Формальные языки и грамматики

Цепочки символов. Операции над цепочками символов

Алфавит - это конечное множество символов, которое в дальнейшем будем обозначать **V**. Предполагается, что термин "символ" имеет достаточно ясный интуитивный смысл и не нуждается в дальнейшем уточнении.

Цепочкой символов *в алфавите V* называется любая конечная последовательность символов этого алфавита. Далее цепочки символов будем обозначать греческими буквами: **α**, **β**, **γ**, …
Цепочка, которая не содержит ни одного символа, называется ***пустой цепочкой***. Для ее обозначения будем использовать символ **ε**.

Более формально цепочка символов в алфавите **V** определяется следующим образом:
(1) **ε** - цепочка в алфавите **V**;
(2) если **α** - цепочка в алфавите **V** и **a** - символ этого алфавита, то **αa** - цепочка в алфавите **V**;
(3) **β** - цепочка в алфавите **V** тогда и только тогда, когда она является таковой в силу (1) и (2).

Длиной цепочки называется число составляющих ее символов. Например, если α = abcdefg, то длина α равна 7. Длину цепочки α будем обозначать | α |. Длина ε равна 0.

Основной операцией над цепочками символов является операция ***конкатенации*** - это дописывание второй цепочки в конец первой, т.е. если α и β - цепочки, то цепочка αβ называется их конкатенацией. Например, если α = ab и β = cd, то αβ = abcd. Для любой цепочки α всегда αε = εα = α.
Операция конкатенации не обладает свойством коммутативности, т.е. αβ≠βα, но обладает ассоциативностью (αβ)γ = α(βγ).

Обращением *(*или ***реверсом****)* цепочки α называется цепочка, символы которой записаны в обратном порядке. Обращение цепочки α будем обозначать $\alpha^{R.}$
Например, если α = abcdef, то α^R = fedcba. Для пустой цепочки: ε = ε^R.
Для операции обращения справедливо следующее равенство "a,b: $(ab)^R = b^R a^R$.

N-ой степенью цепочки a (будем обозначать α^n) называется конкатенация n цепочек α.
$\alpha^0 = \varepsilon$; $\alpha^n = \alpha\alpha^{n-1} = \alpha^{n-1}\alpha$.

Понятие языка. Формальное определение языка. Методы задания языков

В общем случае ***язык*** - это заданный набор символов и правил, устанавливающих способы комбинации этих символов между собой для записи осмысленных текстов. Основой любого естественного или искусственного языка является алфавит, определяющий набор допустимых символов языка.

Язык в алфавите **V** - это подмножество цепочек конечной длины из множества всех цепочек над алфавитом **V**. Из этого определения следует два вывода: во-первых, множество цепочек языка не обязано быть конечным; во-вторых, хотя каждая цепочка символов, входящих в язык, обязана иметь конечную длину, эта длина может быть сколь угодно большой и формально ничем не ограничена.
Все существующие языки попадают под это определение. Большинство реальных естественных и искусственных языков содержат бесконечное множество цепочек. Часто цепочку символов, принадлежащую заданному языку, называют ***предложением*** языка.

Обозначим через $\mathbf{V}^*$ множество, содержащее все цепочки в алфавите **V**, включая пустую цепочку **ε**. Например, если V = {0,1}, то V* = {ε, 0, 1, 00, 11, 01, 10, 000, 001, 011, ...}.

Обозначим через $\mathbf{V}^+$ множество, содержащее все цепочки в алфавите **V**, исключая пустую цепочку **ε**. Следовательно, верно равенство $\mathbf{V^* = V^+ \cup \{\varepsilon\}}$.

Известно несколько различных *способов описания языков*:
1. Перечислением всех допустимых цепочек языка.
2. Указанием способа порождения цепочек языка (заданием грамматики языка).
3. Определением метода распознавания языка.

Первый из методов является чисто формальным и на практике не применяется, т.к. большинство языков содержат бесконечное число допустимых цепочек и перечислить их просто невозможно.

Второй способ предусматривает некоторое описание правил, с помощью которых строятся цепочки языка. Тогда любая цепочка, построенная с помощью этих правил из символов алфавита языка, будет принадлежать заданному языку.

Более всего интересующий нас третий способ предусматривает построение некоторого логического устройства (распознавателя) - автомата, который на входе получает цепочку символов, а на выходе выдает ответ: принадлежит или нет эта цепочка заданному языку. Например, читая этот текст, вы сейчас в некотором роде выступаете в роли распознавателя (надеемся, что ответ о принадлежности текста русскому языку будет положительным).

Синтаксис и семантика языка

Синтаксис языка - это набор правил, определяющий допустимые конструкции языка. Синтаксис определяет «форму языка» - задает набор цепочек символов, которые принадлежат языку. Чаще всего синтаксис языка можно задать в виде строгого набора правил, но полностью это утверждение справедливо только для чисто формальных языков.
Например, строка «3+2» является арифметическим выражением, «3 2 +» - не является.

