Compiladores

Análise sintática (2)

Análise Top-Down

Lembrando...
Gramáticas Livres de Contexto

- Análise sintática = parsing.
- Baseada em GLCs

Gramática: $S \rightarrow A\ B$
$A \rightarrow c \mid \varepsilon$
$B \rightarrow cbB \mid ca$

String:
ccbca

Top-Down		Bottom-Up	
$S \Rightarrow AB$	$S \rightarrow AB$	$ccbca \Leftarrow Acbca$	$A \rightarrow c$
$\Rightarrow cB$	$A \rightarrow c$	$\Leftarrow AcbB$	$B \rightarrow ca$
$\Rightarrow ccbB$	$B \rightarrow cbB$	$\Leftarrow AB$	$B \rightarrow cbB$
$\Rightarrow ccbca$	$B \rightarrow ca$	$\Leftarrow S$	$S \rightarrow AB$

Caraterísticas de Gramáticas

- Gramática fatorada à esquerda:
  - GLC que **não** possui produções do tipo $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ para alguma forma sentencial α.
- Gramática recursiva à esquerda:
  - GLC que permite a derivação

OBS: RECONHECEDOR TOP-DOWN não aceita gramáticas recursivas à esquerda

indireta, como símbolo mais à esquerda de uma subpalavra gerada.

Eliminação de recursividade à esquerda

- Exemplo

$A \rightarrow Aa \mid b$

Com a palavra vazia	Sem a palavra vazia
$A \rightarrow bX$	$A \rightarrow b \mid bX$
$X \rightarrow aX \mid \varepsilon$	$X \rightarrow a \mid aX$

Obs: pode ainda haver recursão indireta!

Exemplo

- E -> E+T | T
- T -> T*F | F
- F -> (E)| **Id**

Fatoração de uma gramática

- Elimina indecisão de qual produção aplicar quando duas ou mais produções iniciam com a mesma forma sentencial

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$$

Se torna:

$$A \rightarrow \alpha X$$
$$X \rightarrow \beta_1 \mid \beta_2$$

Exemplo de Fatoração a Esquerda

Cmd $\rightarrow$ if Expr then Cmd else Cmd
Cmd $\rightarrow$ if Expr then Cmd
Cmd $\rightarrow$ Outro

- Fatorando a esquerda:

Cmd $\rightarrow$ if Expr then Cmd ElseOpc
Cmd $\rightarrow$ Outro
ElseOpc $\rightarrow$ else Cmd | ε

Eliminação de produções vazias (1)

- Objetivo:
  - eliminar produções da forma $A \rightarrow \varepsilon$.
- Algoritmo: seja G = (N,T,P,S) uma GLC
  - Etapa 1:
    - construir N_ε , o conjunto de não-terminais que geram a palavra vazia:

$N_\varepsilon = \{A \mid A \rightarrow \varepsilon\}$;
Repita
$N_\varepsilon = N_\varepsilon \cup \{X \mid X \rightarrow X_1 \ldots X_n \in P \text{ tq } X_1, \ldots, X_n \in N_\varepsilon\}$
Até que o cardinal de N_ε não aumente.

Eliminação de produções vazias (2)

- Etapa 2:
  - construir o conjunto de produções sem produções vazias:

gera $G_1 = (N,T,P_1,S)$, onde P_1 é construído como segue:

$P_1 = \{A \rightarrow \alpha \mid \alpha \neq \varepsilon\}$;
Repita
Para toda $A \rightarrow \alpha \in P_1$ e $X \in N_\varepsilon$ tal que
$\alpha = \alpha_1 X \alpha_2$ e $\alpha_1\alpha_2 \neq \varepsilon$
Faça $P_1 = P_1 \cup \{A \rightarrow \alpha_1\alpha_2\}$
Até que o cardinal de P_1 não aumente

Eliminação de produções vazias (3)

