Compiladores
Análise Sintática
Introdução

Profa. Dra. Ana Carolina Lorena
UNIFESP

Análise Sintática

- Tarefa do analisador sintático é determinar a estrutura sintática de um programa a partir das marcas produzidas pela varredura (análise léxica)
  - Constrói no processo, explícita ou implicitamente, uma árvore de análise sintática, que representa essa estrutura

  - As marcas são solicitadas pelo analisador sintático (Parser) pela ativação de `getToken`

Análise Sintática

- Um problema mais difícil é o tratamento de erros
  - O analisador sintático precisa registrar uma mensagem de erro e também se recuperar a condição sem erros e seguir em frente (para descobrir o maior número de erros possível)
  - Por vezes, ele pode efetuar correção de erros (normalmente só em casos simples)

Sintaxe

- Sintaxe de uma linguagem de programação: normalmente especificada por regras gramaticais de uma gramática livre de contexto
  - Maior diferença em relação a gramáticas regulares de análise léxica é que regras sintáticas podem ser recursivas
    - Ex. if pode ser aninhado com outros ifs, o que exige recursão
    - Ex de regra: `exp → exp op exp|(exp)|número`
      `op → + | - | *`

Gramáticas Livres de Contexto

- Definição: G = (T, N, P, S), em que
  - T é um **conjunto de terminais**
    - Terminam derivações
  - N é um **conjunto de não-terminais**, disjunto de T
    - Não terminam derivações, sempre podem ser substituídos
  - P é um **conjunto de produções** ou regras gramaticais, na forma $A \rightarrow \alpha$, em que $A \in N$ e $\alpha \in (T \cup N)^*$
    - Produzem cadeias
  - $S \in N$ é um **símbolo inicial**

  - Conjunto $T \cup N$ é denominado conjunto de símbolos de G
  - Cadeia $\alpha \in (T \cup N)^*$ é denominada forma sentencial

Gramáticas Livres de Contexto

Passo de derivação sobre G: tem a forma $\alpha A \gamma \Rightarrow \alpha \beta \gamma$, em que:

α, β e $\gamma \in (T \cup N)^*$

$A \rightarrow \beta \in P$

- Derivação sobre G: tem a forma $S \Rightarrow^* w$, em que:
  - $w \in T^*$ (cadeia apenas com terminais ou sentença)
  - Uma derivação é uma sequência de substituições de não-terminais por escolhas à direita das regras gramaticais
  - A cada passo da derivação, uma única substituição é feita com base em uma escolha de regra gramatical
  - Regras **definem** e os passos de derivação **constroem** por substituição

Gramáticas Livres de Contexto

- Linguagem gerada por G: $L(G) = \{w \in T^* \mid \exists$ uma derivação $S \Rightarrow^* w$ de G$\}$
  É o conjunto de sentenças deriváveis de G

- Derivação à esquerda: $S \Rightarrow^*_{lm} w$ é uma derivação em que cada passo $\alpha A \gamma \Rightarrow \alpha \beta \gamma$ é tal que $\alpha \in T^*$ (tem apenas terminais)

- Derivação à direita: $S \Rightarrow^*_{rm} w$ é uma derivação em que cada passo $\alpha A \gamma \Rightarrow \alpha \beta \gamma$ é tal que $\gamma \in T^*$

Gramáticas Livres de Contexto

- Um conjunto de cadeias L é denominado linguagem livre de contexto se existir uma gramática livre de contexto G tal que L = L(G)

Exemplos

- G com a regra E → (E)|a
  Tem um não-terminal E e três terminais (,) e a
  Gera linguagem L(G) = {a, (a), ((a)), …} = $({}^{n}a)^{n}$

- G com regra E → (E) não gera cadeia alguma
  L(G) = {}
  Não possui caso base para a recursão e fica em recursão infinita

- G com regra E → E+a | a
  Exercício 1: a) quais são os terminais e não-terminais?
  b) qual a linguagem gerada?

