Construção de Compiladores

Análise Sintática Ascendente LR
Exercício

Profa. Helena Caseli
helenacaseli@dc.ufscar.br

Análise Sintática Ascendente LR

- Exercício
  - Dada a gramática de expressões aritméticas

    (1) <E>::=<E>+<T>
    (2) <E>::=<T>
    (3) <T>::=<T>*<F>
    (4) <T>::=<F>
    (5) <F>::=(<E>)
    (6) <F>::=id

    a) Construa a tabela SLR
    b) Reconheça a cadeia x+y*z

ASA Simple LR

- Construindo a tabela sintática SLR
  - Construção do conjunto canônico de itens LR(0)
    1) Acrescentar à gramática a produção S' →S (onde S é o símbolo inicial da gramática)
       - Permite a identificação do fim da análise, mais especificamente, com a aplicação de S'→S.
    2) Computar as funções fechamento e desvio para a nova gramática
    - Fechamento
      - Seja I um conjunto de itens LR(0)
        - Todo item em I pertence ao fechamento(I)
        - Se A→α.Xβ está em fechamento(I) e X→γ é uma produção, então adiciona-se X→.γ ao conjunto
    - Desvio
      - Seja I um conjunto de itens LR(0), *desvio*(I,X): consiste em
        - Avançar o indicador (.) através do símbolo gramatical X (terminal ou não terminal) das produções correspondentes em I e
        - Calcular a função *fechamento* para o novo conjunto

ASA Simple LR

- Algoritmo para obter o conjunto canônico de itens

*C := { I_0 = fechamento({S'→ **.**S}) }*

repita

para cada conjunto I em C e X símbolo de G, tal que desvio(I,X) ≠ ∅

adicione desvio(I,X) a C

até que todos os conjuntos tenham sido adicionados a C

ASA Simple LR

- Algoritmo para construir a tabela de análise SLR

(**Após a construção do conjunto canônico C como descrito anteriormente, as tabelas AÇÃO e TRANSIÇÃO são construídas conforme esse algoritmo no qual a representa um terminal e A um não terminal**)
Seja C={I_0, I_1, ..., I_n}, os **estados** do analisador são 0 (inicial), 1, ...n (**estados**)
A linha i da tabela é construída pelo conjunto I_i como segue: (**linhas**)
Ações na tabela (**estado X terminal**)
Se desvio(I_i,a)=I_j, **então** AÇÃO[i,a]=s*j*
Com exceção da regra <S'>::= <S> adicionada, para todas as outras regras,
Se <A>::=α**.** está em I_i, **então**, para todo a ∈ Seguidor(A), faça
AÇÃO[i,a]=reduz *n*, em que *n* é o número da produção <A>::=α
Se <S'>::=<S>**.** está em I_i, **então** faça AÇÃO[i,$]=ACEITA
Transições na tabela (**estado X não terminal**)
Se desvio(I_i,A)=I_j, **então** TRANSIÇÃO(i,A)=j

- Entradas não definidas indicam erros
- Ações conflitantes indicam que a gramática não é SLR

Análise Sintática Ascendente LR

- Exercício
  - Dada a gramática de expressões aritméticas

(0) <E'>::=<E>
(1) <E>::=<E>+<T>
(2) <E>::=<T>
(3) <T>::=<T>*<F>
(4) <T>::=<F>
(5) <F>::=(<E>)
(6) <F>::=id

Na tabela, tem-se que:
- s*i* indica “empilhar *i*”
- r*i* indica “reduzir por regra *i*”

Tabela sintática LR

Estados	Ações						Transições		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				OK			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Análise Sintática Ascendente LR

- Exercício
  - Reconhecer a cadeia x+y*z

Estados	Ações						Transições		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				OK			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Pilha	Cadeia	Ação
0	id+id*id$	

(0) <E'>::=<E>
(1) <E>::=<E>+<T>
(2) <E>::=<T>
(3) <T>::=<T>*<F>
(4) <T>::=<F>
(5) <F>::=(<E>)
(6) <F>::=id

Análise Sintática Ascendente LR

- Algoritmo de análise sintática LR

empilha o estado inicial s_0;
concatena o símbolo delimitador $ no final da cadeia de entrada;
faz ip apontar para o primeiro símbolo da cadeia;
do *(*seja* s_n *o estado no topo da pilha e a o símbolo apontado por ip *)*
  if (*ação*[s_n,*a*]=*"empilhar* s_{n+1}*"*) **then**
    empilha a; *(* empilha *)*
    empilha s_{n+1};
    avança ip; *(* avança na leitura da entrada*)*
  else if (*ação*[s_n,*a*]=*"reduzir A→ β"*) **then**
      desempilha 2|β| elementos*; *(* s está no topo da pilha *)*
      empilha A; *(* reduz *)*
      empilha o estado indicado por transição[$s_{n-|\beta|}$, *A*]; (*$s_{n-|\beta|}$ no topo*)
    else if (*ação*[s_n,*a*]=*"aceitar"*) **then** ACEITA
      else ERRO;
until ACEITA **or** ERRO;

Análise Ascendente LR

(0) <E'>::=<E>
(1) <E>::=<E>+<T>
(2) <E>::=<T>
(3) <T>::=<T>*<F>
(4) <T>::=<F>
(5) <F>::=(<E>)
(6) <F>::=id

- Exercício
  - Reconhecer a cadeia x+y*z

Estados	Ações						Transições		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				OK			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Pilha	Cadeia	Ação
0	id+id*id$	s5
0id5	+id*id$	r6
0F3	+id*id$	r4
0T2	+id*id$	r2
0E1	+id*id$	s6
0E1+6	id*id$	s5
0E1+6id5	*id$	r6
0E1+6F3	*id$	r4
0E1+6T9	*id$	s7
0E1+6T9*7	id$	s5
0E1+6T9*7id5	$	r6
0E1+6T9*7F10	$	r3
0E1+6T9	$	r1
0E1	$	**ACEITA**