Файлы грамматики TranBuilder

TranBuilder принимает на вход спецификацию контекстно-свободной грамматики и строит таблицу правил с возможностью её вывода во внешний файл.

1 Структура грамматики TranBuilder

Файл грамматики TranBuilder содержит четыре основные секции, показанные здесь с соответствующими разделителями.

%{
Объявления C
%}

Объявления TranBuilder

%%
Правила грамматики
%%

1.1 Секция объявлений C

Секция *объявлений C* содержит макроопределения и объявления функций и переменных, используемых в действиях правил грамматики. Они копируются в начало файла анализатора так, чтобы они предшествовали определению yyparse. Если вам не нужны какие-либо объявления C, вы можете опустить ограничивающие эту секцию ‘%{’ и ‘%}’.

1.2 Секция объявлений TranBuilder

Секция *объявлений TranBuilder* содержит объявления, определяющие терминальные и нетерминальные символы, задающие приоритет и т.д. В некоторых простых грамматиках вам могут быть не нужны никакие объявления.

1.3 Секция правил грамматики

Секция *правил грамматики* содержит одно или более правил грамматики TranBuilder и ничего более.

Должно быть по меньшей мере одно правило грамматики, и первый ограничитель ‘%%’ (предшествующий правилам грамматики) не может быть опущен, даже если это первая строка файла.

2 Терминальные и нетерминальные символы

Символы в грамматиках TranBuilder представляют грамматическую классификацию языка.

Терминальный символ (также известный как *тип лексемы*) представляет класс синтаксически эквивалентных лексем. Вы используете такой символ в правилах грамматики, чтобы обозначить, что допустима лексема этого класса. В анализаторе

TranBuilder символ представляется числовым кодом, и функция yylex возвращает код типа лексемы чтобы показать, лексема какого вида прочитана. Вам не нужно знать, каково значение кода, для его обозначения можно использовать сам символ.

Нетерминальный символ обозначает класс синтаксически эквивалентных групп. Имя символа используется при написании правил грамматики.

Имена символов могут содержать буквы, цифры (но не начинаться с цифры), знаки подчёркивания и точки. Точки имеют значение только в нетерминалах.

В TranBuilder существует два способа записи терминальных символов в грамматике:

- *Именованный тип лексемы* записывается идентификатором, как идентификаторы C. Каждое такое имя должно быть определено в объявлении TranBuilder, например, ***%token***.
- Лексема также может записываться в угловых скобках: ***< лексема >***. Такой тип записи лексемы может быть представлен либо с указанием группы и типа лексемы (1), либо с указанием только группы данной лексемы (2):

  (1) < группа, тип >

  (2) < группа >

Способ записи терминального символа не влияет на его грамматический смысл. Он зависит только от того, где он встречается в правилах, и когда функция анализатора его возвращает.

3 Синтаксис правил грамматики

Правило грамматики TranBuilder имеет следующий общий вид:

```
результат:   компоненты...
          ;
```

где *результат* – это описываемый правилом нетерминальный символ, а *компоненты* – различные терминальные и нетерминальные символы, объединяемые этим правилом.

Например:

```
exp:   exp Oper_Add exp
   ;
```

говорит о том, что две группы типа ***exp*** и лексема ***Oper_Add*** между ними могут быть объединены в более крупную группу типа ***exp***.

Пробельные литеры в правилах только разделяют символы. Вы можете по своему усмотрению вставлять дополнительные промежутки.

Между компонентами могут быть разбросаны *действия*, определяющие семантику правила. Действие заключается в фигурные скобки: ***{ действие }*** . Подробнее о *действиях* в пункте **5**.

Для одного *результата* можно написать несколько правил, отдельно или же соединённых литерой вертикальной черты ‘|’ как здесь:

```
результат:   компоненты первого правила...
           |  компоненты второго правила...
              ...
           ;
```

В любом случае правила рассматриваются как различные, даже если они таким образом объединены.

Есть в правиле нет *компонентов*, это означает, что *результат* может соответствовать пустой строке. Например, вот как определяется последовательность нуля или более групп ***exp***, разделённых запятыми:

```
expseq:    /* пусто */
       |   expseq1
       ;

expseq1:   exp
       |   expseq1 Separator_Comma exp
       ;
```

4 Рекурсивные правила

Правило называется *рекурсивным*, если нетерминал его *результата* появляется также в его правой части. Почти все грамматики TranBuilder должны использовать рекурсию, потому что это единственный способ определить последовательность из произвольного числа элементов. Рассмотрим рекурсивное определение последовательности одного или более выражений, разделённых запятыми:

```
expseq1:   exp
       |   expseq1 Separator_Comma exp
       ;
```

Поскольку рекурсивный символ ***expseq1*** – самый левый в правой части, мы называем это *левой рекурсией*. И наоборот, вот та же конструкция, определённая с использованием *правой рекурсии*:

```
expseq1:   exp
       |   exp Separator_Comma expseq1
       ;
```

Косвенная или *взаимная* рекурсия возникает, когда результат правила не появляется непосредственно в правой части, но встречается в правилах для других нетерминалов, появляющихся в его правой части.

Например:

```
expr:   primary
    |   primary Oper_Add primary
    ;
```

```
primary:    constant
          |   Separator_LeftBracket expr Separator_RightBracket
          ;
```

определяет два взаимнорекурсивных правила, поскольку каждое из них ссылается на другое.

5 Использование действий и их виды

Для облегчения построения синтаксических и семантических деревьев, а также для генерации кода используются заранее определенные действия. Действия помещают в фигурные скобки ***{ действие }***, как уже упоминалось ранее. Их обычно ставят в конце правил, однако это не означает, что они не могут стоять в середине или даже в самом начале правила.

Существует несколько типов Действий, каждый из которых отвечает за свою независимую функцию. Условно их можно разделить на две части:

1) ***TREE, CODE, MAKE, YACC, ALL;***
2) ***NODE.***

Разберем первую группу действий:

1. TREE – строит синтаксическое дерево с элементами семантики;
2. CODE – отвечает за генерацию кода;
3. MAKE – выполняет функцию TREE и CODE одновременно;
4. YACC – используется для работы со стеками;
5. ALL – выполняет функцию TREE, CODE и YACC одновременно.

Синтаксис таких действий имеет следующий общий вид:

{ *Тип_Действия*(*Идентификатор, Параметр*); }

где ***Параметр*** может отсутствовать, а ***Идентификатор*** – это имя действия.

Во второй группе находится особый вид действий – NODE. Его особенность заключается в том, что он создает только синтаксическое дерево и для его работы требуются дополнительные параметры – набор целых чисел.

Синтаксис данного вида действий выглядит следующим образом:

{ *NODE*(*Идентификатор*, (*Целые_числа*), *Параметр*); }

где ***Параметр*** может отсутствовать, ***Идентификатор*** – это тип собираемого действием синтаксического узла, а ***(Целые_числа)*** – обязательная часть, где должно присутствовать хотя бы одно число, либо несколько чисел, не превышающих по количеству число 5, через запятую).

6 Режимы работы TranBuilder и их переключение

TranBuilder имеет два режима работы ***‘FSYNTAX’*** и ***‘YSYNTAX’***. Результат работы не зависит от режима работы, но внутренний механизм сильно различается. Если в ***‘FSYNTAX’*** грамматика входного файла проверяется нисходящим синтаксическим анализом, то в ***‘YSYNTAX’*** проверка грамматики приближена к Yacc’у, где используется заранее построенная таблица грамматики формата ***‘.tab.c’***.