точка;
О 2 – до знака порядка «e» только «.»;
О 3 – лишний знак «e»;
О 4 – знаки «+», «–» стоят неверно;
О 5 – неверно задан порядок;
О 6 – число состоит из одной точки;
О 7 – до знака «e» пусто.

Алгоритм построения автомата по грамматике

Пусть слова заданы грамматикой класса 3, т. е. правилами вида $U \to a$, $U \to U'a$, где U, U' – нетерминальные символы, a – терминальный.

Строки таблицы КР соответствуют терминальным символам, столбцы – нетерминальным. Нетерминальные символы можно закодировать целыми числами, они сопоставляются состояниям автомата.

Первый столбец заполняется по правилам $U \to a$, т. е. в строке, соответствующей терминальному символу a, стоит состояние U (или его номер).

Остальные столбцы соответствуют правилам $U \to U'a$, т. е. на пересечении строки U' и столбца a стоит состояние U.

Множество заключительных состояний соответствуют правильным словам.

Пустые клетки соответствуют ошибкам в задании слов.

2.2. Минимизация конечных распознавателей

Очевидно, что два распознавателя следует считать *эквивалентными*, если они распознают один и тот же язык. Проблема минимизации распознавателя A состоит в нахождении такого распознавателя A', который, распознавая тот же язык, имеет наименьшее число состояний. Оказывается, эта проблема только в некоторых чертах отличается от подобной проблемы для конечных автоматов.

Два состояния p и q конечного распознавателя *неотличимы* ($p = q$), если при подаче любого слова автомат из этих состояний попадает либо в заключительные, либо в незаключительные состояния, т. е.

$$(\forall \alpha \in X^*)\ \psi(\alpha, p) \in F \Leftrightarrow \psi(\alpha, q) \in F.$$

Неотличимость является отношением эквивалентности. Объединив неотличимые состояния в классы, можно построить автомат A', эквивалентный A и содержащий наименьшее число состояний. Поиск классов неотличимости можно осуществить алгоритмом Мура либо Хопкрофта, взяв в качестве начального разбиение на два класса: заключительных и незаключительных состояний.

2.3. Лемма о накачке

Лемма о накачке является важным теоретическим результатом, позволяющим во многих случаях проверить, является ли данный язык автоматным. Поскольку все конечные языки являются автоматными, эту проверку имеет смысл проводить только для бесконечных языков.

Лемма о накачке (1). *Пусть L – автоматный язык над алфавитом V. Тогда*

$$\exists n: \quad (\forall \alpha \in L : |\alpha| \geq n)(\exists u, v, w \in V^*) : \\ (\alpha = uvw)\,\&\,(|uv| \leq n)\,\&\,(|v| \geq 1)\,\&\,(\forall i :\ uv^i w \in L).$$

Лемма о накачке (2). *Пусть L – некоторый язык над алфавитом V. Если*

$$\forall n: \quad (\exists \alpha \in L : |\alpha| \geq n)(\forall u, v, w \in V^*) : \\ (\alpha = uvw)\,\&\,(|uv| \leq n)\,\&\,(|v| \geq 1)\,\&\,(\exists i :\ uv^i w \notin L),$$

то L – не автоматный.

Доказательство. Пусть L содержит бесконечное число слов. Предположим, что L распознается конечным автоматом A с n состояниями. Для проверки автоматности языка L выберем произвольное слово α длины n. Если автомат распознает L, то слово α допускается этим автоматом, т. е. в автомате A существует путь длины n из начального q_0 в одно из заключительных состояний q_K, помеченный символами слова α. Путь этот не может быть простым, т. к. он должен проходить через $n + 1$ состояние, в то время как автомат A имеет n состояний. Значит, путь содержит по крайней мере один цикл, т. е. имеет вид:

$$q_0 ... q ... q q' ... q_K.$$

Разобьем слово α на три слова: $\alpha = uvw$, где u переводит автомат из состояния q_0 в состояние q, v переводит автомат из состояния q снова в состояние q, а w переводит автомат из состояния q в q_K. Заметим, что u и w могут быть пустыми, а v – нет. Но тогда слово $uv^i w$ также переводит автомат из q_0 в q_K, т. е. автомат A допускает слово $uv^i w$ для любого i. Лемма доказана.

Термин «накачка» в названии леммы отражает возможность многократного повторения некоторой части слова в любом слове подходящей длины любого бесконечного автоматного языка.

Пример 1. С помощью леммы о накачке можно показать неавтоматность языка, порождаемого грамматикой G_1.

Для произвольного n рассмотрим выражение

$$\alpha = ((...(x_1) + x_2) + ... + x_n),$$

т. е. выражение, начинающееся с n открывающих скобок. Согласно лемме о накачке, $uv = ((...($, и язык должен допускать слова $uv^i w$. Но в любом таком слове нарушен баланс скобок.

Пример 2. $L = \{\beta c\overline{\beta}\},\ X = \{0, 1\}$ – не автоматный.

2.4. Недетерминированные конечные распознаватели

Недетерминированные конечные распознаватели (НКР) являются обобщением детерминированных. НКР могут быть двух типов: либо существует переход, помеченный пустым словом ε, либо из одного состояния существует несколько переходов, помеченных одним и тем же входным символом.

Недетерминированным конечным распознавателем называется пятерка объектов $A =< X, Q, \psi, q_0, F >$, где:

X – входной алфавит;

Q – множество состояний;

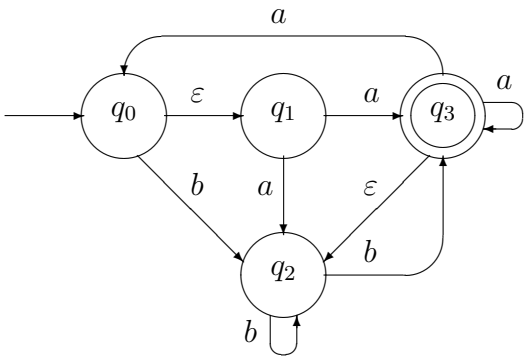
$q_0 \in Q$ – начальное состояние;

$\psi :\ (X \cup \varepsilon) \times Q \to 2^Q$ – функция переходов;

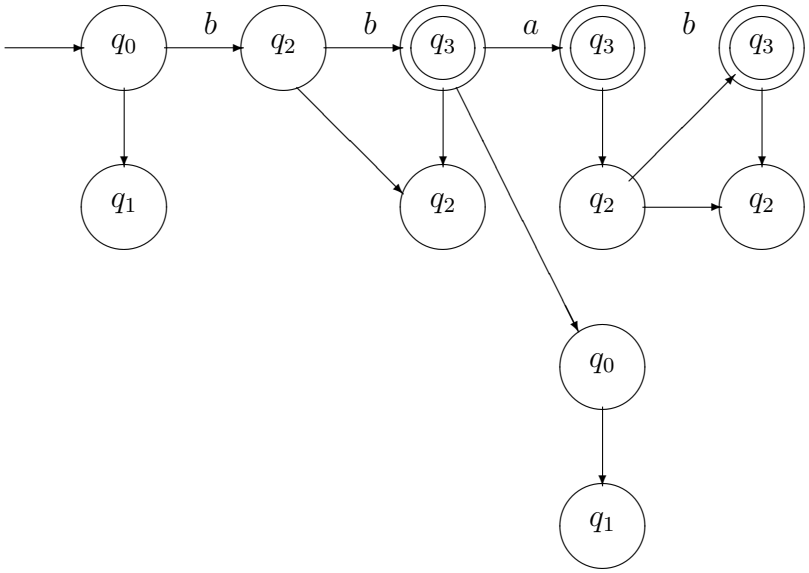
$F \subseteq Q$ – множество финальных состояний.

Здесь 2^Q – множество всех подмножеств (булеан) множества Q.

Пример. Переход из состояния q_0, помеченный пустым символом ε, говорит о том, что в любой момент, если автомат находится в состоянии q_0, он спонтанно, *без подачи входного сигнала*, может оказаться в состоянии q_1, а может и остаться в состоянии q_0. Два перехода из состояния q_1, помеченные a, разрешают автомату переход в любое из состояний q_2 или q_3 под воздействием a. Причем один из этих переходов приводит в заключительное состояние q_3, другой – нет.



Рассмотрим поведение автомата при подаче на вход слова *bbab*.



Как видно из диаграммы, возможны несколько вариантов поведения. Автомат может спонтанно выполнить переход, помеченный пустым словом ε (вертикальные дуги). Каждый переход, помеченный символом, может быть выполнен только при подаче этого входа.

НКР *допускает слово* δ, если существует путь, помеченный символами слова δ и, возможно, символом ε, из начального в одно из заключительных состояний автомата. НКР *распознает язык* L, если он распознает все слова этого языка, и только их.

Пример. Автомат из предыдущего примера допускает слова bb, $bbab$, не допускает слова ε, b.

Теорема. *Для любого недетерминированного конечного распознавателя существует эквивалентный ему детерминированный конечный распознаватель.*

Доказательство (*конструктивное*). Пусть $A_\varepsilon =< X, Q_\varepsilon, \psi_\varepsilon, q_0^\varepsilon, F_\varepsilon >$ – НКР с ε-переходами. Приведем его к НКР $A' =< X, Q', \psi', Q_0, F' >$, не содержащему ε-переходов.

Назовем ε-*замыканием* состояния $q \in Q_\varepsilon$ (обозначается $[q]$) назовем состояния, которые достижимы из q без подачи входного сигнала.

Пример. $[q_0] = \{q_0, q_1\}$, $[q_1] = \{q_1\}$, $[q_2] = \{q_2\}$, $[q_3] = \{q_2, q_3\}$.

Множеством состояний A' являются ε-замыкания состояний A_ε. Множеством начальных состояний A является $Q_0 = \bigcup\limits_{q \in [q_0]} [q]$, т. е. все состояния, достижимые из начального без подачи входного сигнала.

Пример. $Q_0 = \{[q_0], [q_1]\}$.

Множеством заключительных состояний являются такие ε-замыкания (т. е. состояния автомата A'), которые содержат хотя бы одно финальное осстояние автомата A_ε.

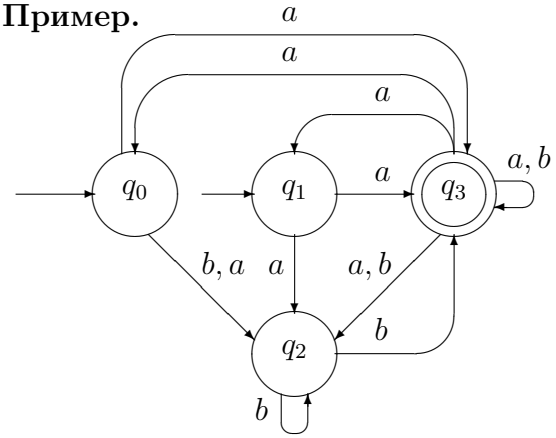
Пример. $F' = \{[q_3]\}$.

