Módulo II - Análise Sintática

Análise da Estrutura

Fundamentos de Compiladores

UNEB

Prof. Antonio Atta

Análise Sintática – Gramáticas Livres de Contexto

- **Gramática**

Uma especificação formal da sintaxe de uma linguagem de programação

Componentes

- **Símbolos terminais ou tokens (T)**: os símbolos básicos com os quais os comandos da linguagem são formados
- **Símbolos não-terminais (V)**: variáveis sintáticas que denotam partes "relacionadas" a construções válidas da linguagem; ex.: comandos, expressões numéricas
- **Símbolo de partida (S)**: um símbolo não-terminal específico a partir do qual derivam as construções sintáticas válidas da linguagem, com $S \in V$
- **Regras de produção (P)**: regras que especificam como as cadeias podem ser derivadas na gramática

Formato: $\alpha \rightarrow \beta$ com $\alpha \in V$ e $\beta \in (T \cup V)^*$

Análise Sintática – Gramáticas Livres de Contexto

- **Aplicação em Linguagens de Programação**

Particularmente adequada por conta das *construções aninhadas* de ocorrência comum em LPs. Tais construções podem ser verificadas em:

- Estruturação de expressões aritméticas (constructo de subexpressões)
- Emprego de parênteses
- Construção de blocos de comandos
- Estruturação do fluxo de controle (aninhamento de comandos e funções)

Exemplo 1

Seja L_1 a linguagem formada por cadeias de parênteses casados. A gramática G_1 abaixo gera cadeias de L_1.

$$G_1: (\{S\}, \{(,)\}, P, S\}$$

onde P: $S \rightarrow (S)$

$S \rightarrow \varepsilon$

Análise Sintática – Gramáticas Livres de Contexto

Exemplo 2

Seja L_2 a linguagem formada por cadeias de parênteses casados e aninhados. A gramática G_2 abaixo gera cadeias de L_2.

$$G_2: (\{S\}, \{(,)\}, P, S\}$$

onde P: $S \rightarrow SS$

$S \rightarrow (S)$

$S \rightarrow \varepsilon$

- **Forma sentencial**

Qualquer cadeia obtida a partir de uma ou mais operações de *derivação direta* (⇒) a partir do símbolo de partida da gramática

$$\alpha\rho\beta \Rightarrow \alpha\gamma\beta$$

A operação de derivação acima só é possível se existir uma regra de produção no formato $\rho \rightarrow \gamma$ válida para a gramática dessa linguagem

Análise Sintática – Gramáticas Livres de Contexto

- **Derivação**

Processo de obtenção de formas sentenciais da linguagem, a partir do símbolo de partida da gramática, pela substituição de símbolos variáveis de acordo com as regras de produção

Assim, se α e β são formas sentenciais de uma linguagem:

$\Rightarrow$: indica uma derivação direta ou em um passo

$\Rightarrow^*$: indica uma derivação com zero ou mais passo

$$\alpha \Rightarrow^* \alpha \quad \text{(derivação com zero passo)}$$

$$\alpha \Rightarrow^* \beta \text{ se } \alpha \Rightarrow \gamma \text{ e } \gamma \Rightarrow^* \beta$$

Exemplo

A partir de G_2 anterior:

1. $S \Rightarrow (S) \Rightarrow (SS) \Rightarrow ((S)S) \Rightarrow ((S)(S)) \Rightarrow ((\varepsilon)(S)) \Rightarrow ((\varepsilon)(\varepsilon)) = ((\ )(\ ))$
2. $S \Rightarrow^* ((\ )(\ ))$

Análise Sintática – Gramáticas Livres de Contexto

- **Linguagem gerada**

  Seja G a uma gramática com símbolo de partida S. A linguagem gerada por G é dada por $L(G)$, onde $w \in L(G)$, se somente se:

  1. w é uma sequência de símbolos terminais, e
  2. $S \Rightarrow^* w$

- **Linguagem equivalente**

  Uma gramática G' é dita equivalente a G, se somente se $L(G') = L(G)$.

Análise Sintática – Gramáticas Livres de Contexto

Derivação mais à esquerda

Derivação onde, a cada passo, a substituição é feita apenas no símbolo mais à esquerda.

