Compiladores e Computabilidade

Prof. Leandro C. Fernandes
UNIP – Universidade Paulista, 2018

(2ª fase da Análise)

ANÁLISE SINTÁTICA DESCENDENTE

Tipos de Analisadores Sintáticos

Métodos Descendentes (*Top Down*):

- Constroem a árvore sintática de cima para baixo (da raiz para as folhas), ou seja, do símbolo inicial da gramática para a sentença.
  - Analisadores Descendentes Recursivos
  - Analisadores LL(k)

A análise sintática ...

- **Tarefa:** Dada uma gramática livre de contexto G e uma sentença *s*, o analisador sintático deve verificar se *s* pertence a linguagem gerada por G.
  - O analisador tenta construir a árvore de derivação para *s* segundo as regras de produção dadas pela gramática G.
  - Se esta tarefa for possível, o programa é considerado sintaticamente correto.

A análise sintática ...

- Observe que o analisador não precisa efetivamente construir a árvore, mas sim comprovar que é possível construí-la.
  - Esse processo pode ser emulado utilizando-se uma pilha de dados.

ANALISADORES DESCENDENTES RECURSIVOS

Análise Descendente Recursiva

- A técnica chamada de descida recursiva (*recursive descent*), é uma forma de análise descendente em que a gramática é transcrita na forma de rotinas responsáveis por processar sua derivação.

- Cada símbolo não-terminal se transforma em um procedimento.
  - Uma chamada ao procedimento *A* tem a finalidade de encontrar a maior cadeia que pode ser derivada do não-terminal *A*, a partir do ponto inicial de análise.

Análise Descendente Recursiva

- Quando realizamos a expansão pela regra $A \rightarrow x_1 x_2 ... x_n$, o procedimento *A* faz uma série de chamadas correspondentes aos elementos $x_1, x_2, ..., x_n$:
  - Se x_i é um não-terminal, o procedimento x_i é chamado;
  - Se x_i é um terminal, fazemos uma chamada *check*(x_i) que verifica a presença do terminal x_i na posição corrente da entrada e aciona o analisador léxico para a obtenção do próximo símbolo.

Análise Descendente Recursiva

- Suponha a gramática:

G=({L,S,I},{(,,,),a},P,S)

P = { L → (S)

S → I,S | I

I → a | L }

- Nossa gramática apresenta três regras, então teremos três procedimentos, cada um responsável por verificar o produto derivado a partir cada símbolo não-terminal.
- Veja como ficaria ...

Análise Descendente Recursiva

- Suponha a gramática:

G=({L,S,I},{(,,,),a},P,S)

P = { L → (S)

S → I,S | I

I → a | L }

```
void parseL() {
  check("(");
  parseS();
  check(")");
}
```

Análise Descendente Recursiva

- Suponha a gramática:

G=({L,S,I},{(,,,),a},P,S)

P = { L → (S)

S → I,S | I

I → a | L }

```
void parseS() {
  parseI();
  while (tk==",") {
     check(",");
     parseI();
  }
}
```

Análise Descendente Recursiva

- Suponha a gramática:

G=({L,S,I},{(,,,),a},P,S)

P = { L → (S)

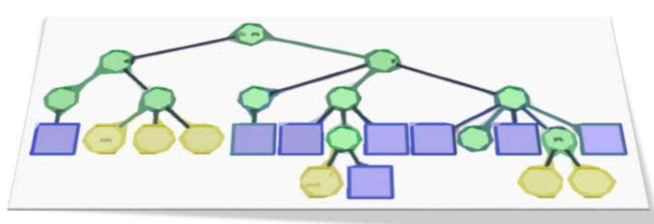
S → I,S | I

I → a | L }

```
void parseI() {
  switch (tk) {
    case "a":
       check("a");
       break;
    case "(":
       parseL();
       break;
    default: ERRO();
  }
}
```

Análise Descendente Recursiva

- Se a gramática considerada é LL(1), a construção dos procedimentos pode ser feita de maneira automática.
- Por outro lado, se a gramática não é LL(1), ainda é possível construir um analisador de descida recursiva, embora deixa de ser automática.
  - Na prática, este é o caso mais interessante, uma vez que gramática é LL(1), o próximo analisador é sempre mais eficiente do que o de descida recursiva.



