Grammatiche ad attributi

Una *grammatica ad attributi* è costituita dalle seguenti entità:

1) una grammatica non contestuale $G = \langle V_T, V_N, P, Z \rangle$
2) due insiemi disgiunti di simboli, detti rispettivamente insieme degli *attributi ereditati* ($\mathcal{E}_A$) ed insieme degli *attributi sintetizzati.* ($\mathcal{S}_A$), associati ad ogni simbolo A della grammatica (terminale o no).
   L'attributo $\underline{a}$ del simbolo A si indica con $\underline{a}.A$; si definisce $\mathcal{A}_A = \mathcal{E}_A \cup \mathcal{S}_A$
3) un insieme di valori ammissibili per ogni attributo $\underline{a}$, detto *dominio semantico* di $\underline{a}$ (D_a).
4) un insieme di regole semantiche associate ad ogni produzione. Data la produzione:
   $p: A_0 ::= A_1\ A_2\ \dots\ A_n$
   il valore dell'occorrenza $\underline{a}.A_k$ è calcolata mediante la funzione $f_a^{pk}(\underline{x}_1.A_{i_1}, \dots \underline{x}_m.A_{i_m})$.

Calcolo degli attributi

- se l'assioma Z ha attributi ereditati, questi devono avere un valore iniziale.
- se un simbolo terminale ha attributi sintetizzati, questi devono avere un valore iniziale.
- per ogni produzione $p: A_0 ::= A_1\ A_2\ \dots\ A_n$
  - esiste una regola $f_a^{p0}(\underline{x}_1.A_{i_1}, \dots \underline{x}_m.A_{i_m})$, $0 \le i_r \le n$, per ogni attributo sintetizzato $\underline{a}.A_0$.

- esiste una regola $f_a^{pk}(\underline{x}_1.A_{i_1}, \dots \underline{x}_m.A_{i_m})$, $0 \le i_r \le n$, per ogni attributo ereditato $\underline{a}.A_k$, $1 \le k \le n$.
- le occorrenze di attributi che compaiono nella parte sinistra di una regole sono dette *interne*: sia $\mathcal{I}^p$ l'insieme di tali occorrenze ($\mathcal{I}^p = \mathcal{S}_{A_0} \cup_{k=1} \cup^n \mathcal{E}_{A_k}$)
- le occorrenze di attributi che non compaiono nella parte sinistra di una regola sono dette *esterne*: sia $\mathcal{E}^p$ l'insieme di tali occorrenze ($\mathcal{E}^p = \mathcal{E}_{A_0} \cup_{k=1} \cup^n \mathcal{S}_{A_k}$)

Gli insiemi di regole definite per ogni produzione associano complessivamente ad ogni albero sintattico un insieme di *equazioni semantiche* , le cui incognite sono i valori degli attributi dei nodi.

Data una frase α si definisce *significato* di α la soluzione del sistema di equazioni semantiche dell'albero sintattico di α.

Dato l'albero sintattico di una frase, il sistema di equazioni semantiche ammette una ed una sola soluzione a condizione che:

1) ogni occorrenza di attributi sia definita da una ed una sola regola semantica.
2) non vi siano circolarità nelle definizioni
3) i valori iniziali degli attributi sintetizzati dei terminali e degli attributi ereditati dell'assioma siano noti.

Una grammatica ad attributi si dice *ben definita* se le condizioni precedenti sono verificate per ogni frase del linguaggio. In caso di mancanza di alcuni valori iniziali, si possono ottenere soluzioni parametriche in funzione dei valori mancanti.

Una grammatica sintatticamente ambigua produce, per una stessa frase, alberi sintattici diversi ciascuno con un proprio sistema di equazioni semantiche (ambiguità semantica .

Esempio : correttezza dei salti

programma corretto

```
begin label a
goto a
      begin
     a: x
      end
end
```

programma errato

```
begin
goto a
      begin label a
     a: x
      end
end
```

attributi:

a *boolean* : *true* se il programma è corretto {sint.}
d *set of label* : etichette dichiarate in un blocco {sint.}
v *set of label* : etichette visibili in un blocco {ereditato}
i *label* : identificatore di label {sint. con valore iniziale}

P ::= B	a.P := a.B v.B := Φ
B ::= **begin** D L **end**	a.B := a.L v.L := v.B ∪ d.D
B ::= **begin** L **end**	a.B := a.L v.L := v.B
D^0 ::= D^1 e	d.D^0::= d.D^1 ∪ i.e
D ::= **label** e	d.D::= i.e
L^0 ::= e : x L^1	a.L^0 := a. L^1 v.L^1 := v. L^0
L^0 ::= x L^1	a.L^0 := a. L^1 v.L^1 := v. L^0
L^0 ::= **goto** e L^1	a.L^0 := (i.e ∈ v. L^0) ∧ a. L^1 v.L^1 := v. L^0
L ::= B	a.L := a.B v.B := v.L
L ::= ε	a.L := *true*

Esempio: controllo dei tipi (espressioni aritmetiche)

Si considerano espressioni con valori interi e booleeani e operatori di somma (+), confronto (>) e congiunzione (∧).

attributi:

a *boolean* : *true* se il programma è corretto {sint.}
v *integer o boolean* : valore dell'espressione {sint.}
t { *b,i* } : tipo dell'espressione {sint.}

