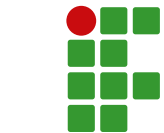
Análise Sintática Bottom-up

Compiladores

Prof. Daniel Saad Nogueira Nunes



INSTITUTO FEDERAL
Brasília

Campus
Taguatinga

Sumário

Sumário

Introdução

- Durante a análise sintática bottom-up, a construção da árvore sintática é feita de baixo para cima, ou seja, a partir dos terminais.
- Os terminais são **reduzidos** a símbolos não-terminais seguindo as regras da gramática.
- O processo termina quando o símbolo inicial, a raiz da árvore sintática, é alcançado.

Introdução

Tome a seguinte gramática:

$$
\begin{array}{ll}
1 & E \rightarrow E\ + T \\
2 & E \rightarrow T \\
3 & T \rightarrow T \cdot F \\
4 & T \rightarrow F \\
5 & F \rightarrow (E) \\
6 & F \rightarrow id
\end{array}
$$

Introdução

Tome a seguinte gramática:

$$
\begin{array}{ll}
1 & E \rightarrow E + T \\
2 & E \rightarrow T \\
3 & T \rightarrow T \cdot F \\
4 & T \rightarrow F \\
5 & F \rightarrow (E) \\
6 & F \rightarrow id
\end{array}
$$

Como a string $id \cdot id$ pode ser reduzida ao símbolo E?

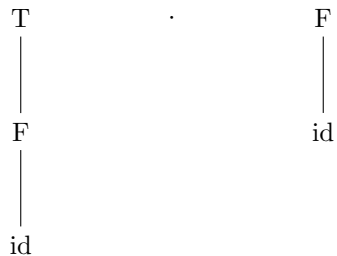
Introdução

$$\text{id} \quad \cdot \quad \text{id}$$

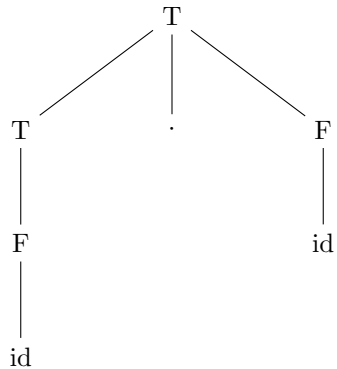
Introdução

Introdução

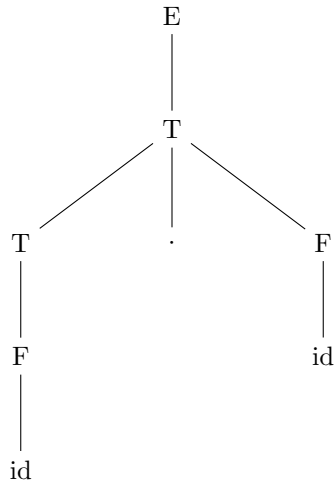
Introdução



Introdução



Introdução



Introdução

- Os símbolos são reduzidos aos seus respectivos não-terminais conforme ao inverso da ordem das derivações mais à direita:

$$E \underset{\text{rm}}{\Rightarrow} T \underset{\text{rm}}{\Rightarrow} T \cdot F \underset{\text{rm}}{\Rightarrow} T \cdot id \underset{\text{rm}}{\Rightarrow} F \cdot id \underset{\text{rm}}{\Rightarrow} id \cdot id$$

- Na prática, ocorre uma busca em pós-ordem na árvore sintática.
- As substrings substituídas pelos seus não terminais são chamadas de **handles**.

Sumário

Shift-reduce Parsing

- Shift-reduce é um processo bottom-up de análise sintática.
- Uma pilha é usada para armazenar símbolos da gramática e um buffer armazena o restante dos símbolos da entrada a serem analisados.
- O próximo símbolos do buffer de entrada é inserido na pilha, operação denominada de **shift**.
- Sempre que um handle é encontrado no podo da pilha, ele é substituído ao seu não-terminal correspondente. Essa operação é chamada de **reduce**.
- O processo termina quando a pilha contém apenas o símbolo inicial da gramática e o buffer está vazio.
- Quando um erro de sintaxe é encontrado, o processo pode ser interrompido com uma mensagem de erro.