Derivação mais à direita

Derivação onde, a cada passo, a substituição é feita apenas no símbolo mais à direita.

Derivação mais à esquerda:

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow num + E * E \Rightarrow num + num * E \Rightarrow num + num * num$$

Derivação mais à direita:

$$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * num \Rightarrow E + num * num \Rightarrow num + num * num$$

Nenhuma das duas:

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow E + num * E \Rightarrow num + num * E \Rightarrow num + num * num$$

Análise Sintática – Gramáticas Livres de Contexto

- **Árvore de Derivação (*Parse tree*)**

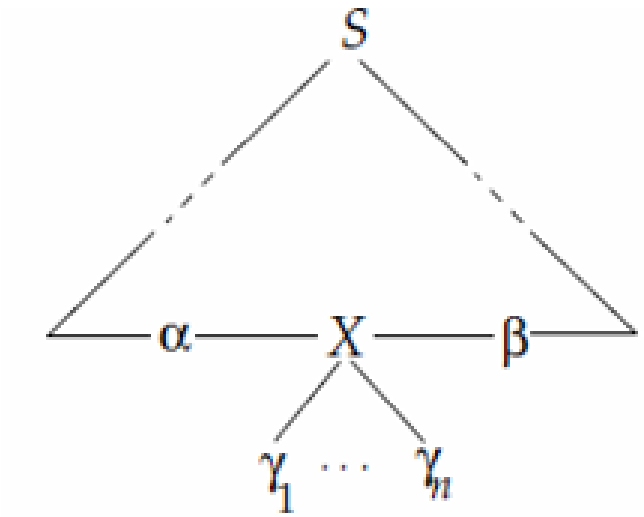
Uma árvore de derivação é uma representação gráfica de uma forma sentencial que mostra em sua estrutura as regras de produção usadas no processo de derivação

Construindo a árvore de derivação: dada uma gramática com símbolo de partida S,

- **Inicialmente, a forma sentencial é S e a árvore de derivação possui um único nodo denominado S;**
- **Suponha que T é a árvore de derivação para a forma sentencial $w = \alpha X \beta$ e que derivemos dela a forma sentencial**

$$w' = \alpha\gamma_1 \dots \gamma_n\beta$$

usando a regra de produção '$X \rightarrow \gamma_1 \dots \gamma_n$', então a árvore de derivação para w' é obtida acrescentando n filhos nomeados $\gamma_1 \dots \gamma_n$ ao nodo X em T



Análise Sintática – Gramáticas Livres de Contexto

- **Análise Descendente Recursiva**
  - Tipo de análise sintática que usa o processo de montagem de árvores sintáticas associadas a formas sentenciais (comandos válidos de uma linguagem de programação, por exemplo), para validar essas formas sentenciais como pertencentes à linguagem
  - Forma mais adotada na construção manual de analisadores sintáticos
  - Inicia pelo símbolo de partida e "tenta" chegar à forma sentencial no final do processo de derivação

    Isso é feito usando o próximo símbolo (token) de entrada e o estado corrente de construção da árvore sintática associada para decidir apropriadamente qual o próximo passo de derivação escolher

  - Podemos mapear a essa execução em termos de uma diagrama de transições
    - Existe um procedimento associado a cada *não terminal*: esse procedimento é responsável por fazer a análise sintática (*parse*) das formas sentenciais derivadas daquele *não terminal*;
    - Uma transição sobre um *token* (*terminal*) significa que essa transição deve ser executada se o próximo símbolo de entrada coincide com esse *token*
    - Uma transição sobre um *não terminal* implica na chamada do procedimento associado a ele

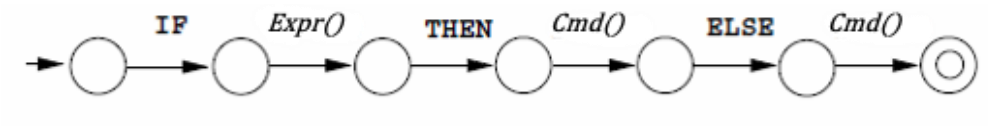
Análise Sintática – Gramáticas Livres de Contexto

