Синтаксический анализатор

При выполнении синтаксического анализа подтверждает то, что цепочка символов является программой, а отдельные подцепочки составляют синтаксически правильные программные объекты. Осуществляется анализ их семантической корректности на основе накопленной информации, затем проводится добавление новых объектов в объектную модель программы или промежуточное представление.

Синтаксический анализатор должен распознать весь текст исходной программы, поэтому в отличие от ЛА ему нет необходимости искать границы распознаваемой строки символов. Он должен воспринимать всю информацию, поступающую ему на вход, либо подтвердить ее принадлежность входному языку, либо сообщить об ошибке в исходной программе.

На этапе синтаксического анализа нужно установить имеет ли цепочка лексем структуру заданную синтаксисом языка и зафиксировать эту структуру, следовательно снова необходимо решать задачу разбора. Дана цепочка лексем и надо определить выводима ли она в грамматике, определяющая синтаксис языка. Структура таких конструкций как выражение, оператор и тд более сложная чем структура идентификаторов и чисел, поэтому для описания синтаксиса языков программирования нужны более мощные грамматики чем регулярные. Обычно для этого используют укорачивающие контекстно-свободные грамматики. Грамматики этого класса с одной стороны позволяют достаточно полно описать синтаксическую структуру реальных ЯП, а с другой стороны для разных подклассов укорачивающих контекстно свободных грамматик существуют достаточно эффективные алгоритмы разбора.

Имея грамматику входного языка разработчик синтаксического анализатора должен в первую очередь выполнить ряд формальных преобразований над этой грамматикой облегчающих построение распознавателя. После этого он должен проверить попадает ли полученная грамматика под один из известных классов для которых существуют линейные распознаватели. Если такой класс найден, то можно строить распознаватель. Если такой класс контекстно-свободных языков найти не удалось, то разработчик должен попытаться выполнить некоторые преобразования над грамматикой, чтобы привести ее к одному из известных классов. Для каждого класса контекстно-свободных языков существует свой класс распознавателей, но все они функционируют на основе общих принципов.

Все распознаватели для контекстно-свободных языков можно разделить на 2 группы:

1. Нисходящие. Нисходящие методы начинают с правила грамматики определяющего конечную цель анализа с корня дерева грамматического разбора и пытаются его наращивать так, чтобы последующие узлы дерева соответствовали синтаксису анализируемого предложения.
2. Восходящие. Начинают с конечных узлов дерева грамматического разбора и пытаются объединить их построением узлов все более и более высокого уровня до тех пор пока не будет достигнут корень дерева.

В общем случае эти 2 алгоритма универсальны, они строятся на основе любой контекстно-свободной грамматики после некоторых формальных преобразований и поэтому могут быть использованы для разбора цепочки любого контекстно-свободного языка.

**Восходящий** метод грамматического разбора основан на анализе пар последовательно расположенных операторов исходной программы и решении вопроса о том который из них должен выполнятся первым.

При использовании метода грамматического разбора основанного на отношении операторного предшествования анализируемое предложение просматривается слева направо до тех пор пока не будет найдено подвыражение операторы которого имеют более высокий уровень предшествования чем соседние операторы. Далее это подвыражение распознается в терминал правил вывода используемой грамматики. Этот процесс продолжается до тех пор пока не будет достигнут корень дерева что и будет означать конец грамматического разбора. Первым шагом при разработке процессора грамматического разбора основанного на методе предшествования должно быть установление отношений предшествования между операторами грамматики. При этом под оператором понимается любой терминальный символ, т.е. любая лексема. Если у пар отсутствуют отношения это значит, что они не могут находится рядом ни в одном правильном грамматическом предложении. Существуют алгоритмы автоматического построения матриц предшествования на основе формального описания грамматики. Для применимости метода операторного предшествования необходимо чтобы методы предшествования были заданы однозначно.

Для **нисходящих** распознавателей используется алгоритм с подбором альтернатив. Для каждого нетерминала грамматики создается своя процедура носящая ее имя. Если такую подцепочку считать не удается, то процедура завершает свою работу вызовом процедуры обработки ошибки. Если такая подцепочка найдена, то работа процедуры считается нормально завершенной и осуществляется возврат в точку вызова.

Тело каждой такой процедуры пишется непосредственно по правила вывода соответствующего нетерминала; для правой части каждого правила осуществляется поиск подцепочки выводимой и этой правой части. Терминалы распознаются самой процедурой, а нетерминалы соответствуют вызову процедур носящие их имена.

Восходящий синтаксический анализ как правило привлекательнее нисходящего так как для языка программирования часто легче построить правосторонний восходящий распознаватель. Класс языков заданный восходящими распознавателями шире чем класс языков заданных нисходящими распознавателями. С другой стороны нисходящий синтаксический анализ более предпочтителен с точки зрения процесса трансляции поскольку на его основе легче организовать процесс порождения цепочек результирующего языка. Конкретный выбор зависит от реализации конкретного компилятора, а так же от сложности грамматики входного языка программирования. Некоторые компиляторы используют метод рекурсивного спуска для распознавания конструкций относительно высокого уровня, а затем переключаются на метод предшествования для анализа таких конструкций как например арифметические выражения.