$E^0 ::= E^1 + E^2$	if $\underline{a}.E^1 \wedge \underline{a}.E^2 \wedge (\underline{t}.E^1 = i) \wedge (\underline{t}.E^2 = i)$ then $\underline{t}.E^0 := i$ $\underline{v}.E^0 := \underline{v}.E^1 + \underline{v}.E^2$ $\underline{a}.E^0 := true$ else $\underline{a}.E^0 := false$
$E^0 ::= E^1 \wedge E^2$	if $\underline{a}.E^1 \wedge \underline{a}.E^2 \wedge (\underline{t}.E^1 = b) \wedge (\underline{t}.E^2 = b)$ then $\underline{t}.E^0 := b$ $\underline{v}.E^0 := \underline{v}.E^1 \wedge \underline{v}.E^2$ $\underline{a}.E^0 := true$ else $\underline{a}.E^0 := false$
$E^0 ::= E^1 > E^2$	if $\underline{a}.E^1 \wedge \underline{a}.E^2 \wedge (\underline{t}.E^1 = i) \wedge (\underline{t}.E^2 = i)$ then $\underline{t}.E^0 := b$ $\underline{v}.E^0 := \underline{v}.E^1 > \underline{v}.E^2$ $\underline{a}.E^0 := true$ else $\underline{a}.E^0 := false$
$E ::= var$	$\underline{t}.E := \underline{t}.var$ $\underline{v}.E := \underline{v}.var$ $\underline{a}.E := true$

frase : $a > b \wedge c > d + e + f \wedge g$
lettura corretta : $(a > b) \wedge (c > (d + e + f)) \wedge g$
lettura errata : $((a > b) \wedge c) > (d + e + f) \wedge g$

Esempio : definizione scopi di variabili

attributi:

$\underline{e}$	*set of id*	:	insieme degli id dichiarati ed utilizzati nel blocco {ereditato}
$\underline{d}$	*id*	:	id della variabile dichiarata {sint.}
$\underline{o}$	*set of id*	:	insieme degli id dichiarati esternamente al blocco {ereditato}
$\underline{u}$	*set of id*	:	insieme degli id dichiarati interna nente al blocco {sint.}

Prog ::= Block	$\underline{e}$.Block := Φ
Block ::= Dlist Slist	$\underline{o}$.Dlist := $\underline{e}$.Block $\underline{e}$.Slist := $\underline{o}$.Dlist $\cup$ $\underline{u}$.Dlist
$Dlist^0 ::= id\ Dlist^1$	$\underline{u}.Dlist^0 := \underline{d}.id \cup \underline{u}.Dlist^1$ $\underline{o}.Dlist^1 := \underline{o}.Dlist^0$
Dlist ::= id	$\underline{u}$.Dlist := $\underline{d}$.id
$Slist^0 ::= Stat\ Slist^1$	$\underline{e}.Slist^1 := \underline{e}.Slist^0$ $\underline{e}$.Stat := $\underline{e}.Slist^0$
Slist ::= Stat	$\underline{e}$.Stat := $\underline{e}$.Slist
Stat ::= ex-stat	$\underline{e}$.ex-stat := $\underline{e}$.Stat
Stat::= begin Block end	$\underline{e}$.Block := $\underline{e}$.Stat

Dipendenze funzionali

Data la produzione p: $A_0 ::= A_1\ A_2 \ldots A_n$ e la regola semantica $f_a^{pk}(\underline{x}_1.A_{i_1}, \ldots \underline{x}_m.A_{i_m})$:

- l'insieme $D_a^{pk} = \{\underline{x}_1.A_{i_1} \ldots \underline{x}_m.A_{i_m}\}$ costituisce l'insieme delle *dipendenze funzionali* di $\underline{a}.A_k$;

- l'insieme dei nodi $\{\underline{a}.A_k, \underline{x}_1.A_{i_1}, \ldots, \underline{x}_m.A_{i_m}\}$ e degli archi $\{(\underline{x}_1.A_{i_1}, \underline{a}.A_k) \ldots (\underline{x}_m.A_{i_m}, \underline{a}.A_k)\}$ costituisce il grafo delle dipendenze funzionali G_a^{pk} di $\underline{a}.A_k$;

- il grafo G^p delle dipendenze funzionali della produzione p si ottiene fondendo i nodi omonimi di tutti i grafi parziali delle dipendenze dei singoli attributi.

Sia dato un albero sintattico T avente per radice il nodo A_0, con nodi figli $A_1, A_2, \ldots A_n$ (produzione p: $A_0 ::= A_1\ A_2 \ldots A_n$) a cui corrispondono rispettivamente i sottoalberi $T_1, T_2, \ldots T_n$; il grafo G_T delle dipendenze funzionali di T si ottiene (ricorsivamente) fondendo i nodi degli attributi di A_k in G^p, $1 \le k \le n$, con i nodi degli omonimi attributi della radice del sottoalbero T_k.

Un albero T è *aciclico* se tale è il suo grafo G_T. Una grammatica è aciclica se lo è ogni albero di ogni frase.

NOTA: gli algoritmi di verifica della aciclicità di una grammatica hanno complessità esponenziale.

Dipendenze approssimate

Sia dato un albero sintattico T avente per radice il nodo A_0, con nodi figli A_1, A_2, ... A_n (produzione p: $A_0 ::= A_1\ A_2 \ldots\ A_n$) a cui corrispondono rispettivamente i sottoalberi T_1, T_2, ... T_n . Se il grafo $\mathcal{G}_T$ delle dipendenze di T presenta un percorso dal nodo $\underline{a}.A_0$ al nodo $\underline{s}.A_0$ ($\underline{s}.A_0$ è sintetizzato), l'arco $(\underline{a}.A_0, \underline{s}.A_0) \in \mathcal{G}_T^{IO}$ (*grafo di ingresso- scita* dell'albero T).

