Analisadores sintáticos

Professor: Luciano Ferreira Silva, Dr.

Analisadores sintáticos

- **É um programa construído para uma gramática G que:**
  - ✓ Recebe como entrada uma seqüência de símbolos terminais do alfabeto de G;
  - ✓ Verifica a validade da seqüência em G;
  - ✓ Caso válida ele constrói sua árvore sintática;
  - ✓ Caso não válida ele deve apresentar uma indicação de erro.

Analisadores sintáticos

- **Há duas estratégias possíveis para construção da árvore sintática:**

  - ✓ Ascendente:

    - O analisador sintático varre a sentença e procura aplicar as produções que permitam substituir seqüências de símbolos da sentenças pelo lado esquerdo das produções (usa-se derivações mais a direita);
    - A sentença é reconhecida quando, na raiz da árvore sintática, o único símbolo restante é o símbolo sentencial (usa a estratégia de varredura pós-ordem);

Analisadores sintáticos

- Esta técnica é fortemente adequada, juntamente com técnicas de precedência e associatividade de operadores, para a análise de expressões aritméticas.

✓ Descendente:

- O analisador sintático parte do símbolo sentencial e aplica derivações mais a esquerda para atingir a sentença de entrada;
- A construção da árvore sintática parte da raiz como o símbolo sentencial e usa a estratégia de varredura pré-ordem;
- Analisadores sintáticos descendentes normalmente são recursivos e usados para analisar comandos que não sejam expressões aritméticas.

Autômato de pilha

- **É uma estrutura formal que incorpora a memória necessária para o reconhecimento de sentenças em livres de contexto:**
  - ✓ As linguagens livres de contexto demandam no processamento que alguma forma de memória esteja disponível devido a sua propriedade de auto-incorporação;
  - ✓ O autômato finito opera sem nenhum tipo de memória pois considera apenas o estado corrente e o próximo símbolo da string.

Autômato de pilha

- **Formalmente representável por uma sêxtupla M = (K, Σ, Γ, δ, s, F):**

1. K é um conjunto finito de estados;
2. Σ é o alfabeto de entrada finito;
3. Γ é o alfabeto finito de pilha;
4. δ é a função de transição, $\delta: K \times \Sigma \times \Gamma \rightarrow K \times \Gamma$
5. s é o estado inicial, sendo que $s \in K$;
6. F é o conjunto de estados finais. Sendo $F \subseteq K$.

Autômato de pilha

- **Pilha:**
  - ✓ Estrutura de dados linear com restrição na política de acesso aos seus elementos;
    - Política de acesso LIFO (*last in*, *first out*), ou seja, o último elemento que entra é o primeiro que sai.
  - ✓ A biblioteca STL de C++ oferece a classe `stack` que implementa uma pilha para qualquer tipo elemento;

Autômato de pilha

- **Funções stack C++:**
  - ✓ `push`: para inserir um elemento no topo da pilha;
  - ✓ `pop`: para remover o elemento no topo da pilha;
  - ✓ `top`: para inspecionar o elemento que está no topo;
  - ✓ `empty`: para testar se a pilha está vazia;
  - ✓ `size`: para obter a quantidade de elementos na pilha;

Autômato de pilha

- **Exemplo de uso:**

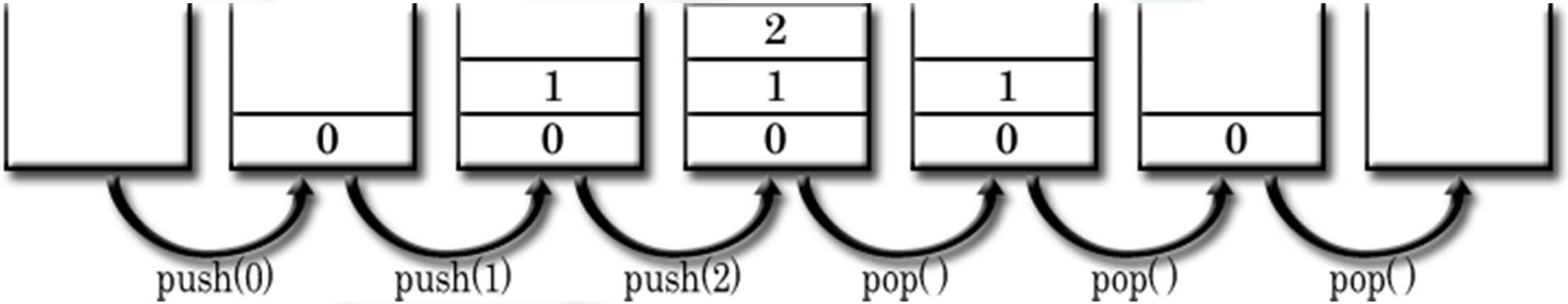
```
#include <stack>
stack<int> simbolos;
for(int pos=0; pos<3;++pos)
   simbolos.push(pos);
while(!simbolos.empty()){
        cout<<“Pilha tem ”<<simbolos.size();
        cout<<“elementos, topo ”<<simbolos.top();
        cout<<endl;
   simbolos.pop();
}
```

