UNIVERSITÀ
DEGLI STUDI
DI TORINO

Schemi di traduzione e valutazione top-down

a.a. 2018-2019

Riferimenti bibliografici

Compilatori: principi, tecniche e strumenti, A.V. Aho, M. S. Lam, R. Sethi, J.D.Ullman

Traduzione guidata dalla sintassi [cap. 5]
- 5.2 Ordine di valutazione delle SDD
  - 5.2.3 Definizioni S-attribuite
  - 5.2.4 Definizioni L-attribuite
- 5.3 Applicazioni della traduzione guidata dalla sintassi
  - 5.3.1 Costruzione degli alberi sintattici
- 5.4 Schemi di traduzione guidati dalla sintassi
  - 5.4.5 Schemi di traduzione per definizioni L-attribuite
- 5.5 Implementazione di SDD L-attribuite
  - 5.5.1 Traduzione durante il parsing a discesa ricorsiva

Schemi di traduzione: SDT

Gli **schemi di traduzione** (**SDT**) sono un'utile notazione per specificare la traduzione durante la parsificazione.

Uno schema di traduzione è una definizione guidata dalla sintassi in cui le azioni semantiche, racchiuse tra parentesi graffe, sono inserite nei membri destri delle produzioni, in posizione tale che il valore di un attributo sia disponibile quando un'azione fa ad esso riferimento.

Gli schemi di traduzione impongono un ordine di valutazione da sinistra a destra e permettono che nelle azioni semantiche siano contenuti frammenti di programma e in generale side-effect che non influiscano sulla valutazione degli attributi.

Definizioni S-attribuite

Una SDD è **S-attribuita** se **tutti gli attributi sono sintetizzati**

Esempio:

$E \rightarrow E_1 + T$	E.*val* = E_1.*val* + T.*val*
$E \rightarrow E_1 - T$	E.*val* = E_1.*val* - T.*val*
$E \rightarrow T$	E.*val* = T.*val*
$T \rightarrow (E)$	T.*val* = E.*val*
$T \rightarrow$ **num**	T.*val* = num.*val*

Lo schema di traduzione si ottiene ponendo le azioni semantiche alla fine delle produzioni.

$E \rightarrow E_1 + T$	{E.*val* = E_1.*val* + T.*val*}
$E \rightarrow E_1 - T$	{E.*val* = E_1.*val* - T.*val*}
$E \rightarrow T$	{E.*val* = T.*val*}
$T \rightarrow (E)$	{T.*val* = E.*val*}
$T \rightarrow$ **num**	{T.*val* = num.*val*}

Definizioni L-attribuite

Una SDD è **L-attribuita** se gli attributi sono
- **sintetizzati** e/o
- **ereditati**, che soddisfano il seguente vincolo:

per ogni produzione $A \rightarrow X_1 X_2 \ldots X_n$, ogni attributo ereditato di X_j dipende solo da:

- attributi ereditati o sintetizzati dei simboli $X_1 X_2 \ldots X_{j-1}$ a sinistra di X_j nella produzione;
- attributi ereditati di A
- altri attributi di X_j, purchè non vi siano cicli nel grafo delle dipendenze formati dagli attributi di questa occorrenza di X_j.

Nel grafo delle dipendenze gli archi tra gli attributi associati a una produzione possono andare da sinistra a destra, ma non da destra a sinistra (**L**eft to right).

Schemi di traduzione e SDD L-attribuite

Trasformare una SDD L-attribuita in un SDT:

- Inserire le azioni che calcolano gli attributi ereditati per un non terminale A immediatamente prima dell'occorrenza di A nel corpo della produzione.
- Se diversi attributi ereditati per A dipendono uno dall'altro, ordinare la valutazione degli attributi in modo che quelli necessari prima siano calcolati per primi.
- Porre la azioni che calcolano un attributo sintetizzato per la variabile a sinistra in una produzione alla fine del corpo della produzione stessa.

