Análise LL(1)

Construção de compiladores I

Objetivos

Objetivos

- Apresentar o algoritmo para construção de tabelas LL(1).
- Apresentar o algoritmo de análise sintática preditiva.

Objetivos

- Apresentar uma implementação Haskell do algoritmo.

Gramáticas LL(1)

Gramáticas LL(1)

- Na última aula, definimos os conjuntos first e follow.
- Usaremos esses conjuntos para definir gramáticas LL(1).

Gramáticas LL(1)

- Para isso, vamos definir o conceito de $first^+$:

$first^+(A \to \alpha) = first(\alpha) \quad \lambda \notin first(\alpha)$
$first^+(A \to \alpha) = first(\alpha) \cup follow(A) \quad \lambda \in first(\alpha)$

Gramáticas LL(1)

- Dizemos que uma gramática é LL(1) se:

$\forall 1 \leq i, j \leq n, i \neq j \to$
$first^+(A \to \alpha_i) \cap first^+(A \to \alpha_j) = \emptyset$

Gramáticas LL(1)

- Gramáticas LL(1) admitem analisadores sem retrocesso.
- Veremos como tal analisador pode ser contruído.

Análise preditiva

Análise preditiva

- Determina a produção da gramática a ser usada com base no próximo token da entrada.

Análise preditiva

- Analisadores preditivos utilizam uma tabela para decidir qual regra será utilizada.
- Tabela construída utilizando os conjuntos first e follow.

Análise preditiva

- Tabela indexada por V e símbolos de Σ mais o marcador de final de entrada.
- O símbolo \$ marca o final da entrada.

Análise preditiva

- Tabela armazena produções da gramática.
- Entrada $M[A, a]$ armazena a regra a ser utilizada se $a \in first(\alpha)$.

Construção da tabela

Construção da tabela

- Dada uma gramática G, calcule os conjuntos first e follow de cada não terminal de G.
- Para cada regra $A \to \alpha$ da gramática, faça os seguintes passos:

Construção da tabela

- Para cada $a \in first(A)$, inclua $A \to \alpha$ em $M[A, a]$.
- Se $\lambda \in first(\alpha)$, inclua $A \to \alpha$ em $M[A, b]$ para cada $b \in follow(A)$.

Construção da tabela

- Se $\lambda \in first(\alpha)$ e \$ $\in follow(A)$, coloque $A \to \alpha \in$ M[A,\$].

Construção da tabela

- Gramática de exemplo

$$
\begin{array}{lcl}
E & \to & TE' \\
E' & \to & +TE' \mid \lambda \\
T & \to & FT' \\
T' & \to & *FT' \mid \lambda \\
F & \to & (E) \mid \mathbf{id}
\end{array}
$$

Construção da tabela

- first(F) = first (T) = first(E) = $\{(, id\}$.
- first(E') = $\{+, \lambda\}$.
- first(T') = $\{*, \lambda\}$.

Construção da tabela

- follow(E) = follow(E') = {),\$}.
- follow(T) = follow(T') = {+,),\$}.
- follow(F) = {+, *,), \$}.

Construção da tabela

- Produção $E \to TE'$.
  - first(TE') = first(T) = {(,id}.
  - $M[E, id] = M[E, (] = E \to TE'$.

Construção da tabela

- Produção $E' \rightarrow +TE'$.
  - $first(+TE') = +$.
  - $M[E', +] = E' \rightarrow +TE'$.

Construção da tabela

- Produção $E' \rightarrow \lambda$.
  - $first(\lambda) = \lambda$.
  - follow(E') = {),$}.
  - M[E',)] = M[E',$] = $E' \rightarrow \lambda$

Construção da tabela

- Produção $T \rightarrow FT'$
  - $first(T) = first(FT') = \{(, id\}$.
  - $M[T, (] = M[T, id] = T \rightarrow FT'$.

Construção da tabela

- Produção $T' \rightarrow *FT'$
  - $first(*FT') = \{*\}$.
  - $M[T', *] = T' \rightarrow FT'$.