Семантика языка - это раздел языка, определяющий значение предложений языка. Семантика определяет «содержание языка» - задает значение для всех допустимых цепочек языка.
Например, используя семантику алгебры мы можем сказать, что строка «3+2» есть сумма чисел 3 и 2, а также то, что «3+2 = 5» - это истинное выражение.

Лексика - это совокупность слов (словарный запас) языка. Слово или лексическая единица (лексема) языка - это конструкция, которая состоит из элементов алфавита языка и не содержит в себе других конструкций.

Определение грамматики. Форма Бэкуса - Наура

Грамматика - это описание способа построения предложений некоторого языка. Иными словами, грамматика - это математическая система, определяющая язык.

Формально ***порождающая грамматика*** **G** - это четверка **G (VT, VN, P, S)**, где
VT - множество *терминальных символов (терминалов)*,
VN - множество *нетерминальных символов (нетерминалов)*, не пересекающееся с VT,
P - множество правил (продукций) грамматики вида $\alpha \to \beta$, где $\alpha \in V^+$, $\beta \in V^*$,
S - *целевой* (*начальный) символ* грамматики, $S \in VN$.

Для записи правил вывода с одинаковыми левыми частями

$$\alpha \to \beta_1 \square \alpha \to \beta_2 \ \ldots \ \alpha \to \beta_n$$

будем пользоваться сокращенной записью

$$\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n.$$

Каждое β_i , i = 1, 2, ... ,n , будем называть *альтернативой* правила вывода из цепочки α.
Такую форму записи правил грамматики называют ***формой Бэкуса-Наура***. Она предусматривает также, что все нетерминальные символы берутся в <> скобки.

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

Классификация грамматик и языков по Хомскому

Формальные грамматики классифицируются по структуре их правил. Если все без исключения правила грамматики удовлетворяют некоторой заданной структуре, то ее относят к определенному типу. Достаточно иметь в грамматике одно правило, не удовлетворяющее требованиям структуры правил, и она уже не попадет в заданный тип.

По классификации Хомского выделяют четыре типа грамматик.

ТИП 0:
Грамматика **G** = (**VT**, **VN**, **P**, **S**), **V** = **VN** ∪ **VT** называется ***грамматикой типа 0***, если на ее правила вывода не накладывается никаких ограничений, т.е. правила имеют вид $\alpha \to \beta$, где $\alpha \in V^+$, $\beta \in V^*$.

ТИП 1:
Грамматику типа 1 можно определить как контекстно-зависимую либо как неукорачивающую.
Грамматика **G** = (**VT**, **VN**, **P**, **S**), **V** = **VN** ∪ **VT** называется ***контекстно-зависимой***, если каждое правило из P имеет вид $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, где $\alpha_1, \alpha_2 \in V^*$, $A \in VN$, $\beta \in V^+$.
Грамматика **G** = (**VT**, **VN**, **P**, **S**), **V** = **VN** ∪ **VT** называется ***неукорачивающей*** *грамматикой*, если каждое правило из P имеет вид $\alpha \to \beta$, где $\alpha, \beta \in V^+$ и $|\beta| \geq |\alpha|$.

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

ТИП 2:
Грамматику типа 2 можно определить как контекстно-свободную либо как укорачивающую контекстно-свободную.
Грамматика **G** = (**VT**, **VN**, **P**, **S**), **V** = **VN** ∪ **VT** называется ***контекстно-свободной***, если каждое правило из P имеет вид $A \to \beta$, где $A \in VN$, $\beta \in V^+$.
Грамматика **G** = (**VT**, **VN**, **P**, **S**), **V** = **VN** ∪ **VT** называется ***укорачивающей контекстно-свободной***, если каждое правило из P имеет вид $A \to \beta$, где $A \in VN$, $\beta \in V^*$.

Возможность выбора обусловлена тем, что для каждой укорачивающей контекстно-свободной грамматики существует почти эквивалентная ей контекстно-свободная грамматика.

ТИП 3:
Грамматику типа 3 можно определить либо как праволинейную, либо как леволинейную.
Грамматика **G** = (**VT**, **VN**, **P**, **S**), **V** = **VN** ∪ **VT** называется ***праволинейной***, если каждое правило из P имеет вид $A \to \gamma B$ либо $A \to \gamma$, где $A, B \in VN$, $\gamma \in VT^*$.
Грамматика **G** = (**VT**, **VN**, **P**, **S**), **V** = **VN** ∪ **VT** называется ***леволинейной***, если каждое правило из P имеет вид $A \to B\gamma$ либо $A \to \gamma$, где $A, B \in VN$, $\gamma \in VT^*$.

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

Определение: язык L(G) является ***языком типа k***, если его можно описать грамматикой типа k.
Языки классифицируются в соответствии с типами грамматик, с помощью которых они заданы.