Gramáticas Livres de Contexto

- Pela hierarquia de Chomsky, as Gramáticas Livres de Contexto são mais gerais e incluem as Gramáticas Regulares
  - Daria então para projetar um analisador sintático para ler diretamente caracteres do arquivo fonte e eliminar o sistema de varredura, mas isso comprometeria eficiência

Gramáticas Livres de Contexto

- Nome livre de contexto: não-terminais aparecem livres à esquerda da seta
  - $A \rightarrow \alpha$ indica que A pode ser substituída por α em *qualquer ponto*, independentemente de onde ocorra A

- Há requisitos em linguagens de programação que poderiam ser mais facilmente especificados por gramáticas sensíveis ao contexto, que são mais poderosas
  - Possui regras $\beta\ A\ \gamma \rightarrow \beta\ \alpha\ \gamma$, em que $\alpha \neq \varepsilon$ (regra se aplica apenas se β ocorre antes e γ ocorre depois do não-terminal)

Gramáticas Livres de Contexto

- Exemplo em que gramática sensível a contexto se aplicaria: regra de declaração de variável antes de uso

```
{ int x;
…
… x …
}
(nome deve aparecer em declaração antes de ser usado)
```

- Problema é gramáticas sensíveis a contexto são mais difíceis de serem usadas como base para um analisador sintático
  - Isso é então deixado para a análise semântica

Gramáticas Livres de Contexto

- Nos compiladores:
  - Os símbolos terminais serão as marcas produzidas pelo sistema de varredura
  - Os não-terminais são nomes de estruturas sintáticas

Especificação BNF

- Forma de Backus-Naur (BNF) é normalmente usada para especificar as regras gramaticais das linguagens de programação
  - Possui meta-símbolo de escolha (|)
  - Concatenação também é usada
  - Não possui meta-símbolo de repetição, que é obtida pela recursão
  - Uso de seta → para expressar definições de nomes
  - Podem usar expressões regulares como componentes

Não há padrão universal para convenções, vamos adotar uma comumente utilizada

Especificação BNF

- Regra gramatical em BNF define a estrutura cujo nome está à esquerda da seta
  - Exemplo: `exp → exp op exp | (exp)| número`
    `op → + | - | *`

Primeira regra (recursiva) define uma estrutura de expressão
Segunda regra define um operador

Especificação BNF

- Repetição pode ser construída por recursão
  - Exemplo: A → Aa | a ou A → aA | a
  - geram a linguagem $\{a^n \mid n \geq 1\}$

Primeira regra (A → Aa | a) é recursiva à esquerda, pois não-terminal A aparece como primeiro símbolo à direita da regra que define A

Segunda regra (A → aA | a) é recursiva à direita

Especificação BNF

- Considere uma regra recursiva à esquerda $A \rightarrow A\ \alpha|\beta$
  - α e β representam cadeias arbitrárias e β não começa com A
  - Ela gera todas as cadeias da forma β, $\beta\alpha$, $\beta\alpha\alpha$, …
- Considere uma regra recursiva à direita $A \rightarrow\ \alpha\ A|\beta$
  - α e β representam cadeias arbitrárias e β não termina com A
  - Ela gera todas as cadeias da forma β, $\alpha\beta$, $\alpha\alpha\beta$, …
  - β também pode ser a cadeia vazia ε

Derivações e linguagem

- As regras da gramática definem o conjunto de cadeias de símbolos para marcas consideradas sintaticamente legais
- Exemplo: `exp → exp op exp |(exp)| número`
  `op → + | - | *`

Expressão aritmética (34-3)*42 corresponde à cadeia legal com sete marcas: (número – número) * número

Já a cadeia (34-3*42 não é uma expressão legal

Derivações e linguagem

- Regras da gramática determinam cadeias legais de símbolos para marcas fazendo uso de derivações
  - Uma derivação começa com um único nome de estrutura e termina com uma cadeia de símbolos para marcas
  - Exemplo: `exp → exp op exp | (exp)| número`
    `op → + | - | *`

Expressão (34-3)*42

(1) exp ⇒ exp op exp
(2) ⇒ exp op número
(3) ⇒ exp * número
(4) ⇒ (exp) * número
(5) ⇒ (exp op exp) * número
(6) ⇒ (exp op número) * número
(7) ⇒ (exp – número) * número
(8) ⇒ (número – número) * número

