INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

Curso de Engenharia de Computação
ECM253 – Linguagens Formais, Autômatos e Compiladores

Análise Sintática Ascendente

Slides da disciplina ECM253 – Linguagens Formais, Autômatos e Compiladores
Curso de Engenharia de Computação
Instituto Mauá de Tecnologia – Escola de Engenharia Mauá
Prof. Marco Antonio Furlan de Souza
<marco.furlan@maua.br>

Análise sintática ascendente

- **Conceitos**
  - **Analisadores sintáticos** *parsers* **ascendentes** *bottom-up* **constroem** uma **árvore sintática** começando com suas **folhas** e **trabalhando em direção à raiz**;
  - O parser **constrói** um nó de **folha** na **árvore** para cada **palavra** retornada pelo **scanner**. Essas folhas **formam** a **borda inferior** da **árvore** sintática;
  - Para montar uma **derivação**, o parser **acrescenta camadas** de **não terminais** em **cima das folhas** em uma **estrutura orientada** tanto **pela gramática quanto** pela **parte inferior parcialmente completa** da árvore sintática;
  - Em qualquer **estágio** da análise, a **árvore parcialmente completa** representa seu **estado**;
  - Cada **palavra que o scanner tiver retornado** é representada por uma **folha**. Os **nós acima das folhas codificam todo o conhecimento que o parser já derivou**;
  - O **parser funciona junto à fronteira superior** dessa árvore sintática parcialmente completa; essa **fronteira corresponde** à **forma sentencial atual** na derivação sendo montada pelo parser.

Análise sintática ascendente

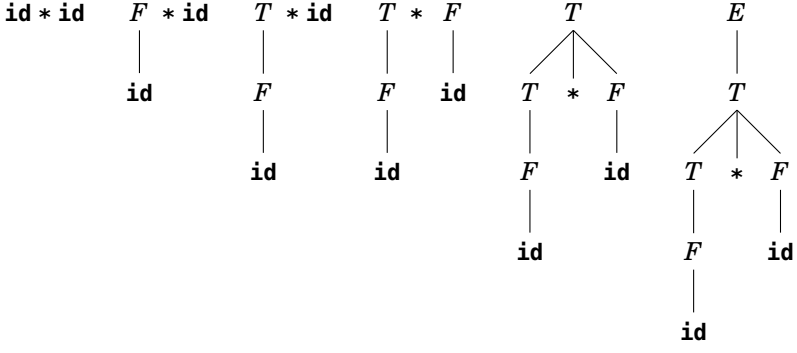
- **Conceitos**(cont.)
  - Para **estender** a fronteira **para cima**, o parser examina a **fronteira atual** em busca de uma **substring** que **corresponda** ao **lado direito** de alguma produção $A \rightarrow \beta$;
  - Por **exemplo**, considere uma **gramática de expressões** assim:

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \texttt{id}$$

  - E a seguinte **entrada** (considerar `id` qualquer identificador): **`id * id`**.

Análise sintática ascendente

- **Conceitos**(cont.)
  - O **processo** de **análise ascendente** está ilustrado a seguir:



  - O objetivo da análise ascendente é construir uma árvore de derivação ao inverso. A derivação a seguir corresponde à figura anterior: $E \Rightarrow T \Rightarrow T * F \Rightarrow T * \mathbf{id} \Rightarrow F * \mathbf{id} \Rightarrow \mathbf{id} * \mathbf{id}$.

Análise sintática ascendente

- **Conceitos**(cont.)
  - Para estender a fronteira para cima, **o parser examina a fronteira atual** em **busca** de uma **substring** que corresponda ao **lado direito** de alguma **produção** $A \rightarrow \beta$. Se **encontrar** β na **fronteira**, com sua **extremidade direita** em k, pode **substituir** β por A, para **criar** uma nova fronteira;
  - Se a **substituição** de A **por** β na **posição** k for a **próxima etapa** em uma **derivação válida** para a string de entrada, **então** o **par** $\langle A \rightarrow \beta, k \rangle$ é um **handle** (alça) na derivação atual, e o parser deverá **substituir** β por A. Essa **substituição** é **chamada redução**, porque **reduz** o **número** de **símbolos** na **fronteira**, a menos que $|\beta| = 1$;
  - Se o parser estiver montando uma árvore sintática, **constrói** um **nó** para A, **acrescenta-o** à **árvore** e **conecta** os nós **representando** β como **filhos** de A;
  - O parser bottom-up repete um **processo simples**. **Encontra** um **handle** $\langle A \rightarrow \beta, k \rangle$ na **fronteira**, e substitui a ocorrência de β em k por A. Esse **processo continua até** que ele: (1) **reduza** a **fronteira** para **um único nó** que representa o **símbolo-alvo** da gramática, **ou** (2) **não possa encontrar um handle**. No **primeiro caso**, o parser **encontrou** uma **derivação**; se também tiver consumido todas as palavras no fluxo de entrada (ou seja, a próxima palavra é **`eof`**), então a **análise** tem **sucesso**. No **segundo caso**, ele não pode montar uma derivação para o fluxo de entrada e deve relatar esta **falha**.