Exemplo

Fragmento de gramática:

$$Cmd \longrightarrow \text{if } Expr \text{ then } Cmd \text{ else } Cmd$$

Diagrama de transição:



Fragmento de código correspondente:

```
Cmd()
{
      reconhece(IF);
      Expr();
      reconhece(THEN);
      Cmd();
      reconhece(ELSE);
      Cmd();
}
```

Análise Sintática – Gramáticas Livres de Contexto

- **Armadilhas na Análise Descendente Recursiva**
  - **Ambiguidade**: mais de uma produção pode ser aplicável em uma derivação; qual usar?
  - **Recursão à Esquerda**: regras de produção no formato

$$E \rightarrow E + E$$

    Causam *loop* infinito no Analisador.
  - **Backtracking**: pode ser necessário retroceder na montagem da árvore, por exemplo, em regras de produção do tipo:

$$A \rightarrow aB \mid aC$$

  - **Gerência erros**: o que fazer se a montagem da árvore "ficar presa"?

Análise Sintática – Gramáticas Livres de Contexto

- **Ambiguidade**

Uma gramática é *ambígua* se existe alguma forma sentencial na sua linguagem para a qual existe mais de uma árvore sintática associada

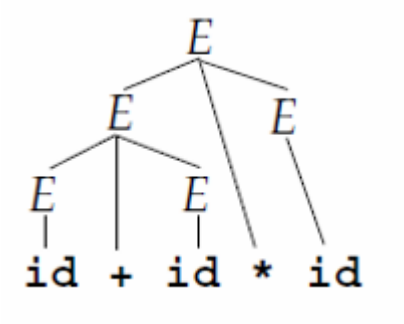
Exemplo

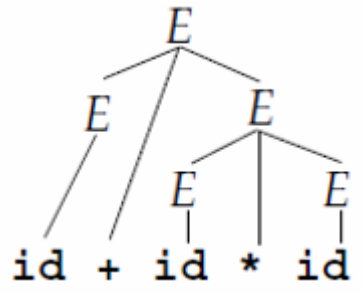
A gramática cujas regras de produção são:

$$E \rightarrow E + E \mid E * E \mid id$$

Com {+, *, *id*} símbolos *terminais*, { E } *não terminal* e símbolo de partida, é ambígua!

Possíveis árvores de derivação para "$id + id * id$ “:





Análise Sintática – Gramáticas Livres de Contexto

- **Problemas com a ambiguidade**

  Em geral, podemos não estar aptos a determinar qual árvore de derivação usar

- **Solução**

  1. **Encontrar uma gramática equivalente (transformar a gramática original) não ambígua.**

  2. **Usar regras de "desambiguidade" com especificações de qual árvore de derivação usar em caso de ambiguidade.**

Observação: **Decidir se uma GLC arbitrária é ambígua é um problema recursivamente insolúvel**

Análise Sintática – Gramáticas Livres de Contexto

Eliminando a ambiguidade: *Fatoração à esquerda*

Ideia básica: Atrasar a decisão sobre qual árvore sintática usar pela fatoração de qualquer prefixo comum entre as regras de produção de um *não terminal*

Exemplo: Dada as produções

$S \longrightarrow$ if E then S
$S \longrightarrow$ if E then S else S

A fatoração à esquerda produz

$S \longrightarrow$ if E then $S\ S'$
$S' \longrightarrow$ else $S \mid \varepsilon$

E associamos a seguinte regra de *desambiguidade*:

if $(prox_{token} ==' ELSE')$ *processe o ELSE*
else *saia por* ε

Observação: Essa solução efetivamente associa cada else com o if mais próximo

Análise Sintática – Gramáticas Livres de Contexto

Eliminando a ambiguidade: *Precedência e Associatividade*

Ideia básica: Adicione mais regras de produção para forçar a ocorrência da precedência e associatividade:

- **Introduza um *não terminal* para cada nível de precedência;**
- **Modifique as regras de produção de forma que as "*precedências mais baixas*" estejam orientadas à associatividade**

Exemplo: A gramática

$$E \rightarrow E + E \mid E * E \mid id$$

onde * tem maior precedência que + e ambos são associativos à esquerda,

se torna

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * id \mid id$$

Análise Sintática – Gramáticas Livres de Contexto

- **Recursão à Esquerda**

Essa armadilha está relacionada à produções no formato

$$A \rightarrow A\alpha$$

Exemplo: $E \rightarrow E + E$

Problema: esse tipo de contrução coloca a implementação computacional do analisador sintático descendente recursivo em laço infinito