ТИП 0: языки с фразовой структурой
Это самые сложные языки, которые могут быть заданы только грамматикой, относящейся к типу 0. Для распознавания цепочек таких языков требуются вычислители равномощные машине Тьюринга. Поэтому можно сказать, что если язык относится к типу 0, то для него невозможно построить компилятор, который гарантированно выполнял бы разбор предложений языка за ограниченное время на основе ограниченных ресурсов.
Примером таких языков являются практически все языки общения между людьми.

ТИП 1: контекстно-зависимые языки
Второй по сложности тип языков. В общем случае время на распознавание предложений языка экспоненциально зависит от длины исходной цепочки символов.
Языки и грамматики типа 1 применяются в анализе и переводе текстов на естественных языках. Распознаватели, построенные на их основе, позволяют анализировать тексты с учетом контекстной зависимости в предложениях входного языка.

ТИП 2: контекстно-свободные языки
Контекстно-свободные языки лежат в основе синтаксических конструкций большинства современных языков программирования, на их основе функционируют некоторые довольно сложные командные процессоры, допускающие управляющие команды цикла и условия. Эти обстоятельства определяют распространенность данного класса языков.
В общем случае время на распознавание предложений языка, относящегося к типу 1, полиноминально зависит от длины исходной цепочки символов (кубическая, квадратная или другая зависимость).
Однако среди контекстно-свободных языков существует много классов языков, для которых эта зависимость линейна. Многие языки программирования можно отнести к одному из таких классов.

ТИП 3: регулярные языки
Это самый простой тип языков. Время на распознавание предложений регулярного языка линейно зависит от длины исходной цепочки символов.
Данные языки лежат в основе простейших конструкций языков программирования, на их основе строятся многие мнемокоды команд, а также командные процессоры, символьные управляющие команды и другие подобные структуры.
Регулярные языки - очень удобное средство. Для работы с ними можно использовать регулярные множества и выражения, конечные автоматы.

Вывод. Цепочки вывода

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

Цепочка $\beta \in V^*$ ***выводима*** из цепочки $\alpha \in V^+$ в грамматике G = (VT, VN, P, S) (обозначим $\alpha \Rightarrow \beta$), если существуют цепочки $\gamma_0, \gamma_1, \ldots, \gamma_n$ (n >= 0), такие, что $\alpha = \gamma_0 \to \gamma_1 \to \ldots \to \gamma_n = \beta$. Последовательность $\gamma_0, \gamma_1, \ldots, \gamma_n$ называется ***выводом длины n***.

Тогда ***языком, порождаемым грамматикой*** **G** = (**VT**, **VN**, **P**, **S**), называется множество $L(G) = \{\alpha \in VT^* \mid S \Rightarrow \alpha\}$.
Другими словами, L(G) - это все цепочки в алфавите VT, которые выводимы из S с помощью P.

Цепочка $\alpha \in V^*$, для которой $S \Rightarrow \alpha$, называется ***сентенциальной формой*** в грамматике G = (VT, VN, P, S). Таким образом, язык, порождаемый грамматикой, можно определить как множество терминальных сентенциальных форм.

Цепочка принадлежит языку, порождаемому грамматикой, только в том случае, если существует ее вывод из цели этой грамматики. Процесс построения такого вывода (а, следовательно, и определения принадлежности цепочки языку) называется ***разбором***.

Вывод цепочки $\beta \in (VT)^*$ из $S \in VN$ в контекстно-свободной грамматике G = (VT, VN, P, S), называется ***левым (левосторонним)***, если в этом выводе каждая очередная сентенциальная форма получается из предыдущей заменой самого левого нетерминала.

Вывод цепочки $\beta \in (VT)^*$ из $S \in VN$ в контекстно-свободной грамматике G = (VT, VN, P, S), называется ***правым (правосторонним)***, если в этом выводе каждая очередная сентенциальная форма получается из предыдущей заменой самого правого нетерминала.

В грамматике для одной и той же цепочки может быть несколько выводов, эквивалентных в том смысле, что в них в одних и тех же местах применяются одни и те же правила вывода, но в различном порядке.
Например, для цепочки a+b+a в грамматике G = ({a,b,+}, {S,T}, {S → T | T+S; T → a | b}, S) можно построить выводы:
(1) S→T+S→T+T+S→T+T+T→a+T+T→a+b+T→a+b+a
(2) S→T+S→a+S→a+T+S→a+b+S→a+b+T→a+b+a
(3) S→T+S→T+T+S→T+T+T→T+T+a→T+b+a→a+b+a

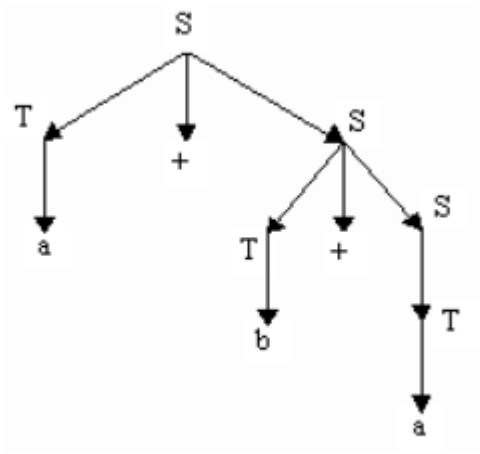
Здесь (2) - левосторонний вывод, (3) - правосторонний, а (1) не является ни левосторонним, ни правосторонним, но все эти выводы являются эквивалентными в указанном выше смысле.

