

Compiladores - Análise Ascendente

Fabio Mascarenhas – 2015.2

<http://www.dcc.ufrj.br/~fabiom/comp>

Análise Descendente vs. Ascendente

- As técnicas de análise que vimos até agora (recursiva com retrocesso, recursiva preditiva, LL(1) de tabela) usam a mesma estratégia de análise: a *análise descendente*, ou *top-down*
- Vamos ver agora uma outra estratégia de análise, a *ascendente*, ou *bottom-up*, e as técnicas que a utilizam
- A diferença mais visível entre as duas é a forma de construção da árvore: na análise descendente construímos a árvore de cima para baixo, começando pela raiz, e na ascendente de baixo para cima, começando pelas folhas

Análise Ascendente

- A análise ascendente é mais complicada de implementar, tanto para um analisador escrito à mão (o que é muito raro) quanto para geradores
- Mas é mais geral, o que quer dizer que impõe menos restrições à gramática
- Por exemplo, recursão à esquerda e prefixos em comum não são problemas para as técnicas de análise ascendente
- Vamos usar um exemplo que deixa essas vantagens bem claras

Gramática de Expressões

- Vamos usar como exemplo uma gramática de expressões simplificada:

~~)-E~~
E \rightarrow E + T
E \rightarrow E - T
E \rightarrow T
T \rightarrow T * F
T \rightarrow F
F \rightarrow - F
F \rightarrow num
F \rightarrow (E)

- Vamos analisar a cadeia num * num + num

+ *

Reduções

- A análise ascendente analisa uma cadeia através de *reduções*, aplicando as regras da gramática ao contrário:

num * num + num
~~F~~ * num + num
~~T~~ * num + num
 T * ~~F~~ + num
~~T~~ + num
 E + num
 E + ~~F~~
 E + T
 E

- Vamos ler a sequência de reduções de trás para a frente: $E \rightarrow E + T \rightarrow E + F \rightarrow E + \text{num} \rightarrow T + \text{num} \rightarrow T * F + \text{num} \rightarrow T * \text{num} + \text{num} \rightarrow F * \text{num} + \text{num} \rightarrow \text{num} * \text{num} + \text{num}$
 → derivação à direita

Reduções vs derivações

- A sequência de reduções da análise ascendente equivale a uma *derivação mais à direita*, lida de trás pra frente
- Lembre-se que, para uma gramática não ambígua, cada entrada só pode ter uma única derivação mais à direita
- Isso quer dizer que a sequência de reduções também é única! O trabalho do analisador é então achar qual a próxima redução que tem que ser feita a cada passo

Exercício

- Qual a sequência de reduções para a cadeia - (num + num) - num

$$-(\underline{num} + num) - num$$

$$-(F + num) - num$$

$$-(T + num) - num$$

$$-(E + num) - num$$

$$-(E + F) - num$$

$$-(E + T) - num$$

$$-(E) - num$$

$$- F - num$$

$$F - num$$

$$T - num$$

$$E - num$$

$$E - F$$

$$E - T$$

$$E$$

$$E \rightarrow E + T$$

$$E \rightarrow E - T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

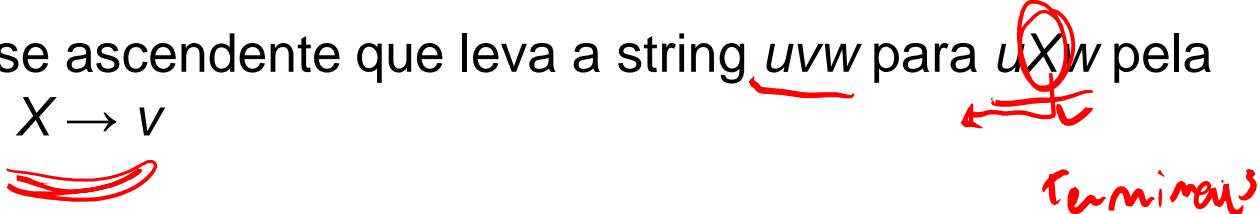
$$T \rightarrow F$$

$$F \rightarrow - F$$

$$F \rightarrow num$$

$$F \rightarrow (E)$$

Análise shift-reduce

- As reduções da análise ascendente formam uma derivação mais à direita de trás para frente
- Tomemos o passo da análise ascendente que leva a string uvw para uXw pela redução usando uma regra $X \rightarrow v$

- O pedaço w da string só tem terminais, pois essa redução corresponde ao passo $uXw \rightarrow uvw$ de uma derivação mais à direita
- Isso implica que a cada passo da análise temos um *sufixo* que corresponde ao resto da entrada que ainda não foi reduzido

Análise shift-reduce

| w
└─┬─┘
entrada

- Vamos marcar o foco atual da análise com uma |
 - À direita desse foco temos apenas terminais ainda não reduzidos
 - À esquerda temos uma mistura de terminais e não-terminais
 - Imediatamente à esquerda do foco temos um potencial candidato à redução
 - O foco começa no início da entrada
- A *análise shift-reduce* funciona tomando uma de duas ações a cada passo: shift, que desloca o foco para à direita, e reduce, que faz uma redução

Shift e reduce

- Shift: move o foco uma posição à direita

CONTINUAÇÃO

- $A B C | x y z \Rightarrow A B C x | y z$ é uma ação *shift*

- Reduce: reduz o que está imediatamente à esquerda do foco usando uma produção

- Se $A \rightarrow x y$ é uma produção, então $C b x y | i j k \Rightarrow C b A | i j k$ é uma ação *reduce* $A \rightarrow x y$

- Acontece um erro sintático quando não se pode tomar nenhuma das duas ações, e reconhecemos a entrada quando o chegamos a $S |$, onde S é o símbolo inicial

Exercício

- Qual a sequência de ações para a cadeia - (num + num) - num

Handwritten analysis of the sequence of actions for the string - (num + num) - num. The analysis is organized into three columns of partial strings and their corresponding indices.

Column 1 (Left):

- $\triangleright | - (num + num) - num$
- $\triangleright - | (num + num) - num$
- $\triangleright - (| num + num) - num$
- $num - (num | + num) - num$
- $num - (num | + num) - num$
- $num - (num | + num) - num$
- $num - (num | + num) - num$

Column 2 (Middle):

- $- (E | + num) - num \triangleright$
- $- (E + | num) - num \triangleright$
- $- (E + num |) - num \triangleright 6$
- $- (E + num |) - num \triangleright 4$
- $- (E + T |) - num \triangleright 0$
- $- (E |) - num \triangleright$
- $- (E) | - num \triangleright 7$
- $- F | - num \triangleright 5$

Column 3 (Right):

- $E | - num \triangleright 4$
- $T | - num \triangleright 2$
- $E | - num \triangleright$
- $E - | num \triangleright$
- $E - num | \triangleright 6$
- $E - F | \triangleright 4$
- $E - T | \triangleright 4$
- $(E |) \checkmark$**

Legend:

- 0 E -> E + T
- 1 E -> E - T
- 2 E -> T
- 3 T -> T * F
- 4 T -> F
- 5 F -> - F
- 6 F -> num
- 7 F -> (E)

Implementação

- O que está à esquerda do foco pode ser implementado usando uma pilha
- O foco é o topo da pilha mais uma posição na entrada
- A ação de *shift* empilha o próximo token e incrementa a posição
- A ação de *reduce* $A \rightarrow w$:
 - Desempilha $|w|$ símbolos (que devem formar w , ou a redução estaria errada)
 - Empilha A