Per ogni nonterminale A, si definisce l'insieme $\mathcal{Tg}_A^{IO} = \{\mathcal{G}_T^{IO} \mid T$ è un albero sintattico con radice A$\} = \{\mathcal{G}_{T_1}^{IO}, \mathcal{G}_{T_2}^{IO}, \ldots \mathcal{G}_{T_n}^{IO}\}$.

Per ogni nonterminale A si definisce il *grafo approssimato di ingresso-uscita* $\mathcal{Ga}_A^{IO}$ ottenuto fondendo i nodi omonimi dei grafi dell'insieme $\mathcal{Tg}_A^{IO}$. Si avrà $\mathcal{Ga}_A^{IO} \supseteq \mathcal{G}_T^{IO}$ per ogni $\mathcal{G}_T^{IO} \in \mathcal{Tg}_A^{IO}$.

Dato il grafo $\mathcal{G}^p = (N^p, E^p)$, si definisce *grafo approssimato delle dipendenze* della produzione p: $A_0 ::= A_1, A_2, \ldots\ A_n$ il grafo $\mathcal{Ga}^p = (Na^p, Ea^p)$ tale che:

- $Na^p = N^p$;
- $Ea^p = E^p \cup \{(\underline{a}.A_i, \underline{s}.A_i) \mid (\underline{a}.A_i, \underline{s}.A_i) \in \mathcal{Ga}_{A_i}^{IO},\ 1 \le i \le n\}$

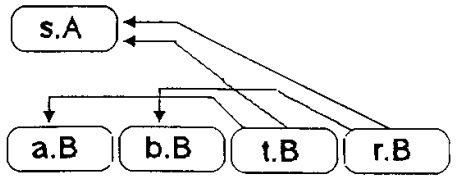
Una grammatica si dice *assolutamente aciclica* se e solo se, per ogni produzione p: $A_0 ::= A_1\ A_2 \ldots\ A_n$, il grafo $\mathcal{Ga}^p$ è aciclico.

Se una grammatica è assolutamente aciclica, allora è aciclica.

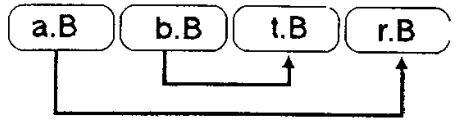
NOTA: la valutazione della aciclicità assoluta può essere fatta con complessità polinomiale.

Esempio : dipendenze approssimate

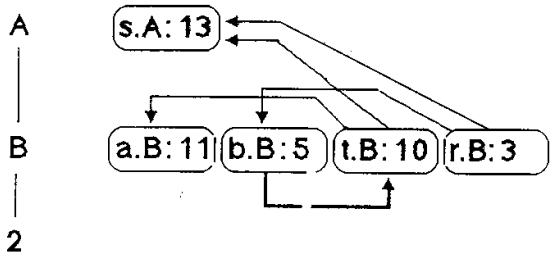
A :: = B	$\underline{s}.A := \underline{t}.B + \underline{r}.B$ $\underline{a}.B := \underline{t}.B + 1$ $\underline{b}.B := \underline{r}.B + 2$
B ::= 2	$\underline{t}.B := 2 * \underline{b}.B$ $\underline{r}.B := 3$
B ::= 3	$\underline{r}.B := 3 * \underline{a}.B$ $\underline{t}.B := 4$



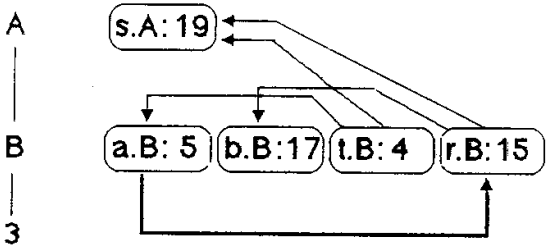
$\mathcal{G}^p$



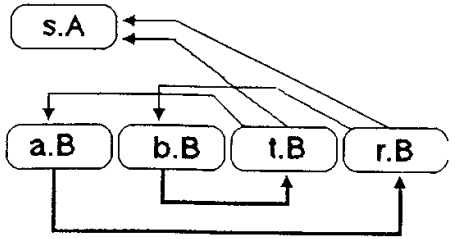
$\mathcal{G}a_B^{IO}$



$\mathcal{G}_{T_1}^{IO}$



$\mathcal{G}_{T_2}^{IO}$



$\mathcal{G}a^{P}$

Valutazione degli attributi

Dato l'albero sintattico T, sia $\mathcal{A}_T$ l'insieme delle variabili (attributi) del suo sistema di equazioni semantiche. Se la grammatica è aciclica, la valutazione degli attributi richiede un ordinamento totale su $\mathcal{A}_T$, che rispetti le relazioni di dipendenza funzionale descritte in $\mathcal{G}_T$.

Classi di strategie:

- *schedulazione fissa* : la visita dei nodi di T avviene in un ordine prestabilito indipendente dalla grammatica
- *schedulazione statica* : la visita dei nodi di T avviene in un ordine dipendente dalla grammatica, ma non da T, determinato prima dell'analisi semantica mediante l'esame delle dipendenze approssimate fra gli attributi.
- *schedulazione dinamica* : la visita dei nodi di T avviene in un ordine dipendente da T, determinato dinamicamente nel corso dell'analisi semantica.