Для контекстно-свободных грамматик можно ввести удобное графическое представление вывода, называемое деревом вывода, причем для всех эквивалентных выводов деревья вывода совпадают.

Дерево называется ***деревом вывода*** *(*или ***деревом разбора****) в контекстно-свободной грамматике* G = {VT, VN, P, S), если выполнены следующие условия:

(1) каждая вершина дерева помечена символом из множества $(VN \cup VT \cup \varepsilon)$, при этом корень дерева помечен символом S; листья - символами из $(VT \cup \varepsilon)$;
(2) если вершина дерева помечена символом $A \in VN$, а ее непосредственные потомки - символами $a_1, a_2, \ldots, a_n$, где каждое $a_i \in (VT \cup VN)$, то $A \to a_1a_2...a_n$ - правило вывода в этой грамматике;
(3) если вершина дерева помечена символом $A \in VN$, а ее единственный непосредственный потомок помечен символом ε, то $A \to \varepsilon$ - правило вывода в этой грамматике.

Пример дерева вывода для цепочки a+b+a в грамматике G:



Дерево вывода можно строить ***нисходящим*** либо ***восходящим*** способом.

При нисходящем разборе дерево вывода формируется от корня к листьям; на каждом шаге для вершины, помеченной нетерминальным символом, пытаются найти такое правило вывода, чтобы имеющиеся в нем терминальные символы “проектировались” на символы исходной цепочки.
Метод восходящего разбора заключается в том, что исходную цепочку пытаются “свернуть” к начальному символу S; на каждом шаге ищут подцепочку, которая совпадает с правой частью какого-либо правила вывода; если такая подцепочка находится, то она заменяется нетерминалом из левой части этого правила.

Если грамматика однозначная, то при любом способе построения будет получено одно и то же дерево разбора.

Эквивалентность и неоднозначность грамматик

Грамматики G1 и G2 называются ***эквивалентными***, если L(G1) = L(G2).

Например,
G1 = ({0,1}, {A,S}, P1, S) и G2 = ({0,1}, {S}, P2, S)
P1: $S \to 0A1$ P2: $S \to 0S1 \mid 01$
$0A \to 00A1$
$A \to \varepsilon$
эквивалентны, т.к. обе порождают язык $L(G1) = L(G2) = \{0^n1^n \mid n>0\}$.

Грамматики G1 и G2 называются ***почти эквивалентными***, если $L(G1) = L(G2) \cup \{\varepsilon\}$. Другими словами, грамматики почти эквивалентны, если языки, ими порождаемые, отличаются не более, чем на ε.

Например,
G1 = ({0,1}, {A,S}, P1, S) и G2 = ({0,1}, {S}, P2, S)
P1: $S \to 0A1$ P2: $S \to 0S1 \mid \varepsilon$
$0A \to 00A1$
$A \to \varepsilon$
почти эквивалентны, т.к. $L(G1)=\{0^n1^n \mid n>0\}$, а $L(G2)=\{0^n1^n \mid n>=0\}$, т.е. L(G2) состоит из всех цепочек языка L(G1) и пустой цепочки, которая в L(G1) не входит.

Контекстно-свободная грамматика G называется ***неоднозначной***, если существует хотя бы одна цепочка $\alpha \in L(G)$, для которой может быть построено два или более различных деревьев вывода.
Это утверждение эквивалентно тому, что цепочка α имеет два или более разных левосторонних (или правосторонних) выводов. В противном случае грамматика называется ***однозначной***.

Язык, порождаемый грамматикой, называется ***неоднозначным***, если он не может быть порожден никакой однозначной грамматикой.

Пример неоднозначной грамматики:
G = ({if, then, else, a, b}, {S}, P, S),
где P = {$S \to$ 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. Однако это не означает, что язык L(G) обязательно неоднозначный. Определенная нами **неоднозначность - это свойство грамматики, а не языка**, т.е. для некоторых неоднозначных грамматик существуют эквивалентные им однозначные грамматики. Если грамматика используется для определения языка программирования, то она должна быть однозначной.
В приведенном выше примере разные деревья вывода предполагают соответствие else разным then. Если договориться, что else должно соответствовать ближайшему к нему then, и подправить грамматику G, то неоднозначность будет устранена:
$S \to$ if b then S | if b then S' else S | a
$S' \to$ if b then S' else S' | a

Проблема, порождает ли данная контекстно-свободная грамматика однозначный язык (т.е. существует ли эквивалентная ей однозначная грамматика), является **алгоритмически неразрешимой.**

Распознаватели. Задача разбора

Общая схема распознавателя

Для каждого языка программирования важно не только уметь построить текст программы на этом языке, но и определить принадлежность имеющегося текста к данному языку. Именно эту задачу решают компиляторы в числе прочих задач. В отношении исходной программы компилятор выступает как распознаватель, а человек, создавший программу на некотором языке, выступает в роли генератора цепочек этого языка.

