Лабораторная работа №7.1
«Синтаксический анализатор на основе предсказывающего анализа»

Коновалов А. В.

11 апреля 2016

1 Цель работы

Целью данной работы является изучение алгоритма построения таблиц предсказывающего анализатора.

2 Исходные данные

В данной лабораторной работе требуется разработать синтаксический анализатор на основе предсказывающего анализа, который принимая на входе текст на *входном языке*, на выходе порождает дерево вывода для входного текста.

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

3 Задание

Выполнение лабораторной работы состоит из следующих этапов:

1. Составление описаний лексической структуры и грамматики входного языка на основе примера из таблицы 1.
2. Разработка лексического анализатора для входного языка.
3. Составление (вручную) таблицы предсказывающего разбора для входного языка.
4. Разработка алгоритма предсказывающего разбора, работающего на основе порождённой таблицы. Реализация этого алгоритма.

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

В качестве языков реализации разрешается использовать C++, Java/C#, Go, Ruby или Python. Также допустимо использование языков Scheme/Clojure, Scala, Rust и других, при условии, что выбранный язык поддерживает тип данных «массив» с константным временем доступа по индексу.

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

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

4

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

5

```
tokens <plus sign>, <star>, <n>.
<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>.
tokens <left paren>,
       <right paren>.
<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 ")"
]
```