Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

prof. S. Crespi Reghizzi, prof. Angelo Morzenti
(prof. Luca Breveglieri)

Prova scritta - 5 febbraio 2008 - Parte I: Teoria

CON SOLUZIONI - a scopo didattico le soluzioni sono molto estese e commentate variamente - non si richiede che il candidato svolga il compito in modo altrettanto ampio, bensì che risponda in modo appropriato e a suo giudizio ragionevole

NOME:

COGNOME:

MATRICOLA: FIRMA:

ISTRUZIONI - LEGGERE CON ATTENZIONE:

- L'esame si compone di due parti:
  - I (80%) Teoria:
    1. espressioni regolari e automi finiti
    2. grammatiche libere e automi a pila
    3. analisi sintattica e parsificatori
    4. traduzione sintattica e analisi semantica
  - II (20%) Esercitazioni Flex e Bison
- Per superare l'esame l'allievo deve avere sostenuto con successo entrambe le parti (I e II), in un solo appello oppure in appelli diversi, ma entro un anno.
- Per superare la parte I (teoria) occorre dimostrare di possedere conoscenza sufficiente di tutte le quattro sezioni (1-4).
- È permesso consultare libri e appunti personali.
- Per scrivere si utilizzi lo spazio libero e se occorre anche il tergo del foglio; è vietato allegare nuovi fogli o sostituirne di esistenti.
- Tempo: Parte I (teoria): 2h.30m - Parte II (esercitazioni): 45m

1 Espressioni regolari e automi finiti 20%

1. Sono date le due espressioni regolari R_1 e R_2 seguenti:

$$R_1 = (a \mid b)^* \; b \; (a \mid b \mid c)^*$$

$$R_2 = (a \mid c)^* \; b \; (a \mid b)^*$$

entrambe definite sull'alfabeto $\Sigma = \{a, b, c\}$, che definiscono i linguaggi regolari L_1 e L_2, rispettivamente.

Si risponda alle domande seguenti:

(a) Si indichino le quattro stringhe più corte appartenenti ai linguaggi differenza $L_1 \setminus L_2$ e $L_2 \setminus L_1$. Per rispondere si compili la tabella riportata di seguito.

(b) Si costruisca l'automa riconoscitore del linguaggio differenza $L_1 \setminus L_2$ (a scelta se indeterministico o deterministico).

(c) Se necessario, si renda deterministico e minimo l'automa costruito al punto precedente

---

#	$L_1 \setminus L_2$	$L_2 \setminus L_1$
1		
2		
3		
4		

Soluzione

(a) Valgono le osservazioni seguenti:

i. le stringhe di L_1 e L_2 contengono tutte almeno una lettera b

ii. nelle stringhe di L_1 a sinistra delle lettere c (se ce ne sono) deve figurare almeno una lettera b

iii. nelle stringhe di L_2 tutte le lettere c (se ce ne sono) stanno a sinistra di tutte le lettere b

iv. le stringhe di L_1 e L_2 hanno una distribuzione di lettere a del tutto libera, e libera anche di lettere b e c ma vincolatamente a quanto detto prima

A motivo dei punti (ii) e (iii), i linguaggi L_1 e L_2 non hanno in comune nessuna stringa contenente almeno una lettera c (o più di una). Al massimo L_1 e L_2 potrebbero condividere le stringhe non contenenti lettere c. Si vede subito che L_1 e L_2 hanno effettivamente in comune tutte (e sole) le stringhe che non contengono lettere c, perché le due espressioni regolari ridotte (ottenute cancellando le c):

$$R'_1 = (a \mid b)^* \; b \; (a \mid b)^*$$

$$R'_2 = a^* \; b \; (a \mid b)^*$$

sono palesemente equivalenti (R'_1 è ambigua come anche R_1). Pertanto si ha:

- $L_1 \setminus L_2$ = tutte le stringhe di L_1 che contengono almeno una lettera c
- $L_2 \setminus L_1$ = tutte le stringhe di L_2 che contengono almeno una lettera c

In conclusione, ecco le prime 7 e 6 stringhe più corte dei linguaggi differenza (per risolvere l'esercizio ne bastano 4 a scelta):

#	$L_1 \setminus L_2$	$L_2 \setminus L_1$
1	b c	c b
2	a b c	c b a
3	b a c	c b b
4	b b c	c a b
5	b c a	a c b
6	b c b	c c b
7	b c c	

Si noti che le due colonne di stringhe sono disgiunte, come deve essere.