Функция переходов автомата A' имеет вид:

$$\psi'(x,[q]) = \bigcup_{s\in\psi_\varepsilon(x,[q])} [s].$$

Иными словами, при воздействии входного сигнала x автомат A' переходит из $[q]$ в ε-замыкания тех состояний, в которые исходный автомат A_ε переходит из всех состояний, достижимых из q под воздействием ε.

Пример.



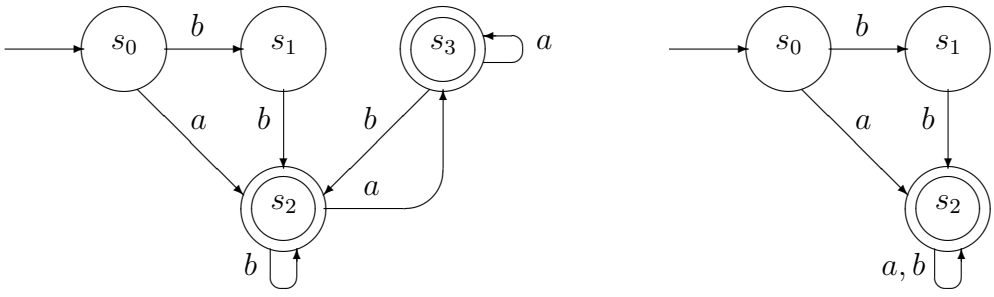
Пусть $A' =< X, Q', \psi', Q_0, F' >$ – НКР без ε-переходов. Приведем его к детерминированному автомату $A =< X, 2^{Q'}, \psi, q_0, F >$. в качестве состояний возьмем все возможные подмножества состояний исходного недетерминированного автомата A', в качестве начального состояния – множество начальных состояний исходного автомата A'. Функцию переходов определим следующим образом

$$\psi(x,Q) = \bigcup_{q\in Q} \psi'(x,q).$$

Множество заключительных состояний состоит из таких множеств состояний, которые включают хотя бы одно заключительное состояние исходного автомата.

Пример. Слева – искомый автомат, справа – он же посли минимизации,

$$s_0 = \{q_0, q_1\}, \quad s_1 = \{q_2\}, \quad s_2 = \{q_2, q_3\}, \quad s_3 = \{q_0, q_1, q_2, q_3\}$$



Теорема доказана.

2.5. Регулярные множества и регулярные выражения

Рассмотрим класс множеств слов, которые легко описать формулами некоторого вида. Такие множества называются *регулярными*.

Пусть V_1 и V_2 – множества слов. Определим три операции над этими множествами.

Объединение. $V_1 \cup V_2 = \{\alpha : \alpha \in V_1 \text{ или } \alpha \in V_1\}$.

Конкатенация. $V_1V_2 = \{\alpha\beta : \alpha \in V_1,\ \beta \in V_2\}$.

Обозначим через V^n конкатенацию n множеств V: $V^n = VV\ldots V$, $V^0 = \{\varepsilon\}$.

Итерация. $V^* = V^0 \cup V_1 \cup \ldots = \bigcup\limits_{n\geq 0} V^n$.

Пример.

$$V_1 = \{abc, ba\},\ V_2 = \{b, cb\}: \quad V_1V_2 = \{abcb, abccb, bab, bacb\};$$
$$V = \{a, bc\}: \quad V^* = \{\varepsilon, a, bc, aa, abc, bcbc, bca, aaa, aabc, ...\}.$$

Класс *регулярных множеств* над конечным словарем V определяется так:

1) $\emptyset$ – регулярное множество;

2) $\{\varepsilon\}$ – регулярное множество;

3) $(\forall\alpha \in V)$ $\{\alpha\}$ – регулярное множество;

4) если S и T – регулярные множества, то регулярны их объединение, конкатенация и итерации;

5) если множество не может быть построено конечным числом применения правил 1)–4), то оно не регулярное.

Пример. $\{ab, ba\}^*\{aa\}$, $\{b\}\{\{c\}\cup\{d, ab\}^*\}$ – регулярные множества, множество $\{a^nb^n|n > 0\}$ – не регулярное.

Любое регулярное множество можно задать специальной формулой – *регулярным выражением*.

Класс *регулярных выражений* над конечным словарем V определяется так:

1) $\emptyset$ – регулярное выражение;

2) ε – регулярное выражение;

3) $(\forall\alpha \in V)$ α – регулярное выражение;

4) если S и T – регулярные выражения, то регулярны их сумма, конкатенация и итерации;

5) если выражение не может быть построено конечным числом применения правил 1)–4), то оно не регулярное.

Пример. $(ab + ba)^*aa$, $b(c + (d, ab)^*)$ – регулярные выражения, a^nb^n – не регулярное.

Для уменьшения числа скобок используются приоритеты: итерация самая приоритетная, затем конкатенация и сумма.

Таким образом, регулярное выражение – это конечная формула, задающая бесконечное число слов, т. е. бесконечный язык. Если R – регулярное выражение, то задаваемый им язык обозначается $\overline{R}$.

Одно и то же множество слов может быть представлено различными регулярными выражениями.

Пример. aa^*a, a^*aa, a^*aaa^* описывают слова из символов a длины не менее двух.

Два регулярных выражения R_1 и R_2 называются *эквивалентными*, если и только если $\overline{R}_1 = \overline{R}_2$.

Следующая теорема дает способ определения эквивалентности двух регулярных выражений.

Теорема. *Для любых регулярных выражений R, S и T справедливо*:
1) $R + S = S + R$, $R + R = R$, $(R + S) + T = R + (S + T)$, $\emptyset + R = R$;
2) $R\varepsilon = \varepsilon R = R$, $(RS)T = R(ST)$, $\emptyset R = R\emptyset = \emptyset$;
3) $R(S + T) = RS + RT$, $(R + S)T = RT + ST$;
4) $R^* = \varepsilon + R + \ldots + R^k R^*$, $RR^* = R^* R$, $R(SR)^* = (RS)^* R$;
5) $R^* RR^* = RR^*$, $RR^* + \varepsilon = R^*$.

Пример. Применим теорему для упрощения регулярного выражения

$$b(b + aa^*b) = b(\varepsilon b + aa^*b) = b(\varepsilon + aa^*)b = ba^*b.$$

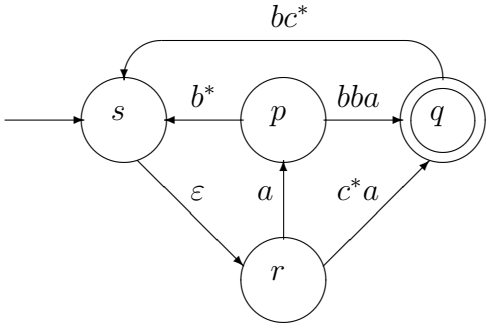
Регулярные выражения, как и конечные автоматы, задают языки. Обозначим через Λ_A множество автоматных языков, через Λ_R – множество регулярных языков.

Теорема Клини. *Классы регулярных множеств и автоматных языков совпадают, т. е.* $\Lambda_A = \Lambda_R$.

Доказательство. Докажем теорему конструктивно в два шага. На первом шаге докажем, что любой автоматный язык является регулярным множеством. На втором шаге докажем, что любое регулярное множество является автоматным языком.

Введем в рассмотрение модель графа переходов как обобщение модели конечного автомата. Граф содержит одну начальную и в общем случае несколько заключительных вершин, а дуги помечены, в отличие от конечного автомата, не символами, а регулярными выражениями. Граф переходов допускает слово α, если α принадлежит множеству слов $\overline{R}$, где $R = R_1 R_2 \ldots R_n$, R_i – регулярные выражения, которые помечают путь из начальной вершины в любую из заключительных вершин. Множество слов, допускаемых графом переходов, образуют допускаемый им язык.

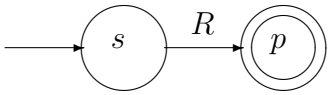
Пример. Граф переходов допускает, например, цепочку $abbca$, поскольку путь $srpsrq$, который ведет в заключительное состояние q, помечен цепочкой регулярных выражений $\varepsilon ab^*\varepsilon c^*a$.



Конечный автомат является частным случаем графа переходов. Поэтому все языки, которые допускаются автоматами, допускаются и графами переходов.

Утверждение 1. *Каждый автоматный язык является регулярным множеством, т. е.* $\Lambda_A \subseteq \Lambda_R$.

Доказательство. Граф переходов с одной начальной и одной заключительной вершиной, единственная дуга которого, исходящая из начальной и заходящая в заключительную вершину, помечена регулярным выражением R, допускает язык $\overline{R}$.

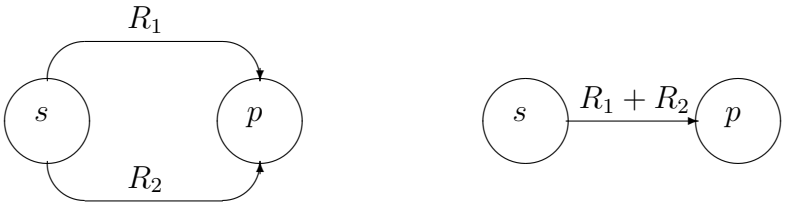


Докажем теперь, что любой граф переходов может быть приведен к такому виду без изменения допускаемого им языка.

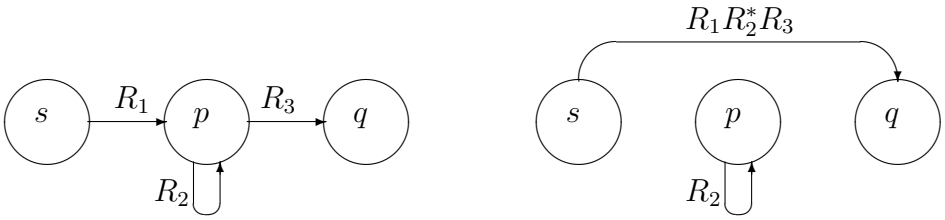
Любой граф переходов, в том числе диаграмму переходов конечного автомата, можно представить в нормализованной форме с одной начальной вершиной s_0 и одной заключительной вершиной s_K. Эти вершины добавляются в граф вместе с ε-переходами: из s_0 в начальные вершины исходного графа, из заключительных вершин исходного графа в s_K.

С графом переходов, представленным в нормализованной форме, могут быть выполнены две операции редукции – редукция ребра и редукция вершины.

Редукция дуги. Слева – подграф исходного графа, справа – он же после редукции ребра.



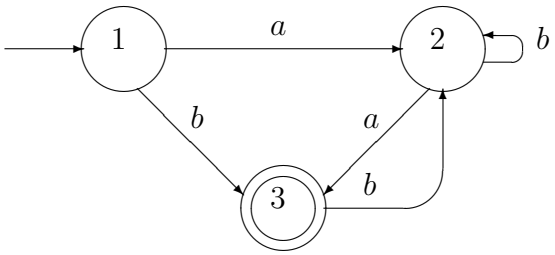
Редукция вершины. Слева – подграф исходного графа, справа – он же после редукции вершины. Далее вершина p выбрасывается как не достижимая из начальной.