Распознаватель - это специальный алгоритм, который позволяет определить принадлежность цепочки символов некоторому языку. Задача распознавателя заключается в том, чтобы на основании исходной цепочки дать ответ, принадлежит ли она заданному языку или нет.

В общем виде распознаватель можно отобразить в виде условной схемы, отображающей работу алгоритма распознавателя, представленной на рисунке.

Как видно из рисунка, распознаватель состоит из следующих основных компонентов:

- ленты, содержащей исходную цепочку входных символов, и считывающей головки, обозревающей очередной символ в этой цепочке;
- устройства управления (УУ), которое координирует работу распознавателя, имеет некоторый набор состояний и конечную память (для хранения своего состояния и некорой промежуточной информации);
- внешней (рабочей) памяти, которая может хранить некоторую информацию в процессе работы распознавателя и в отличие от памяти УУ может иметь неограниченный объем.

Распознаватель работает по шагам или тактам. В начале такта, как правило, считывается очередной символ из входной цепочки, и в зависимости от этого символа УУ определяет, какие действия необходимо выполнить. Вся работа распознавателя состоит из последовательности тактов. В начале каждого такта состояние распознавателя определяется его конфигурацией. В процессе работы конфигурация меняется.

Конфигурация распознавателя определяется следующими параметрами:
- содержимое входной цепочки символов и положение считывающей головки в ней;
- состояние УУ;
- содержимое внешней памяти.

Для распознавателя всегда задается определенная конфигурация, которая считается начальной конфигурацией. В начальной конфигурации считывающая головка обозревает первый символ входной цепочки, УУ находится в заданном начальном состоянии, а внешняя память либо пуста, либо содержит строго определенную информацию.

Кроме начального состояния для распознавателя задается одна или несколько конечных конфигураций. В конечной конфигурации считывающая головка, как правило, находится за концом исходной цепочки.

Распознаватель ***допускает входную цепочку символов*** α, если, находясь в начальной конфигурации и получив на вход эту цепочку, он может проделать последовательность шагов, заканчивающуюся одной из его конечных конфигураций.

Виды распознавателей

Распознаватели можно классифицировать в зависимости от вида составляющих их компонентов: считывающего устройства, устройства управления (УУ) и внешней памяти.

По видам считывающего устройства распознаватели могут быть ***двусторонние*** и ***односторонние***. Односторонние распознаватели допускают перемещение считывающей головки по ленте входных символов только в одном направлении. А двусторонние распознаватели допускают, что считывающая головка может перемещаться относительно ленты входных символов в обоих направлениях.

По видам устройства управления распознаватели бывают ***детерминированные*** и ***недетерминированные***. Распознаватель называется *детерминированным* в том случае, если для каждой допустимой конфигурации распознавателя, которая возникла на некотором шаге его работы, существует единственно возможная конфигурация, в которую распознаватель перейдет на следующем шаге работы. В противном случае распознаватель называется *недетерминированным*.

По видам внешней памяти распознаватели бывают следующих типов:
- распознаватели без внешней памяти;
- распознаватели с ограниченной внешней памятью;
- распознаватели с неограниченной внешней памятью.

Вместе эти три составляющих позволяют организовать общую классификацию распознавателей.
Тип распознавателя в классификации определяет сложность создания такого распознавателя, а следовательно сложность разработки соответствующего программного обеспечения для компилятора.

Классификация распознавателей по типам языков

Классификация распознавателей по видам составляющих их компонентов определяет сложность алгоритма работы распознавателя. Но сложность распознавателя также напрямую связана с типом языка, входные цепочки которого может принимать распознаватель.

Для каждого из четырех типов языков существует свой тип распознавателя с определенным составом компонентов и, следовательно, с заданной сложностью алгоритма работы.

Для ***языков с фразовой структурой*** (тип 0) необходим распознаватель, равномощный машине Тьюринга - недетерминированный двусторонний автомат, имеющий неограниченную внешнюю память.

Для ***контекстно-зависимых языков*** (тип 1) распознавателями являются двусторонние недетерминированные автоматы с линейно ограниченной внешней памятью.

Для ***контекстно-свободных языков*** (тип 2) распознавателями являются односторонние недетерминированные автоматы с магазинной (стековой) внешней памятью.

Для ***регулярных языков*** (тип 3) распознавателями являются односторонние недетерминированные автоматы без внешней памяти - конечные автоматы.