Schemi di traduzione: lista delle differenze

$C \rightarrow N\#L$	C.*list* = L.*list* ; L.*elem* = N.*val*
$L \rightarrow N;L_1$	L.*list* = *cons* (N.*val* - L.*elem* , L_1.*list*) ; L_1.*elem* = L.*elem*)
$L \rightarrow N$	L.*list* = *cons* (N.*val* - L.*elem*, *null*)
$N \rightarrow$ **digit**	N.*val* = **digit**.*lexval*

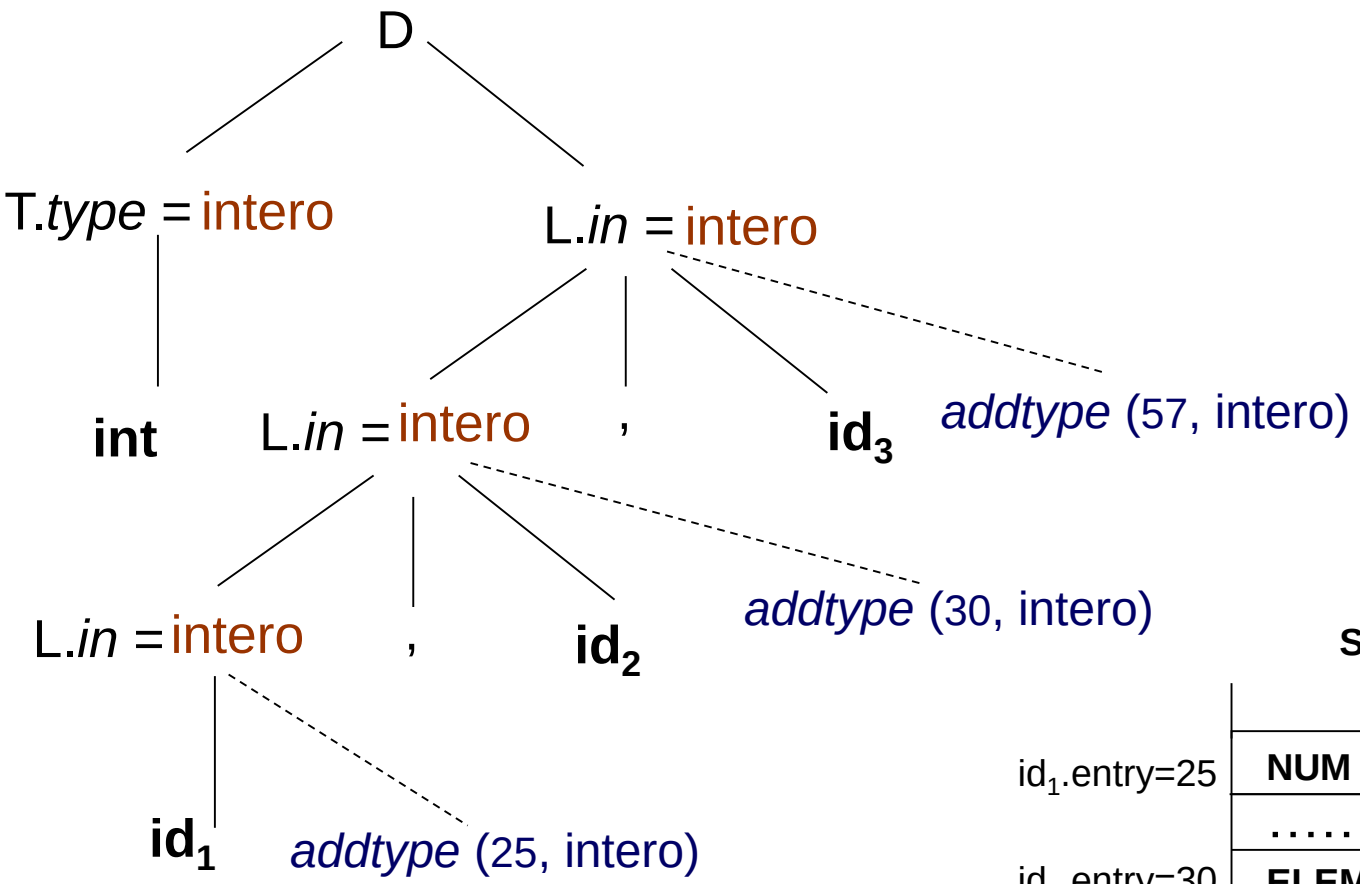
$C \rightarrow N\#$ {L.*elem* = N.*val*}
L {C.*list* = L.*list*}