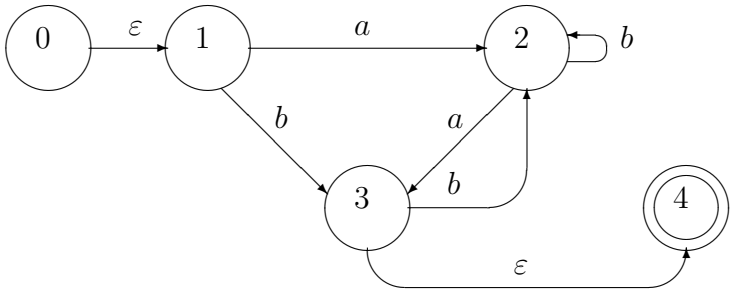
Каждая операция редукции сокращает число ребер либо вершин графа, не меняя языка, в результате получим искомый граф.

Утверждение доказано.

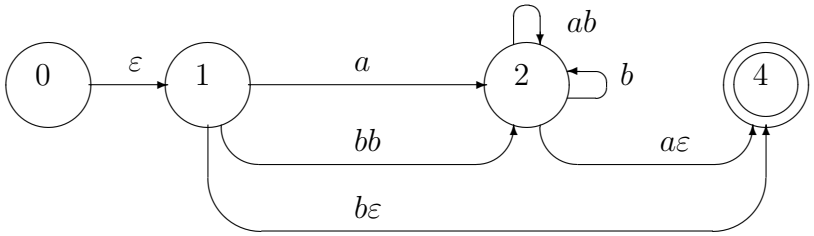
Пример. Задан конечный автомат.



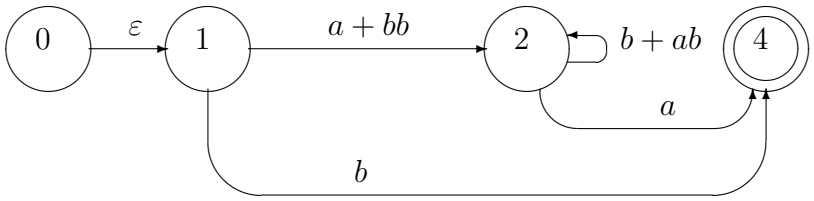
Построим его нормализованную форму.



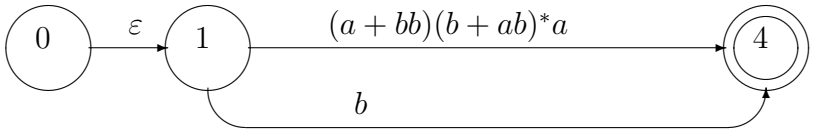
Выполним редукцию вершины 3.



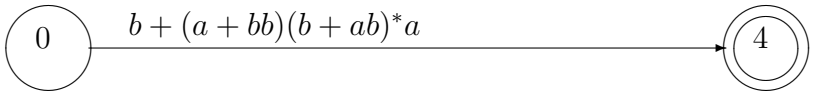
Выполним редукцию дуг и применим правило $R\varepsilon = R$.



Выполним редукцию вершины 2.



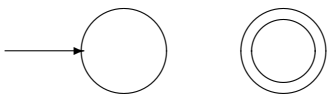
Выполним редукцию дуги и вершины 1.



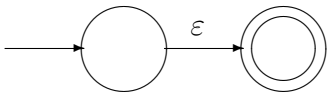
Утверждение 2. *Каждое регулярное множество является автоматным языком, т. е.* $\Lambda_R \subseteq \Lambda_A$.

Доказательство. Покажем, что для каждого регулярного выражения R может быть построен конечный автомат A_R (возможно, недетерминированный), распознающий язык, задаваемый R. Определение таких автоматов дадим рекурсивно.

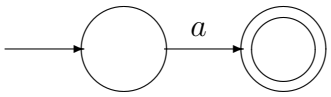
1) Если $R = \emptyset$, то A_R состоит из двух несвязанных состояний.



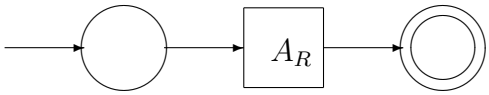
2) Для $R = \varepsilon$ автомат имеет вид



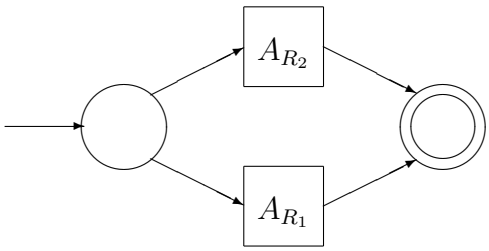
3) Для $R = a$ автомат имеет вид



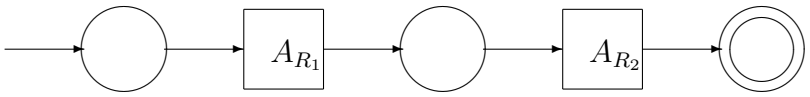
Пусть теперь автомат A_R допускает язык $\overline{R}$.



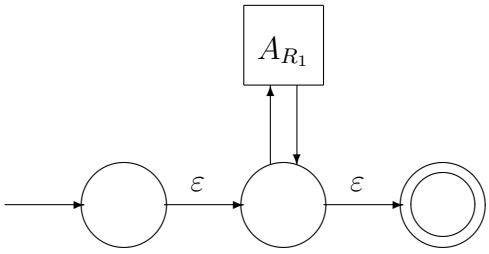
4) Если $R = R_1 + R_2$, то A_R имеет вид (начальные и заключительные состояния автоматов A_{R_1} и A_{R_2} совмещаются).



5) Если $R = R_1 R_2$, то A_R имеет вид (начальное состояние автомата A_{R_2} и заключительное состояние автомата A_{R_1} совмещаются).



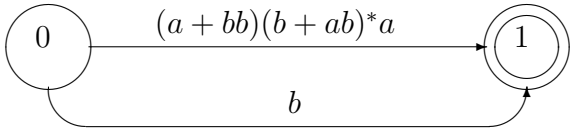
6) Если $R = R_1^*$, то A_R имеет вид (начальное и заключительное состояние автомата A_{R_1} совмещаются).



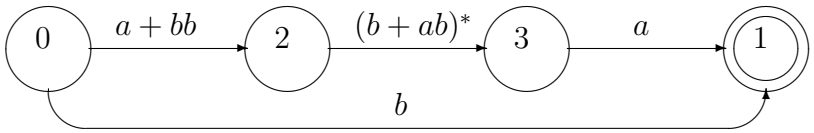
Утверждение доказано.

Пример. Пусть задано регулярное выражение $R = b + (a + bb)(b + ab)^* a$. Построим автомат A_R.

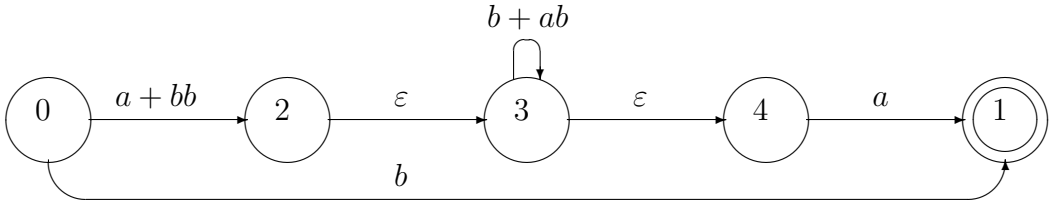
По правилу 4)



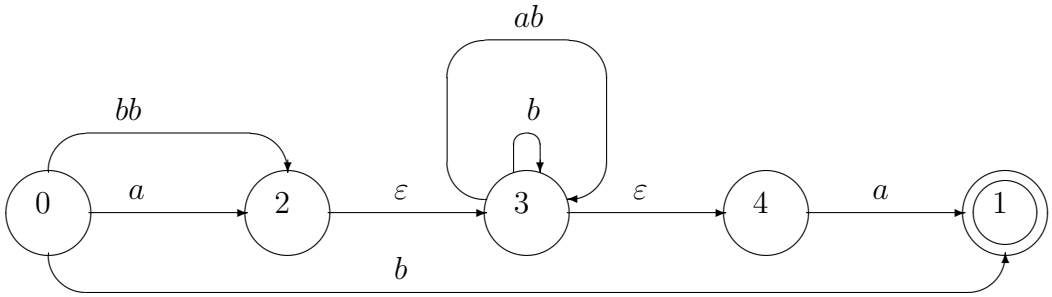
По правилу 5)



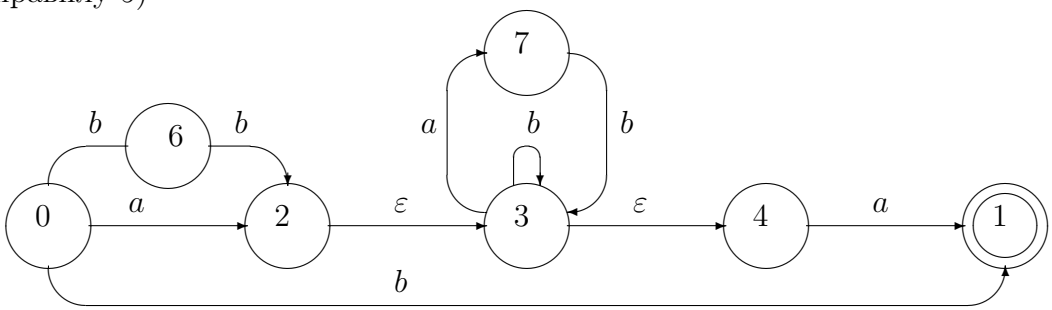
По правилу 6)



По правилу 4)



По правилу 5)



После минимизации получим автомат, эквивалентный исходному в предыдущем примере.

Теорема доказана.

Пример. Регулярные выражения применяются для поиска в текстах по образцам в системе UNIX.

– $[abc]$ – множество, состоящее из символов a, b, c;

– $[\hat{}abc]$ – множество, состоящее из всех символов, кроме a, b, c;

– $[a-z0-7]$ – множество, состоящее из всех букв (от a до z) и восьмеричных цифр (от 0 до 7);

– $\backslash s$ – непосредственное значение символа s а не его специальное управляющее значение);

– . – любой символ кроме EOF;

– $*$ – повторение 0 или более раз;

– $+$ – повторение 1 или более раз;

– ? – конец строки (*знак доллара*);
– | – символ альтернативы (или);
– ^ – начало строки.

