Readme

This project is the code for the T4 of the following question.

Question

Consider the following grammar:

$$exp \rightarrow exp + factor \mid factor$$
$$factor \rightarrow (exp) \mid \mathbf{number}$$

1. Eliminate the left recursion using BNF;
2. Design an L-SDD to compute the value of the expression generated by the grammar of (1);
3. Convert the SDD of (2) to SDT;
4. Implement the SDT of (3) as a recursive-descent parser.

T1

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

T2

PRODUCTION	SEMANTIC RULES
$exp \rightarrow factor\ exp'$	$exp'.inh = factor.syn$ $exp.syn = exp'.syn$
$exp'_1 \rightarrow +factor\ exp'_2$	$exp'_2.inh = exp'_1.inh + factor.syn$ $exp'_1.syn = exp'_2.syn$
$exp' \rightarrow \epsilon$	$exp'.syn = exp'.inh$
$factor \rightarrow (exp)$	$factor.syn = exp.syn$
$factor \rightarrow \mathbf{number}$	$factor.syn = \mathbf{number}.lexval$

T3

PRODUCTION	SEMANTIC RULES	ACTIONS
$exp \rightarrow factor\ exp'$	$exp'.inh = factor.syn$ $exp.syn = exp'.syn$	$exp \rightarrow factor\{exp'.inh = factor.syn\}$ $exp'\{exp.syn = exp'.syn\}$
$exp'_1 \rightarrow +factor\ exp'_2$	$exp'_2.inh = exp'_1.inh + factor.syn$ $exp'_1.syn = exp'_2.syn$	$exp'_1 \rightarrow +factor\{exp'_2.inh = exp'_1.inh + factor.syn\}$ $exp'_2\{exp'_1.syn = exp'_2.syn\}$
$exp' \rightarrow \epsilon$	$exp'.syn = exp'.inh$	$exp' \rightarrow \epsilon\{exp'.syn = exp'.inh\}$
$factor \rightarrow (exp)$	$factor.syn = exp.syn$	$factor \rightarrow (exp)\{factor.syn = exp.syn\}$
$factor \rightarrow \mathbf{number}$	$factor.syn = \mathbf{number}.lexval$	$factor \rightarrow \mathbf{number}\{factor.syn = \mathbf{number}.lexval\}$

T4

- Code: Please refer to the source code.