$L \rightarrow N;$ {L_1.*elem* = L.*elem*}
L_1 {*L.list* = *cons* (N.*val* - L.*elem*, L_1.*list*)}

$L \rightarrow N$ {L.*list* = *cons* (N.*val* - L.*elem*, null)}

$N \rightarrow$ **digit** {N.*val* = **digit**.*val*}

Schemi di traduzione: dichiarazioni di tipo



$D \rightarrow T\ L$

$L \rightarrow L_1$, **id**

$L \rightarrow$ **id**

$T \rightarrow$ **int**

$T \rightarrow$ **real**

Symbol Table

id_1.entry=25	**NUM**	intero
	.....	.....
id_2.entry=30	**ELEM**	intero
	.....	.....
	.....	
id_3.entry=57	**K**	intero

int NUM, ELEM, K

Schemi di traduzione: dichiarazioni di tipo

int NUM, ELEM, K

Produzioni	Regole semantiche
$D \rightarrow T\ L$	L.*in* = T.*type*
$L \rightarrow L_1$, **id**	L_1.*in* = L.*in*; *addtype* (**id**.*entry*, L.*in*)
$L \rightarrow$ **id**	*addtype* (**id**.*entry*, L.*in*)
$T \rightarrow$ **int**	T.*type* = intero
$T \rightarrow$ **real**	T.*type* = reale

Schema di traduzione

$D \rightarrow T\ \{L.in = T.type\}\ L$

$L \rightarrow \{L_1.in = L.in\}\ L_1$, **id** {*addtype* (**id**.*entry*, L.*in*)}

$L \rightarrow$ **id** {*addtype* (**id**.*entry*, L.*in*)}

$T \rightarrow$ **int** {T.*type* = intero}

$T \rightarrow$ **real** {T.*type* = reale}

SDD e schemi di traduzione

Due importanti classi di SDD:

SDD S-attribuite
SDD L-attribuite

garantiscono l'esistenza di un ordine di valutazione poiché non ammettono grafi di dipendenza ciclici e le regole possono essere convertite in un SDT con azioni specificate al momento giusto.

Uno schema di traduzione può essere implementato costruendo prima l'albero di parsificazione e poi eseguendo le azioni nell'ordine di visita in profondità, da sinistra verso destra.

Tipicamente però gli SDT sono implementati durante la parsificazione, senza costruzione esplicita dell'albero, in particolare in due casi:

- SDD L-attribuita e grammatica LL-parsificabile
- SDD S-attribuita e grammatica LR-parsificabile

Valutazione top-down di SDD L-attribuite

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

Durante la parsificazione un’azione semantica nel corpo di una produzione (nello schema di traduzione) 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 come parametri in input i valori degli attributi ereditati della variabile e restituisce i valori dei suoi attributi sintetizzati.

La funzione per un non terminale ha una variabile locale per ogni attributo ereditato o sintetizzato per i simboli che compaiono nelle parti destre delle produzioni da quel non terminale.

Valutazione top-down di SDD L-attribuite

main **traduzione_discesa_ricorsiva**( )
  var risultato
        cc ← PROSS
        risultato ← S( )
        if (cc = '$') stampa ("stringa corretta, la sua traduzione è:" risultato)
        else ERRORE(…)

function A(e_1, . . . e_n)
    var s_1, . . . , s_m, $X_1\_a_1$, . . . , $X_1\_a_k$, . . . , $X_h\_a_1$, . . . , $X_h\_a_r$
        if (cc $\in$ Gui(A $\rightarrow \alpha_1$)) body'(α_1)
        elseif (cc $\in$ Gui(A $\rightarrow \alpha_2$)) body'(α_2)
        ….
        elseif (cc $\in$ Gui(A $\rightarrow \alpha_k$)) body'(α_k)
        else ERRORE(…)
        return (<s_1, . . . , s_m>)

Valutazione top-down di SDD L-attribuite

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

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

- 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.
  Poiché la SDD è L-attribuita, gli argomenti (attributi) saranno già stati calcolati e memorizzati nelle variabili locali.