Autômato de pilha

- **Na tela aparecerá:**

```
Pilha tem 3 elementos, topo: 2
Pilha tem 2 elementos, topo: 1
Pilha tem 1 elementos, topo: 0
```

- **Evolução do estado da pilha:**



Autômato de pilha

- **Observações:**
  - ✓ Assim como para os autômatos finitos, os autômatos de pilha também são computáveis por meio de uma tabela de transições;
  - ✓ A estratégia de construção dessa tabela a partir da gramática depende do tipo de analisador que é desenvolvido;

Analisador sintático preditivo

- **É baseado na técnica de construção descendente;**
- **Não pode operar com gramáticas que tenham produções com recursão à esquerda;**
  - ✓ Se for este o caso, é necessário inicialmente reescrever essas produções de modo a substituir a recursão à esquerda por recursão à direita.

Conversão de produções recursivas

- A estratégia é substituir a recursão à esquerda por uma recursão à direita;
- Seja *A* o símbolo não-terminal de uma produção recursiva à esquerda,

$$A \rightarrow A\beta$$

- Onde β é uma seqüência qualquer de símbolos não iniciada pelo símbolo *A*.
- Além dessa produção, deve haver outra produção não-recursiva para o símbolo A,

$$A \rightarrow \delta$$

- Onde δ é outra seqüência qualquer de símbolos não iniciada pelo símbolo *A*.

Conversão de produções recursivas

- Expansão do símbolo não-terminal A:

$$A \Rightarrow \delta$$

$$A \Rightarrow A\beta \Rightarrow \delta\,\beta$$

$$A \Rightarrow A\beta \Rightarrow A\beta\,\beta \Rightarrow \delta\,\beta\,\beta$$

$$A \Rightarrow A\beta \Rightarrow A\beta\,\beta \Rightarrow A\beta\,\beta\,\beta \Rightarrow \delta\,\beta\,\beta\,\beta$$

$$A \Rightarrow \ldots$$

- São seqüências iniciadas pela seqüência δ seguidas por zero ou mais ocorrências da seqüência de símbolos β.

Conversão de produções recursivas

- O par de produções $A \rightarrow A\beta$ e $A \rightarrow \delta$ deve ser substituído por três produções:

1. $A \rightarrow \delta A`$
2. $A` \rightarrow \beta A`$
3. $A` \rightarrow \varepsilon$

- Elas descrevem exatamente as mesmas formas sentenciais que o par de produções originais cuja recursão era à esquerda;

Conversão de produções recursivas

- Expansão do símbolo não-terminal A usando as novas produções:

$$A \Rightarrow \delta A` \Rightarrow \delta$$

$$A \Rightarrow \delta A` \Rightarrow \delta \beta A` \Rightarrow \delta \beta$$

$$A \Rightarrow \delta A` \Rightarrow \delta \beta A` \Rightarrow \delta \beta \beta A` \Rightarrow \delta \beta \beta$$

$$A \Rightarrow \delta A` \Rightarrow \delta \beta A` \Rightarrow \delta \beta \beta A` \Rightarrow \delta \beta \beta \beta A` \Rightarrow \delta \beta \beta \beta$$

$$A \Rightarrow ...$$

Conversão de produções recursivas

- **Considere a gramática G` abaixo.**

Gramática G`, equivalente à gramática G mas sem ambiguidade

$E \rightarrow E + M$	Produção 1: adição com recursividade
$E \rightarrow M$	Produção 2: marca o fim da recursividade da adição
$M \rightarrow M \times P$	Produção 3: multiplicação com recursividade, contempla a associatividade
$M \rightarrow P$	Produção 4: marca o fim recursividade da multiplicação
$P \rightarrow (E)$	Produção 5: possibilita o uso de parênteses
$P \rightarrow v$	Produção 6: terminal

Conversão de produções recursivas

- A primeira produção de G` apresenta recursão a esquerda: $E \rightarrow E + M$;
- A produção 2 é a produção não recursiva do símbolo *E*: $E \rightarrow M$;
- O par de produções originais seria substituído por:

$$E \rightarrow ME`$$

$$E` \rightarrow +ME`$$

$$E` \rightarrow \varepsilon$$

Conversão de produções recursivas

- A produção 3 de G` também apresenta recursão a esquerda: $M \rightarrow M \times P$;
- A produção 4 é a produção não recursiva do símbolo *M*: $M \rightarrow P$;
- O par de produções originais seria substituído por:

$$M \rightarrow PM`$$

$$M` \rightarrow \times PM`$$

$$M` \rightarrow \varepsilon$$

Conversão de produções recursivas

- Uma nova gramática (G``), equivalente a G`, que remove as produções com recursão à esquerda, encontra-se abaixo (com E sentencial).

Gramática G``, equivalente à gramática G` sem ambiguidade recursões à esquerda

E → ME`	Produção 1: adição
E` → +ME`	Produção 2: adição
E` → ε	Produção 3: adição
M → PM`	Produção 4: multiplicação
M` → xPM`	Produção 5: multiplicação
M` → ε	Produção 6: multiplicação
P → (E)	Produção 7: possibilita o uso de parênteses
P → v	Produção 8: terminal