Задача разбора (постановка задачи)

Грамматики и распознаватели - два независимых метода, которые идеально могут быть использованы для определения какого-либо языка. Однако при разработке компилятора для некоторого языка программирования возникает задача, которая требует связать между собой эти методы задания языков.

Разработчики компилятора всегда имеют дело с уже определенным языком программирования. Грамматика для синтаксических конструкций этого языка известна. Она, как правило, четко описана в стандарте языка. Задача разработчиков заключается в том, чтобы построить распознаватель для заданного языка, который затем будет основой синтаксического анализатора в компиляторе.

Таким образом, задача разбора в общем виде заключается в следующем: на основе имеющейся грамматики некоторого языка построить распознаватель для этого языка. Заданная грамматика и распознаватель должны быть эквивалентны, то есть определять один и тот же язык.

Далее будет приведено описание методов решения задачи разбора.

Преобразования грамматик. Лексический разбор

Общие соглашения

1. Если при описании грамматики указаны только правила вывода **P**, то будем считать, что большие латинские буквы обозначают нетерминальные символы, **S** - цель грамматики, все остальные символы - терминальные.
2. Предположим, что анализируемая цепочка заканчивается специальным символом # - ***признаком конца цепочки***.

Алгоритм разбора

Рассмотрим методы и средства, которые обычно используются при построении лексических анализаторов. В основе таких анализаторов обычно лежат регулярные грамматики.

Для грамматик, например, леволинейного типа существует алгоритм определения того, принадлежит ли анализируемая цепочка языку, порождаемому этой грамматикой или нет (***алгоритм разбора***):
(1) первый символ исходной цепочки $a_1a_2...a_n\#$ заменяем нетерминалом A, для которого в грамматике есть правило вывода $A \to a_1$ (другими словами, производим "свертку" терминала a_1 к нетерминалу A);
(2) затем многократно (до тех пор, пока не считаем признак конца цепочки) выполняем следующие шаги: полученный на предыдущем шаге нетерминал A и расположенный непосредственно справа от него очередной терминал a_i исходной цепочки заменяем нетерминалом B, для которого в грамматике есть правило вывода $B \to Aa_i$ (i = 2, 3,.., n) и т.д.
Это эквивалентно построению дерева разбора методом "снизу-вверх": на каждом шаге алгоритма строим один из уровней в дереве разбора, "поднимаясь" от листьев к корню.

При работе этого алгоритма возможны следующие ситуации:
(1) прочитана вся цепочка; на каждом шаге находилась единственная нужная "свертка"; на последнем шаге свертка произошла к символу S. Это означает, что исходная цепочка принадлежит языку ($a_1a_2...a_n\# \in L(G)$).
(2) прочитана вся цепочка; на каждом шаге находилась единственная нужная "свертка"; на последнем шаге свертка произошла к символу, отличному от S. Это означает, что исходная цепочка не принадлежит языку ($a_1a_2...a_n\# \notin L(G)$).
(3) на некотором шаге не нашлось нужной свертки, т.е. для полученного на предыдущем шаге нетерминала A и расположенного непосредственно справа от него очередного терминала ai исходной цепочки не нашлось нетерминала B, для которого в грамматике было бы правило вывода $B \to Aa_i$. Это означает, что исходная цепочка не принадлежит языку ($a_1a_2...a_n\# \notin L(G)$).
(4) на некотором шаге работы алгоритма оказалось, что есть более одной подходящей свертки, т.е. в грамматике разные нетерминалы имеют правила вывода с одинаковыми правыми частями, и поэтому непонятно, к какому из них производить свертку. Это говорит о *недетерминированности разбора.* Анализ этой ситуации будет приведен далее.

Таблица сверток и диаграмма состояний

Для того, чтобы при лексическом разборе быстрее находить правило с подходящей правой частью, фиксируют все возможные свертки в описании грамматики (это определяется только грамматикой и не зависит от вида анализируемой цепочки).
Это можно сделать в виде таблицы, строки которой помечены нетерминальными символами грамматики, столбцы - терминальными. Значение каждого элемента таблицы - это нетерминальный символ, к которому можно свернуть пару "нетерминал-терминал", которыми помечены соответствующие строка и столбец.

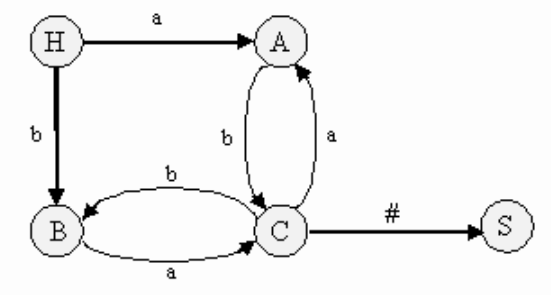
Например, для грамматики G = ({a, b, #}, {S, A, B, C}, P, S), такая таблица будет выглядеть следующим образом:
P: S → C#
C → Ab | Ba
A → a | Ca
B → b | Cb

	a	b	#
C	A	B	S
A	-	C	-
B	C	-	-
S	-	-	-

Знак "-" ставится в том случае, если для пары "терминал-нетерминал" свертки нет.