Schedulazione fissa con una sola passata discendente

Condizioni sufficienti sulla struttura delle regole semantiche (grammatica con dipendenze di tipo L). Per ogni produzione p: $A_0 ::= A_1\ A_2 \ldots A_n$:

1) Esiste una regola $f_a^{p0}(\underline{x}_1.A_{i_1}, \ldots \underline{x}_m.A_{i_m})$, $0 \le i_r \le n$, per ogni attributo sintetizzato $\underline{a}.A_0$. Gli attributi di A_0 presenti in $\mathcal{D}_a^{p0}$ devono essere *ereditati*.
$$\mathcal{D}_a^{p0} \cap \mathcal{S}_{A_0} = \Phi$$
2) Esiste una regola $f_a^{pk}(\underline{x}_1.A_{i_1}, \ldots \underline{x}_m.A_{i_m})$, $0 \le i_r < k$, per ogni attributo ereditato $\underline{a}.A_k$, $1 \le k \le n$. Gli attributi di A_0 presenti in $\mathcal{D}_a^{pk}$ devono essere *ereditati*.
$$\mathcal{D}_a^{pk} \cap \{\mathcal{S}_{A_0} \cup {}_{i=k}\cup^{n} \mathcal{A}_{A_i}\} = \Phi$$

Algoritmo di valutazione ricorsivo (a stack implicito)

Per ogni simbolo nonterminale A, definito dall'insieme di produzioni:

$A ::= b_{p_1 1} \ldots b_{p_1 n_1} \mid b_{p_2 1} \ldots b_{p_2 n_2} \mid b_{p_m 1} \ldots b_{p_m n_m}$

definire una procedura ricorsiva

```
procedure A(in ℰ_A ; out 𝒮_A);
 [ si dichiara una variabile locale per ogni a ∈ 𝒮_A e per
   ogni a ∈ ℰ_x, per ogni x ∈ {b_pknk} ]
 begin case input of
    𝒢(A::= b_p11 ... b_p1n1):
    begin
     for k := 1 to n1 do begin
      if b_p1k ∈ V_N
        then begin {per ogni a ∈ ℰ_x , x = b_p1k}
                a = f_a^1k( ... );
                ...
                P_p1k(in ℰ_x , out 𝒮_x )
             end;
        else if input = b_p1k
                then input := NEXT
                else ERRORE
     end;
     {per ogni a ∈ 𝒮_A}  a = f_a^p1k( ... );
    end;
    ......
    𝒢(A::= B_pm1 ... B_pmnm):
    begin
     ......
    end;
  end; {case}
end; {A}
```

Schedulazione fissa con una sola passata ascendente (analisi LR)

Una grammatica è nella *forma normale di Bochmann* se, per ogni produzione, gli attributi interni dipendono solo da attributi esterni: $\mathcal{D}_a^{pk} \cap \mathcal{T}^p = \Phi$, $\forall$ a, k, p.

Condizioni sufficienti sulla struttura delle regole semantiche (grammatica con dipendenze di tipo S):

1) non esistono attributi ereditati;
2) la grammatica è in forma normale di Bochmann.

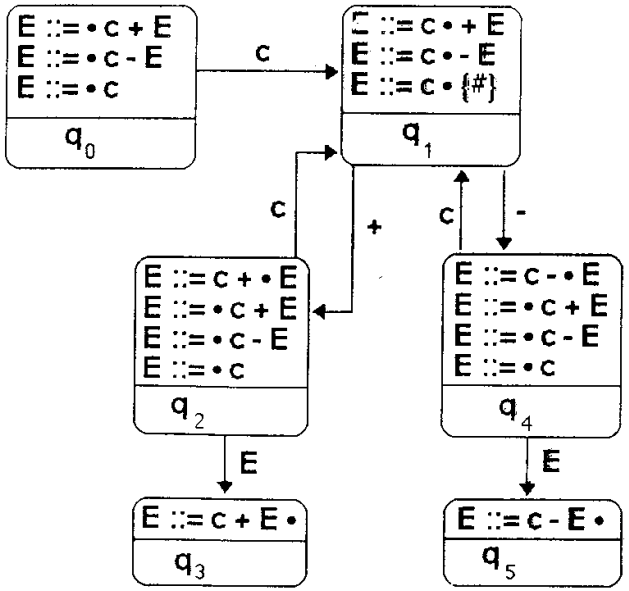
NOTA: la rinuncia agli attributi ereditati non riduce la capacità espressiva di una grammatica.

Un analizzatore LR(k) [SLR(k), LARL(k)] con attributi si ottiene modificando un automa $\mathcal{A}_{LR(k)}$ [$\mathcal{A}_{SLR(k)}$, $\mathcal{A}_{LARL(k)}$] nel modo seguente:

- ad ogni simbolo A nello stack si associa l'insieme ordinato dei valori dei suoi attributi $\mathcal{S}_A$ (componente semantica). La componente semantica viene inserita o rimossa contemporaneamente alla componente sintattica (coppia {simbolo,stato})
- in corrispondenza della transizione $\langle q_0, a\alpha, h_i\gamma\rangle$ |- $\langle q_0, \alpha, h_k a h_i \gamma\rangle$ (dove $h_i \in Q_p$ è uno stato di spostamento e $t_p(h_i,a) = h_k$) si inseriscono di valori di $\mathcal{S}_a$

Esempio :grammatica SLR(1) con dipendenze di tipo S

$E^0 ::= c + E^1$	$\underline{a}.E^0 := [(\underline{v}.c + \underline{v}.E^1) > min] \wedge [(\underline{v}.c + \underline{v}.E^1) < max] \wedge \underline{a}.E^1$ $\underline{v}.E^0 := \underline{v}.c + \underline{v}.E^1$
$E^0 ::= c - E^1$	$\underline{a}.E^0 := [(\underline{v}.c - \underline{v}.E^1) > min] \wedge [(\underline{v}.c - \underline{v}.E^1) < max] \wedge \underline{a}.E^1$ $\underline{v}.E^0 := \underline{v}.c - \underline{v}.E^1$
$E ::= c$	$\underline{v}.E := \underline{v}.c$ $\underline{a}.E := true$