Пример выражения в синтаксисе UNIX:

– $[a-z]\text{main}\backslash\ *($ – подстрока, включающая слово main, перед которым стоит символ (не буква); за ним может следовать произвольное число пробелов (возможно, их нет), за которыми идет открывающая скобка.

3. Контекстно-свободные грамматики и языки

3.1. Контекстно-свободные грамматики

Все порождающие правила контекстно-свободной (КС) грамматики имеют вид

$$U \to u,$$

где U – нетерминальный символ, u – любая строка.

КС-грамматики описывают более широкий класс языков, чем автоматные грамматики. Они были придуманы Н. Хомским как способ описания естественных языков, но их оказалось недостаточно. Однако они широко применяются для описания рекурсивно определяемых понятий. Рассмотрим некоторые применения КС-грамматик.

1. Грамматики используются для описания языков программирования. Существует способ превращения описания языка, вроде КС-грамматики, в синтаксический анализатор – часть компилятора, которая изучает структуру исходной программы и представляет ее с помощью дерева разбора. Это приложение является одним из самых ранних использований КС-грамматик.

Например, обычные языки программирования используют круглые и квадратные скобки во вложенном и сбалансированном виде. С помощью леммы о накачке легко показать, что множество строк из сбалансированных скобок не является регулярным языком. Оно описывается КС-грамматикой $G_b = (\{B\}, \{(,)\}, P, B)$

$$B \to BB;$$
$$B \to (B);$$
$$B \to \varepsilon.$$

Многие объекты типичного языка программирования ведут себя подобно сбалансированным скобкам, например, начала и окончания блоков кода – *begin* и *end* в языке Паскаль, { и } в Си.

Есть еще один способ балансирования, отличающийся тем, что левые скобки могут быть несбалансированы, но правые обязаны быть сбалансированы. Примером является обработка *if* и *else* в Си. Грамматика, порождающая возможные последовательности слов *if* и *else*, содержит следующие правила:

$$S \to \varepsilon;$$
$$S \to S\ S;$$
$$S \to if\ S$$
$$S \to if\ S\ else\ S.$$

2. Развитие языков описания документов (*Markup Language*) призвано облегчить электронную коммерцию тем, что ее участникам доступны соглашения о форматах ордеров, описаний товаров, и многих других видов документов. Существенной частью таких языков является *определение типа документа* (*DTD – Document Type Definition*), представляющее собой КС-грамматику, которая описывает допустимые дескрипторы (тэги – *tags*) и способы вложения их друг в друга.

Пример. Рассмотрим текст и его выражение в HTML:

Вещи, которые я *ненавижу*:

1. Кипяченое молоко.

2. Дождь со снегом.

<P>Вещи, которые я <EM>ненавижу</EM>:

<OL>

<LI> Кипяченое молоко.

<LI> Дождь со снегом.

</OL>

Парные дескрипторы <EM> и </EM> выделяют текст, <OL> и </OL> указывают на упорядоченный пронумерованный список, непарные дескрипторы <P> и <LI> вводят соответственно абзацы и элементы списка.

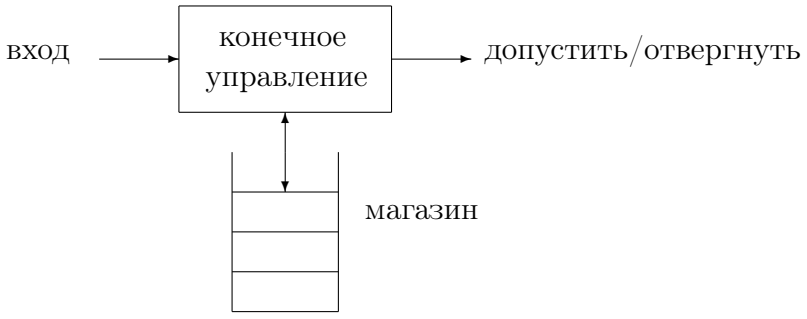
Однако, грамматики используются не только для описания языков описания документов, но и являются частью процесса использования языка. Они применяются не только для форматирования текста, а для описания его семантики, т. е. тэги говорят о том, что именно описывает документ.

Пример. Например, можно заключить в скобки <PHONE> и </PHONE> последовательности символов, интерпретируемые как телефонные номера.

Язык, задаваемый контекстно-свободной грамматикой, называется *контекстно-свободным языком.*

3.2. Магазинные автоматы

КС-языки задаются магазинными автоматами, которые являются расширением недетерминированного конечного автомата с ε-переходами. Магазин (стек) – это бесконечное устройство хранения информации, работающее по принципу «последний пришел – первый вышел» (*LIFO – last-in-first-out*). Присутствие магазина означает, что в отличие от конечного автомата магазинный автомат может «помнить» бесконечное количество информации. Неформально магазинный автомат можно рассматривать следующим образом.



«Конечное управление» читает входные символы по одному. Магазинный автомат может читать символ с вершины магазина и совершать переход на основе текущего состояния, входного символа и символа на вершине магазина. Он также может выполнить спонтанный переход по символу ε. За один переход автомат совершает следующие действия.

1) Читает и пропускает входной символ, используемый при переходе. Если в качестве входа используется ε, то входные символы не пропускаются.

2) Переходит в новое состояние, которое может и не отличаться от предыдущего.

3) Заменяет символ на вершине магазина некоторой цепочкой. Цепочкой может быть ε, что соответствует снятию с вершины магазина. Это может быть тот же символ, который был ранее на вершине магазина, т. е. магазин не изменяется. Автомат может заменить магазинный символ, что равносильно изменению вершины магазина без снятий и заталкиваний. Наконец, символ может быть заменен несколькими символами – это равносильно тому, что (возможно) изменяется символ на вершине, а затем туда помещаются один или несколько новых символов.

Пример. Рассмотрим язык палиндромов четной длины над символами 0 и 1 (например, 010010, 110011, 1001) и дадим неформальное описание автомата, распознающего этот язык.

1. Работа начинается в состоянии q_0, представляющем «догадку», что не достигнута середина входного слова. В этом состоянии символы читаются и их копии записываются в магазин.

2. В любой момент можно предположить, что достигнута середина входного слова, т. е. левая половина слова находится в магазине. Этот выбор отмечается спонтанным переходом в состояние q_1. Поскольку автомат недетерминированный, в действительности предполагаются обе возможности, т. е. можно также оставаться в состоянии q_0 и продолжать читать входные символы и записывать их в магазин.

3. В состоянии q_1 входные символы сравниваются с символами на вершине магазина. Если они совпадают, то входной символ пропускается, магазинный удаляется, и работа продолжается. Если же они не совпадают, то предположение о середине слова неверно. Эта ветвь вычислений отбрасывается, хотя другие могут продолжаться и вести к тому, что слово допустимо.

4. Если магазин опустошается и конец слова достигнут, то входное слово является палиндромом.

Дадим теперь формальное определение магазинного автомата.

Магазинный автомат – это семерка объектов $M =< V, Q, \Gamma, \psi, q_0, Z_0, F >$:

V – конечное множество входных символов (как у конечного автомата);

Q – конечное множество состояний (как у конечного автомата);

Γ – конечный магазинный алфавит (не имеет конечноавтоматного аналога) – множество символов, которые можно помещать в магазин;

ψ – функция переходов, управляющая поведением автомата: $\psi(x, Z, q) = (Z', q')$. Здесь $x \in \varepsilon \cup X$ – символ входного алфавита либо пустое слово, $Z \in \Gamma$ – магазиный символ, $q \in Q$ – текущее состояние, $q' \in Q$ – новое состояние, $\gamma \in \Gamma^*$ – цепочка магазинных символов, замещающих Z на вершине магазина. Если $\gamma = \varepsilon$, то магазинный символ снимается, если $\gamma = Z$, магазин не меняется, если $\gamma = Z_1 \dots Z_m$,

то Z заменяется на Z_m, а символы $Z_{m-1} \dots Z_1$ добавляются в магазин (Z_1 теперь на вершине);

$q_0 \in Q$ – начальное состояние;

Z_0 – начальный магазинный символ (маркер дна), в начале работы в магазине находится только этот символ;

F – множество заключительных состояний.

Пример. Автомат из предыдущего примера выглядит следующим образом.

$$M =< \{0,1\}, \{q_0, q_1, q_2\}, \{0, 1, Z_0\}, \psi, q_0, Z_0, \{q_2\} > .$$

Функция ψ определяется правилами:

– $\psi(0, Z_0, q_0) = (0Z_0, q_0)$, $\psi(1, Z_0, q_0) = (1Z_0, q_0)$. Одно из этих правил применяется в начале работы, когда автомат находится в состоянии q_0 и обозревает начальный символ Z_0 на вершине магазина. Читается первый символ и помещается в магазин; Z_0 остается под ним для отметки дна.

– $\psi(0, 0, q_0) = (00, q_0)$, $\psi(0, 1, q_0) = (01, q_0)$, $\psi(1, 0, q_0) = (10, q_0)$, $\psi(1, 1, q_0) = (11, q_0)$. Эти правила позволяют оставаться в состоянии q_0 и читать входные символы, помещая каждый из них на вершину магазина над предыдущим верхним символом.

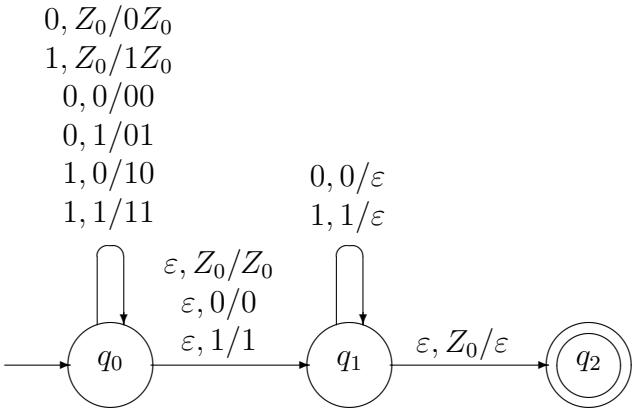
– $\psi(\varepsilon, Z_0, q_0) = (Z_0, q_1)$, $\psi(\varepsilon, 0, q_0) = (0, q_1)$, $\psi(\varepsilon, 1, q_0) = (1, q_1)$. Эти правила позволяют автомату спонтанно (без чтения входа) переходить из состояния q_0 в состояние q_1, не изменяя верхний символ магазина, каким бы он ни был.

– $\psi(0, 0, q_1) = (\varepsilon, q_1)$, $\psi(1, 1, q_1) = (\varepsilon, q_1)$. В состоянии q_1 входные символы проверяются на совпадение с символами на вершине магазина. При совпадении последние выталкиваются.

– $\psi(\varepsilon, Z_0, q_1) = (\varepsilon, q_2)$. Наконец, если обнаружен маркер дна магазина Z_0 и автомат находится в состоянии q_1, то обнаружен палиндром. Автомат переходит в заключительное состояние q_2, магазин опустошается.

