Valutazione top-down di SDT L-attribuiti

Valutazione top-down di definizioni L-attribuiti I

L'analizzatore a discesa ricorsiva per grammatiche LL(1) può essere *modificato* in modo da valutare gli L-attributi.

Durante la parsificazione un'azione nel corpo di una produzione è eseguita non appena tutti i simboli della grammatica a sinistra dell'azione sono stati "matched" (presi in esame).

Ad ogni non terminale si associa una funzione che ha
- parametri in input: i valori degli attributi ereditati della variabile e
- ritorna il valore (o i valori) dei suoi attributi sintetizzati.

Spesso è conveniente che la funzione per un non terminale abbia una variabile locale per ogni attributo ereditato o sintetizzato per ogni simbolo che occorre nelle corpo di una produzioni per la variabile stessa.

Valutazione top-down di SDT L-attribuiti

Program **traduzione_discesa_ricorsiva**

```
    begin cc ← PROSS
              risultato ← S()
              if (cc = '$') "stringa accettata"
              else ERRORE(…)
    end

function A(e1, . . . en)
    begin if (cc ∈ Gui(A → α1)) body (α1)
              else if (cc ∈ Gui(A → α2)) body (α2)
              ….
              else if (cc ∈ Gui(A → αk)) body (αk)
              else ERRORE(…)
              return < s1, . . . , sm>
    end
```

Valutazione top-down di definizioni L-attribuite III

Codice per le parti destre delle produzioni (body(α_i)):

- Per ogni non terminale B si genera un'assegnazione
  $$c \leftarrow B(b_1, \ldots b_n),$$
  che è una chiamata alla funzione associata a B, dove **$b_1, \ldots b_n$** sono gli attribui ereditati di B e **c** il valore dell'attributo sintetizza.

- Per ogni terminale i valori degli attributi sintetizzati vengono assegnati alle corrispondenti variabili e l'esame passa al simbolo successivo.

- Le azioni semantiche vengono ricopiate dopo aver sostituito i riferimenti agli attributi con le variabili corrispondenti.

Valutazione top-down di definizioni L-attribuite IV

Esempio: Lista delle differenze I

$C \rightarrow N\#$	{L.*elem* = N.*val*}
L	{C.*list* = L.*list*}
$L \rightarrow N;$	{L_1.*elem* = L.*elem*}
L_1	{*L.list* = *cons* (N.*val* - L.*elem*, L_1.*list*)}
$L \rightarrow \varepsilon$	{L.*list* = <>|}
$N \rightarrow$ **num**	{N.*val* = **num**.*val*}

La grammatica è LL(1):

	Insiemi guida
$C \rightarrow N\#L$	**{num}**
$L \rightarrow N;L_1$	**{num}**
$L \rightarrow \varepsilon$	**{$}**
$N \rightarrow$ **num**	**{num}**

La grammatica è stata ottenuta da quella di un esempio visto in precedenza sostituendo la produzione $L \rightarrow N$ con $L \rightarrow \varepsilon$ e modificando di conseguenza la regola semantica associata. Questa grammatica, a differenza della precedente, è LL(1).

Valutazione top-down di definizioni L-attribuite V

Esempio: Lista delle differenze II

```
Program traduzione_discesa_ricorsiva
main
    begin cc ← PROSS
          list ← C()
          if (cc = '$') "stringa accettata"
          else ERRORE(…)
    end
```

N → **num** {N.*val* = **num**.*val*}

```
function N
    var N_val
    begin if (cc = num) N_val ← num.val
                              cc ← PROSS
          else ERRORE(…)
          return N_val
    end
```

Valutazione top-down di definizioni L-attribuite VI

Esempio: Lista delle differenze III

C → N #		{L.*elem* = N.*val*}
	L	{C.*list* = L.*list*}

```
function C
    var C_list, N_val, L_elem, L_list
    begin if (cc = num)
                N_val ← N()
                if (cc = '#')
                        cc ← PROSS
                        L_elem ← N_val
                        L_list ← L (L_elem)
                        C_list ← L_list
                else ERRORE(…)
            else ERRORE(…)
            return C_list
    end
```