Solução: transformar a *recursão à esquerda* em uma *recursão à direita*

RECURSÃO À ESQUERDA

$A \rightarrow A\alpha \mid \beta$

Deriva formas sentenciais:

$$\{\beta, \beta\alpha, \beta\alpha\alpha, \beta\alpha\alpha\alpha, \ldots\}$$

RECURSÃO À DIREITA

$A \rightarrow \beta R$
$R \rightarrow \alpha R \mid \varepsilon$

Deriva formas sentenciais:

$$\{\beta, \beta\alpha, \beta\alpha\alpha, \beta\alpha\alpha\alpha, \ldots\}$$

Análise Sintática – Gramáticas Livres de Contexto

Eliminando Recursão à Esquerda Imediata

Em geral: dada as regras de produção no formato

$$A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$$

onde nenhum β_i inicia com A, transforme para

$$A \rightarrow \beta_1 R \mid \ldots \mid \beta_n R$$
$$R \rightarrow \alpha_1 R \mid \ldots \mid \alpha_m R \mid \varepsilon$$

Nota: Essa solução não remove a recursão à esquerda *indireta*

similar a

$$A \rightarrow Ba$$
$$B \rightarrow Ab$$

Análise Sintática – Gramáticas Livres de Contexto

- ***Backtracking*** **(retrocesso na análise sintática)**

Ocorre quando os lados direitos de duas regras de produção possuem um prefixo comum para o mesmo não terminal do lado esquerdo dessas regras. Nesse caso não sabemos qual produção usar.

Exemplo: $A \rightarrow cdA \mid ceB$

<u>Solução imediata</u>: fatoração à esquerda

$$\begin{cases} A \rightarrow cX \\ X \rightarrow dA \mid eB \end{cases}$$

Infelizmente, a fatoração á esquerda não consegue "enxergar mais adiante" nas regras de produção para detectar sobreposição indireta de prefixos

$$A \rightarrow aB \mid C$$
$$C \rightarrow a$$

Análise Sintática – Gramáticas Livres de Contexto

- ***Backtracking* (cont.)**

Solução Alternativa: tentar seguir por uma das regras de produção e verificar se ela funciona. "Lembrar" do caminho seguido de modo que o analisador possa *retroceder* e tentar outra regra de produção aplicável se a tentativa anterior não funcionar

Apesar de computacionalmente possível, essa solução apresenta aspectos sérios de ineficiência. Não vamos considera-la, portanto, daqui para a frente.

Solução definitiva: não permitir o *backtracking*

Observação: Isso significa que deve ser possível *processar* a gramática antes de implementar o analisador e detectar qualquer possibilidade de *backtracking*. Isso pode ser conseguido computando os conjuntos de *PRIMEIROS* (*FIRST*)

Análise Sintática – Gramáticas Livres de Contexto

Conjunto de PRIMEIROS (*FIRST Sets*):

Definição: Se α é qualquer forma sentencial constituída de símbolos da gramática, então PRIMEIROS(α) é o conjunto de *terminais* que iniciam formas sentenciais derivadas de α

$$PRIMEIROS(\alpha) = \{a \mid a \text{ é um símbolo terminal e } \alpha \Rightarrow^{*} a\,\beta\}$$

Se $\alpha \Rightarrow^{*} \varepsilon$ então ε está também em PRIMEIROS(α)

Exemplo: Considere as seguintes regras de produção de uma gramática de expressões aritméticas com adição e multiplicação:

$$E \rightarrow T\,E'$$
$$E' \rightarrow +\,T E' \mid \varepsilon$$
$$T \rightarrow F\,T'$$
$$T' \rightarrow \times\, F\,T' \mid \varepsilon$$
$$F \rightarrow (\,E\,) \mid id$$

PRIMEIROS(E) = PRIMEIROS(T) = PRIMEIROS(F) = { $($, id }

PRIMEIROS(E') = { $+$, ε }

PRIMEIROS(T') = { $\times$, ε }