- Etapa 3:
  - incluir geração da palavra vazia, se necessário.
  - Se a palavra vazia pertence à linguagem, então a gramática resultante é
  - $G_2 = (N,T,P_2,S)$, onde $P_2 = P_1 \cup \{S \rightarrow \varepsilon\}$

Plano da aula

- **Como reconhecer se uma sentença está de acordo com uma gramática?**
  - **Pode-se implementar um reconhecedor de sentença**
    - **Recursivamente, com retrocesso**
    - **Com um mecanismo preditivo**
      - **Lookahead.**
      - **Primitivas First e Follow.**
  - **Para usar reconhecedores, deve-se transformar a GLC.**
    - Eliminação de produções vazias
    - Eliminação de recursividade à esquerda:
      - recursão direta
      - recursão indireta
    - Fatoração de uma gramática

Análise Top-Down

- Como implementar um reconhecedor para uma GLC?
- Constrói-se a árvore de derivação, lendo a sentença de esquerda para a direita, e substituindo sempre o não-terminal mais à esquerda.
- Existe três tipos principais de parser top-down:
  - Recursivo com retrocesso
  - Recursivo preditivo
  - Tabular preditivo.

Top-Down: Backtracking

$S \rightarrow A\ B$
$A \rightarrow c \mid \varepsilon$
$B \rightarrow cbB \mid ca$

S	cbca	tentar S→AB
AB	cbca	tentar A→c
cB	cbca	casar c
B	bca	sem-saída, tentar A→ε
εB	cbca	tentar B→cbB
cbB	cbca	casar c
bB	bca	casar b
B	ca	tentar B→cbB
cbB	ca	casar c
bB	a	sem-saída, tentar B→ca
ca	ca	casar c
a	a	casar a -> Fim!

- Gera $S \Rightarrow^* cbca$?

Implementação do reconhecedor

- Cria-se um procedimento por não-terminal
  - Ele testa a aplicação de cada produção associada ao não-terminal;
  - Lê no texto de entrada o próximo token
    - Chama o analisador lexical (yylex()) !
- É necessário lembrar onde se fez uma escolha de uma alternativa, para poder retroceder neste ponto.

Exemplo:
TIPO -> TIPO_SIMPLES
| ^ **idf**
| **array [** TIPO_SIMPLES **] of** TIPO
TIPO_SIMPLES -> **integer** | **char**
| **num pp num**

Reconhecedor recursivo

```
ReconheceTIPO() {
  switch(token) {
  case {integer, char, num}: ReconheceTIPOSIMPLES();        break;
  case ^: reconhece( id) ; break;
  }
}
```

NÃO há necessidade de retrocesso!

```
ReconheceTIPOSIMPLES() {
    switch(token){
    case integer: reconhece(integer); break;
    case char: reconhece(char); break;
    case num: reconhece(num); reconhece(pp); reconhece(num); break;
    default: erro();
  }
}
```

Observações sobre o método recursivo com retrocesso

- É fácil de implementar.
- É necessário:
  1. Que a gramática não seja recursiva à esquerda
     - A -> Aa se tornará
       ReconheceA() { ReconheceA();... }
     - Recursão infinita!
  2. Que a gramática seja fatorada à esquerda
     - Senão, deve-se fazer retrocesso.
  3. Que os primeiros terminais deriváveis possibilitem a decisão de uma produção a aplicar!
     - Não há retrocesso sobre não-terminais...

