Этапы анализа

В работе любого **транслятора** или **интерпретатора** присутствует фаза разбора входной программы, представленной, как привило, в виде текста. Задача транслятора – преобразовать одно представление программы в эквивалентное другое представление. **Компиляторы** – частный случай трансляторов, которые переводят программу в представление, понятное компьютеру – ассемблерный код или байт-код. Задача интерпретатора – выполнить входную программу без преобразования в другое представление.

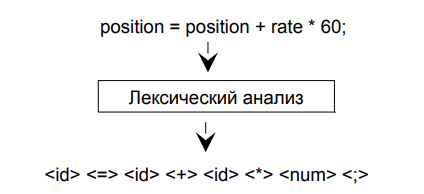
Грань между трансляторами и интерпретаторами достаточно условна и размыта, так как современные интерпретаторы перед интерпретацией обязательно переводят входную программу в какое-либо промежуточное представление, удобное для интерпретации (в частных случаях в качестве такого промежуточного представления может быть непосредственно код целевой платформы, т.е. по сути выполняется компиляция кода «на лету»).

Также следует уточнить, что под входной программой, о которой говорится в предыдущих абзацах, не следует узко понимать исключительно код на языке программирования, в частных случаях это могут быть, например, данные, представленные в текстовых форматах XML, JSON и т.д.

Процесс разбора входной программы, часто неформально называемый **парсингом**, как правило, состоит из двух этапов: **лексического анализа** и **синтаксического анализа** (стоит отметить, что существуют подходы к парсингу, в которых этап лексического анализа явно не выделен).

На этапе лексического анализа входная строка из последовательности символов переводится в последовательность **лексем (токенов)**.

Лексический анализ:

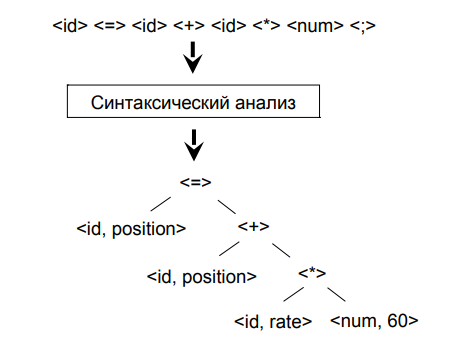


Видно, что суть лексического анализа заключается, если говорить просто, в выделении из последовательности «букв» входной программы последовательности «слов», неделимых с точки зрения последующего анализа. Эти «слова» называются лексемами или токенами.

На этапе синтаксического анализа линейная последовательности лексем сопоставляется с неким формальным набором правил построения предложений естественного или формального языка (далее в данном пособии речь пойдет исключительно о формальных языках, наиболее яркими примерами которых являются языки программирования).

В простейшем случае синтаксический анализ позволяет получить ответ, принадлежит ли входная строка языку, с которым мы ее пытаемся сопоставить или нет. Однако на практике одного такого ответа (принадлежит или нет) недостаточно и, как правило, параллельно с синтаксическим разбором происходят какие-либо действия, направленные на дальнейшее преобразование входной строки (на этапе лексического анализа уже не просто входной строки, а последовательности лексем).

Синтаксический анализ:



В простейшем случае для простых языков на этапе синтаксического анализа могут уже производиться вычисления (например, вычисление арифметического выражения), однако чаще на этом этапе строится внутреннее представление программы, удобное для дальнейшего анализа и обработки. Часто в качестве такого представления используется **абстрактное синтаксическое дерево** (AST-дерево), пример которого можно видеть на предыдущем рисунке.

Как можно видеть, AST-дерево представляет собой структурное представление исходной программы, очищенное от элементов конкретного синтаксиса (в рассматриваемом примере в AST-дерево не попал «разделитель», т.к. он не имеет отношения непосредственно к семантике данного фрагмента программы, а лишь к конкретному синтаксису языка). В качестве узлов в AST-дереве выступают операторы, к которым присоединяются их аргументы, которые в свою очередь также могут быть составными узлами. Часто узлы AST-дерева получаются из лексем, выделенных на этапе лексического анализа, однако могут встречаться и узлы, которым ни одна лексема не соответствует.

Как правило, лексический и синтаксический анализаторы строятся (даже если построение осуществляется вручную) на основе формализованного описания лексики и синтаксиса языка. Лексика языка (правила выделения лексем) может быть описана в виде регулярных выражений или грамматик, синтаксис всегда описывается с помощью формальных грамматик (или их графического изображения в виде схем).