Valutazione top-down di definizioni L-attribuite VII

Esempio: Lista delle differenze IV

$L \rightarrow N$;	$\{L_1.elem = L.elem\}$
L_1	$\{L.list = \underline{cons}\ (N.val - L.elem, L_1.list)\}$
$L \rightarrow \varepsilon$	$\{L.list = <>\}$

```
function L (L_elem)
    var L_list, N_val, L1_elem, L1_list
    begin if (cc = num)
                N_val ← N()
                if (cc = ';')
                        cc ← PROSS
                        L1_elem ← L_elem
                        L1_list ← L (L1_elem)
                        L_list ← cons (N_val – L_elem, L1_list)
                else ERRORE(…)
          else if (cc = '$')   L_list ← <>
          else ERRORE(…)
          return L_list
    end
```

Valutazione top-down di definizioni L-attribuite VIII

Esempio: Lista delle differenze V

4 # 6 ; 3 ; **$**

↑

```
function C
var .....
begin if (cc = num)
        N_val ← N()
        if (cc = num) N_val ← num.val
                cc ← PROSS ; return N_val
        if (cc = '#') cc ← PROSS
            L_elem ← N_val
            L_list ← L (L_elem)
             if (cc = num)  N_val← N()
                        if (cc = num) N_val ← num.val
                                cc ← PROSS ; return N_val
                    if (cc = ';') cc ← PROSS
                        L1_elem[1] ← L_elem
                        L1_list ← L(L1_elem)
                          if (cc = num)  N_val ← N()
                                    if (cc = num) N_val ← num.val
                                            cc ← PROSS ; return N_val
                                if (cc = ';') cc ← PROSS
                                    L1_elem ← L_elem
                                    L1_list ← L (L1_elem[2])
                                       if (cc = '$') L_list ← <>
                                            return L_list
                                    L_list ← cons (N_val – L1_elem, L1_list)
                                return L_list
                        L_list ← cons (N_val – L_elem, L1_list)
                    return L_list
            C_list ← L_list
        return C_list
end
```

{N_val = 4}

{L_elem = 4}
{L(4)} =>

{N_val = 6}

{L1_elem = 4}
{L(4)} =>

{N_val = 3}

{L1_elem= 4}
{L(4)} =>

{L_list = <>}
{L1_list = <>}

{L_list = < -1 >}
{L1_list = <-1>

{L_list = < 2,-1 >}

{C_list = < 2,-1 >}

Piu' attributi sintetizzati: Traduzione binario -> decimale I

Esempio I

$N \rightarrow D\ K$ {N.*val* = D.*val* $*$ $2^{K.left}$ + K.*val*) ; N.left = K.*left* }
$K \rightarrow N$ {K.*val* = N.*val* ; K.*left* = *N.left* + 1}
$K \rightarrow \varepsilon$ {K.*val* =0 ; K.left = 0}
$D \rightarrow 0$ {D.*val* =0}
$D \rightarrow 1$ {D.*val* =1}

Dove .*val* rappresena il valore, .*left* il mumero di cifre del numero che deve essere allungato..

La grammatica è LL(1) in quanto le produzioni da uno stesso non terminale hanno insiemi guida disgiunti.

Gui ($K \rightarrow N$) = {0,1} Gui ($D \rightarrow 0$) = {0} Gui ($N \rightarrow DK$) = {0,1}
Gui ($K \rightarrow \varepsilon$) = {⊣} Gui ($D \rightarrow 1$) = {1}

Traduzione binario -> decimale II

I due parametri ereditati vengono passati come unico valore di ritorno costituito da una coppia < , > i cui componenti rappresentano rispettivamente .*val* e . *left* e p_1, p_2 sono le due proiezioni che individuano la prima e la seconda componente della coppia, rispettivamente).

N $\rightarrow$ D K {N.*pair*= **<** D.*val* x 2 $^{\mathbf{p_2}(\text{K.}pair)}$ + $\mathbf{p_1}$(K.*pair*), $\mathbf{p_2}$(K.*pair*)**>**}
K $\rightarrow$ N {K.*pair*= **<** $\mathbf{p_1}$(N. *pair*), $\mathbf{p_2}$(N. *pair*) + 1**>**}
K $\rightarrow$ ε {K. *pair* = **<**0,0**>**}
D $\rightarrow$ 0 {D. *val* = 0}
D $\rightarrow$ 1 {D. *val* = 0}

NB. Questa sarebbe ovviamente inutile in linguaggi che permettono per esempio piu' parametri di ritorno (o con I parametri di tipo ref dei inguaggi Pascal-like)