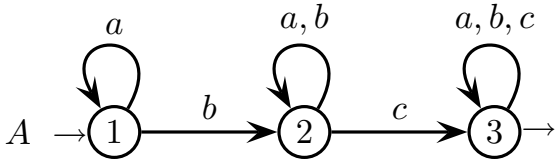
(b) Un'espressione regolare R_3 facile da ottenere modificando l'espressione regolare R_1, e che genera il linguaggio differenza $L_1 \setminus L_2$, è la seguente (l'espressione R_3 forza la presenza di almeno una lettera c):

$$R_3 = (a \mid b)^* \; b \; (a \mid b \mid c)^* \; c \; (a \mid b \mid c)^*$$

Tuttavia R_3 è ambigua perché lo è già in partenza R_1, pertanto è preferibile l'espressione regolare R_4 seguente, che è equivalente a R_3 ma non è ambigua (le lettere b e c forzate da R_4 sono necessariamente le più a sinistra):

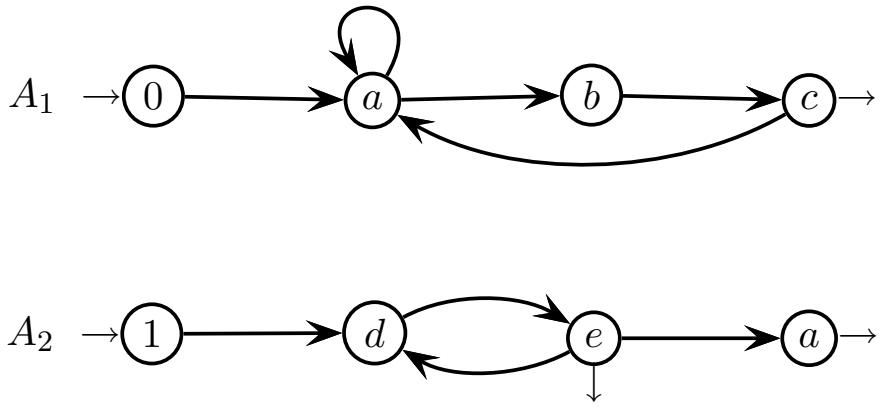
$$R_4 = a^* \; b \; (a \mid b)^* \; c \; (a \mid b \mid c)^*$$

Ecco l'automa A corrispondente all'espressione R_4 (ottenuto tramite la costruzione di Tomphson o modulare un po' semplificata):



(c) L'automa A è già deterministico. Esso è anche evidentemente minimo: lo stato 3 è finale e pertanto distinguibile dagli stati 1 e 2 (non finali), e gli stati 1 e 2 sono distinguibili perché 1 non ha arco c uscente.

2. Sono dati i due automi locali A_1 e A_2 seguenti:



sugli alfabeti di ingresso $\Sigma_1 = \{a, b, c\}$ e $\Sigma_2 = \{a, d, e\}$, rispettivamente.

Si risponda alle domande seguenti:

(a) Si determinino le due espressioni regolari R_1 e R_2 che generano i linguaggi $L(A_1)$ e $L(A_2)$, rispettivamente.

(b) Si verifichi se i linguaggi concatenamento:

$$L_C = L(A_2) \cdot L(A_1)$$

e unione

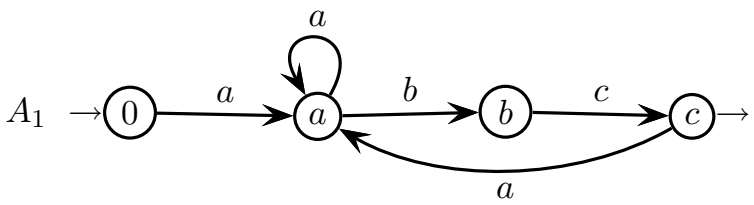
$$L_U = L(A_1) \cup L(A_2)$$

siano anch'essi linguaggi locali oppure no (si noti che il linguaggio L_C concatena prima $L(A_2)$ e poi $L(A_1)$).