Магазинный автомат также может быть представлен диаграммой. На дугах пишется $(x, Z/\gamma)$ – вход и магазинный символ, по которому происходит переход, и через слэш – цепочка, замещающая верхний магазинный символ.

Пример. Автомат из предыдущего примера имеет вид (ля упрощения записи дуги не дублируются, на одной дуге выписаны все возможные пометки).



В отличие от конечного автомата, где для описания дальнейшего поведения автомата достаточно знать его состояние, для магазинного автомата значимым является и содержимое магазина. Полезно знать и непрочитанную часть входа.

Конфигурация магазинного автомата представляется тройкой (q, γ, w) где q – состояние, γ – содержимое магазина, w – непрочитанная часть входного слова.

Определим отношение «$\vdash$» (*непосредственной выводимости*) следующим образом: если $\psi(a, X, q) = (\alpha, p)$, то $\forall w \in V^*$, $\beta \in \Gamma^*$ $(q, aw, X\beta) \vdash (p, w, \alpha\beta)$ ($(p, w, \alpha\beta)$ *непосредственно выводима из* $(q, aw, X\beta)$).

Это отношение отражает идею того, что, прочитывая на входе символ a (возможно, ε), и заменяя X на вершине магазина цепочкой α можно перейти из состояния q в состояние p. Заметим, что оставшаяся часть входа w и содержимое магазина под его вершиной β не влияют в данный момент на действия автомата, они просто сохраняются и, возможно, будут влиять на события в дальнейшем.

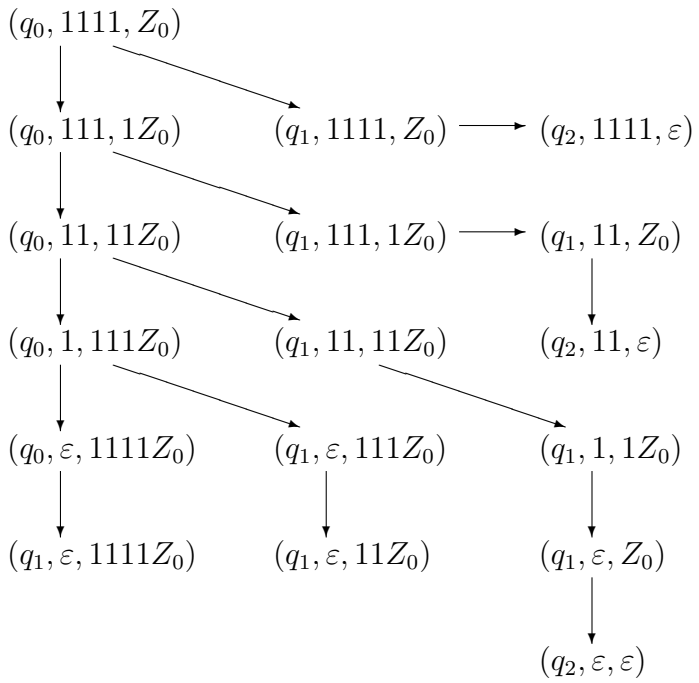
Любую конфигурацию вида (q_0, w, Z_0) называют *начальной*, а любую конфигурацию вида $(q_f, \varepsilon, \varepsilon)$ – *заключительной*. Обратим внимание, что заключительная конфигурация – это конфигурация с пустым магазином. Таким образом, множество всех заключительных конфигураций находится во взаимно-однозначном соответствии с множеством всех заключительных состояний автомата. Из заключительной конфигурации не может быть выведена ни одна конфигурация. Не заключительную конфигурацию магазинного автомата, из которой не выводима ни одна конфигурация, называют *тупиковой*.

Выводом на множестве конфигураций магазинного автомата M называют последовательность $C_0, C_1, \ldots, C_n, \ldots$ (конечную или бесконечную) таких его конфигураций, что $\forall i \geq 0 : C_i \vdash C_{i+1}$ (если C_{i+1} существует).

Конфигурацию C' называют *выводимой* из конфигурации C, $(C \vdash^* C')$ если существует связывающий их вывод. В частности, любая конфигурация выводима сама из себя.

Таким образом, понятие выводимости для конфигураций автомата полностью аналогично таковому понятию для грамматики.

Пример. Рассмотрим работу автомата из предыдущего примера для входа 1111.



Здесь мы получили единственную цепочку, приводящую к заключительной конфигурации $(q_2, \varepsilon, \varepsilon)$, остальные заканчиваются тупиковыми конфигурациями (или мы не попали в заключительное состояние, или прочитан не весь вход, или магазин не пуст).

Входное слово w называют *допустимым словом* магазинного автомата, если на множестве конфигураций автомата существует вывод, связывающий начальную конфигурацию (q_0, w, z_0) с заключительной конфигурацией $(q_f, \varepsilon, \varepsilon)$.

Язык, допускаемый автоматом $L(M)$ – это множество его допустимых слов.

3.3. Эквивалентность магазинных автоматов и контектно-свободных грамматик

Магазинный автомат M называют *эквивалентным КС-грамматике* G, если язык, допускаемый M, совпадает с языком, порождаемым G т. е. если $L(M) = L(G)$.

Заметим, что языки, допускаемые автоматами по заключительному состоянию, эквивалентны языкам, допускаемым по пустому магазину. Далее мы покажем, что языки, допускаемые по пустому магазину, эквивалентны языкам, допускаемым грамматиками.

3.3.1. От грамматики к автомату

По данной грамматике $G =< V, T, P, A >$ строится магазинный автомат, имитирующий ее левые порождения. Любую левовыводимую цепочку, которая не является терминальной, можно записать в виде $vU\alpha$, где U – крайний слева нетерминал, v – цепочка терминалов слева от U, α – цепочка терминалов и переменных

справа. $U\alpha$ называется *остатком* этой левовыводимой цепочки. У терминальной левовыводимой цепочки остатком является ε.

Идея построения магазинного автомата по грамматике состоит в том, чтобы магазинный автомат имитировал последовательность левовыводимых цепочек, используемых в грамматике для порождения данной нетерминальной цепочки w. Остаток каждой цепочки $U\alpha$ появляется в магазине с переменной U на вершине. При этом v «представлен» прочитанными на входе символами, а символы цепочки w после v считаются непрочитанными.

Предположим, что автомат находится в конфигурации $(q, y, U\alpha)$, представляющей левовыводимую цепочку $vU\alpha$. Он угадывает правило грамматики, используемое для расширения U, скажем, $U \to \beta$. Переход автомата состоит в том, что U на вершине магазина заменяется цепочкой β, и достигается конфигурация $(q, y, \beta\alpha)$. Состояние при этом не меняется, и вообще, у этого автомата всего одно состояние q.

Теперь $(q, y, \beta\alpha)$ может не быть представлением следующей левовыводимой цепочки, поскольку β может начинаться с терминальных символов. Также β может вообще не содержать переменных, а α может начинаться с терминалов. Все терминалы в начале цепочки $\beta\alpha$ нужно удалить до появления следующей переменной на вершине магазина.Эти терминалы сравниваются со следующими входными символами для того, чтобы убудиться, что наши предположения о левом порождении входной цепочки w правильны, и правило $U \to \beta$ выбрано верно; в противном случае эта ветвь вычислений отбрасывается.

Если таким способом нам удается угадать левое порождение w, то в конце мы дойдем до левовыводимой цепочки w. В этот момент все символы в магазине или расширены (если это переменные) или совпали с выходными (если это терминалы). Магазин пуст, автомат допускает слово w по пустому магазину.

Дадим формальное онисание алгоритма.

Алгоритм построения магазинного автомата по контектно-свободной грамматике

Пусть $G =< V, T, P, A >$ – КС-грамматика. Построим магазинный автомат $M =< T, \{q\}, V, \psi, q, S >$, который допускает $L(G)$ по пустому магазину. Функция переходов ψ определяется следующим образом:

– $\psi(\varepsilon, U, q) = \{(\beta, q) | U \to \beta \in P\}$, $\forall U \in V \setminus T$;

– $\psi(x, x, q) = \{(\varepsilon, q)\}$, $\forall x \in T$;

других правил перехода нет.

Пример. Порождающая грамматика для арифметических выражений, использующих операции сложения и умножения, скобки и идентификаторы.

<Выражение> → <Терм>

<Выражение> → <Выражение> + <Терм>

<Терм> → <Множитель>

<Терм> → <Терм> × <Множитель>

<Множитель> → <Идентификатор>

<Множитель> → (<Выражение>)

<Идентификатор> → <Буква>

<Идентификатор> → <Идентификатор><Буква>
<Идентификатор> → <Идентификатор><Цифра>
<Буква> → a ... <Буква> → z
<Цифра> → 0 ... <Цифра> → 9
Сократим запись:
<В> → <Т>
<В> → <В> + <Т>
<Т> → <М>
<Т> → <Т> × <М>
<М> → <И>
<М> → (<В>)
<И> → <Б>
<И> → <И><Б>
<И> → <И><Ц>
<Б> → a ... <Б> → z
<Ц> → 0 ... <Ц> → 9

Преобразуем эту грамматику в магазинный автомат. Входной алфавит X – это все терминальные символы грамматики, т. е.

$$V = \{a, \ldots, z, 0, \ldots, 9, (,), +, \times\}.$$

Магазинный алфавит Г – это терминальные символы и переменные:

$$\Gamma = V \cup \{<В>,<Т>,<М>,<И>,<Б>,<Ц>\}.$$

Начальный символ Z_0=<В>.

Функция переходов определяется следующим образом:
1) $\psi(\varepsilon, <В>, q) = \{(<Т>, q), (<В> + <Т>, q)\}$;
2) $\psi(\varepsilon, <Т>, q) = \{(<М>, q), (<Т> \times <М>, q)\}$;
3) $\psi(\varepsilon, <М>, q) = \{(<И>, q), (<(В)>, q)\}$;
4) $\psi(\varepsilon, <И>, q) = \{(<Б>, q), (<И><Б>, q), (<И><Ц>, q)\}$;
5) $\psi(\varepsilon, <Б>, q) = \{(a, q), \ldots, (z, q)\}$;
6) $\psi(\varepsilon, <Ц>, q) = \{(0, q), \ldots, (9, q)\}$;
7) $\psi(a, a, q) = (\varepsilon, q), \ldots, \psi(z, z, q) = (\varepsilon, q)$;
8) $\psi(0, 0, q) = (\varepsilon, q), \ldots, \psi(9, 9, q) = (\varepsilon, q)$;
9) $\psi((, (, q) = (\varepsilon, q)$;
10) $\psi(),), q) = (\varepsilon, q)$;
11) $\psi(+, +, q) = (\varepsilon, q)$;
12) $\psi(\times, \times, q) = (\varepsilon, q)$.

Здесь первые шесть переходов появились по первому правилу алгоритма, последние шесть – по второму. Других переходов в автомате нет.