Derivações e linguagem

- Conjunto de cadeias de símbolos para marcas obtido por derivações para exp é a linguagem definida pela gramática de expressões
  - Contém todas as expressões sintaticamente legais
  - L(G) = {s | exp $\Rightarrow$* s}, em que G representa a gramática de expressões

Exemplos

- Gramática de declarações simplificada

```
declaração → if-decl | outra
if-decl → if (exp) declaração |
          if (exp) declaração else declaração
exp → 0|1
```

Linguagem = declarações if aninhadas

Exemplos: if (0) outra

if (1) outra else outra

if (1) outra else if (0) outra else outra

Exemplos

- Gramática de declarações simplificada (versão 2)

```
declaração → if-decl | outra
if-decl → if (exp) declaração else-parte
else-parte → else declaração | ε
exp → 0|1
```

ε–produção indica que estrutura else-parte é opcional

Exercício 2: mostre como é a derivação das seguintes sentenças:

a) if (0) outra

b) if (1) outra else outra

c) if (1) outra else if (0) outra else outra

Exemplos

- Gramática de parênteses balanceados
  A → (A) A | ε

- Gramática para uma sequência de declarações
  `decl-sequencia → decl ; decl-sequencia | decl`
  `decl → s`
- Gera sequências com uma ou mais declarações separadas por ;
- L(G) = {s, s;s, s;s;s, …}

- Exercício 3: o que ocorre se permitirmos a cadeia vazia na primeira regra anterior?
  `decl-sequencia → decl ; decl-sequencia | ε`

Árvore de Análise Sintática

- Derivações não representam unicamente a estrutura das cadeias construídas
  - Existem em geral muitas derivações para a mesma cadeia

Exemplo: expressão (34-3)*42

(1) exp ⇒ exp op exp
(2) ⇒ (exp) op exp
(3) ⇒ (exp op exp) op exp
(4) ⇒ (número op exp) op exp
(5) ⇒ (número op número) op exp
(6) ⇒ (número - número) op exp
(7) ⇒ (número – número) * exp
(8) ⇒ (número – número) * número

Árvore de Análise Sintática

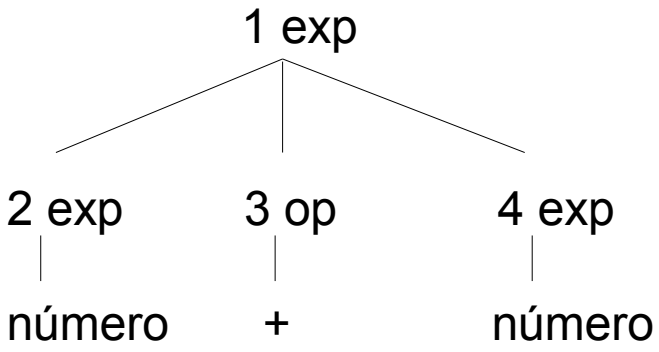
- Uma árvore de análise sintática correspondente a uma derivação é uma árvore com a raiz rotulada e com as propriedades:
  - Cada nó é rotulado com um terminal, um não-terminal ou ε
  - O nó-raiz é rotulado com o símbolo inicial S
  - Cada nó folha é rotulado com um terminal ou ε
  - Cada nó não-folha é rotulado com um não-terminal
  - Se um nó com rótulo $A \in N$ tiver n filhos com rótulos $X_1, X_2, \ldots, X_n$ (que podem ser terminais ou não-terminais), então $A \rightarrow X_1X_2\ldots X_n \in P$

Árvore de Análise Sintática

- Cada derivação leva a uma árvore de análise sintática
- Em geral, muitas derivações podem levar à mesma árvore de análise sintática
- Cada árvore de análise sintática tem, entretanto, uma derivação à esquerda (percurso em pré-ordem) e uma à direita (inverso do percurso em pós-ordem) que são únicas e que levam a ela

Árvore de Análise Sintática

- Exemplo: (1) exp ⇒ exp op exp
  (2) ⇒ número op exp
  (3) ⇒ número + exp
  (4) ⇒ número + número