ANALISADORES LL(1)

Analisadores LL(1)

O nome LL(1) indica que:

- L: *left-to-right*
  - a cadeia de entrada é examinada da esquerda para a direita
- L: *leftmost*
  - O analisador procura construir uma derivação esquerda
- 1: *lookahead*
  - Apenas um símbolo do restante da entrada é examinado.

Analisadores LL(1)

- É um analisador descendente, portanto, constrói a árvore de derivação correspondente ao programa *cima para baixo*. Isto é, parte do símbolo inicial S (da raiz) em direção as folhas, onde se encontra os tokens do programa.
- Nos métodos descendentes (*top-down*), temos de decidir qual a regra $A \rightarrow \beta$ a ser aplicada a um nó rotulado por um não-terminal A.
  - A *expansão* de A é feita criando nós filhos rotulados com os símbolos de β.

Exemplo

- Suponha a gramática:

(1) $E \rightarrow E + T$
(2) $E \rightarrow T$
(3) $T \rightarrow T * F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow a$

Considere a cadeia: a+a*a

Podemos obtê-la:

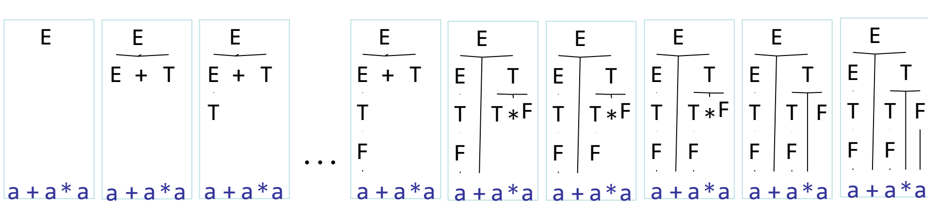
$E \Rightarrow E{+}T \Rightarrow T{+}T \Rightarrow F{+}T \Rightarrow a{+}T \Rightarrow a{+}T{*}F \Rightarrow a{+}F{*}F \Rightarrow a{+}a{*}F \Rightarrow a{+}a{*}a$

Derivação *left-most*.

Exemplo

Construindo a árvore de derivação para a cadeia: a+a*a
Tomemos como base a derivação *left-most*:

$$E \Rightarrow E{+}T \Rightarrow T{+}T \Rightarrow F{+}T \Rightarrow a{+}T \Rightarrow a{+}T{*}F \Rightarrow a{+}F{*}F \Rightarrow a{+}a{*}F \Rightarrow a{+}a{*}a$$



Análise Descendente

- A representação do processo será feita através de uma pilha de dados
- Para tal, temos:
  - configurações (a, y),
    - a = o conteúdo da pilha
    - y = o *resto* da entrada ainda não analisada.
- Existem duas formas de transição:
  - *Expansão de um não-terminal pela regra* $A \rightarrow \beta$:
    permite passar da configuração ($A\alpha$, y) para a configuração ($\beta\alpha$, y).
  - *Verificação de um terminal* a: permite passar da configuração (aα, ay) para a configuração (α, y).

Como escolher a regra certa?

- A idéia é utilizar duas informações:
  - o não-terminal A a ser expandido
  - e o primeiro símbolo a do resto da entrada.
- Uma tabela M com essas duas entradas nos dará a regra a ser utilizada: M[A, a].
- Essa técnica só pode ser usada para uma classe restrita de gramáticas, a classe das gramáticas LL(1).

Exemplo

- Suponha a gramática:
  - (1) E → T E'
  - (2) T → F T'
  - (3) F → (E)
  - (4) F → a
  - (5) E' → + T E'
  - (6) E' → ε
  - (7) T' → * F T'
  - (8) T' → ε

Essa gramática é LL(1).
Com a seguinte tabela M:

	(	a	+	*	)	$
E	1	1	–	–	–	–
T	2	2	–	–	–	–
F	3	4	–	–	–	–
E'	–	–	5	–	6	6
T'	–	–	8	7	8	8

Exemplo

Pilha	Entrada	Regra
E	a+a*a	M[E,a] = 1
TE'	a+a*a	M[T,a] = 2
FT'E'	a+a*a	M[F,a] = 4
aT'E'	a+a*a	–
T'E'	+a*a	M[T',+] = 8
E'	+a*a	M[E',+] = 5
+TE'	+a*a	–
TE'	a*a	M[T,a] = 2
FT'E'	a*a	M[F,a] = 4
aT'E'	a*a	–
T'E'	*a	M[T',*] = 7