- in corrispondenza della transizione $\langle q_0, \alpha, h_{i_n} b_n \ldots h_{i_1} b_1 h_{i_0} \rangle$ $\vdash \langle q_0, \alpha, h_k A h_{i_0} \rangle$ (dove $h_{i_n} \in Q_p$ è uno stato di riduzione con candidata $\langle A ::= b_1 b_2 \ldots b_n^{\nabla} \rangle$ $(A \neq Z)$ e $t_p(h_{i_0}, A) = h_k$) si calcolano di valori di $\mathcal{S}_A$.
- in corrispondenza della transizione $\langle q_0, \varepsilon, h_{i_n} b_n \ldots h_{i_1} b_1 Z_0 \rangle$ $\vdash \langle q_F, \varepsilon, Z_0 \rangle$ (dove $h_{i_n} \in Q_p$ è uno stato di riduzione con candidata $\langle Z ::= b_1 b_2 \ldots b_n^{\nabla} \rangle$ e Z è l'assioma di G) si calcolano di valori di $\mathcal{S}_Z$.

Schedulazione fissa con passate multiple

Per ridurre la complessità dei domini degli attributi può essere necessario rinunciare ad ottenere dipendenze di tipo L o S e calcolare i valori degli attributi con più visite dell'albero sintattico.

Si ha *schedulazione fissa* quando l'ordine delle visite è definito a priori e non dipende dal particolare albero sintattico.

Strategie possibili:

- passate discendenti destrorse
- passate discendenti alternativamente destrorse e sinistrorse
- passate discendenti e ascendenti, destrorse e sinistrorse, variamente alternate.

Esempio : definizione scopi di variabili (1) .

In ogni blocco le dichiarazioni delle variabili devono precedere gli statement eseguibili; la grammatica ha dipendenze di tipo L

attributi:

e	*set of id*	:	insieme degli id dichiarati ed utilizzati nel blocco {ereditato}
d	*id*	:	id della variabile dichiarata {sint.}
o	*set of id*	:	insieme degli id dichiarati esternamente al blocco {ereditato}
u	*set of id*	:	insieme degli id dichiarati internamente al blocco {sint.}

Prog ::= Block	e.Block := Φ
Block ::= Dlist Slist	o.Dlist := e.Block e.Slist := e.Block ∪ u.Dlist
$Dlist^0$ ::= id $Dlist^1$	$u.Dlist^0$:= d.id ∪ $u.Dlist^1$ $o.Dlist^1$:= $o.Dlist^0$
Dlist ::= id	u.Dlist := d.id
$Slist^0$ ::= Stat $Slist^1$	$e.Slist^1$:= $e.Slist^0$ e.Stat := $e.Slist^0$
Slist ::= Stat	e.Stat := e.Slist
Stat ::= ex-stat	e.ex-stat := e.Stat
Stat::= begin Block end	e.Block := e.Stat

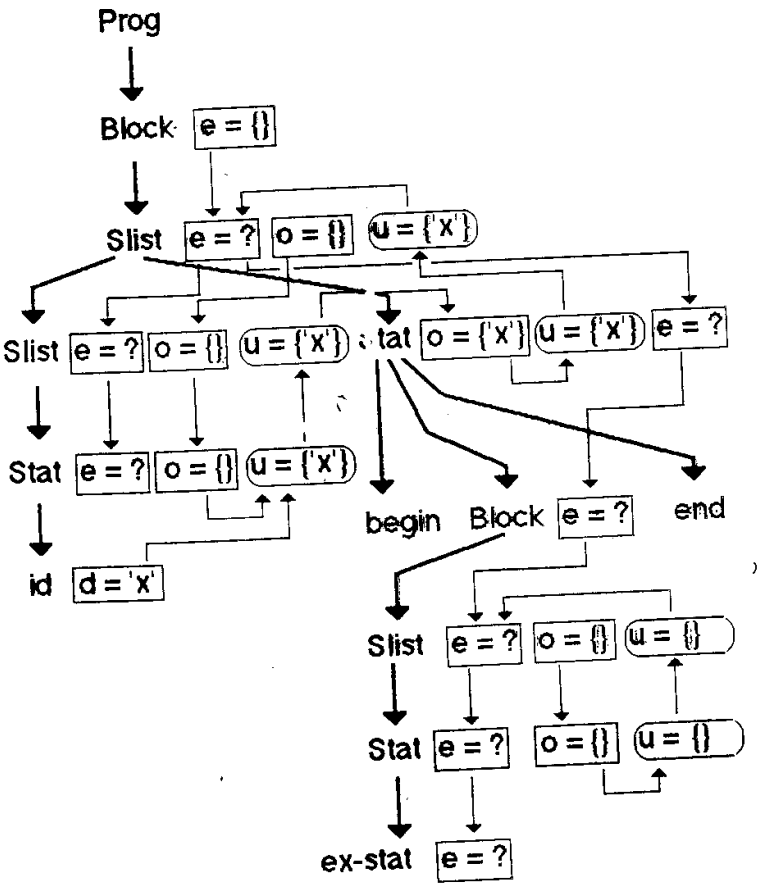
NOTA: la grammatica non è LL(k)

Esempio : definizione scopi di variabili (2) .