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

Valutazione di SDD L-attribuite: lista delle differenze

$C \rightarrow N\#$	{L.*elem* = N.*val*}
$\quad L$	{C.*list* = L.*list*}
$L \rightarrow N;$	{L_1.*elem* = L.*elem*}
$\quad L_1$	{*L.list* = *cons* (N.*val* - L.*elem*, L_1.*list*)}
$L \rightarrow \varepsilon$	{L.*list* = null}
$N \rightarrow$ **digit**	{N.*val* = **digit**.*val*}

La grammatica è LL(1):

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

N.B. 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 di SDD L-attribuite: lista delle differenze

```
main traduzione_discesa_ricorsiva( )
  var list
      cc ← PROSS
      list ← C( )
      if (cc = '$')
              write ("stringa accettata. La sua traduzione è" list)
      else ERRORE(…)
```

N → **digit** {N.*val* = **digit**.*lexval*}

```
function N( )
   var N_val
        if (cc = digit) N_val ← digit. lexval
                          cc ← PROSS
        else ERRORE(…)
        return (N_val)
```

Valutazione di SDD L-attribuite: lista delle differenze

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

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

Valutazione di SDD L-attribuite: lista delle differenze

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

```
function L (L_elem)
var L_list, N_val, L1_elem, L1_list
        if (cc = digit)
            N_val ← N( )
            if (cc = ';') cc ← PROSS
            else ERRORE(…)
            L1_elem ← L_elem
            L1_list ← L (L1_elem)
            L_list ← cons (N_val – L_elem, L1_list)
        elseif (cc = '$')   L_list ← null
        else ERRORE(…)
        return (L_list)
```

Valutazione di SDD L-attribuite: stringhe binarie

Nel seguente schema di traduzione, < , > è una coppia e p_1, p_2 sono le proiezioni (cioè p_1(D.*val*) e p_2(D.*val*) individuano la prima e la seconda componente dell'attributo D.*val*, rispettivamente).

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

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

Valutazione di SDD L-attribuite: stringhe binarie

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

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

Valutazione di SDD L-attribuite: stringhe binarie

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

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

Valutazione di SDD L-attribuite: stringhe binarie

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

```
function D( )
   var D_val
        if (cc = 0)
             D_val ← < 0,0>
             cc ← PROSS
        else if (cc = 1)
             D_val ← <1,0>
             cc ← PROSS
        else ERRORE (…)
        return (D_val)
```

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

Esercizi

1. Data la seguente grammatica:

$$N \rightarrow N0 \mid N1 \mid 0 \mid 1$$

   - Definire le regole semantiche per calcolare un attributo sintetizzato *.val*, che contenga la rappresentazione decimale del numero binario rappresentato da una qualunque stringa generata dalla grammatica;
   - Eliminare dalla grammatica le ricorsioni sinistre e per la grammatica ottenuta
     a) definire delle regole semantiche che realizzino la traduzione dalla rappresentazione binaria alla rappresentazione decimale.
     b) scrivere il traduttore a discesa ricorsiva.

2. Eliminare la ricorsione sinistra dalla seguente grammatica e costruire uno schema di traduzione sia per la grammatica data, sia per la grammatica ricorsiva a destra, che associ ad ogni stringa generata la coppia formata dal numero di "⋆" e dal numero di "#" in essa presenti.

$$S \rightarrow B \qquad B \rightarrow B\#$$
$$B \rightarrow A\# \qquad A \rightarrow \star A\#$$
$$A \rightarrow \varepsilon$$

   Scrivere il traduttore a discesa ricorsiva per la grammatica ricorsiva a destra.

Esercizi

3. Data la seguente grammatica:

$S \rightarrow SD$
$S \rightarrow D$
$D \rightarrow TL;$
$T \rightarrow int$
$T \rightarrow real$
$T \rightarrow char$
$L \rightarrow L, id$
$L \rightarrow id$

definire delle regole semantiche per calcolare, per ogni stringa generata, il numero di identificatori dichiarati “int”, “real” e “char”, rispettivamente:

$S.num = <n^o\ int, n^o\ real, n^o\ char>$