В общем случае грамматики состоят из правил преобразования последовательностей символов в другие последовательности. В качестве символов в грамматиках присутствуют **терминалы** (терминальные символы) и **нетерминалы** (нетерминальные символы).

Способ представления грамматики

Основной задачей синтаксиса является определение формы и вида допустимых языковых конструкций. Эту задачу возможно решить путем перечисления описаний всех языковых конструкций. Одним из механизмов такого описания является нотация БНФ.

**Простая форма Бэкуса — Наура** (БНФ) это формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. БНФ используется для описания контекстно-свободных формальных грамматик, обычно используется для описания синтаксиса языков программирования, форматов документов, наборов инструкций и протоколов связи. Применяются везде, где необходимо точное описание синтаксиса: например, в официальных спецификациях, руководствах и учебниках.

**Терминал** или терминальный символ — объект, непосредственно присутствующий в словах языка, соответствующего грамматике, и имеющий конкретное, неизменяемое значение.

**Нетерминал** или нетерминальный символ — объект, обозначающий какую-либо сущность языка (например, формула, арифметическое выражение, команда) и не имеющий конкретного символьного значения.

БНФ-конструкция определяет конечное число нетерминалов и определяет правила замены символа на какую-то последовательность терминалов и нетерминалов.

Процесс построения цепочки терминалов можно проследить поэтапно:

* Изначально имеется только один нетерминал. Обычно нетерминалы представляются в виде некого названия заключенного в угловые скобки.
* Затем этот нетерминал заменяется некоторой последовательностью терминалов и нетерминалов, согласно одному из описанных правил.
* Затем процесс повторяется (на каждом шаге один из нетерминалов заменяется на последовательность, согласно правилу).

В конце концов, получается цепочка, состоящая из терминалов.

Например, грамматика целых чисел без знака, описанная БНФ, имеет вид:

*<число> := <цифра>|<цифра><число>*

*<цифра> := 0|1|2|…|9*

В правой стороне от стрелочек задаются выражения грамматики («шаблоны» для распознавания), которые состоят из терминалов (фрагменты в кавычках, за которыми скрываются классы лексем) и нетерминалов (названия правил). Круглые скобки группируют часть выражения, чтобы для этих частей задать дополнительные правила применения, в данном примере это:

* <выражение>\* – повторять <выражение> произвольное количество раз (ноль или более);
* <выражение 1> | <выражение 2> | … | <выражение N> – применить любое из перечисленных выражений (альтернатива).

Кроме того, в грамматиках встречаются следующие правила применения:

* <выражение>+ – повторять <выражение> произвольное количество раз, но не менее одного раза (один или более);
* <выражение>? – <выражение> может как присутствовать, так и не присутствовать в разбираемой строке (опционально, ноль или один).

Создание лексического анализатора

**Лексический анализатор (лексер)** — это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы. На выходе лексер возвращает последовательность токенов.

**Токен** — объект, создающийся из лексемы в процессе лексического анализа.

Распознавание лексем в контексте грамматики обычно производится путём их идентификации (или классификации), согласно идентификаторов (или классов) токенов, определяемых грамматикой языка.

Каждый токен можно представить в виде структуры, содержащей идентификатор токена (или идентификатор класса токена) и, если нужно, последовательность символов лексемы выделенной из входного потока (строку, число и т. д.).

Например, дав на вход лексеру следующую строку кода “a = 14 + 88;”, на выходе получим список токенов:

|  |  |  |
| --- | --- | --- |
| Тип токена | Лексема | Описание |
| variable | a | идентификатор |
| assign | = | оператор присваивания |
| number | 14 | число |
| plus | + | операция сложения |
| number | 88 | число |

Цель такой конвертации обычно состоит в том, чтобы подготовить входную последовательность для следующего этапа – синтаксического анализа.

Конечные автоматы являются мощным инструментом для работы. Компиляторы работает на основе конечных автоматов. Любой алгоритм которые вы пишете может быть представлен в виде конечного автомата.

В конечном автомате количество состояний конечно, а результат его работа определяется по его конченому состоянию.