	(	a	+	*	)	$
E	1	1	–	–	–	–
T	2	2	–	–	–	–
F	3	4	–	–	–	–
E'	–	–	5	–	6	6
T'	–	–	8	7	8	8

(1) E → T E'
(2) T → F T'
(3) F → (E)
(4) F → a
(5) E' → + T E'
(6) E' → ε
(7) T' → * F T'
(8) T' → ε

Exemplo

Pilha	Entrada	Regra
T'E'	*a	M[T',*] = 7
*FT'E'	*a	–
FT'E'	a	M[F,a] = 4
aT'E'	a	–
T'E'	ε	M[T',$] = 8
E'	ε	M[E',$] = 6
ε	ε	–

	(	a	+	*	)	$
E	1	1	–	–	–	–
T	2	2	–	–	–	–
F	3	4	–	–	–	–
E'	–	–	5	–	6	6
T'	–	–	8	7	8	8

(1) $E \to T\ E'$
(2) $T \to F\ T'$
(3) $F \to (\ E\ )$
(4) $F \to a$
(5) $E' \to +\ T\ E'$
(6) $E' \to \varepsilon$
(7) $T' \to *\ F\ T'$
(8) $T' \to \varepsilon$

Antes de continuar ...

- A partir deste ponto, você precisa saber como calcular:
  – Geradores de ε,
  – Conjunto First(α), e
  – Conjunto Follow(α).

  ... antes de poder construir a tabela para o analisador LL(1).

- Faça uma pausa e estude o material:
  “Coletando informações sobre a gramática”

Como construir a tabela?

- O símbolo *a* é o primeiro símbolo derivado do não-terminal a ser expandido A, e faz parte de First(A). Neste caso, α deve pertencer a First(α), onde $A \rightarrow \alpha$ é uma das alternativas de regra para A.
  - a regra $F \rightarrow (E)$ foi usada com o símbolo (
- Outra possibilidade é a de que não seja A o não-terminal responsável pela geração do símbolo *a*, mas sim algum outro não-terminal encontrado depois de A. Neste caso, devemos ter *a* pertencendo ao Follow(A).
  - a regra $T' \rightarrow \varepsilon$ foi usada com o símbolo +

Como construir a tabela?

- Para construir a tabela M, vamos examinar o conjunto de regras de produção e para cada *regra i*:
  - $A \rightarrow \alpha$, temos M[A, a] = i, para cada a em First(α).
  - $A \rightarrow \alpha$, se $\alpha \Rightarrow^* \varepsilon$, temos M[A, a]=i, para cada a em Follow(A).

- Cada entrada de M receberá no máximo um valor, mas se isso não acontecer, dizemos que houve um conflito, e que a gramática não é LL(1).
- As entradas de M que não receberem nenhum valor devem ser marcadas como entradas de erro.

Exemplo

(1) $E \rightarrow T\ E'$	First(TE') = { (, a }	M[E, (] = 1 e M[E, a] = 1
(2) $T \rightarrow F\ T'$	First(FT') = { (, a }	M[T, (] = 1 e M[T, a] = 2
(3) $F \rightarrow (\ E\ )$	First((E)) ={ (}	M[F, (] = 3
(4) $F \rightarrow a$	First(a) = { a }	M[F, a] = 4
(5) $E' \rightarrow +\ T\ E'$	First(+TE') = { + }	M[E', +] = 5
(6) $E' \rightarrow \varepsilon$	Follow(E') = { $,) }	M[E', $] = 6 e M[E',)] = 6
(7) $T' \rightarrow *\ F\ T'$	First(*FT') = { * }	M[T', *] = 7
(8) $T' \rightarrow \varepsilon$	Follow(T') = { $,+,)}	M[T', $] = 8 , M[T', +] = 8 e M[T',)] = 8

Exemplo

(1) $E \rightarrow E + T$	First(E+T)= { (, a }	M[E, (] = 1 e M[E, a] = 1
(2) $E \rightarrow T$	First(T)= { (, a }	M[E, (] = 2 e M[E, a] = 2
(3) $T \rightarrow T * F$	First(T*F)= { (, a }	M[T, (] = 3 e M[T, a] = 3
(4) $T \rightarrow F$	First(F)= { (, a }	M[T, (] = 4 e M[T, a] = 4
(5) $F \rightarrow (\ E\ )$	First((E))= { (}	M[F, (] = 5
(6) $F \rightarrow a$	First(a)= { a }	M[F, a] = 6

- A gramática não é LL(1), por causa dos conflitos.
- Múltiplas definições para M[E, (], M[E, a], M[T, (] e M[T, a].