Construção da tabela

- Produção $T' \rightarrow \lambda$
  - $first(\lambda) = \lambda$
  - follow(T') = {+,),$}.
  - M[T',+] = M[T',)] = M[T',$] = $T' \rightarrow \lambda$.

Construção da tabela

- Produção $F \rightarrow id$
  - first(id) = {id}
  - M[F,id] = id

Construção da tabela

- Produção $F \to (E)$
  - first((E)) = {(}
  - $M[F, (] = F \to (E)$

Implementação em Haskell

Implementação em Haskell

- Definição da tabela

```
type Table = Map (Nonterminal, Terminal) [Production]
```

Implementação em Haskell

- Construção da tabela

```
buildTable :: Grammar -> Table
buildTable g = foldr step Map.empty (productions g)
  where
    firstG = first g
    followG = follow g
    insertTable p s tbl
       = Map.insertWith union (leftHand p, s) [p] tbl
```

Implementação em Haskell

- Construção da tabela

```
step p tbl
   = let
       firstM = Map.fromList firstG
       lhs = leftHand p
       rhs = rightHand p
       firstP = firstForWord rhs firstM
       followP = maybe [] id (lookup lhs followG)
       tbl1 = foldr (insertTable p) tbl [x | x <- firstP, x /= Lambda]
       tbl2 = if Lambda `elem` firstP
              then foldr (insertTable p) tbl1 followP
              else tbl1
```

```
in if Lambda ‘elem‘ firstP &&
      Dollar ‘elem‘ followP
   then insertTable p Dollar tbl2
   else tbl2
```

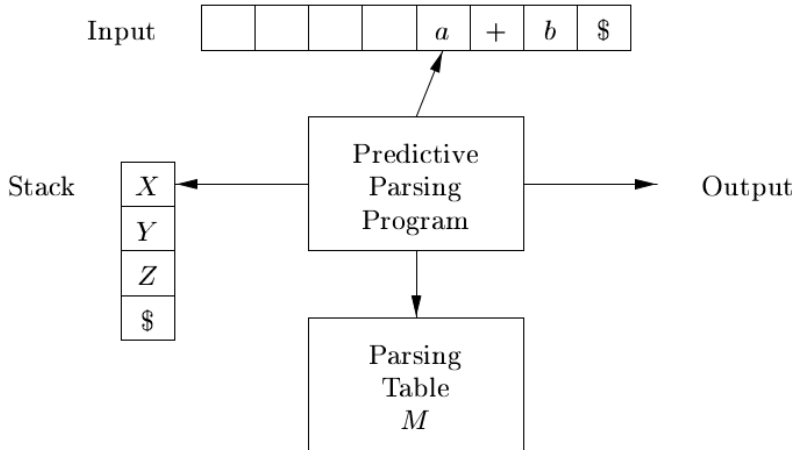
Análise preditiva

Análise preditiva

- O algoritmo utiliza:
  - Tabela
  - Pilha

Análise preditiva

- Estrutura do analisador



Análise preditiva

- Inicialização
  - Entrada w$
  - Pilha: Símbolo de partida no topo, $ no fundo.

Análise preditiva

- Seja X o símbolo de topo da pilha.
- Seja a o primeiro token da entrada.
- Se $X = a$, desempilhe X e obtenha próximo token.

Análise preditiva

- Se X é um não terminal, seja $r = M[X, a]$.
- Se r é erro, pare.
- Se $r = X \rightarrow Y_1 ... Y_k$
  - Desempilhe X.
  - Empilhe $Y_k ... Y_1$.

Análise preditiva

- Vamos considerar a gramática

$$
\begin{array}{lcl}
E & \rightarrow & TE' \\
E' & \rightarrow & +TE' \mid \lambda \\
T & \rightarrow & FT' \\
T' & \rightarrow & \mathbf{*}FT' \mid \lambda \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

Análise preditiva

- Vamos considerar a string id + id.

Análise preditiva

- Inicialização
  - Entrada: id + id$
  - Pilha: E$

Análise preditiva

- Temos que:
  - $X = E$
  - $a = id$

Análise preditiva

- Temos que $M[E, id] = E \rightarrow TE'$
  - Entrada: id+id$
  - Pilha: TE'$

Análise preditiva

- Temos que:
  - $X = T$
  - $a = id$

Análise preditiva

- Temos que $M[T, id] = T \rightarrow FT'$
  - Entrada: id+id$
  - Pilha: FT'E'$.