Algoritmo LR(1)

- **Conceitos**
  - **LR(1)**: **lê o texto de entrada** da **esquerda** (**L**eft) **para** a **direita** e realiza **derivações** à **direita** (**R**ight) **utilizando 1 símbolo** de **verificação** à frente (*lookahead*);
  - Trata-se de um método **dirigido por tabela**;
  - A **localização eficiente do handle** é a **chave** para a análise ascendente **eficiente**;
  - Um parser LR(1) usa um **autômato de localização de handles** codificado em **duas tabelas**, chamadas **Action** e **Goto**;
  - Ao invés de montar uma árvore sintática explícita, **o algoritmo mantém** a **fronteira superior atual da árvore parcialmente construída** em uma **pilha**, **intercalada** com **estados** do **autômato** de **localização** de **handles** que lhe permitem **encadear** as **reduções** para uma análise;
  - Em qualquer ponto na análise, a **pilha contém um prefixo da fronteira atual**. **Além deste prefixo**, a **fronteira** consiste em **nós-folha**. Uma **variável** do algoritmo **mantém** a **primeira palavra** no **sufixo** que se encontra **além** do **conteúdo da pilha** – este é o símbolo de **verificação à frente**.

Algoritmo LR(1)

- **Tabela de análise**
  - **Dirige** a execução do **algoritmo** LR(1). Aqui se omitirão os detalhes de como esta tabela é construída – consultar a bibliografia desta aula:
    - Primeiramente elabora-se um **DFA** cujos **estados representam** fechamento de **itens** (handles) a partir de um item inicial. As transições são rotuladas por **não terminais** e **terminais** de acordo com regras que podem ou não alterar o símbolo de verificação à frente;
    - As **linhas** da tabela são rotuladas com os **estados** do **DFA**;
    - Um **estado** pode **possuir** tanto **carregamentos** quanto **reduções**;
    - Cada **célula** da **tabela** na parte de **Entrada** pode ser **rotulada** como **carrega** (s) ou **reduz** (r), seguido respectivamente do **número** do **estado** e do **número** da **regra** a ser reduzida;
    - Acrescentam-se colunas para atender o **carregamento de não terminais** após uma redução (coluna **Goto**). Desloca-se para o número do estado indicado na coluna **Goto** após uma redução.

Algoritmo LR(1)

- **Exemplo**
  - Considerar a gramática a seguir, onde as regras serão enumeradas para serem aplicadas pelo algoritmo:

    (1) $A \rightarrow (A)$

    (2) $A \rightarrow$ a
  - A partir dela, pode-se construir a seguinte tabela (processo omitido):

Estado	**Entrada**				**Goto**
	(	a	)	$	A
0	s2	s3			1
1				aceita	
2	s5	s6			4
3				r2	
4			s7		
5	s5	s6			8
6			r2		
7				r1	
8			s9		
9			r1		

Algoritmo LR(1)

- **Exemplo**
  - A simulação envolve uma pilha e análise sintática e a entrada. O símbolo $ representa tanto o topo da pilha quanto o fim da entrada. Considerar como entrada **(((a)))**:

Estado	Entrada				Goto
	(	a	)	$	A
0	s2	s3			1
1				aceita	
2	s5	s6			4
3				r2	
4			s7		
5	s5	s6			8
6			r2		
7				r1	
8			s9		
9			r1		

	Pilha de análise sintática	Entrada	Ação
1	$0	(((a)))$	carrega 2
2	$0(2	((a)))$	carrega 5
3	$0(2(5	(a)))$	carrega 5
4	$0(2(5(5	a)))$	carrega 6
5	$0(2(5(5a6	)))$	reduz com 2
6	$0(2(5(5	)))$	goto 8 em A
7	$0(2(5(5$A$8	)))$	carrega 9
8	$0(2(5(5$A$8)9	))$	reduz com 1
9	$0(2(5	))$	goto 8 em A
10	$0(2(5$A$8	))$	carrega 9
11	$0(2(5$A$8)9	)$	reduz com 1
12	$0(2	)$	goto 4 em A
13	$0(2$A$4	)$	carrega 7
14	$0(2$A$4)7	$	reduz com 1
15	$0	$	goto 1 em A
16	0A$1	$	aceita

Referências bibliográficas

AHO, A. V.; SETHI, R.; LAM, M. S. **Compiladores: princípios, técnicas e ferramentas**. 2. ed. [s.l.] Pearson, 2007.

COOPER, K.; TORCZON, L. **Construindo compiladores**. 2. ed. Rio de Janeiro: Elsevier, 2014.

LOUDEN, K. C. **Compiladores: princípios e práticas**. [s.l.] Pioneira Thomson Learning, 2004.