Traduzione binario -> decimale III

Esempio II

$N \rightarrow D\ K$ {N.*pair*= **<** D.*val* x 2 $^{\mathbf{p_2}(K.pair)}$ + $\mathbf{p_1}$(K.pair), $\mathbf{p_2}$(K.*pair*)**>**}

```
function N()
var N_pair, K_val, D_val
      begin if (cc ∈ {0,1})
                  D_val ← D()
                  K_pair← K()
                  N_pair ← <D_val x 2 p2(K_pair) + p1(K_pair), p2(K_pair)>
            else ERRORE (…)
            return N_val
      end
```

Traduzione binario -> decimale IV

$K \rightarrow N$ {K.*pair* = < p_1(N.*pair*), p_2(N.*val*) + 1>}
$K \rightarrow \varepsilon$ {K.*pair* = <0,0>}

```
function K()
    var N_pair, K_pair
    begin if (cc ∈ {0,1})
                N_pair ← N()
                K_pair ← <p1(N_pair), p2(N_pair) + 1>
          else if (cc = $ )   K_pair ← < 0,0>
          else ERRORE (…)
          return K_pair
    end
```

Traduzione binario -> decimale V

$D \rightarrow 0$ {D.*val* = 0}
$D \rightarrow 1$ {D.*val* = 1}

```
function D()
   var D_val
   begin if (cc = 0)
                D_val ← 0
                cc ← PROSS
         else if (cc = 1)
                D_val ← 1
                cc ← PROSS
         else ERRORE (…)
         return D_val
   end
```

Che cosa associa il traduttore ad ogni stringa prodotta dalla grammatica ?

Schema L-attribuita per valutazione espressioni

E → T {E'.*p*= T.*val*
E' {E.*val*= E'.*val*}

E' → + T {E_1'.*p* = E'.*p* + T.*val*}
E' {E'.*val* = E_1'.*val*}
E' → ε {E'.*val*= E'.*p*}

T → F {T'.p = F.*val*}
T' {T.*val* = T'.*val*}

T' → * F {T_1'.*p* = T'.*p* × F.*val*}
T_1' {T'.*val* = T_1'.*val*}
T' → ε {T'.*val*= T'.*p*}

F → (E) {F.*val* = E.*val*}
F → **num** {F.*val* = **num**.*lexval*}

.p : *sintetizzato*
.val: *ereditato*

Funzioni

E → T {E'.*p*= T.*val*}
E' {E.*val*= E'.*val*}

```
function E()
    begin
        if cc = '(' or cc = num  then
          E'_p  =  T( )
          E'_val = E'(E'_p)
          return E'_val
     else ERROR ()
```

Funzioni

$E' \rightarrow + T \ \{E_1'.p = E'.p + T.val\}$
$E_1' \ \{E'.val = E_1'.val\}$

$E' \rightarrow \varepsilon \ \{E'.val = E'.p\}$

```
function E'(E'_p)
    begin
        if cc = '+' then
            T_val  = T()
            E1'_p  = E'.p + T_val
            E1'_val = E'(E1'_p)
        else if  cc = ')' or cc = $  then E'_val = E'_p
        else ERROR ()
        return E'_val
    end
```

Funzioni

T → F {T'.*p*= F.*val*}
T' {T.*val*= T'.*val*}

```
function T ()
    begin
        if cc = '(' or cc = num then
            T'_p = F( )
            T'_val = T'(T'_p)
            return T'_val
        else ERROR ()
        end
```

Funzioni

$T' \rightarrow * F \ \{T_1'.p = T'.p \times F.val\}$
$T_1' \ \{T'.val = T_1'.val\}$

$T' \rightarrow \varepsilon \ \{T'.val = T'.p\}$

```
function T'(T'_p)
    begin
        if cc = '*' then
            F_val  = F()
            T1'_p  = T'.p + F_val
            T_val = T'(T1'_p)
        else if  cc = '+' or ')' or cc = $  then T'_val = T'_p
        else ERROR ()
        return T'_val
    end
```

Funzioni

F → (E) {F.*val* = E.*val*}

F → **num** {F.*val* = **num**.*lexval*}

```
function F()
      begin
            if  cc = ‘num‘ then
                        F_val = num.lexval
             else if cc = ‘(‘ then  cc  = PROSS
                        F_val = E()
                        if cc = ‘)’ then cc = PROSS
            else ERROR()
             return F_val
       else ERROR ()
     end
```