Compiladores - 27/02/2014

Análise Sintática

1. Processamento de gramáticas
2. Gramática reduzida: todos os não-terminais participam na derivação de strings da linguagem. Não-terminais problemáticos:
   (a) não-terminais que não são alcançáveis a partir do símbolo inicial
   (b) não-terminais onde qualquer produção contendo ele não pode ser derivado a ponto de conter somente terminais
   (c) Exemplo: $S \rightarrow A|B; A \rightarrow a; B \rightarrow Bb; C \rightarrow c$
   (d) Existem algoritmos para identificar/remover esses dois tipos de produções da gramática
3. Tradução de BNF para gramática padrão
   (a) Conversão de []

   `para cada produção` p `na forma` $A \rightarrow \alpha[X_1...X_n]\beta$
   `seja` N `um novo não-terminal`
   `substitua` p `por` $A \rightarrow \alpha N\beta$
   `adicione` $N \rightarrow X_1...X_n$
   `adicione` $N \rightarrow \epsilon$

   (b) Conversão de []

   `para cada produção` p `na forma` $A \rightarrow \alpha\{X_1...X_n\}\beta$
   `seja` N `um novo não-terminal`
   `substitua` p `por` $A \rightarrow \alpha N\beta$
   `adicione` $N \rightarrow X_1...X_nN$ *// recursão à direita*
   `adicione` $N \rightarrow \epsilon$

4. Parsers top-down e bottom-up
   (a) Parser é top-down se gera uma árvore começando pela raiz da árvore
      i. expandindo-a e aplicando produções numa maneira depth-first
      ii. equivale a um caminhamento pre-order da árvore
      iii. Parsing top-down é **preditivo** porque precisa prever a produção a ser casada antes que o casamento comece
   (b) Parser é bottom-up se começa pelas folhas da árvore e caminha em direção à raiz
      i. Um nó é inserido somente depois que seus filhos foram inseridos
      ii. corresponde a um caminhamento post-order
5. Estratégias mais conhecidas e usadas de parsing:

(a) LL - Left-to-right Leftmost-parse
   i. top-down, produz derivação à esquerda
   ii. subconjunto das gramáticas livres de contexto determinísticas

(b) LR - Left-to-right Rightmost-parse
   i. bottom-up, constrói derivação à direita em reverso

(c) Especificamos ainda o número de símbolos de lookahead: LL(1) e LR(1) usam um símbolo de lookahead
   i. LL trata subconjunto das linguagens livres de contexto determinísticas
   ii. LR(1) trata todas as linguagens livres de contexto determinísticas

6. Analisador top-down

(a) Usa uma pilha para indicar o que falta ser lido

(b) Começa com (S, w)
   i. S é o estado inicial na pilha
   ii. w é a string de entrada completa

(c) Se o topo da pilha é $X \in V$ e eu tenho uma regra $X \to A_1...A_n$ posso desempilhar X e empilhar $A_1...A_n$

(d) Se o topo da pilha é $\alpha \in T$ e a entrada é αw, leio α e retiro o α do topo da pilha

(e)
$$\begin{array}{lcl} L & \to & A|\epsilon \\ A & \to & nome|nome\text{","}\,A \end{array}$$
não é LL(1)!

(f)
$$\begin{array}{lcl} L & \to & A|\epsilon \\ A & \to & nome\,B \\ B & \to & \epsilon|\text{","}\,A \end{array}$$
é LL(1)

7. Conjuntos First e Follow

(a) First$(A_1...A_n)$=$\{a \in T | \alpha \Rightarrow^* a\beta\}$
   i. Todo primeiro caracter que pode ser gerado em derivações possíveis de $A_1...A_n$ contendo apenas não-terminais
   ii. O conjunto de "primeiros caracteres" de todas as palavras geráveis em $L(A_1...A_n)$

(b) Follow(A)=$\{x \in T | S \overset{*}{\Rightarrow} \alpha A x \beta\}$
   i. Saindo de S, é possível construir uma sequência onde x sucede imediatamente A

8. Predição

(a) Predict$(X \to \alpha)$=$\begin{cases} First(\alpha) & \text{se } \epsilon \notin L(\alpha) \\ First(\alpha) \cup Follow(X) & \text{se } \epsilon \in L(\alpha) \end{cases}$

(b) Se o topo da pilha é X e o lookahead é a e $a \in Predict(X \to \alpha)$, posso substituir X por α na pilha.

(c) Se existem duas produções $X \to \alpha$ e $X \to \beta$ tal que o símbolo de lookahead atual está no Predict de ambas (i.e., $Predict(X \to \alpha) \cap Predict(X \to \beta) \neq \varnothing$) temos um **conflito** LL(1)

9. Calculando First

```
fun First(a: string): Set of T
   for A in V
      visitado[A] = false
   next
   return RecFirst(a)
end
fun RecFirst(a: string): Set of T
   if a = “”
      return {}
   end
   X:char = a[1]
   resto:string = a[2:]
   if X in T
      return {X}
   end
   r:Set of T = {}
   if not visitado[X]
      visitado[X] = true
      for direita in ProduçõesDe(X) // regras X → direita
         r = r ∪ RecFirst(direita)
      next
   end
   if SímboloDerivaEpsilon(X) // existe X → ϵ ?
      r = r ∪ RecFirst(resto)
   end
   return r
end
```

10. Calculando Follow

```
fun Follow(A: char): Set of T
   for A in T
      visitado[A] = false
   next
   return RecFollow(A)
end
fun RecFollow(A: char): Set of T
   r:Set of T = {}
   if visitado[A]
      return {}
   end
   visitado[A] = true
   for a in Ocorrências(A) // aparecimentos de A em regras: exemplo Z → αAβ
      r = r ∪ First(Após(a)) // β
      if TodosDerivamEpsilon(Após(a))
         r = r ∪ RecFollow(Esquerda(Regra(a))) // Z
      end
   next
   return r
end
fun TodosDerivamEpsilon(s:string): bool
   for c in s
```

```
      if c in T or not SímboloDerivaEpsilon(X)
         return false
      end
   next
   return true
end
```