Definição: Conjuntos "First"

- First(α):
  - Definição informal:
    - conjunto de todos os terminais que começam qualquer seqüência derivável de α.
  - Definição formal:
    - Se existe um $t \in$ **T** e um $\beta \in$ **V*** tal que $\alpha \Rightarrow^* t\ \beta$ então $t \in$ First(α)
    - Se $\alpha \Rightarrow^* \varepsilon$ então $\varepsilon \in$ First(α)

$A \rightarrow B \mid C \mid D$
$B \rightarrow b$
$C \rightarrow c$
$D \rightarrow d$

First(A)={b,c,d}
First(B)={b}
First(C)={c}
First(D)={d}

Condição para que se possa usar um analisador preditivo

- No caso que os First() sejam "simpáticos", não terá retrocesso.
- Isso supõe que para qualquer produção
$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$, se tenha
$\text{First}(\alpha_1) \cap \text{First}(\alpha_2) \cap \dots \cap \text{First}(\alpha_n) = \varnothing$

proc First(α: string of symbols)

```
Repeat {
  Para todas as produções α ⟶ X_1 X_2 X_3 … X_n do
    if X_1 ∈ T then  // caso simples onde X_1 é um terminal
          First(α) := {X_1}
    else {  // X_1 é um não-terminal
           First(α) = First{X_1} \ {ε};
           for (i=1 ; i<=n ; i++) {
              if ε is in First(X_1) and in First(X_2) and in… First(X_{i-1})
                             First(α) := First(α) ∪ First(X_i) \ {ε}
           }
    }
    if (α =>* ε)
        then First(α) := First(α) ∪ {ε}
  end do
} until no change in any First(α)
```

O que acontece se $\varepsilon \in$ First(A)?

$S \rightarrow ACE$
$A \rightarrow a \mid b \mid \varepsilon$
$C \rightarrow c \mid d \mid \varepsilon$
$E \rightarrow e$

Consegue derivar ?
- ace
- bde
- ce
- de
- e

O que acontece se $\varepsilon \in$ First(A)?

$S \rightarrow ACE$
$A \rightarrow a \mid b \mid \varepsilon$
$C \rightarrow c \mid d \mid \varepsilon$
$E \rightarrow e$

```
S () {
switch {
case 'a':
case 'b':
          A();
          C();
          E();
          break;
default:
          abort("syntax_error");
}
```

Consegue derivar ?
- ace
- bde
- ce
- de
- e

O que acontece se $\varepsilon \in$ First(A)?

$S \rightarrow ACE$
$A \rightarrow a \mid b \mid \varepsilon$
$C \rightarrow c \mid d \mid \varepsilon$
$E \rightarrow e$

```
S () {
switch {
case 'a':
case 'b':
case 'c':
case 'd':
          C();
          E();
          break;
default:
          abort("syntax_error");
}
```

Consegue derivar ?
- ace
- bde
- ce
- de
- e

O que acontece se $\varepsilon \in$ First(A)?

$S \rightarrow ACE$
$A \rightarrow a \mid b \mid \varepsilon$
$C \rightarrow c \mid d \mid \varepsilon$
$E \rightarrow e$

```
S () {
switch {
case 'a':
case 'b':
case 'c':
case 'd':
case 'e':
          E();
          break;
default:
          abort("syntax_error");
}
```

Consegue derivar ?
- ace
- bde
- ce
- de
- e

O que acontece se $\varepsilon \in$ First(A)?

$S \rightarrow ACE$
$A \rightarrow a \mid b \mid \varepsilon$
$C \rightarrow c \mid d \mid \varepsilon$
$E \rightarrow e$

```
A () {
switch {
case 'a':
          MatchToken('a'); break;
case 'b':
          MatchToken('b'); break;
case ??:
          break;
default:
          abort("syntax_error");
}
```

Consegue derivar ?
- ace
- bde
- ce
- de
- e

Outro exemplo

- Escrever um reconhecedor recursivo simples para as gramáticas equivalentes:

E -> E+T | T
T -> T*F | F
F -> (E)| **Id**

E -> TE'
E' -> +TE'| ε
T -> FT'
T'-> *FT' | ε
F -> (E)| Id

Para tratar o caso do ε: o conjunto Follow

- Follow(B):
  - conjunto de **terminais** que podem aparecer à direita de um não-terminal B em uma sentença válida.
  - $ passa a denotar um terminal "virtual" que marca o fim da entrada (EOF, CTRL-D,...)
- Formalmente:
  - Se existe um $t \in$ **T** e $\alpha, \beta \in$ **V*** tal que $S \Rightarrow^* \alpha\ B\ t\ \beta$ então $t \in$ Follow(B)
  - Se $S \Rightarrow^* \alpha\ B$ então $\$ \in$ Follow(B)