Análise preditiva

- Temos que:
  - $X = F$
  - $a = id$

Análise preditiva

- Temos que $M[F, id] = F \rightarrow id$
  - Entrada: id + id$
  - Pilha: idT'E'$.

Análise preditiva

- Temos que:
  - $X = id.$
  - $a = id.$

Análise preditiva

- Como $X = a$, desempilhamos X e obtemos próximo token.
  - Entrada: +id$
  - Pilha: T'E'$.

Análise preditiva

- Temos que:
  - $X = T'.$
  - $a = +.$

Análise preditiva

- Temos que $M[T', +] = T' \to \lambda$.
  - Entrada: +idE
  - Pilha: E'$.

Análise preditiva

- Temos que:
  - $X = E'.$
  - $a = +.$

Análise preditiva

- Temos que $M[E', +] = E' \to +TE'$.
  - Entrada: +id$
  - Pilha: +TE'$

Análise preditiva

- Temos que
  - $X = +$
  - $a = +$

Análise preditiva

- Como $X = a$, desempilhamos X e obtemos o próximo token.
  - Entrada: id$.
  - Pilha: TE'$.

Análise preditiva

- Temos que
  - $X = T$
  - $a = id$

Análise preditiva

- Temos que $M[T, id] = T \rightarrow FT'$
  - Entrada: id$
  - Pilha: FT'E'$

Análise preditiva

- Temos que
  - $X = F$
  - $a = id$

Análise preditiva

- Temos que $M[F, id] = F \rightarrow id$
  - Entrada: id$
  - Pilha: idT'E'$.

Análise preditiva

- Temos que
  - $X = id.$
  - $a = id.$

Análise preditiva

- Como $X = a$, desempilhamos X e obtemos o próximo token.
  - Entrada: \$
  - Pilha: T'E'\$.

Análise preditiva

- Temos que:
  - $X = T'$.
  - a = \$.

Análise preditiva

- Temos que M[T',\$] = $T' \rightarrow \lambda$:
  - Entrada: \$
  - Pilha: E'\$

Análise preditiva

- Temos que:
  - $X = E'$.
  - a = \$.

Análise preditiva

- Temos que M[E',\$] = $E' \rightarrow \lambda$:
  - Entrada: \$
  - Pilha: \$

Análise preditiva

- Temos que:
  - X = $
  - a = $

Análise preditiva

- como $X = a$, desempilhamos X e como não há próximo token o algoritmo encerra com sucesso.

Implementação em Haskell

Implementação em Haskell

- Definição do estado do algoritmo

```
data Pred
  = Pred {
      stack :: [Symbol]
    , input :: String
    , table :: Table
    }
```

Implementação em Haskell

- Inicialização

```
initial :: Grammar -> String -> Pred
initial g s
  = Pred stk (s ++ "$") (buildTable g)
    where
      stk = [Var (start g), Symb Dollar]
```

Implementação em Haskell

- Implementação da análise preditiva

```
predictiveM :: Lexer -> PredictiveM ()
predictiveM plex
  = do
```

```
v <- emptyStack
if v then return ()
else do
    r <- nextToken plex
    a <- top
    when (isTerminal a && a /= (Symb r)) (throwError $ expecting a r)
    when (isTerminal a && a == (Symb r)) (pop >> consumeToken plex)
    when (isNonterminal a) $ do
      let nt' = nonTerminal a
      v <- pop
      p <- lookupTable nt' r
      when (isNothing p) (throwError "Parsing table error!")
      let p' = fromJust p
      push (rightHand p')
    predictiveM plex
```

Concluindo

Concluindo

- Nesta aula apresentamos o algoritmo de análise sintática preditiva.
- Apresentamos uma implementação deste algoritmo em Haskell

Concluindo

- Próxima aula: análise sintática ascendente.

Exercícios

Exercícios

- Estenda o algoritmo de análise preditiva para que este verifique se a gramática fornecida é ou não LL(1).