Note que a enumeração está em pré-ordem

Árvore de Análise Sintática

- Exemplo: a mesma árvore corresponde à derivação:

exp $\Rightarrow$ exp op exp

$\Rightarrow$ exp op número

$\Rightarrow$ exp + número

$\Rightarrow$número + número

- Mas enumerações diferentes dos nós internos seriam aplicáveis

Árvore de Análise Sintática

- A estrutura da árvore sintática depende da estrutura sintática específica da linguagem
  - Em geral é usada uma estrutura de dados dinâmica
  - Nós possuem campos para atributos

Árvore Sintática Abstrata

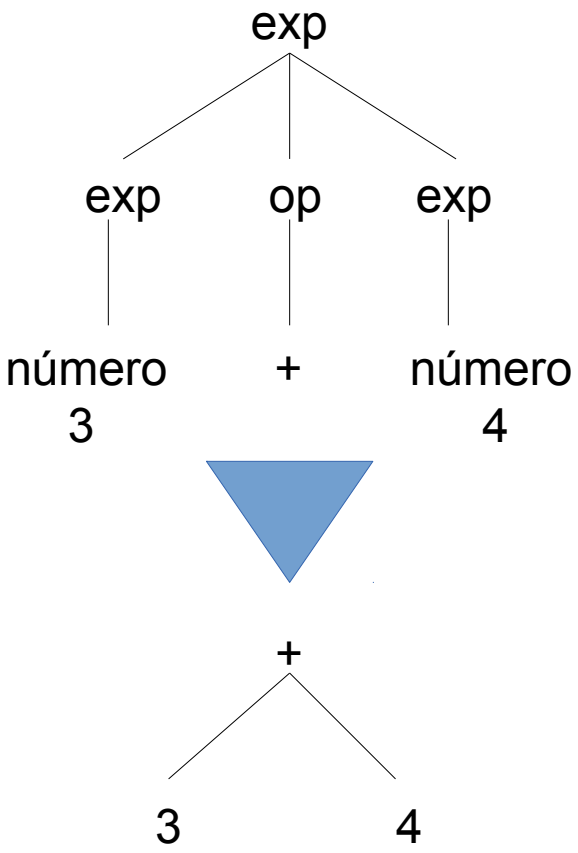
- Árvore de análise sintática é uma representação útil da estrutura de uma cadeia de marcas, que parecem como folhas
  - Os nós internos representam passos de derivação
  - Entretanto, ela contém mais informação do que o absolutamente necessário para um compilador

  - Árvores sintáticas abstratas representam abstrações e contêm toda a informação necessária de forma mais eficiente

  - São representação de uma notação simplificada, sintaxe abstrata

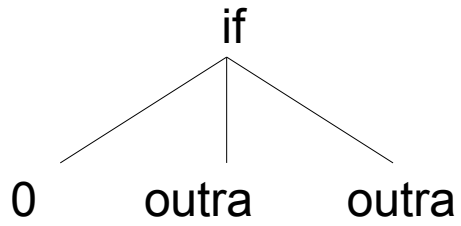
Árvore Sintática Abstrata

- Exemplo 3 + 4

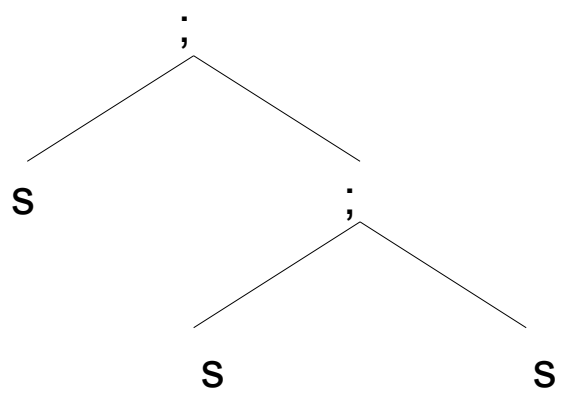


Árvore Sintática Abstrata

- Exemplo if (0) outra else outra



- Exemplo s;s;s



Árvore Sintática Abstrata

- Dependendo de como é construída a árvore sintática, podem ser definidas duas categorias de algoritmos de análise sintática:
  - Ascendentes
  - Descendentes

Análise Sintática

- Bibliografia consultada

Capítulo 3: LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004