Identificando uma gramática LL(1)

- Em alguns casos, como o da gramática do exemplo anterior, é possível concluir que a gramática não é LL(1) por inspeção.

- As duas características mais óbvias são:
  - a recursão à esquerda;
  - e a possibilidade de fatoração.

Recursão à esquerda

- Se uma gramática permite uma derivação $A \Rightarrow^* A\alpha$, para algum não-terminal A e para alguma cadeia não vazia α, a gramática é dita <u>recursiva à esquerda</u>.
- Naturalmente, para que A não seja um não-terminal inútil, deve existir na gramática (pelo menos) uma regra da forma $A \rightarrow \beta$, sem recursão à esquerda.
- A combinação dessas duas regras faz com que First(A) e, portanto, First($A\alpha$) contenham todos os símbolos de First(β), e isso leva necessariamente a um conflito.

Eliminando a recursão à esquerda

- A eliminação da recursão à esquerda pode ser tentada, procurando transformar a gramática em uma gramática LL(1). Basta observar que a combinação:

  $A \rightarrow A\alpha$

  $A \rightarrow \beta$

  permite a geração de cadeias da forma $\beta\alpha\alpha...\alpha$, e que essas mesmas cadeias podem ser geradas de outra maneira. Por exemplo:

  $A \rightarrow \beta A'$

  $A' \rightarrow \alpha A'$

  $A' \rightarrow \varepsilon$

Exemplo

Suponha a gramática:

(1) $E \rightarrow T$

(2) $E \rightarrow E + T$

(3) $T \rightarrow F$

(4) $T \rightarrow T * F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow a$

Fazendo as devidas substituições temos:

(1) $E \rightarrow T\ E'$

(2) $E' \rightarrow +\ T\ E' \mid \varepsilon$

(3) $T \rightarrow F\ T'$

(4) $T' \rightarrow *\ F\ T' \mid \varepsilon$

(5) $F \rightarrow (E)$

(6) $F \rightarrow a$

... que é LL(1) !

Possibilidade de fatoração

- Suponha duas regras que começam pelo mesmo símbolo:

  $A \rightarrow \alpha\beta$

  $A \rightarrow \alpha\gamma$
- Se First(α) $\neq \emptyset$ então existe uma interseção entre First($\alpha\beta$) e First($\alpha\gamma$), e não é possível decidir, olhando apenas para α, qual a regra correta.
- A solução é simples e envolve a fatoração:

  $A \rightarrow \alpha A'$

  $A' \rightarrow \beta \mid \gamma$

Exemplo (combinado)

Suponha a gramática:

(1) $L \rightarrow L \; ; \; S \mid S$

(2) $S \rightarrow$ if E th L el L fi

| if E th L fi

| s

(3) $E \rightarrow e$

Eliminando recursões à esquerda:

(1) $L \rightarrow S \; L'$

(4) $L' \rightarrow \; ; \; S \; L' \mid \varepsilon$

(2) $S \rightarrow$ if E th L el L fi

| if E th L fi

| s

(3) $E \rightarrow e$

Exemplo (combinado)

Observe a possibilidade de fatoração de S:

(1) $L \rightarrow S\ L'$

(4) $L' \rightarrow ;\ S\ L' \mid \varepsilon$

(2) $S \rightarrow$ if E th L el L fi

| if E th L fi

| s

(3) $E \rightarrow e$

Assim, teremos:

(1) $L \rightarrow S\ L'$

(4) $L' \rightarrow ;\ S\ L' \mid \varepsilon$

(2) $S \rightarrow$ if E th L S' | s

(5) $S' \rightarrow$ el L fi | fi

(3) $E \rightarrow e$