In ogni blocco le dichiarazioni delle variabili poss
essere frammiste agli statement eseguibili
dipendenze non sono di tipo L (vd. **)

1° passata DD: si valutano o e u
2° passata DD: si valuta e

Prog ::= Block	e.Block := Φ
Block ::= Slist	o.Slist := Φ e.Slist := e.Block ∪ u.Sli **
Slist^0 ::= Slist^1 Stat	u.Slist^0 := u.Stat e.Slist^1 := e.Slist^0 o.Slist^1 := o.Slist^0 e.Stat := e.Slist^0 o.Stat := u.Slist^1
Slist ::= Stat	e.Stat := e.Slist o.Stat := o.Slist u.Slist := u.Stat
Stat ::= id	u.Stat := o.Stat ∪ d.id
Stat ::= ex-stat	u.Stat := o.Stat e.ex-stat := e.Stat
Stat::= begin Block end	u.Stat := o.Stat e.Block := e.Stat

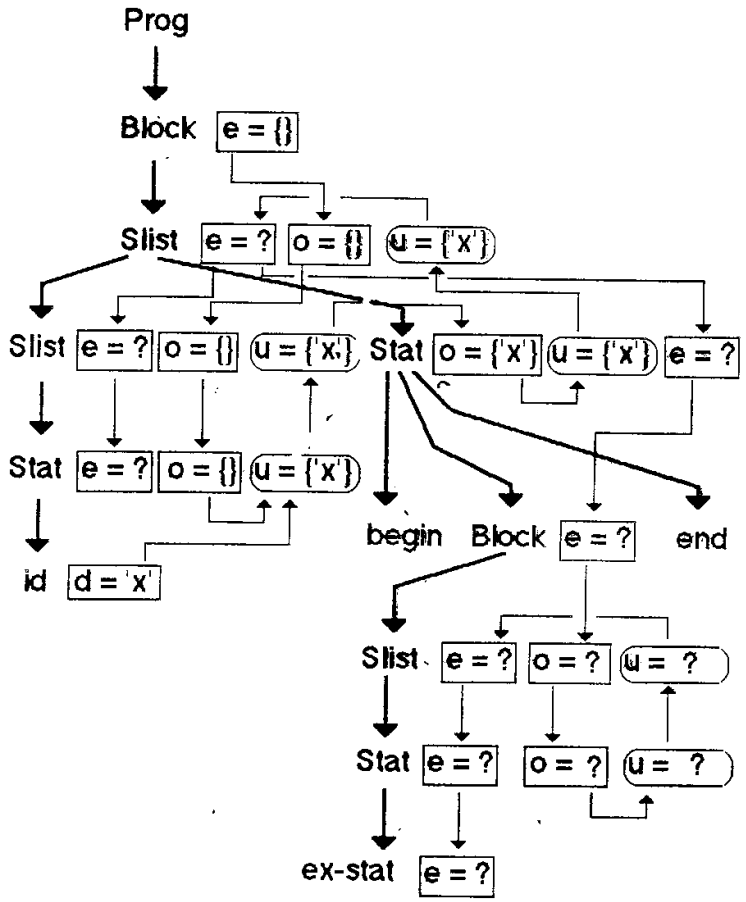


Situazione al termine della prima passata. Qualunque albero può essere valutato in due passate.

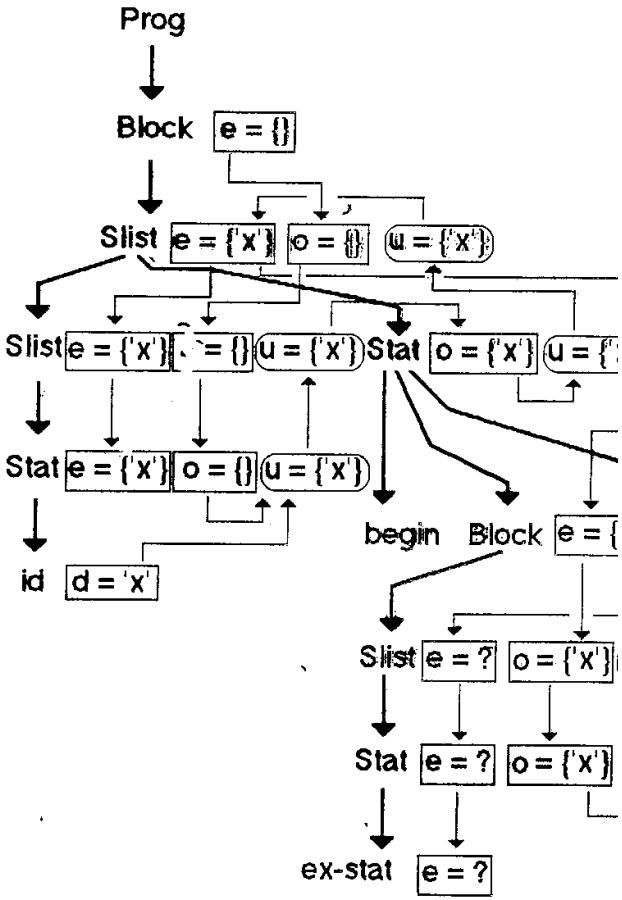
Esempio : definizione scopi di variabili (3) .