Shift-reduce Parsing

Exemplo

O processo de shift reduce para a string $id \cdot id$ da gramática se dá da seguinte forma:

Pilha	Entrada	Ação
	$id \cdot id$	shift
id	$\cdot\, id$	reduce $F \to id$
F	$\cdot\, id$	reduce $F \to T$
T	$\cdot\, id$	shift
$T \cdot$	id	shift
$T \ \cdot id$		reduce $F \to id$
$T \ \cdot F$		reduce $T \to T \cdot F$
T		reduce $E \to T$
E		aceita

Shift-reduce Parsing

- Se um parser shift-reduce reconhece uma gramática ela é dita LR.
- Gramáticas LR são interessantes porque:
  - A maioria das linguagens de programação projetadas sob linguagens livres de contexto são LR.
  - O método de análise sintática shift-reduce é eficiente na prática.
  - Um parser LR consegue detectar um erro de sintaxe tão cedo quanto o possível em uma leitura da entrada da esquerda para a direita.
  - Gramáticas LR são um superconjunto das gramáticas LL, logo, conseguimos construir um parser eficiente mantendo a expressividade.

Shift-reduce Parsing

- Um dos métodos mais comuns de implementar o shift-reduce parsing é o **Simple LR parsing**, também conhecido como SLR.
- O nome vem de que queremos construir um parser LR(0) ou LR(1).
-

Sumário

Sumário

SLR: Itens

- Como um parser shift-reduce sabe quando é a hora de realizar a redução e quanto é a hora de inserir um símbolo da entrada na pilha?
- Ele mantém registro de quão avançado estamos no processo de redução.
- Para isso, usa o conceito de item.

SLR: Itens

Definição (Item LR(0))

Um item LR(0) de uma gramática G é uma regra da gramática adicionada de uma marcação ($\bullet$), que indica o quanto da regra já foi lido. Uma regra da forma $A \rightarrow X\ Y\ Z$ pode possuir até quatro itens:

- $A \rightarrow \bullet X\ Y\ Z$*: o ponto está antes do primeiro símbolo da regra;*
- $A \rightarrow X \bullet Y\ Z$*: o ponto está entre o primeiro e o segundo símbolo da regra;*
- $A \rightarrow X\ Y \bullet Z$*: o ponto está entre o segundo e o terceiro símbolo da regra;*
- $A \rightarrow X\ Y\ Z\bullet$*: o ponto está após o último símbolo da regra.*

Em especial, a produção $A \rightarrow \epsilon$ só possui um item, que é $A \rightarrow \bullet$.

SLR: Itens

- Um item da forma $A \rightarrow \bullet X\ Y\ Z$ indica que esperamos ver uma string derivável a partir de XYZ na entrada.
- Um item da forma $A \rightarrow X \bullet Y\ Z$ indica que já vimos uma string derivável de X na entrada e estamos esperando ver uma string derivável de YZ na entrada.
- Um item da forma $A \rightarrow X\ Y\ Z \bullet$ indica que já vimos uma string derivável de XYZ na entrada e que pode ser a hora de reduzir essa string para o não-terminal A.

Autômato LR(0)

- Uma coleção de conjuntos de itens LR(0) é chamada de coleção LR(0) canônica
- Ela é a base para construção do autômato LR(0).
- Cada estado de um autômato LR(0) é um conjunto de itens LR(0).
- Para construir o autômato LR(0), precisamos definir uma gramática aumentada e duas funções, CLOSURE e GOTO.

Autômato LR(0): gramática aumentada

Definição

(Gramática aumentada) Dada uma gramática G com símbolo inicial S, a gramática aumentada G' é a gramática que contém uma nova regra $S' \rightarrow S$, além, é claro, de todas as regras de G.

Autômato LR(0): gramática aumentada

- O propósito da gramática aumentada é dizer quando o parser deve parar e aceitar a entrada.
- Isso ocorre quando o símbolo S é reduzido a S'.

Autômato LR(0): gramática aumentada

Exemplo

Considere a gramática G:

$$
\begin{array}{ll}
1 & E \rightarrow E\ + T \\
2 & E \rightarrow T \\
3 & T \rightarrow T \cdot F \\
4 & T \rightarrow F \\
5 & F \rightarrow (E) \\
6 & F \rightarrow id
\end{array}
$$

Autômato LR(0): gramática aumentada

Exemplo

A gramática aumentada G' é:

$$
\begin{array}{ll}
1 & E' \to E \\
2 & E \to E \ + T \\
3 & E \to T \\
4 & T \to T \cdot F \\
5 & T \to F \\
6 & F \to (E) \\
7 & F \to id
\end{array}
$$

Autômato LR(0): função CLOSURE

- A função CLOSURE é responsável por calcular o fecho de um conjunto de itens LR(0).
- Intuitivamente, se $A \rightarrow \alpha \bullet B\ \beta \in \text{CLOSURE}(I)$, então, em algum momento, estamos considerando que podemos ver uma string derivável de $B\beta$ na entrada.
- Em outras palavras, a substring derivável de $B\beta$ terá um prefixo derivável de B ao aplicar as produções em que B ocorra do lado esquerdo.
- Então, se $B \rightarrow \gamma$ é uma produção da gramática, então o item $B \rightarrow \bullet\gamma$ deve ser adicionado ao conjunto de itens.