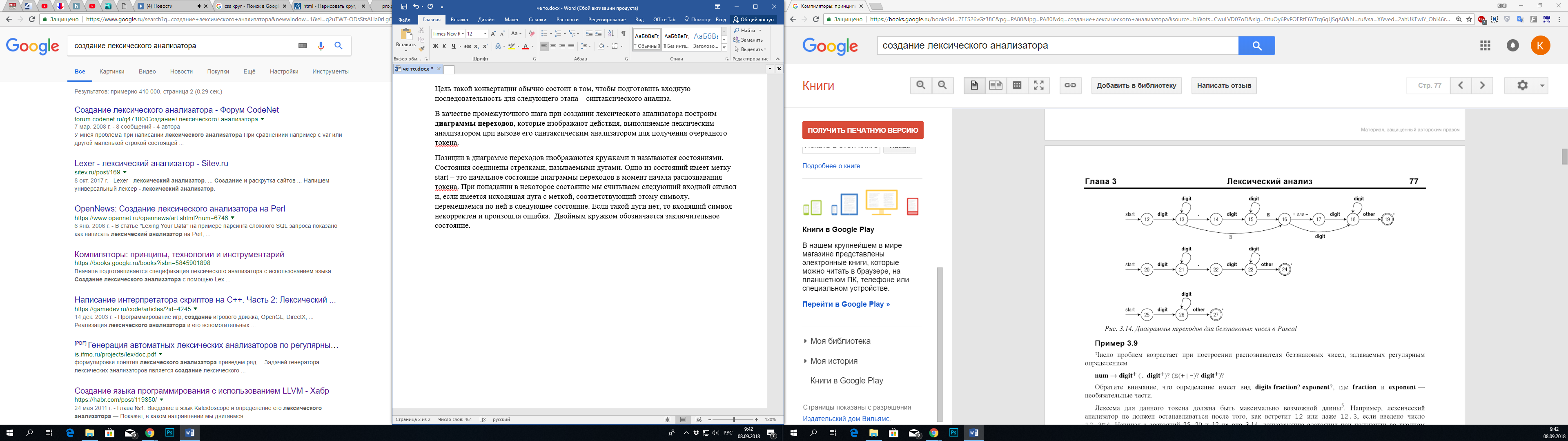
Существует несколько видов конечных автоматов, но мы рассмотрим детерминированный конечный автомат (ДКА). Изначально автомат находится в стартовом состоянии . Автомат считывает символы по очереди. При считывании очередного символа автомат переходит в состояние , где — текущее состояние автомата. Процесс продолжается до тех пор, пока не будет достигнут конец входного слова. ДКА отличается от других автоматов детерминированностью. Это значит, что для любого символа мы можем перейти только в одно состояние из текущего.

Один из способов представления ДКА – диаграмма переходов.

**Диаграмма переходов** — граф, вершины которого соответствуют состояниям автомата, а рёбра — переходам между состояниями.

Позиции в диаграмме переходов изображаются кружками и называются состояниями. Состояния соединены стрелками, называемыми дугами. Одно из состояний имеет метку start – это начальное состояние диаграммы переходов в момент начала распознавания токена. При попадании в некоторое состояние мы считываем следующий входной символ и, если имеется исходящая дуга с меткой, соответствующий этому символу, перемещаемся по ней в следующее состояние. Если такой дуги нет, то входящий символ некорректен и произошла ошибка. Двойным кружком обозначается заключительное состояние.

Диаграммы переходов для беззнаковых чисел в Pascal:



Создание синтаксического анализатора

Синтаксический анализ – это процесс, который определяет, принадлежит ли некоторая последовательность лексем языку, порождаемому грамматикой.

Вход синтаксического анализатора – последовательность токенов, которая являются выходом лексического анализатора. Выход синтаксического анализатора – AST-дерево.

Большинство известных методов анализа принадлежат одному из двух классов, один из которых объединяет **нисходящие** (top-down) алгоритмы, а другой – **восходящие** (bottom-up) алгоритмы. Происхождение этих терминов связано с тем, каким образом строятся узлы синтаксического дерева: либо от корня (аксиомы грамматики) к листьям (терминальным символам), либо от листьев к корню.

Мы будем рассматривать восходящие анализаторы, которые могут анализировать большее количество грамматик, чем нисходящие, и поэтому именно для таких методов существуют программы, которые умеют автоматически строить анализаторы. С восходящими анализаторами связаны LR-грамматики. В этом обозначении буква L означает, что входная цепочка просматривается слева направо (left-to-right scan), а буква R означает, что строится правый вывод цепочки (rightmost derivation). С помощью LR-грамматик можно определить большинство использующихся в настоящее время языков программирования.