In ogni blocco le dichiarazioni delle variabili possono essere frammiste agli statement eseguibili; le dipendenze non sono di tipo L (vd. **)

Prog ::= Block	e.Block := Φ
Block ::= Slist	o.Slist := e.Block e.Slist := u.Slist ****
$Slist^0$ ::= $Slist^1$ Stat	u.$Slist^0$:= u.Stat e.$Slist^1$:= e.$Slist^0$ o.$Slist^1$:= o.$Slist^0$ e.Stat := e.$Slist^0$ c.Stat := u.$Slist^1$
Slist ::= Stat	e.Stat := e.Slist o.Stat := o.Slist u.Slist := u.Stat
Stat ::= id	u.Stat := o.Stat ∪ d.id
Stat ::= ex-stat.	u.Stat := o.Stat e.ex-stat := e.Stat
Stat::= begin Block end	u.Stat := o.Stat e.Block := e.Stat



Situazione al termine della prima passata.



Situazione al termine della seconda pas
numero delle passate necessarie per la val
degli attributi dipende dal numero di blocchi ar

Schedulazione statica

Sia G una grammatica assolutamente aciclica; un algoritmo di schedulazione statica (dipendente dalla struttura delle produzioni e delle regole semantiche ma non dal particolare albero sintattico) può essere costruito nel modo seguente:

- per ogni produzione p: $A_0 ::= A_1\ A_2 \ldots\ A_n$ si definisce il grafo approssimato $\mathcal{Ga}^p = (N^p, E^p)$. Per ogni attributo sintetizzato $\underline{s}.A_0$ si costruisce la procedura H_s^p(**in** $\mathcal{E}_{A_0}, T_{A_0}$; **out** $\underline{s}.A_0$) nel modo seguente [T_{A_0} è il (sotto)albero sintattico corrente di radice A_0]:
  1) si cancellano i nodi e gli archi di $\mathcal{Ga}^p$ che non appartengono a percorsi che portano a $\underline{s}.A_0$; sia $\mathcal{Ga}^p(s) = (N_s^p, E_s^p)$ tale grafo.
  2) si costruisce l'insieme $V_s^p = N_s^p - \mathcal{E}_{A_0}$.
  3) si ordinano i nodi di V_s^p secondo un ordinamento "$<_s^p$" compatibile con $\mathcal{Ga}^p(s)$ (ne esiste almeno uno perchè il grafo $\mathcal{Ga}^p$ è aciclico), tale che:

     se $(a,b) \in E_s^p$ allora $a <_s^p b$

4) per ogni attributo $\underline{a} \in V_s^p$ [non ancora valutato], procedendo secondo l'ordinamento "$<_s^p$":
   - se $\underline{a} \in \mathcal{E}_{A_k}$, $1 \le k \le n$ oppure $\underline{a} = \underline{s}.A_0$ si calcola il valore mediante la funzione $f_a^{pk}(\underline{x}_1.A_{i_1}, \ldots \underline{x}_m.A_{i_m})$.
   - se $\underline{a} \in \mathcal{S}_{A_k}$, $1 \le k \le n$ si calcola il valore mediante chiamata alla procedura (definita nel seguito) R_{a,A_k}(**in** $\mathcal{E}_{A_k}$,T_{A_k}; **out** $\underline{a}.A_k$).

- per ogni simbolo non terminale A e per ogni attributo sintetizzato $\underline{s}.A$ si costruisce la procedura $R_{s,A}$(**in** $\mathcal{E}_A$,T_A; **out** $\underline{s}.A$) nel modo seguente:
  1) seleziona la produzione p utilizzata per riscrivere il simbolo A [radice del (sotto)albero T_A]
  2) chiama la procedura H_s^p(**in** $\mathcal{E}_A$,T_A; **out** $\underline{s}.A$)

Schemi di traduzione

Uno *schema di traduzione ad attributi con azioni semantiche* (più brevemente *schema ad attributi*) è costituito da una grammatica di traduzione ad attributi $G_t = \langle V_{T1} \cup V_{T2}, V_N, P_t, Z \rangle$, dove gli attributi sintetizzati dei simboli terminali della grammatica oggetto possono essere funzione di altri attributi dello stesso simbolo:

- data la produzione p: $A_0 ::= A_1\ A_2\ \ldots\ A_n$ per ogni $\underline{a} \in \mathcal{S}_{A_k}$, $A_k \in V_{T2}$, $1 \le k \le n$, $\mathcal{A}_{A_k} \supseteq \mathcal{D}_a^{pk}$

Esempio

A ::= var ← E *store*	i.*store* := ind.var j.*store* := temp.E
$E^0 ::= E^1 + E^2$ *add*	i.*add* := temp.E^1 j.*add* := temp.E^2 k.*add* := *nuovacella* temp.E^0:= k.*add*
$E^0 ::= E^1 * E^2$ *mult*	i.*mult* := temp.E^1 j.*mult* := temp.E^2 k.*mult* := *nuovacella* temp.E^0:= k.*mult*
E ::= var	temp.E := ind.var
$E^0 ::= (E^1)$	temp.E^0:= temp.E^1

La stringa terminale:

v[7] ← v[5] + v[2] * v[3] *mult*[2,3,t_{200}] *add*[5,t_{200},t_{201}] *store*[7,t_{201}]

traduce la frase "a ← b + c * d" (dove le variabili a,b,c,d sono rispettivamente nei registri 7,5,2,3) nella sequenza di istruzioni-macchina:

mult 2,3,t_{200}
add 5,t_{200},t_{201}
store 7,t_{201}

Valutazione discendente con automa a stack esplicito

Uno schema ad attributi $G_t = \langle V_{T1} \cup V_{T2}, V_N, P_t, Z\rangle$ di tipo L è in forma di *assegnamenti semplici* se, per ogni produzione p: $A_0 ::= A_1\ A_2\ \ldots\ A_n$, valgono le seguenti condizioni:

1) per ogni attributo ereditato $\underline{a}.A_k$, $A_k \in V_{T1} \cup V_N$, $1 \le k \le n$, e per ogni attributo sintetizzato $\underline{a}.A_0$, $f_a^{pk}(\underline{x}_1.A_{i_1}, \ldots. \underline{x}_m.A_{i_r})$ è nella forma "$\underline{x}.A_r$" (i.e. il calcolo di $\underline{a}.A_k$ si effettua mediante una *regola di copiatura* $\underline{a}.A_k = \underline{x}.A_r$).
2) Limitatamente all'insieme delle regole di copiatura, la grammatica è in forma normale di Bochmann.