Прочитаем слово $(a1 + b \times c) \times a + c$. Работу автомата представим в виде таблицы, в которой представлена конфигурация: остаток строки и состояние магазина. Состояние автомата всегда одно и то же.

Вход	Магазин
$(a1+b\times c)\times a+c$	<В>
$(a1+b\times c)\times a+c$	<В> + <Т>
$(a1+b\times c)\times a+c$	<Т> + <Т>
$(a1+b\times c)\times a+c$	<Т> × <М> + <Т>
$(a1+b\times c)\times a+c$	<М> × <М> + <Т>
$(a1+b\times c)\times a+c$	(<В>) × <М> + <Т>
$a1+b\times c)\times a+c$	<В>) × <М> + <Т>
$a1+b\times c)\times a+c$	<В> + <Т>) × <М> + <Т>
$a1+b\times c)\times a+c$	<Т> + <Т>) × <М> + <Т>
$a1+b\times c)\times a+c$	<М> + <Т>) × <М> + <Т>
$a1+b\times c)\times a+c$	<И> + <Т>) × <М> + <Т>
$a1+b\times c)\times a+c$	<И><Ц> + <Т>) × <М> + <Т>
$a1+b\times c)\times a+c$	<Б><Ц> + <Т>) × <М> + <Т>
$a1+b\times c)\times a+c$	a<Ц> + <Т>) × <М> + <Т>
$1+b\times c)\times a+c$	<Ц> + <Т>) × <М> + <Т>
$1+b\times c)\times a+c$	1 + <Т>) × <М> + <Т>
$+b\times c)\times a+c$	+ <Т>) × <М> + <Т>
$b\times c)\times a+c$	<Т>) × <М> + <Т>
$b\times c)\times a+c$	<Т> × <М>) × <М> + <Т>
$b\times c)\times a+c$	<М> × <М>) × <М> + <Т>
$b\times c)\times a+c$	<И> × <М>) × <М> + <Т>
$b\times c)\times a+c$	<Б> × <М>) × <М> + <Т>
$b\times c)\times a+c$	b × <М>) × <М> + <Т>
$\times c)\times a+c$	× <М>) × <М> + <Т>
$c)\times a+c$	<М>) × <М> + <Т>
$c)\times a+c$	<И>) × <М> + <Т>
$c)\times a+c$	<Б>) × <М> + <Т>
$c)\times a+c$	c) × <М> + <Т>
$)\times a+c$	) × <М> + <Т>
$\times a+c$	× <М> + <Т>
$a+c$	<М> + <Т>
$a+c$	<И> + <Т>
$a+c$	<Б> + <Т>
$a+c$	a + <Т>
$+c$	+ <Т>
c	<Т>
c	<М>
c	<И>
c	<Б>
c	c

Автомат моделирует левый вывод

<В> ⇒ <В> + <Т> ⇒ <Т> + <Т> ⇒ <Т> × <М> + <Т> ⇒
<М> × <М> + <Т> ⇒ (<В>) × <М> + <Т> ⇒

$$(<\text{В}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow (<\text{В}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow$$
$$(<\text{Т}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow (<\text{М}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow$$
$$(<\text{И}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow (<\text{И}><\text{Ц}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow$$
$$(<\text{Б}><\text{Ц}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow (\text{a}<\text{Ц}> + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow$$
$$(a1 + <\text{Т}>) \times <\text{М}> + <\text{Т}> \Rightarrow (a1 + <\text{М}>) \times <\text{М}> + <\text{Т}> \Rightarrow$$
$$(a1 + <\text{И}>) \times <\text{М}> + <\text{Т}> \Rightarrow (a1 + <\text{Б}>) \times <\text{М}> + <\text{Т}> \Rightarrow$$
$$(a1 + b) \times <\text{М}> + <\text{Т}> \Rightarrow (a1 + b) \times <\text{И}> + <\text{Т}> \Rightarrow$$
$$(a1 + b) \times <\text{Б}> + <\text{Т}> \Rightarrow (a1 + b) \times c + <\text{Т}> \Rightarrow (a1 + b) \times c + <\text{М}> \Rightarrow$$
$$(a1 + b) \times c + <\text{И}> \Rightarrow (a1 + b) \times c + <\text{Б}> \Rightarrow (a1 + b) \times c + a.$$

3.3.2. От автомата к грамматике

Идея построения грамматики по автомату основана на том, что выталкивание одного символа из магазина вместе с прочтением некоторого входа является основным событием в процессе работы автомата.

Будем рассматривать магазинный автомат, как «игрока», ставящего цели следующего вида: «находясь в состоянии q и имея верхний символ магазина Z, перейти в состояние s». Условимся записывать такую цель в виде тройки $[qZs]$. Как может игрок достичь поставленной цели? Если в автомате есть переход

$$\psi(a, Z, q) = (X_1 X_2 \dots X_k, r),$$

где $X_i \in \Gamma$ – магазинные символы, то по данной функции перехода автомат перейдет в состояние r.

Если $r = s$ то цель достигнута, иначе ставим цель $[rX_1s_1]$, достигнув ее, ставим цель $[s_1X_2s_2]$, и т. д. Достигнув цели $[s_{k-1}X_ks]$, игрок достигает и первоначальной цели $[qZs]$. Так как рассматривается допуск с пустым магазином, то символы X_i должны по очереди покинуть магазин, и только в этом случае будет достигнута «глобальная» цель магазинного автомата: $[q_0Z_0q_f]$, $q \in F$: находясь в начальном состоянии и имея верхним символом магазина начальный магазинный символ, попасть в одно из заключительных состояний, опустошив магазин. Так как, вообще говоря, игрок не знает наперед последовательность состояний $s_1, s_2, \dots s_{k-1}$, ведущих к цели, то он может перебрать все такие последовательности.

Дадим формальное описание алгоритма.

Алгоритм построения контектно-свободной грамматики по магазинному автомату

Пусть дан магазинный автомат $M =< V, Q, \Gamma, \psi, q_0, Z_0, F >$. Определим КС-грамматику $G_M =< V, T, P, A >$.

Терминальный алфавит T совпадает со входным алфавитом V автомата.

Нетерминальный алфавит N есть множество, находящееся во взаимно-однозначном соответствии с множеством всех упорядоченных троек $[qZs]$, где $q, s \in Q$, $Z \in \Gamma$, дополненное специальным стартовым символом A – аксиомой грамматики.

Множество правил вывода P грамматики строится так:

а) если есть переход $\psi(a, Z, q) = (X_1X_2 \dots X_k, r)$, то в P записываются все правила вида:

$$[qZs_k] \to a[rX_1s_1][s_1X_2s_2] \dots [s_{k-1}X_ks_k]$$

для любой последовательности k состояний $s_1, \ldots, s_k$ множества Q;

б) для каждого перехода $\psi(a, Z, q) = (\varepsilon, r)$ в P добавляется правило

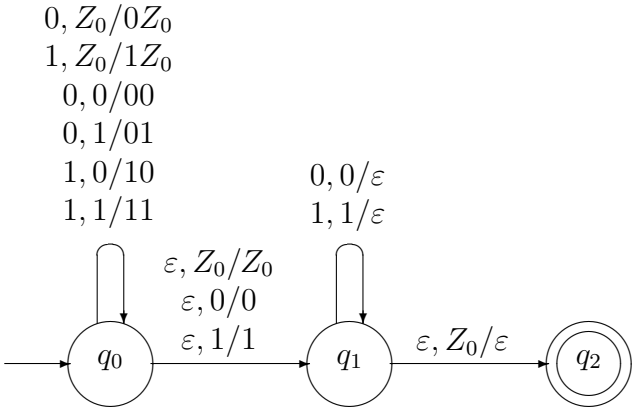
$$[qZr] \to a;$$

в) для каждого заключительного состояния $q_f \in F$ вводится правило

$$A \to [q_0 Z_0 q_f];$$

г) никаких других правил нет.

Пример. Рассмотрим магазинный автомат для распознавания палиндромов. Автомат допускает по пустому магазину.



По первому правилу имеем:

$$\begin{array}{lll}
1) & [q_0 Z_0 s_2] \to 0[q_0 0 s_1][s_1 Z_0 s_2], & s_1, s_2 \in \{q_0, q_1, q_2\}; \\
2) & [q_0 Z_0 s_2] \to 1[q_0 1 s_1][s_1 Z_0 s_2], & s_1, s_2 \in \{q_0, q_1, q_2\}; \\
3) & [q_0 0 s_2] \to 0[q_0 0 s_1][s_1 0 s_2], & s_1, s_2 \in \{q_0, q_1, q_2\}; \\
4) & [q_0 1 s_2] \to 0[q_0 0 s_1][s_1 1 s_2], & s_1, s_2 \in \{q_0, q_1, q_2\}; \\
5) & [q_0 0 s_2] \to 1[q_0 1 s_1][s_1 0 s_2], & s_1, s_2 \in \{q_0, q_1, q_2\}; \\
6) & [q_0 1 s_2] \to 1[q_0 1 s_1][s_1 1 s_2], & s_1, s_2 \in \{q_0, q_1, q_2\}; \\
7) & [q_0 Z_0 s_1] \to [q_1 Z_0 s_1], & s_1 \in \{q_0, q_1, q_2\}; \\
8) & [q_0 0 s_1] \to [q_1 0 s_1], & s_1 \in \{q_0, q_1, q_2\}; \\
9) & [q_0 1 s_1] \to [q_1 1 s_1], & s_1 \in \{q_0, q_1, q_2\}.
\end{array}$$

Обратим внимание, что в первых шести строках на самом деле содержится не по одному, а по девять правил (число размещений с повторениями двух элементов множества из трех), а в последних четырех строках – по три правила.

По второму правилу имеем:

$$\begin{array}{ll}
10) & [q_1 0 q_1] \to 0; \\
11) & [q_1 1 q_1] \to 1; \\
12) & [q_1 Z_0 q_2] \to \varepsilon.
\end{array}$$

По третьему правилу имеем:

$$13) \quad A \to [q_0 Z_0 q_2].$$

Выведем в этой грамматике цепочку 0110:

$$A \Rightarrow_{13)}\ [q_0Z_0q_2] \Rightarrow_{1)\ s_1=q_0,\ s_2=q_2}\ 0[q_00q_0][q_0Z_0q_2] \Rightarrow_{5)\ s_1=q_1,\ s_2=q_0}$$
$$01[q_01q_1][q_10q_1][q_0Z_0q_2] \Rightarrow_{9)\ s_1=q_1}\ 01[q_11q_1][q_10q_1][q_0Z_0q_2] \Rightarrow_{11}\ 011[q_10q_1][q_0Z_0q_2] \Rightarrow_{10)}$$
$$0110[q_0Z_0q_2] \Rightarrow_{7)\ s_1=q_2}\ 0110[q_1Z_0q_2] \Rightarrow_{12)}\ 0110\varepsilon = 0110.$$

3.4. Нормальные формы контекстно-свободных грамматик

В этом разделе мы покажем, что каждый КС-язык, не допускающий ε, порождается грамматикой, все правила которой имеют одну из двух форм

$$U \to u;$$
$$U \to VW$$

где U, W, V – нетерминальные символы, u – терминальный символ. Эта форма называется *нормальной формой Хомского.*

3.4.1. Удаление бесполезных символов

Определение. Символ X называется *полезным* в грамматике $G =< V, T, P, A >$, если существует вывод вида $A \Rightarrow^* \alpha X\beta \Rightarrow^* w$, где $w \in T^*$.

