Compiladores: Parsing ascendente

Christiano Braga

March 2021

Parsing ascendente

- Das folhas até o topo: redução de uma string ao símbolo inicial da gramática.
- Gramática 4.1, livro-texto, pg. 244

```
E -> E + T | T
T -> T * F | F
F -> (E) | id
```

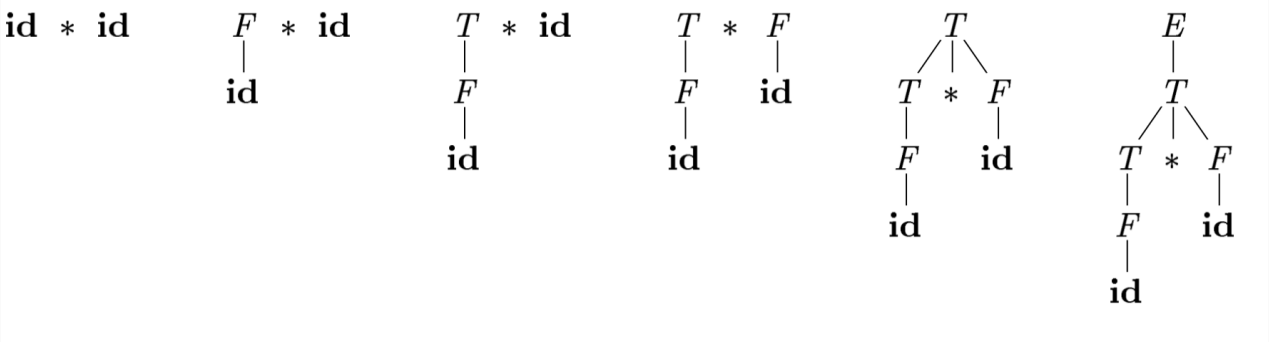

Figure 1: Parsing ascendente para id * id

Shift-reduce

- *Shift-reduce parsing* (empilha-reduz) é uma forma de parsing ascendente. Uma pilha guarda os símbolos gramaticais e um *buffer* guarda (o resto da) string sendo analisada.

STACK	INPUT	ACTION
$\$$	$\mathbf{id}_1 * \mathbf{id}_2 \$$	shift
$\$ \mathbf{id}_1$	$* \mathbf{id}_2 \$$	reduce by $F \to \mathbf{id}$
$\$ F$	$* \mathbf{id}_2 \$$	reduce by $T \to F$
$\$ T$	$* \mathbf{id}_2 \$$	shift
$\$ T *$	$\mathbf{id}_2 \$$	shift
$\$ T * \mathbf{id}_2$	$\$$	reduce by $F \to \mathbf{id}$
$\$ T * F$	$\$$	reduce by $T \to T * F$
$\$ T$	$\$$	reduce by $E \to T$
$\$ E$	$\$$	accept

Figure 2: Configurações do parsing shift-reduce para id * id

Items

- Como decidir entre shift e reduce? Itens.

