Тема: Синтаксически управляемая трансляция.

В любом языке програььирования есть набор правил составления программ, которае не охватываются грамматикой.

Например, в языке МИКРОЛИСП грамматика не может описать такие правила:

1. **Каждая процедура, вызванная в программе, должна быть определена, причем только один раз.**
2. **Количество аргументов вызова процедуры должно совпадать с количеством ее параметров.**

Все правила, которые не охватываются грамматикой, относятся к правилам семантики, их проверка составляет сферу деятельности семантического анализатора.

Еще один набор правил должен быть формально описан и реализован в компиляторе. Это правила перевода исходного текста в эквивалентный текст на целевом языке. Реализация првил перевода составляет сферу деятельности генератора кода.

Во всех реально применяемых технологиях конструирования компиляторов алгоритмы семантического анализа и перевода разрабатываются вручную и записываютя на каком-то языке высокого уровня. Чаще всего используются языки С или С++, Но есть очень интересный опыт реализации языка Паскаль, выполненной Никлаусом Виртом. Полное описание компилятора он составил на самом Паскале, а затем применил метод «раскрутки». Сначала он вручную написал примитивный промежуточный компилятор. Затем он пропустил текст полного компилятора через промежуточный и получил окончательную реализацию языка. С тех пор язык считается полноценным языком программирования, если на нем можно описать компилятор для самого себя.

Метод синтаксически управляемой трансляции позволяет хорошо структурировать программный код компилятора, разбив его на небольшие функции с четко прописанными интерфейсами.

Суть метода в следующем.
По мере восстановления дерева разбора, каждому узлу сопоставляется набор атрибутов: чисел, строк, булевских флагов и т.п. Сигнатуру и назначение атрибутов разработчик выбирает с учетом информационных потребностей конкретного компилятора.

Атрибуты характеризуют свойства синтаксических классов, существенные для компилятора. Например, определение процедуры имеет такие существенные для семантического анализатора свойства, как имя процедуры и количество параметров. Для генератора кода существенным будет такое свойство, как фрагмент эквивалентной целевой программы.

Пометка узла дерева разбора нетерминалом грамматики обозначает одну цепочку токенов – экземпляр синтаксического класса. Для этой цепочки фиксируются значения атрибутов. Например, имя процедуры – "f", количество параметров – 2.

Каждой продукции грамматики сопоставляется функция преобразования атрибутов. Эта функция применяется в момент свертки основы сентенциальной формы к нетерминалу левой части продукции. Функция манипулирует значениями атрибутов дочерних узлов и вычисляет значения атрибутов родительского узла, помеченного нетерминалом. Этот процесс называется синтезом атрибутов. Он начинается от листьев дерева и постепенно распространяется к корню.

Функцию преобразования атрибутов удобно называть продукцией атрибутов, подчеркивая ее неразрывную связь с продукцией грамматики.

Таким образом, продукции граммати A->a1 a2…an сопоставляется продукция атрибутов
p(Sa1, Sa2,…,San)->SA , символ S обозначает атибут.

Потребности компилятора языка Микролисп полностью покрываются кортежем из пяти атрибутов.

```
struct tSA{
  std::string line;
  string name;
  int count;
  int types;
  string obj;
}
```

В дальнейшем весь кортеж мы будем называть одним словом «атрибут», имея я виду составную природу этого объекта. Таким унифицированным атрибутом мы будем снабжать каждый узел дерева разбора.
Значение атрибута будем записывать так
[line| name | count | types | obj]
Для листьев дерева, помеченных токенами, значение атрибута имеет вид [line| lexeme | 0 | 0 |]
Таким образом, через поле name этого атрибута компилятор получает исходные данные об именах и литералах.
В поле line записан номер строки, содержащей лексему.

Продукции атрибутов это функции вида
int pi() , i уникальный номер продукции грамматики.
Функция возвращает значение 0, если она не обнаруживает ошибок, в противном случае 1.

Продукции обмениваются значениями атрибутов через стек. Управляет стеком атрибутов синтаксический анализатор.
Когда на вершине стека символов формируется основа сентенциальной формы
#...x a1 a2 ... an ,
на вершине стека атрибутов находятся атрибуты соответствующих узлов дерева разбора.
...Sx S1 S2 ... Sn
Действует очень важное соглашение о передаче параметров. Продукция вызывается перед тем, как основа сворачивается к нетеминалу A. Она анализирует значения атрибутов S1,S2, ... Sn и синтезирует новое

значение атрибута для родительского узла, помеченного A. Это значение записывается по адресу S1.

pi(S1,S2, … Sn) -> S1

После того как продукция отработала, анализатор заменяет основу нетерминалом A и удаляет из стека атрибутов n-1 элемент.

 # …x A

…Sx S1

Это соглашение имеет очень важный побочный эффект, который можно назвать «транзитом» атрибутов. Даже еслсли функция pi имеет вид

int pi(){return 0;} ,

то есть не выполняет никаких операций с атрибутами, атрибут первого символа основы без изменений передается родительскому узлу. Атрибут остается на вершине стека и просто меняет «владельца».