Отметим, что X может быть как переменной, так и терминалом, а выводимая цепочка $\alpha X\beta$ – первой или последней в выводе.

Определение. Если символ X не является полезным, то он называется называется *бесполезным.*

Очевидно, что исключение бесполезных символов не изменяет порождаемого грамматикой языка, поэтому все такие символы можно удалить.

Определение. Символ X называется *порождающим* в грамматике $G =< V, T, P, A >$, если существует вывод вида $X \Rightarrow^* w$, где $w \in T^*$.

Заметим, что каждый терминал является порождающим символом, поскольку выводится сам из себя за 0 шагов.

Определение. Символ X называется *достижимым* в грамматике $G =< V, T, P, A >$, если существует вывод вида $A \Rightarrow^* \alpha X\beta$ для некоторых α, β.

Полезный символ, как следует из определения, является и порождающим, и достижимым. Поэтому, если удалить из грамматики сначала непорождающие, а затем недостижимые символы, то останутся только полезные, как будет показано далее.

Пример. Рассмотрим грамматику $G =< \{A, B, C, a, b\}, \{a, b\}, P, A >$ с правилами

$$A \to BC;$$
$$A \to a;$$
$$B \to b.$$

Все символы, кроме C, являются порождающими, поскольку терминалы a и b порождают самих себя, A порождает a, и B порождает b. Удаление B приводит к удалению правила $A \to BC$, т.е. к грамматике $G =< \{A, B, a, b\}, \{a, b\}, P, A >$ с правилами

$$A \to a;$$
$$B \to b.$$

Нетрудно получить, что из A достижимы лишь A и a. Удаление символов B и b приводит к грамматике $G =< \{A, a\}, \{a\}, P, A >$ с единственным правилом

$$A \to a.$$

Она порождает язык $\{a\}$, как и исходная грамматика.

Заметим, что если начать с проверки достижимости, то все символы грамматики окажутся достижимыми. Если затем удалить C как непорождающий символ, то останется грамматика с бесполезными символами B и b.

Поиск порождающих символов.

Шаг 1. Каждый терминальный символ является порождающим.

Шаг 2. Пусть есть правило $U \to \alpha$ и известно, что каждый символ в α является порождающим. Тогда U – порождающий. Пусть $U \to \varepsilon$, тогда U – порождающий.

Поиск достижимых символов.

Шаг 1. Символ A является достижимым.

Шаг 2. Пусть переменная U достижима. Тогда если $U \to \alpha$, все символы из α тоже достижимы. каждый символ в α является порождающим. Тогда U –

Правильность данных алгоритмов легко доказывается индукцией по длине вывода.

3.4.2. Удаление ε-правил

Определение. *ε-правилом* называется правило вида $U \to \varepsilon$.

Вообще, ε-правила бывают удобны для описания грамматики, однако, они могут быть удалены специальным образом без изменения языка, порождаемого грамматикой. Единственным исключением является случай, когда пустое слово ε входит в язык. Таким образом, для любой грамматики G с языком $L(G)$ можно найти грамматику G', не содержащую ε-правил, с языком $L(G') = L(G) \setminus \{\varepsilon\}$. Это различие легко устраняется добавлением правила $A \to \varepsilon$.

Определение. Переменная U называется *ε-порождающей*, если $U \Rightarrow^* \varepsilon$.

Если U – ε-порождающая, то где бы в правилах она ни встречалась, из нее можно вывести ε. Пусть правило имеет вид $V \to \alpha U \beta$, тогда существует также правило $V \to \alpha\beta$. Тогда можно использовать оба правила: $V \to \alpha U \beta$ и $V \to \alpha\beta$, и не разрешать выводить ε из U.

Поиск ε-порождающих символов.

Шаг 1. Если в G есть правило $U \to \varepsilon$, то U – ε-порождающая.

Шаг 2. Если в G есть правило $B \to C_1 C_2 \dots C_k$, где все C_i являются ε-порождающими, то B – ε-порождающая.

Отметим, что ε-порождающим символом может быть только переменная, так что рассматриваем только правила, в правой части которых нет терминалов.

Правильность алгоритма доказывается индукцией по длине вывода ε.

Пример. Рассмотрим грамматику

$$\begin{aligned} A &\to BC \\ B &\to bBB \\ B &\to \varepsilon \\ C &\to cCC \\ C &\to \varepsilon \end{aligned}$$

Здесь B и C являются ε-порождающими по шагу 1, и A является ε-порождающей по шагу 2, поскольку существует правило $A \to BC$. Таким образом, все переменные являются ε-порождающими.

Из правила $A \to BC$ получаются три правила

$$A \to BC$$
$$A \to B$$
$$A \to C$$

Из правила $B \to bBB$ получаются три правила

$$B \to bBB$$
$$B \to bB$$
$$B \to b$$

Аналогично, из правила $C \to cCC$ получаются три правила

$$C \to cCC$$
$$C \to cC$$
$$C \to c$$

Теперь можно удалить ε-правила. В результате получаем грамматику G' вида

$$A \to BC$$
$$A \to B$$
$$A \to C$$
$$B \to bBB$$
$$B \to bB$$
$$B \to b$$
$$C \to cCC$$
$$C \to cC$$
$$C \to c$$

3.4.3. Удаление цепных правил

Определение. *Цепным правилом* называется правило вида $U \to W$, где U и W являются переменными.

Цепные правила могут быть удобны для построения грамматики, но могут и усложнять некоторые доказательства и создавать излишние шаги в выводе.

Пример. Рассмотрим грамматику арифметических выражений G_1

<В> → <Т>
<В> → <В> + <Т>
<Т> → <М>
<Т> → <Т> × <М>
<М> → <И>
<М> → (<В>)
<И> → <Б>
<И> → <И><Б>

<И> → <И><Ц>
<Б> → a ... <Б> → z
<Ц> → 0 ... <Ц> → 9

Она содержит четыре цепных правила:
<В> → <Т>
<Т> → <М>
<М> → <И>
<И> → <Б>

Можно избавиться от этих правил, заменив в правой части переменные всеми возможными способами.

Правило	Замена
<И> → <Б>	<И> → a ... <И> → z
<М> → <И>	<М> → a ... <М> → z <М> → <И><Б> <М> → <И><Ц>
<Т> → <М>	<Т> → a ... <Т> → z <Т> → <И><Б> <Т> → <И><Ц> <Т> → (<В>)
<В> → <Т>	<В> → a ... <В> → z <В> → <И><Б> <В> → <И><Ц> <В> → (<В>) <В> → <Т> × <М>

В итоге получаем грамматику
<В> → a ... <В> → z
<В> → <И><Б>
<В> → <И><Ц>
<В> → (<В>)
<В> → <Т> × <М>
<В> → <В> + <Т>
<Т> → a ... <Т> → z
<Т> → <И><Б>
<Т> → <И><Ц>
<Т> → (<В>)
<Т> → <Т> × <М>
<М> → a ... <М> → z
<М> → <И><Б>
<М> → <И><Ц>
<М> → (<В>)
<И> → a ... <И> → z
<И> → <И><Б>
<И> → <И><Ц>
<Б> → a ... <Б> → z

<Ц> → 0 ... <Ц> → 9

Такая техника удаления цепных правил работает довольно часто, но теряет смысл, если в грамматике есть цикл из цепных правил, например,

$$A \to B$$
$$B \to C$$
$$C \to A$$

В этом случае необходимо выявить все пары переменных (U, W), для которых $U \Rightarrow^* W$ лишь с использованием цепных правил. Заметим, что вывод $U \Rightarrow^* W$ возможен и без использования цепных правил, например, $U \to BW$ и $B \to \varepsilon$. Определив все такие пары, можно для любого вывода

$$U \Rightarrow B_1 \Rightarrow \ldots \Rightarrow B_k \Rightarrow \alpha,$$

где $B_k \to \alpha$ – нецепное правило, добавить правило $U \to \alpha$.

Определение. *Цепной парой* назовем пару переменных (U, W), для которых $U \Rightarrow^* W$ лишь с использованием цепных правил.

Поиск цепных пар.

Шаг 1. Для любой переменной (U, U) является цепной парой, поскольку $U \Rightarrow^* U$ за нуль шагов.

Шаг 2. Если пара (U, W) – цепная, и $W \to Y$ – цепное правило. Тогда пара (U, Y) также цепная.

Пример. Рассмотрим все цепные правила предыдущего примера.

<В> → <Т>
<Т> → <М>
<М> → <И>
<И> → <Б>

На шаге 1 получаем цепные пары (<В>,<В>), (<Т>,<Т>), (<М>,<М>), (<И>,<И>). На втором шаге получаем следующие пары:

(<В>,<В>) и <В> → <Т> дают (<В>,<Т>);
(<В>,<Т>) и <Т> → <М> дают (<В>,<М>);
(<В>,<М>) и <М> → <И> дают (<В>,<И>);
(<Т>,<Т>) и <Т> → <М> дают (<Т>,<М>);
(<Т>,<М>) и <М> → <И> дают (<Т>,<И>);
(<М>,<М>) и <М> → <И> дают (<М>,<И>).

Итак, получено 10 цепных пар.

Удаление цепных правил.

Начало. Дана грамматика $G =< V, T, P, A >$.

Шаг 1. Найти все цепные пары грамматики G.

Шаг 2. Для каждой пары (U, W) добавить в P' все правила $U \to \alpha$, где $W \to \alpha$ – нецепное правило. Заметим, что для пар (U, U) в P' просто добавляются все нецепные правила.

Конец. Грамматика $G' =< V, T, P', A >$ не содержит цепных правил.

3.4.4. Нормальная форма Хомского

Нами изучены различные правила упрощения грамматик. Если требуется избавиться от ε-правил, цепных правил и бесполезных символов, то эти преобразования выполняются в следующем порядке:

1. Удаление ε-правил.
2. Удаление цепных правил.
3. Удаление бесполезных символов.

Данные три шага должны быть выполнены именно в таком порядке, иначе в грамматике могут остаться элементы, которые требуется удалить. Назовем грамматику, преобразованную таким образом, *приведенной грамматикой*. Все правила такой грамматики имеют вид $U \to u$, где u – терминал, либо $U \to \alpha$, где длина слова α не менее двух.