Autômato LR(0): função CLOSURE

Algorithm 1: CLOSURE(I)

Input: I: conjunto de itens

Output: fecho de I

```
1 ans ← I
2 repeat
3 |   prev ← ans
4 |   foreach A → α • B β ∈ ans do
5 |   |   foreach p ∈ PRODUCTIONS-FOR(B) do
6 |   |   |   ans ← ans ∪ {B → •RHS(p)}
7 until ans = prev
8 return ans
```

Autômato LR(0): função CLOSURE

Exemplo

Considere a gramática G:

$$
\begin{array}{ll}
1 & E' \to E \\
2 & E \to E + T \\
3 & E \to T \\
4 & T \to T \cdot F \\
5 & T \to F \\
6 & F \to (E) \\
7 & F \to id
\end{array}
$$

E tome $I = \{E' \to \bullet E\}$. Vamos computar $\text{CLOSURE}(I)$.

Autômato LR(0): função CLOSURE

Exemplo

1. $\text{CLOSURE}(I) = I$
2. Pela regra 2, $E \rightarrow \bullet E + T$ é adicionado a closure(I)
3. Pela regra 3, $E \rightarrow \bullet T$ é adicionado a closure(I)
4. Pelo item anterior e pela regra 4, $T \rightarrow \bullet T \cdot F$ é adicionado a closure(I)
5. Pelo item 3 e pela regra 5, $T \rightarrow \bullet F$ é adicionado a closure(I)
6. Pelo item anterior e pela regra 6, $F \rightarrow \bullet(E)$ é adicionado a closure(I)
7. Pelo item 5 e pela regra 7, $F \rightarrow \bullet id$ é adicionado a closure(I)

Assim, temos que $\text{CLOSURE}(I) = \{E' \rightarrow \bullet E, E \rightarrow \bullet E + T, E \rightarrow \bullet T, T \rightarrow \bullet T \cdot F, T \rightarrow \bullet F, F \rightarrow \bullet(E), F \rightarrow \bullet id\}$

Autômato LR(0): função GOTO

- A função GOTO é responsável por calcular o próximo estado do autômato LR(0) dado um estado e um símbolo. Isto é, $\text{GOTO}(I, X)$ é o estado alcançado a partir do estado I ao ler o símbolo X.
- Essa função é definida como o fecho de todos os itens $A \to \alpha\ X \bullet \beta$, tais que $A \to \alpha \bullet X\ \beta \in I$. Em outras palavras, $\text{GOTO}(I, X) = \text{CLOSURE}(I')$ em que

$$I' = \{A \to \alpha X \bullet \beta \mid A \to \alpha \bullet X\beta \in I\}$$

.

Autômato LR(0): função GOTO

Algorithm 2: $\text{GOTO}(I, X)$

Input: I: estado do autômato LR(0), X: símbolo da gramática
Output: Próximo estado do autômato LR(0)

1 $I' \leftarrow \text{ADVANCE-DOT}(I, X)$
2 **return** $\text{CLOSURE}(I')$

Algorithm 3: $\text{ADVANCE-DOT}(I, X)$

Input: I: estado do autômato LR(0), X: símbolo da gramática
Output: conjunto de itens $\{A \to \alpha X \bullet \beta \mid A \to \alpha \bullet X\beta \in I\}$

1 $I' \leftarrow \emptyset$
2 **foreach** $A \to \alpha \bullet X\beta \in I$ **do**
3 $\quad I' \leftarrow I' \cup \{A \to \alpha X \bullet \beta\}$
4 **return** I'

Autômato LR(0): função GOTO

Exemplo

Considere a gramática G:

$$
\begin{array}{ll}
1 & E' \to E \\
2 & E \to E\ + T \\
3 & E \to T \\
4 & T \to T \cdot F \\
5 & T \to F \\
6 & F \to (E) \\
7 & F \to id
\end{array}
$$

Tome $I = \{E' \to \bullet E, E \to E \bullet + T\}$. Queremos computar $\text{GOTO}(I, +)$.