Чаще информацию о возможных свертках представляют в виде ***диаграммы состояний*** - неупорядоченного ориентированного помеченного графа, который строится следующим образом:
(1) строят вершины графа, помеченные нетерминалами грамматики (для каждого нетерминала - одну вершину), и еще одну вершину, помеченную символом, отличным от нетерминальных, - H. Эти вершины будем называть ***состояниями.*** H - начальное состояние.
(2) соединяем эти состояния дугами по следующим правилам:
а) для каждого правила грамматики вида W → t соединяем дугой состояния H и W (от H к W) и помечаем дугу символом t;
б) для каждого правила W → Vt соединяем дугой состояния V и W (от V к W) и помечаем дугу символом t.

Диаграмма состояний для грамматики G (см. пример выше):



Тогда для диаграммы состояний применим следующий *алгоритм разбора*:

(1) объявляем текущим состояние H;
(2) затем многократно (до тех пор, пока не считаем признак конца цепочки) выполняем следующие шаги: считываем очередной символ исходной цепочки и переходим из текущего состояния в другое состояние по дуге, помеченной этим символом. Состояние, в которое мы при этом попадаем, становится текущим.

При работе этого алгоритма возможны следующие ситуации (аналогичные ситуациям, которые возникают при разборе непосредственно по регулярной грамматике):
(1) прочитана вся цепочка; на каждом шаге находилась единственная дуга, помеченная очередным символом анализируемой цепочки; в результате последнего перехода оказались в состоянии S. Это означает, что исходная цепочка принадлежит L(G).
(2) прочитана вся цепочка; на каждом шаге находилась единственная "нужная" дуга; в результате последнего шага оказались в состоянии, отличном от S. Это означает, что исходная цепочка не принадлежит L(G).
(3) на некотором шаге не нашлось дуги, выходящей из текущего состояния и помеченной очередным анализируемым символом. Это означает, что исходная цепочка не принадлежит L(G).
(4) на некотором шаге работы алгоритма оказалось, что есть несколько дуг, выходящих из текущего состояния, помеченных очередным анализируемым символом, но ведущих в разные состояния. Это говорит о *недетерминированности разбора*. Анализ этой ситуации будет приведен далее.

Конечный автомат. Недетерминированный разбор

Диаграмма состояний определяет конечный автомат, построенный по регулярной грамматике, который допускает множество цепочек, составляющих язык, определяемый этой грамматикой. Состояния и дуги диаграммы состояний - это графическое изображение функции переходов конечного автомата из состояния в состояние при условии, что очередной анализируемый символ совпадает с символом-меткой дуги. Среди всех состояний выделяется начальное (считается, что в начальный момент своей работы автомат находится в этом состоянии) и конечное (если автомат завершает работу переходом в это состояние, то анализируемая цепочка им допускается).

Конечный автомат (КА) - это пятерка (**K**, **VT**, **F**, **H**, **S**), где
K - конечное множество состояний;
VT - конечное множество допустимых входных символов;
F - функции переходов;
H $\in$ **K** - начальное состояние;
S $\in$ **K** - заключительное состояние (либо конечное множество заключительных состояний).
F(A, t) = B означает, что из состояния A по входному символу t происходит переход в состояние B.
Конечный автомат *допускает цепочку* $a_1a_2...a_n$, если $F(H,a_1) = A_1$; $F(A_1,a_2) = A_2$; . . . ; $F(A_{n-2},a_{n-1}) = A_{n-1}$; $F(A_{n-1},a_n) = S$, где $a_i \in VT$, $A_j \in K$, j = 1, 2 , ... ,n-1; i = 1, 2, ... ,n; H - начальное состояние, S - одно из заключительных состояний.
Множество цепочек, допускаемых конечным автоматом, составляет определяемый им *язык*.

При анализе по регулярной грамматике может оказаться, что несколько нетерминалов имеют одинаковые правые части, и поэтому неясно, к какому из них делать свертку (см. ситуацию 4 в описании алгоритма). В терминах диаграммы состояний это означает, что из

одного состояния выходит несколько дуг, ведущих в разные состояния, но помеченных одним и тем же символом.

Например, для грамматики G = ({a, b, #}, {S, A, B}, P, S), где
P: S $\rightarrow$ A#
A $\rightarrow$ a | Bb
B $\rightarrow$ b | Bb
разбор будет недетерминированным (т.к. у нетерминалов A и B есть одинаковые правые части - Bb). Такой грамматике будет соответствовать недетерминированный конечный автомат.