Automa $\mathcal{A}_{LL(1)}(G)$ per schema G ad attributi di tipo L-semplice

Ogni posizione dello stack contiene una coppia $\langle A, L_A\rangle$, dove $A \in V_{T1} \cup V_{T2} \cup V_N$ e L_A è una lista ordinata di campi associati agli attributi di A:

- per ogni attributo sintetizzato il campo contiene un puntatore alla posizione dello stack dove l'attributo sarà ricopiato.
- per ogni attributo ereditato il campo contiene il valore.

Esempio

Schema G_1 : non di tipo L-semplice

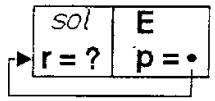
S ::= E *sol*	$\underline{r}.sol := \underline{p}.E$
E^0 ::= + E^1 E^2	$\underline{p}.E^0 := \underline{p}.E^1 + \underline{p}.E^2$
E^0 ::= * E^1 E^2	$\underline{p}.E^0 := \underline{p}.E^1 * \underline{p}.E^2$
E ::= cost	$\underline{p}.E := \underline{q}.cost$

Schema equivalente G_2 : i tipo L-semplice

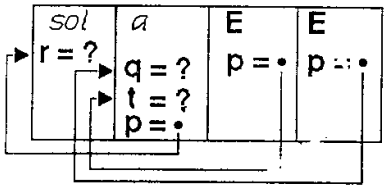
S ::= E *sol*	$\underline{r}.sol := \underline{p}.E$
E^0 ::= + E^1 E^2 *a*	$\underline{p}.E^0 := \underline{p}.a$ $\underline{p}.a := \underline{q}.a + \underline{t}.a$ $\underline{q}.a := \underline{p}.E^1$ $\underline{t}.a := \underline{p}.E^2$
E^0 ::= * E^1 E^2 *m*	$\underline{p}.E^0 := \underline{p}.m$ $\underline{p}.m := \underline{q}.m * \underline{t}.m$ $\underline{q}.m := \underline{p}.E^1$ $\underline{t}.m := \underline{p}.E^2$
E ::= cost	$\underline{p}.E := \underline{q}.cost$

S

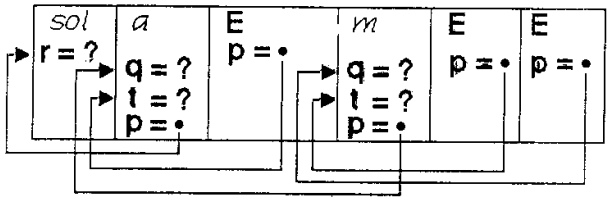
+ * cost[2] cost[7] cost[5]



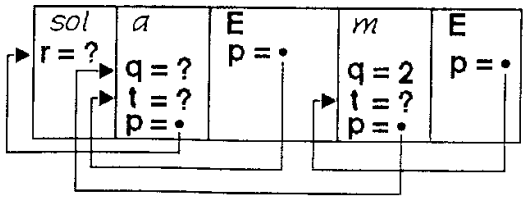
+ * c[2] c[7] c[5]



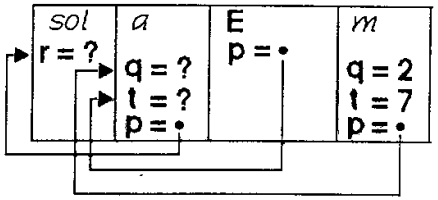
* c[2] c[7] c[5]



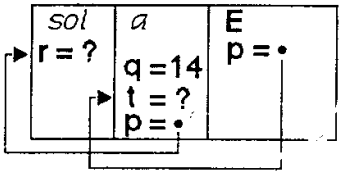
c[2] c[7] c[5]



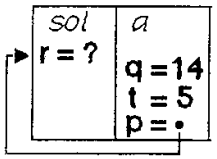
c[7] c[5]

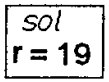


c[5]



c[5]





Stampa r = 19

Analisi sintattica guidata dalla semantica

I predicati semantici possono essere usati per disambiguare strutture sintattiche.

Esempio

P_head ::= P_id "procedure" (Fpl)	e.Fpl := e.P_head r.P_head := r.Fpl
Fpl ::= F_par X	e.X := e.Fpl r.Fpl := r.F_par ∧ r.X
X^0 ::= F_par X^1 | ε [id] [)]	e.X^1 := e.X^0 e.F_par := e.X^0 r.X^0 := r.F_par ∧ r.X^1
F_par ::= L_id Type_id	e.L_id := e.F_par e.Type_id := e.F_par r.F_par := r.L_id
L_id ::= id Y	e.Y := e.L_id r.L_id := (n.id ∉ e.L_id)
Y^0::= id Y^1 | ε [id] [id] *** [t.Y^0 = *true*] [t.Y^0 = *false*]	e.Y^1 := e.Y^0 t.Y^0 := (n.id ∉ e.Y^0)
Type_id ::= id	