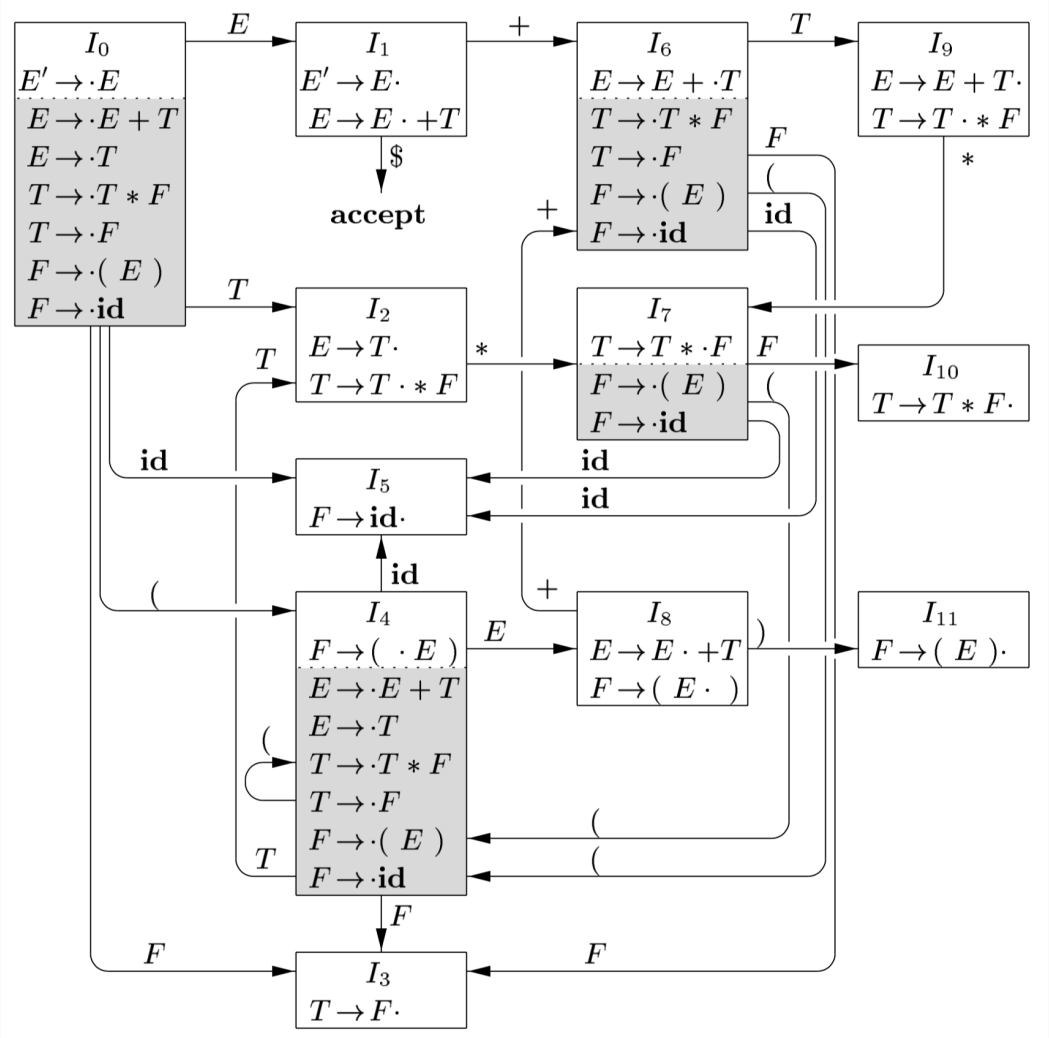

Figure 3: Autômato LR(0) para a gramática 4.1

Closure de um conjunto de itens

- Closure (fecho) de um conjunto de itens representa (parte do) o estado do autômato LR(0).

```
SetOfItems CLOSURE(I) {
        J = I;
        repeat
                for ( each item A → α·Bβ in J )
                        for ( each production B → γ of G )
                                if ( B → ·γ is not in J )
                                        add B → ·γ to J;
        until no more items are added to J on one round;
        return J;
}
```

Figure 4: Algoritmo para o cálculo da closure de um conjunto de itens

Goto

- A função goto define as transições do autômato LR(0).
- A função GOTO(I,X) é definida como a closure do conjunto de todos os itens $A \rightarrow \alpha X \cdot \beta$ desde que $A \rightarrow \alpha \cdot X\beta$ esteja em I.

Cannonical items

- Cannonical items (itens canônicos) representam os estados do autômato LR(0).

```
void items(G') {
        C = {CLOSURE({[S' → ·S]})};
        repeat
                for ( each set of items I in C )
                        for ( each grammar symbol X )
                                if ( GOTO(I, X) is not empty and not in C )
                                        add GOTO(I, X) to C;
        until no new sets of items are added to C on a round;
}
```

Figure 5: Algoritmo para o cálculo dos itens canônicos

Parsing LR

- As tabelas ACTION e GOTO formam a tabela de parsing utilizada pelo driver do parsing.