Из приведенной грамматики можно получить *нормальную форму Хомского*, т.е. грамматику, все правила которой имеют одну из двух форм

$$U \to u$$
$$U \to VW$$

где U, W, V – нетерминальные символы, u – терминальный символ.

Получение нормальной формы Хомского.

Начало. Дана приведенная грамматика $G =< V, T, P, A >$.

Шаг 1. Для каждого терминала x_i ввести переменную X_i и правило $X_i \to x_i$.

Шаг 2. Заменить в правилах $U \to \alpha$, где длина слова α не менее двух, все терминалы на соответствующие переменные.

Шаг 3. Каждое правило $U \to B_1 \dots B_k$, где B_i – переменные, заменить набором правил:

$$U \to B_1 C_1$$
$$C_1 \to B_2 C_2$$
$$\dots$$
$$C_{k-3} \to B_{k-2} C_{k-2}$$
$$C_{k-2} \to B_{k-1} B_k$$

Конец. НФХ получена.

Пример. Рассмотрим грамматику арифметических выражений.

<В> $\to a \dots$ <В> $\to z$
<В> $\to$ <И><Б>
<В> $\to$ <И><Ц>
<В> $\to$ (<В>)
<В> $\to$ <Т> $\times$ <М>
<В> $\to$ <В> $+$ <Т>
<Т> $\to a \dots$ <Т> $\to z$
<Т> $\to$ <И><Б>
<Т> $\to$ <И><Ц>
<Т> $\to$ (<В>)
<Т> $\to$ <Т> $\times$ <М>
<М> $\to a \dots$ <М> $\to z$

<М> → <И><Б>
<М> → <И><Ц>
<М> → (<В>)
<И> → a ... <И> → z
<И> → <И><Б>
<И> → <И><Ц>
<Б> → a ... <Б> → z
<Ц> → 0 ... <Ц> → 9
Выполняя шаг 1, добавим новые переменные и правила
<Left> → (
<Right> →)
<Plus> → +
<Times> → ×

На шаге 2 выполним замены правил

Правило	Замена
<В> → (<В>)	<В> → <Left><В><Right>
<В> → <Т> × <М>	<В> → <Т> <Times> <М>
<В> → <В> + <Т>	<В> → <В> <Plus> <Т>
<Т> → (<В>)	<Т> → <Left><В><Right>
<Т> → <Т> × <М>	<Т> → <Т> <Times> <М>
<М> → (<В>)	<М> → <Left><В><Right>

На шаге 3 выполним замены правил

Правило	Замена
<В> → <Left><В><Right>	<В> → <Left><U_1> <U_1> → <В><Right>
<В> → <Т> <Times> <М>	<В> → <Т> <U_2> <U_2> → <Times> <М>
<В> → <В> <Plus> <Т>	<В> → <В> <U_3> <U_3> → <Plus> <Т>
<Т> → <Left><В><Right>	<Т> → <Left><U_1>
<Т> → <Т> <Times> <М>	<Т> → <Т> <U_2>
<М> → <Left><В><Right>	<М> → <Left><U_1>

В результате получаем грамматику в НФХ
<В> → a ... <В> → z
<В> → <И><Б>
<В> → <И><Ц>
<В> → <В> → <Left><U_1>
<В> → <Т> <U_2>
<В> → <В> <U_3>
<Т> → a ... <Т> → z
<Т> → <И><Б>
<Т> → <И><Ц>

<Т> → <Left><U_1>
<Т> → <Т> <U_2>
<М> → a ... <М> → z
<М> → <И><Б>
<М> → <И><Ц>
<М> → <Left><U_1>
<И> → a ... <И> → z
<И> → <И><Б>
<И> → <И><Ц>
<Б> → a ... <Б> → z
<Ц> → 0 ... <Ц> → 9
<U_1> → <В><Right>
<U_2> → <Times> <М>
<U_3> → <Plus> <Т>
<Left> → (
<Right> →)
<Plus> → +
<Times> → ×

3.5. Лемма о накачке для контекстно-свободных языков

3.5.1. Деревья разбора

Рассмотрим грамматику $G =< V, T, P, A >$.

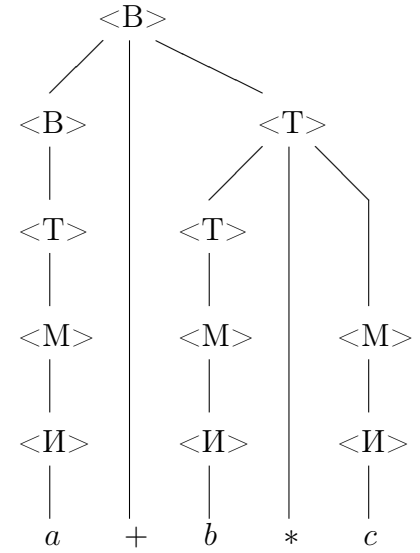
Определение. *Дерево разбора* – это ориентированное упорядоченное дерево со следующими свойствами

1. Каждый внутренний узел отмечен переменной из V.

2. Каждый лист отмечен либо переменной, либо терминалом, либо ε. Если лист отмечен ε, он должен быть единственным сыном своего родителя.

3. Если внутренний узел отмечен U, й его сыновья отмечены слева направо X_1, X_2, ..., X_n, то $U \to X_1X_2 \dots X_n \in P$.

Пример. Дерево разбора строки $a + b * c$ в грамматике G_1.



Определение. *Крона дерева разбора* – это последовательность отметок его листьев слева направо.

Крона дерева всегда является словом, выводимым из переменной, отмечающей корень.

Особый интерес представляют деревья, все листья которых отмечены терминалом, а корень – аксиомой грамматики. Кроны всех таких деревьев образуют язык, порождаемый грамматикой.

Одно из применений НФХ состоит в том, чтобы преобразовать все деревья разбора в бинарные. Такие деревья имеют ряд удобных свойств, одно из которых используется в доказательстве леммы о накачке.

Утверждение. *Пусть дано дерево разбора НФХ-грамматики* $G =< V, T, P, A >$, *и пусть кроной дерева является терминальное слово* w. *Если* n – *высота дерева, то* $|w| \leq 2^{n-1}$.

Доказательство. Индукция по n.

База индукции. Пусть $n = 1$. Такое дерево состоит из корня и листа, отмеченного терминалом u. В этом случае $w = u$, т.е. $|u| = 1 = 2^{1-1}$.

Индуктивный переход. Пусть утверждение верно для высоты $1, 2, \ldots, n-1$. Рассмотрим дерево высоты n. Для корня использовано правило $A \to BC$. Высота левого и правого поддеревьев корня не больше $n-1$, значит, по индукционному предположению, они имеют кроны длины не более, чем 2^{n-2}. Длина кроны всего дерева есть сумма длин крон поддеревьев, т.е. не превосходит $2^{n-2} + 2^{n-2} = 2^{n-1}$.

3.5.2. Лемма о накачке

Лемма о накачке для КС-языков подобна лемме о накачке для регулярных языков. Суть ее в том, что в каждом длинном слове КС-языка можно выделить две достаточно короткие цепочки, одновременное накачивание которых (т.е. повторение их одно и то же число раз) порождает слова из того же языка.

Лемма о накачке для КС-языков (1). *Пусть* L – *КС-язык над алфавитом* V. *Тогда*

$$\exists n: \quad (\forall \alpha \in L : |\alpha| \geq n)(\exists u, v, w, x, y \in V^*) : \\ (\alpha = uvwxy) \,\&\, (|uvx| \leq n) \,\&\, (|vx| \geq 1) \,\&\, (\forall i : \ uv^i wx^i y \in L).$$

Лемма о накачке для КС-языков (2). *Пусть* L – *некоторый язык над алфавитом* V. *Если*

$$\forall n: \quad (\exists \alpha \in L : |\alpha| \geq n)(\forall u, v, w, x, y \in V^*) : \\ (\alpha = uvwxy) \,\&\, (|uvx| \leq n) \,\&\, (|vx| \geq 1) \,\&\, (\exists i : \ uv^i wx^i y \notin L),$$

то L – *не контекстно-свободный.*

Доказательство. Рассмотрим КС-язык L. Сначала приведем грамматику языка L к нормальной форме Хомского. Это невозможно только для случаев $L = \emptyset$ и $L = \{\varepsilon\}$, однако, для таких языков лемма справедлива, поскольку длинных слов в них нет.

Пусть НФХ-грамматика $G =< V, T, P, A >$ имеет m переменных и порождает язык $L \setminus \{\varepsilon\}$. Выберем $n = 2^m$. Пусть $\alpha \in L : |\alpha \geq n$. По предыдущему утверждению высота дерева разбора этой цепочки не менее $m+1$. Пусть эта высота равна

$k+1$, где $k \geq m$. Тогда самый длинный путь в дереве разбора отмечен (начиная с корня) переменными $A_0 = A$, A_1, …, A_k и терминалом a. Поскольку всего переменных m, то среди $A_0, A_1, \ldots, A_k$ как минимум две переменные совпадают. Пусть $A_i = A_j$, где $k - m \leq i \leq j \leq k$. Если таких пар вершин несколько, то выберем пару с наибольшими индексами, т.е. ближайшую к кроне.

Теперь разделим крону α на 5 частей:

– часть w – крона дерева с корнем A_j;

– части v и x находятся слева и справа от w и вместе образуют крону дерева с корнем A_i;

– часть u лежит слева от v, часть y – справа от x, и вместе все пять частей образуют крону всего дерева.

Поскольку $A_i = A_j$, по данному дереву разбора можно строить новые деревья. Если заменить поддерево с корнем A_i на поддерево с корнем A_j, то мы получим дерево с кроной $uwy = uv^0wx^0y$. Замена поддерева с корнем A_j на поддерево с корнем A_i приведет к дереву с кроной $uvvwxxy = uv^2wx^2y$. Вторая такая замена породит дерево с кроной $uvvvwxxxy = uv^3wx^3y$, и вообще, $l-1$ таких замен позволят получить дерево с кроной uv^lwx^ly.

Осталось показать, что $|uvx| \leq n$. Поскольку мы выбирали A_i как можно ближе к кроне дерева, поэтому высота дерева с корнем A_i не больше, чем $m+1$ (на этом пути не больше, чем $m+1$ переменных и один терминал). Согласно утверждению, длина кроны такого дерева не превосходит $2^m = n$.

Пример. Рассмотрим язык $\{0^k1^k2^k, k \geq 1\}$. Если он контекстно-свободный, то существует целое n из леммы о накачке. Рассмотрим слово $\alpha = 0^n1^n2^n$. При любом разбиении $\alpha = uvwxy$ согласно лемме середина слова vwx не содержит одновременно нули и двойки, поскольку $|vwx| \leq 0$. Тогда одновременно накачиваться могут либо нули и/или единицы и/или двойки, и в полученных словах число нулей или двоек всегда останется неизменным, тогда как число других символов увеличится. В итоге слова не будут принадлежать рассматриваемому языку.