Autômato LR(0): função GOTO

Exemplo

$I' = \{E \to E + \bullet T\}$, pois $E \to E \bullet + T$ é o único item com o símbolo $+$ após o ponto. Agora basta calcular

$$\text{CLOSURE}(I') = \left\{ \begin{array}{l} E \to E + \bullet T, T \to \bullet T \cdot F, \\ T \to \bullet F, F \to \bullet(E), F \to \bullet id \end{array} \right\}$$

Autômato LR(0)

Construção do autômato LR(0)

Com posse das funções $\textsc{closure}$ e $\textsc{goto}$, podemos construir Q, o conjunto de estados do autômato LR(0) da seguinte forma:

1. Adicione $\textsc{closure}(\{S' \to \bullet S\})$ a Q. Em que S' é o símbolo inicial da gramática aumentada.
2. Faça $Q' \leftarrow Q$.
3. Para cada estado $I \in Q'$ e cada símbolo X da gramática, calcule $\textsc{goto}(I, X)$.
4. Se $\textsc{goto}(I, X)$ é diferente de $\emptyset$ e não está em Q, adicione-o a Q.
5. Se $Q' = Q$, então pare, caso contrário, volte ao passo 2.

Autômato LR(0)

Algorithm 4: BUILD-LRZERO-AUTOMATA(G)

Input: G: a gramática aumentada

Output: Q: conjunto de estados do autômato LR(0)

```
1  Q ← {CLOSURE({S' → •S})}
2  repeat
3      Q' ← Q
4      foreach I ∈ Q' do
5          foreach X ∈ V ∪ Σ do
6              J ← GOTO(I, X)
7              if( J ≠ ∅ ∧ J ∉ Q )
8                  Q ← Q ∪ {J}
9  until Q' = Q
10 return Q
```

Autômato LR(0)

Exemplo

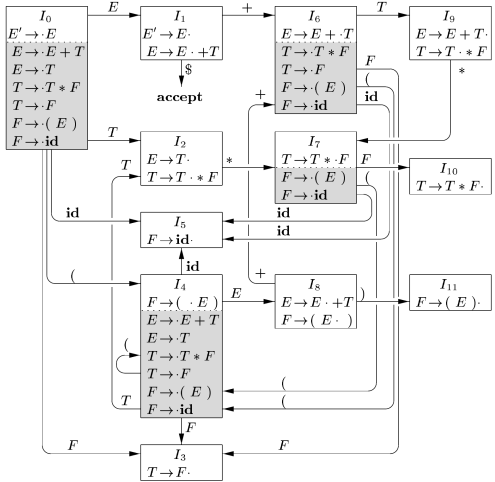
Considere a gramática G:

$$
\begin{array}{ll}
1 & E' \to E \\
2 & E \to E + T \\
3 & E \to T \\
4 & T \to T \cdot F \\
5 & T \to F \\
6 & F \to (E) \\
7 & F \to id
\end{array}
$$

O autômato LR(0) de G segue a seguir. As transições são dadas por $\text{GOTO}(I, X)$, onde I é o estado e X é o símbolo da gramática.

Autômato LR(0)

Exemplo



Autômato LR(0)

- Como o autômato LR(0) funciona?
- Suponha que a string γ de símbolos leva o autômato do estado 0, digamos, ao estado j.
- Uma operação de **shift** é realizada caso o próximo símbolo da entrada seja a e exista uma transição de j para outro estado com o símbolo a.
- Caso contrário, uma operação de **reduce** é realizada.
- No caso de uma redução de uma regra do tipo $A \rightarrow \alpha$, retiramos $|\alpha|$ símbolos do topo da pilha e seguimos a transição de acordo com A a partir do estado que ficou no topo da pilha.
- Não utilizaremos mais símbolos na pilha, mas sim números que representam os estados do autômato LR(0).