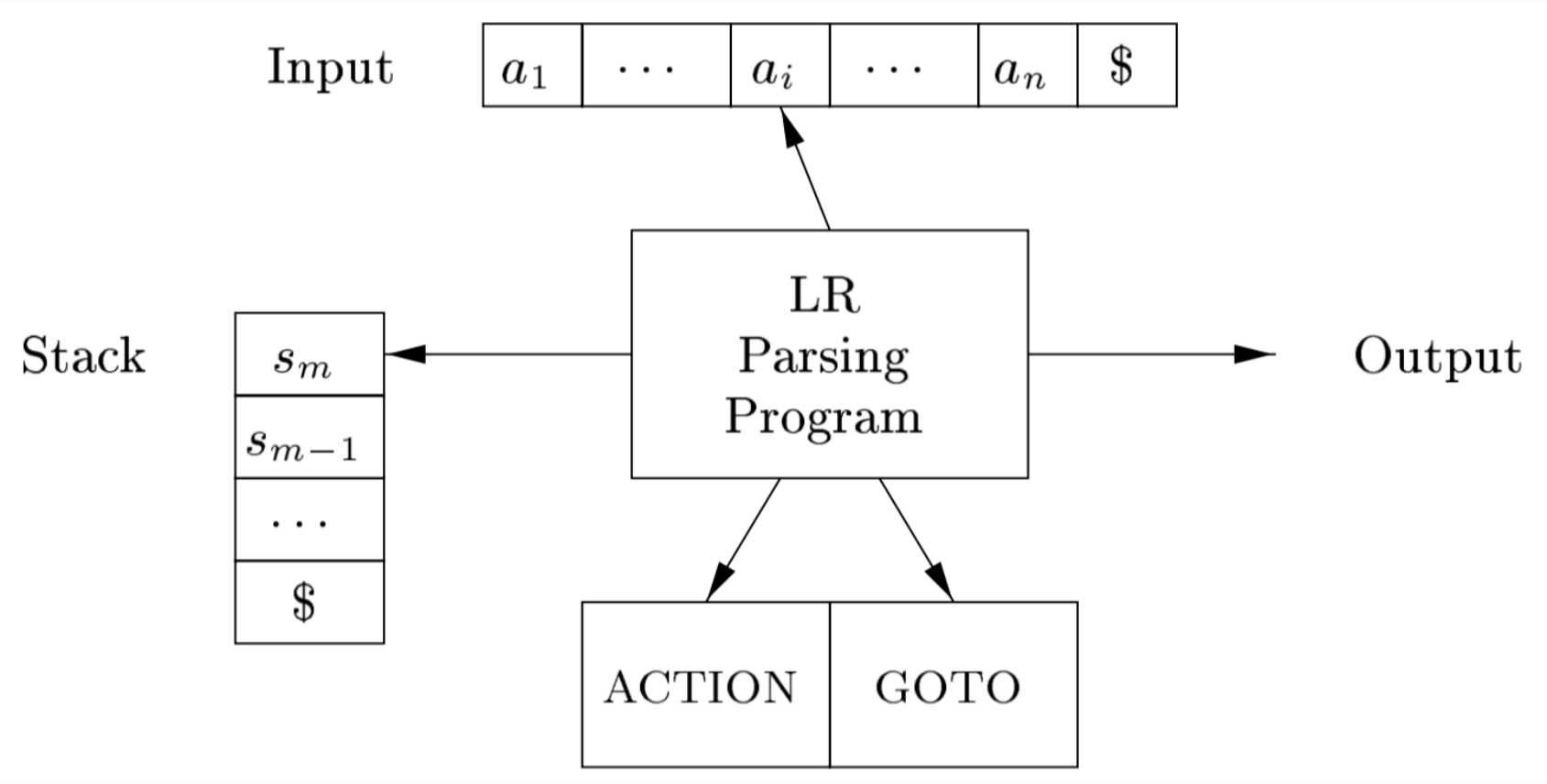

Figure 6: Modelo de um parser LR

Tabelas de parsing SLR

(1)	$E \to E + T$	(4)	$T \to F$
(2)	$E \to T$	(5)	$F \to (E)$
(3)	$T \to T * F$	(6)	$F \to \mathbf{id}$

Figure 7: Enumeração das regras

STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
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			

Figure 8: Tabelas de parsing SLR para a gramática 4.1

Construção da tabela de parsing

1. Construa $C = \{I_0, I_1, \ldots, I_n\}$
2. O i-ésimo estado é construído a partir de I_i. As ações associadas ao i-ésimo estado são:
   - 2.1 Se $A \to \alpha \cdot a\beta$ está em I_i e $GOTO(I_i, a) = I_j$ então $ACTION[i, a] = shift j$, onde a é um terminal.
   - 2.2 Se $A \to \alpha\cdot$ está em I_i, então $ACTION[i, a] = reduce\ A \to \alpha$ para todo a no FOLLOW(A), com $A \neq S'$.
   - 2.3 Se $S' \to S\cdot$ está em I_i, então $ACTION[i, \$] = accept$.
3. As transições para o i-ésimo estado são construídas para todos os não terminais A utilizando a regra se $GOTO(I_i, A) = I_j$ então $GOTO(i, A) = j$.

Algoritmo de parsing LR

METHOD: Initially, the parser has s_0 on its stack, where s_0 is the initial state, and $w\$$ in the input buffer. The parser then executes the program in Fig. 4.36. □

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Figure 9: Algoritmo de parsing LR

Exemplo

- Entrada **id**$

1. O parser começa com a pilha P = $ 0 e a entrada **id**$.
2. Executa a ação **shift 5**: P = $ 0 5 e a entrada $.
3. No estado 5 (no topo da pilha) e entrada $, o parser reduz pela regra 6: F $\rightarrow$ id, então P = $ 0 3 (pois 5 foi retirado, 0 passa a ser o topo e GOTO(0, F) = 3) e a entrada é $.
4. No estado 3 (no topo da pilha), lendo $ da entrada, o parser reduz pela regra 4: T $\rightarrow$ F, a pilha se torna P = $ 0 2 e a entrada $.
5. No estado 2 lendo $ da entrada, o parser reduz pela regra 2, E $\rightarrow$ T. A pilha então fica P = $ 0 1, pois GOTO(0, E) = 1 e a entrada é $.
6. No estado 1 lendo $ o autômato aceita a entrada.