Лабораторная работа №7
«Самоприменимый генератор компиляторов на основе предсказывающего анализа»

Скоробогатов С.Ю.

19 апреля 2014 г.

1 Цель работы

Целью данной работы является изучение алгоритма построения таблиц предсказывающего анализатора.

2 Исходные данные

В данной лабораторной работе требуется разработать простейший генератор компиляторов, который по описанию грамматики рабочего языка, записанному на входном языке, строит таблицы предсказывающего анализа в виде составного литерала на целевом языке.

Здесь *рабочий язык* – это некоторый формальный язык с LL(1)-грамматикой, синтаксический анализатор которого должен быть построен генератором компиляторов.

В качестве *входного языка* генератора компиляторов должен выступать язык представления правил грамматики, варианты лексики и синтаксиса которого можно восстановить по примерам из таблицы 1.

И, наконец, *целевым языком* мы будем называть язык реализации компилятора рабочего языка. Так как разрабатываемый генератор компиляторов должен быть самоприменимым, целевой язык должен совпадать с языком реализации генератора компиляторов.

3 Задание

Выполнение данной лабораторной работы состоит из следующих этапов:

1. Составление описаний лексической структуры и грамматики входного языка на основе примера из таблицы 1 (при этом грамматику входного языка следует записать на самом входном языке).

2. Разработка лексического анализатора для входного языка.

3. Разработка структур данных для внутреннего представления грамматики, которые будут порождаться парсером входного языка.

Таблица 1: Варианты входного языка в примерах описаний грамматик

1

```
non-terminal E, E1, T, T1, F;
terminal '+', '*', '(', ')', n;

E  ::= T E1;
E1 ::= '+' T E1 | epsilon;
T  ::= F T1;
T1 ::= '*' F T1 | epsilon;
F  ::= n | '(' E ')';

axiom E;
```

2

```
$AXIOM E
$NTERM E' T T' F
$TERM  "+"  "*"  "("  ")"  "n"
$RULE  E  = T E'
$RULE  E' = "+" T E'
            $EPS
$RULE  T  = F T'
$RULE  T' = "*" F T'
            $EPS
$RULE  F  = "n"
            "(" E ")"
```

3

```
(axiom E) = (T) (E1).
(E1) = + (T) (E1) | .
(T)  = (F) (T1).
(T1) = * (F) (T1) | .
(F)  = n | \( (E) \).
```

4

```
* E  (T E')
  E' ("+" T E') ()
  T  (F T')
  T' ("*" F T') ()
  F  ("n")
     ("(" E ")")
```

5

```
tokens <plus sign>, <star>, <n>.
tokens <left paren>,
       <right paren>.
<E>   is <T> <E 1>.
<E 1> is <plus sign> <T> <E 1>.
<E 1> is .
<T>   is <F> <T 1>.
<T 1> is <star> <F> <T 1>.
<T 1> is .
<F>   is <n>.
<F>   is <left paren> <E>
         <right paren>.
start <E>.
```

6

```
{ E }, E', T, T', F
[ E  : T E' ]
[ E' : "+" T E' : @ ]
[ T  : F T' ]
[ T' : "*" F T' : @ ]
[ F  : "n"
     : "(" E ")"
]
```

4. Разработка алгоритма построения таблицы предсказывающего разбора и вывода её в виде исходного текста на целевом языке.

5. Раскрутка генератора компиляторов, в результате которой он должен породить таблицу предсказывающего разбора для своего входного яхыка.

6. Разработка алгоритма предсказывающего разбора, работающего на основе порождённой таблицы.

7. Тестирование генератора компиляторов путём применения его к грамматике из таблицы 1 и к грамматике входного языка (другими словами, путём самоприменения).

Отметим, что парсер входного языка должен выдавать сообщения об обнаруженных ошибках, включающие координаты ошибки. Восстановление при ошибках, а также выдачу специфичных текстовых описаний ошибок реализовывать не нужно.

Кроме того, генератор компиляторов должен уметь обнаруживать следующие ошибки в грамматике:

- наличие нетерминального символа, не присутствующего в левой части ни одного правила;
- грамматика не относится к классу LL(1)-грамматик;
- не указана аксиома грамматики;
- указано более одной аксиомы грамматики;
- используется необъявленный символ или символ объявлен дважды (для вариантов входных языков, подразумевающих обязательное объявление терминальных и/или нетерминальных символов);

В качестве языков реализации разрешается использовать C++, Java, Go, Ruby или Python.