

编译原理第六次作业

任凯 2020141460080

Consider the following grammar:

$$\begin{aligned} \text{exp} &\rightarrow \text{exp} + \text{factor} \mid \text{factor} \\ \text{factor} &\rightarrow (\text{exp}) \mid \text{number} \end{aligned}$$

(1) Eliminate left recursion using BNF;

A:

$$\begin{aligned} \text{exp} &\rightarrow \text{factor exp}' \\ \text{exp}' &\rightarrow + \text{factor exp}' \mid \epsilon \\ \text{factor} &\rightarrow (\text{exp}) \mid \text{number} \end{aligned}$$

(2) Design an L-SDD to compute the value of the expressions generated by the grammar of (1);

A:

Production	Semantic Rules
1) $\text{exp} \rightarrow \text{factor exp}'$	$\text{exp}'.\text{inh} = \text{factor.val}$ $\text{exp.val} = \text{exp}'.\text{syn}$
2) $\text{exp}' \rightarrow + \text{factor exp}'_1$	$\text{exp}'_1.\text{inh} = \text{exp}'.\text{inh} + \text{factor.val}$ $\text{exp}'.\text{syn} = \text{exp}'_1.\text{syn}$
3) $\text{exp}' \rightarrow \epsilon$	$\text{exp}'.\text{syn} = \text{exp}'.\text{inh}$
4) $\text{factor} \rightarrow (\text{exp})$	$\text{factor.val} = \text{exp.val}$
5) $\text{factor} \rightarrow \text{number}$	$\text{factor.val} = \text{number.lexval}$

(3) Convert the SDD of (2) to SDT.

A: From (2) the SDT as followed can be obtained:

$$\begin{aligned} \text{exp} &\rightarrow \text{factor} \{ \text{exp}'.\text{inh} = \text{factor.val} \} \\ \text{exp}' &\{ \text{exp.val} = \text{exp}'.\text{syn} \} \\ \text{exp}' &\rightarrow + \\ &\quad \text{factor} \{ \text{exp}'_1.\text{inh} = \text{exp}'.\text{inh} + \text{factor.val} \} \\ &\quad \text{exp}'_1 \{ \text{exp}'.\text{syn} = \text{exp}'_1.\text{syn} \} \\ \text{exp}' &\rightarrow \epsilon \{ \text{exp}'.\text{syn} = \text{exp}'.\text{inh} \} \\ \text{factor} &\rightarrow (\text{exp}) \{ \text{factor.val} = \text{exp.val} \} \\ \text{factor} &\rightarrow \text{number} \{ \text{factor.val} = \text{number.lexval} \} \end{aligned}$$

(4) Implement the SDT of (3) as a recursive-descent parser

A:

```
1. procedure exp
2. begin
3.     factor();
4.     exp'();
5. end exp
6.
7. procedure exp'
8. begin
9.     while token = + do
10.         match(+);
11.         factor();
12.         exp'();
13.     end while;
14. end exp'
15.
16. procedure factor
17. begin
18.     case token of
19.         ( : match(();
20.             exp();
21.             match());
22.         number : match(number);
23.         else error;
24.     end case;
25. end factor
```