Autômato LR(0)

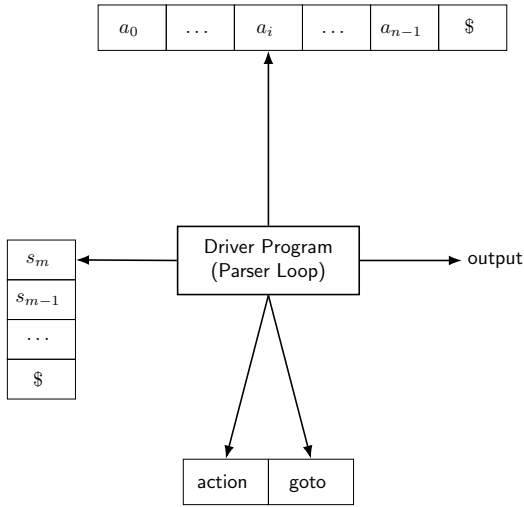
Pilha	Símbolos	Entrada	Ação
0		$id \cdot id\ \$$	shift para 5
$0\ 5$	id	$\cdot id\ \$$	reduce $F \to id$
$0\ 3$	F	$\cdot id\ \$$	reduce $T \to F$
$0\ 2$	T	$\cdot id\ \$$	shift para 7
$0\ 2\ 7$	$T\cdot$	$id\ \$$	shift para 5
$0\ 2\ 7\ 5$	$T \cdot id$	$\$$	reduce $F \to id$
$0\ 2\ 7\ 10$	$T \cdot F$	$\$$	reduce $T \to T \cdot F$
$0\ 2$	T	$\$$	reduce $E \to T$
$0\ 1$	E	$\$$	aceita

Sumário

Algoritmo de parsing LR

- O algoritmo de parsing LR implementa o autômato.
- Para isso, ele depende de uma entrada, uma pilha, um driver e uma tabela de ações, com duas partes, action e goto.
- O driver é o mesmo para todos os parsers LR, o que muda é a tabela de ações.

Algoritmo de parsing LR



Algoritmo de parsing LR

action

A tabela action toma um estado i e um terminal a de forma que action$[i][a]$ possui um dos 4 valores:

1. Shift j, em que j é um estado. Nesta ação, o símbolo a, representado pelo estado j é inserido na pilha.
2. Reduce $A \rightarrow \beta$. Nesta ação, o parser reduz β no topo da pilha ao não-terminal A.
3. Accept. Nesta ação, o parser aceita a entrada.
4. Error. Nesta ação, o parser descobriu um erro de sintaxe.

Algoritmo de parsing LR

goto

A tabela goto toma um estado i e um não-terminal A de forma que goto$[i][A] = j$ implica que o estado i é mapeado ao estado j ao verificar A.

Algoritmo de parsing LR

Algorithm 5: LR-DRIVER

```
1  S ← STACK()
2  ac ← false
3  S.PUSH(0)
4  while ¬ac do
5      a ← input.PEEK()
6      s ← S.TOP()
7      if( action[s][a] = shift t )
8          S.PUSH(t)
9          input.ADVANCE()
10     else if( action[s][a] = reduce A → β )
11         for( i ← 1 to |β| )
12             S.POP()
13             S.PUSH(goto[S.TOP(), A])
14     else if( action[s][a] = accept )  ac ← true
15     else  REPORT-ERROR()
```

Sumário

Construção das tabelas de ações SLR

O método SLR para construir as tabela de ação funciona da seguinte forma:

1. Construa $C = \{I_0, \ldots, I_{n-1}\}$ a coleção de conjunto de itens LR(0) de G'.
2. O estado i da tabela de ações construído de I_i. As ações para o estado i são dadas da seguinte forma:
   - Se $A \to \alpha \bullet a\beta \in I_i$ e $\text{GOTO}(I_i, a) = I_j$, então action$[i][a]$ = shift j. a precisa ser um terminal.
   - Se $A \to \alpha\bullet \in I_i$, então action$[i][a]$ = reduce $A \to \alpha$ para todo terminal $a \in \text{FOLLOW}(A)$. A aqui não pode ser S'.
   - Se $S' \to S\bullet \in I_i$, então action$[i][\$]$ = accept.
3. As transições da tabela goto são dadas por goto$[i][A] = j$ se $\text{GOTO}(I_i, A) = I_j$. A aqui é um não-terminal.
4. Todas as entradas não preenchidas da tabela de ações são marcadas como error.
5. O estado inicial do parser é aquele obtido do conjunto de itens que contém $S' \to \bullet S$.

Construção das tabelas de ações SLR

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			

Construção das tabelas de ações SLR

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept

Figure 4.38: Moves of an LR parser on $\mathbf{id} * \mathbf{id} + \mathbf{id}$