Недетерминированный конечный автомат (НКА) - это пятерка (**K**, **VT**, **F**, **H**, **S**), где
K - конечное множество состояний;
VT - конечное множество допустимых входных символов;
F - функции переходов;
H $\subset$ **K** - конечное множество начальных состояний;
S $\subset$ **K** - конечное множество заключительных состояний.
F(A,t) = {B_1,B_2,...,B_n} означает, что из состояния A по входному символу t можно осуществить переход в любое из состояний B_i, i = 1, 2, ... ,n.

В этом случае можно предложить алгоритм, который будет перебирать все возможные варианты сверток (переходов) один за другим; если цепочка принадлежит языку, то будет найден путь, ведущий к успеху; если будут просмотрены все варианты, и каждый из них будет завершаться неудачей, то цепочка языку не принадлежит. Однако такой алгоритм практически неприемлем, поскольку при переборе вариантов мы, скорее всего, снова окажемся перед проблемой выбора и, следовательно, будем иметь "дерево отложенных вариантов".

Один из наиболее важных результатов теории конечных автоматов состоит в том, что класс языков, определяемых недетерминированными конечными автоматами, совпадает с классом языков, определяемых детерминированными конечными автоматами. ***Это означает, что для любого НКА всегда можно построить детерминированный КА, определяющий тот же язык.***

Алгоритм построения детерминированного КА по НКА

Вход: M = (K, VT, F, H, S) - недетерминированный конечный автомат.
Выход: M' = (K', VT, F', H', S') - детерминированный конечный автомат, допускающий тот же язык, что и автомат M.

Метод:
1. Множество состояний K' состоит из всех подмножеств множества K. Каждое состояние из K' будем обозначать $A_1A_2...A_n$, где $A_i \in K$.
2. Функции переходов F' определим как F' ($A_1A_2...A_n$, t) = $B_1B_2...B_m$, где для каждого 1 <= j <= m F(A_i, t) = B_j для каких-либо 1 <= i <= n.
3. Пусть H = {H_1, H_2, ..., H_k}, тогда H' = H_1, H_2, ..., H_k.
4. Пусть S = {S_1, S_2, ..., S_p}, тогда S' - все состояния из K', имеющие вид ...S_i..., $S_i \in S$ для какого-либо 1 <= i <= p.

Замечание: в множестве K' могут оказаться состояния, которые недостижимы (см. Преобразования грамматик) из начального состояния, их можно исключить.

Проиллюстрируем работу алгоритма на примере.

Пусть задан НКА: M = ({H, A, B, S}, {0, 1}, F, {H}, {S}), где
F(H, 1) = B F(B, 0) = A
F(A, 1) = B F(A, 1) = S ,
тогда соответствующий детерминированный конечный автомат будет определен следующим образом.
Новое множество состояний ДКА: K' = {H, A, B, S, HA, HB, HS, AB, AS, BS, HAB, HAS, ABS, HBS, HABS}
Множество входных сигналов и начальное состояние состояние автомата останутся неизменными.
Новое множество функций переходов:
F'(A, 1) = BS F'(H, 1) = B
F'(B, 0) = A F'(HA, 1) = BS
F'(HB, 1) = B F'(HB, 0) = A
F'(HS, 1) = B F'(AB, 1) = BS
F'(AB, 0) = A F'(AS, 1) = BS
F'(BS, 0) = A F'(HAB, 0) = A
F'(HAB, 1) = BS F'(HAS, 1) = BS
F'(ABS, 1) = BS F'(ABS, 0) = A
F'(HBS, 1) = B F'(HBS, 0) = A
F'(HABS, 1) = BS F'(HABS, 0) = A
Новое множество конечных состояний: S' = {S, HS, AS, BS, HAS, ABS, HBS, HABS}

Достижимыми состояниями в получившемся КА являются H, B, A и BS, т.к. к ним возможен переход из других состояний, поэтому остальные состояния и соответствующие им функции переходов удаляются.
Таким образом, M' = ({H, B, A, BS}, {0, 1}, F', H, {BS}), где
F'(A, 1) = BS F'(H, 1) = B
F'(B, 0) = A F'(BS, 0) = A

Моделировать работу детерминированного КА существенно проще, чем произвольного КА, но при выполнении преобразования число состояний автомата может существенно возрасти и, в худшем случае, составит $\mathbf{(2^n - 1)}$, где **n** - количество состояний исходного КА. В этом случае затраты на моделирование детерминированного КА окажутся больше, чем на моделирование исходного КА. Поэтому не всегда выполнение преобразования автомата к детерминированному виду является обязательным.

Преобразования грамматик

В некоторых случаях контекстно-свободная грамматика может содержать недостижимые символы, которые не участвуют в порождении цепочек языка и поэтому могут быть удалены из грамматики.

Символ $x \in V$ называется ***недостижимым*** в грамматике G = (VT, VN, P, S), если он не появляется ни в одной сентенциальной форме этой грамматики, т.е. к нему не существует ни одного перехода на диаграмме состояний.

Недостижимые символы удаляются из списка состояний КА и также удаляются все функции переходов этих символов.