Exemplo First/Follow

$S \rightarrow A\ B$
$A \rightarrow c \mid \varepsilon$
$B \rightarrow cbB \mid ca$

Exemplo First/Follow

$S \rightarrow A\ B$
$A \rightarrow c \mid \varepsilon$
$B \rightarrow cbB \mid ca$

First(A) = {c, ε}   Follow(A) = {c}
First(B) = {c}   Follow(B) = {$}
First(S) = {c}   Follow(S) = {$}

Algoritmo: proc Follow(B ∈ N)

```
Follow(S) := {$};
Repeat
 foreach p ∈ P do {
   // Varre as produções
   case p == A → αB  // a produção termina por B
        Follow(B) := Follow(B) ∪ Follow(A);

   case p == A → αBβ {// a produção NÃO termina por B
      Follow(B) := Follow(B) ∪ First(β)\{ε};
      if ε ∈ First(β) then
        Follow(B) := Follow(B) ∪ Follow(A);
      end
   }
}
until no change in any Follow(N)
```

Observações First/Follow

- Só terminais entram em First e Follow.
- O algoritmo de cálculo de First(α):
  - É trivial quando α é um terminal t.
  - varre as produções X → tω quando α é um não-terminal X;
  - é chato quando o início de ume derivação de X deriva em ε.
  - Inclui ε apenas quando X pode derivar em ε.
- O algoritmo de cálculo de Follow(X)
  - É reservado aos não-terminais X
  - Inclui o $ em alguns casos triviais (X == o start S)
  - Varre as produções onde X aparece à direita (A → ωX ω')
  - É chato quando X aparece no fim (ou logo antes de algo que deriva em ε)
  - NUNCA inclui ε

Exemplo First/Follow

$S \to XYZ$
$X \to aXb \mid \varepsilon$
$Y \to cYZcX \mid d$
$Z \to eZYe \mid f$

Exemplo First/Follow

$S \to XYZ$
$X \to aXb \mid \varepsilon$
$Y \to cYZcX \mid d$
$Z \to eZYe \mid f$

First(X) = {a, ε} Follow(X) = {c, d, b, e, f}
First(Y) = {c, d} Follow(Y) = {e, f}
First(Z) = {e, f} Follow(Z) = {\$, c, d}
First(S) = {a, c, d} Follow(S) = {\$}

Gramática LL(1)

Condições necessárias:
- sem ambigüidade
- sem recursão a esquerda

Uma gramática G é LL(1) sse
Para quaisquer duas produções de G
$A \to \alpha|\beta \Rightarrow^* t$
1. $First(\alpha) \cap First(\beta) = \varnothing$
2. $\alpha \Rightarrow^* \boldsymbol{\varepsilon}$ implies $!(\beta \Rightarrow^* \boldsymbol{\varepsilon})$
3. $\alpha \Rightarrow^* \boldsymbol{\varepsilon}$ implies $First(\beta) \cap Follow(A) = \varnothing$

(2) Significa que **ε** pertence no máximo ao First de um símbolo.

LL(1) = leitura Left -> right
+ derivação mais a esquerda (Left) +
+ uso de 1 token lookahead.

Sumário

- Gramáticas LL(1) podem ser analisadas por um simples parser descentente recursivo
  - Sem recursão a esquerda
  - Fatorada a esquerda
  - 1 símbolo de look-ahead
- Nem todas as linguagens podem ser tornadas LL(1).
- Ainda se pode usar um mecanismo mais poderoso para reconhecer tais gramáticas.
  - Eliminar a recursividade.

Próxima aula

- Análise LL(1) com tabela preditiva.