编译原理第六次作业

任凯 2020141460080

Consider the following grammar:

$$\exp \rightarrow \exp + factor \mid factor$$
$$factor \rightarrow (\exp) \mid number$$

（1） Eliminate left recursion using BNF;

A:

$$\exp \rightarrow factor\ \exp'$$
$$\exp' \rightarrow + factor\ \exp' \mid \epsilon$$
$$factor \rightarrow (\exp) \mid number$$

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

A:

Production	Semantic Rules
1) $\exp \rightarrow factor\ \exp'$	$\exp'.inh = factor.val$ $\exp.val = \exp'.syn$
2) $\exp' \rightarrow + factor\ \exp_1{}'$	$\exp_1{}'.inh = \exp'.inh + factor.val$ $\exp'.syn = \exp_1{}'.syn$
3) $\exp' \rightarrow \epsilon$	$\exp'.syn = \exp'.inh$
4) $factor \rightarrow (\exp)$	$factor.val = \exp.val$
5) $factor \rightarrow number$	$factor.val = number.lexval$

（3） Convert the SDD of (2) to SDT.

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

$$\exp \rightarrow factor\{\exp'.inh = factor.val\}$$
$$\exp'\{\exp.val = \exp'.syn\}$$
$$\exp' \rightarrow +$$
$$factor\ \{\exp_1{}'.inh = \exp'.inh + factor.val\}$$
$$\exp_1{}'\ \{\exp'.syn = \exp_1{}'.syn\}$$
$$\exp' \rightarrow \epsilon\{\exp'.syn = \exp'.inh\}$$
$$factor \rightarrow (\exp)\{factor.val = \exp.val\}$$
$$factor \rightarrow number\{factor.val = number.lexval\}$$

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

A:

```
procedure exp
begin
    factor();
    exp'();
end exp

procedure exp'
begin
    while token = + do
        match(+);
        factor();
        exp'();
    end while;
end exp'

procedure factor
begin
    case token of
        ( : match(();
            exp();
            match());
        number : match(number);
        else error;
    end case;
end factor
```