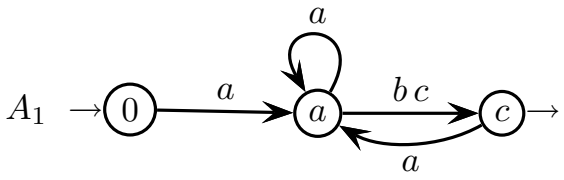
Soluzione

(a) Si procede tramite il metodo di eliminazione dei nodi (Brozozowski). Per semplificare il metodo, non si mettono stati iniziale e finale unici aggiuntivi. Prima di procedere si riportano le lettere sugli archi entranti negli stati.

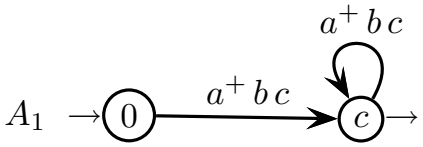
Per l'automa A_1:



Elimina nodo b:



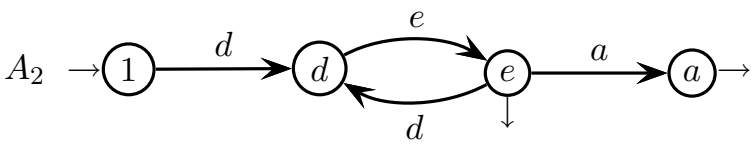
Elimina nodo a:



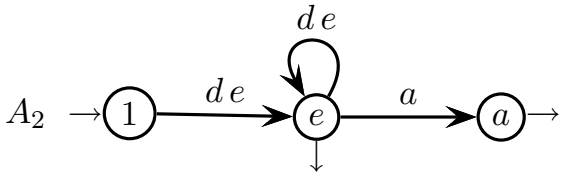
È evidente che l'espressione regolare che genera $L(A_1)$ è la seguente:

$$R_1 = (a^+\ b\ c)^+$$

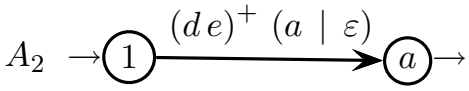
Per l'automa A_2:



Elimina nodo d:



Elimina nodo e (tenendo conto che è finale):

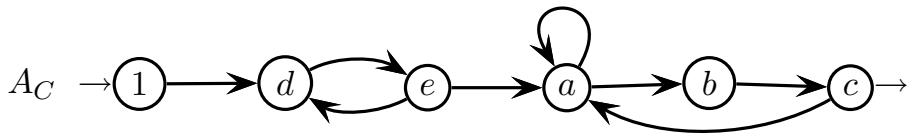


Si ha che l'espressione regolare che genera $L(A_2)$ è la seguente:

$$R_2 = (d\ e)^+\ \ (a\ \mid\ \varepsilon)$$

(b) In generale la famiglia dei linguaggi locali non è chiusa né rispetto al concatenamento né rispetto all'unione. Per linguaggi specifici la chiusura talvolta vale comunque. Qui la situazione è la seguente.

Il linguaggio concatenamento L_C è locale. Ecco l'automa locale A_C che lo riconosce:



Tale automa A_C si caratterizza per i digrammi $d\,e$, $e\,d$, $e\,a$, $a\,a$, $a\,b$, $b\,c$ e $c\,a$, e per lettere iniziale e finale d e c, rispettivamente. Il linguaggio L_C risulta locale perché il linguaggio L_2, che nel concatenamento figura a sinistra, ha lettera finale a, e il linguaggio L_1, che nel concatenamento figura a destra, ammette il digramma $a\,a$ e ha lettera iniziale a, ovvero può iniziare con un numero qualsiasi di lettere a, almeno una. Formalmente si ha: se $u\,a \in L(A_2)$ e $a^h\,v \in L(A_2)$ $(h \geq 1)$, allora $u\,a\,a^h\,v = u\,a^k\,v \in L_C$ $(k \geq 2)$. Si possono pertanto concatenare i due automi A_2 e A_1, mettendo in comune lo stato a (lasciando l'autoanello), e l'automa risultante è di tipo locale. Dunque L_C è un linguaggio locale.

Invece il linguaggio unione L_U non è locale. Infatti il linguaggio L_1 ammette il digramma $a\,a$ e il linguaggio L_2 non lo ammette, ma L_2 ammette la lettera a come carattere finale purché però sia una sola. Per unire i due linguaggi L_1 e L_2 pretendendo che il linguaggio risultante L_U sia ancora descrivibile in modo locale, cioè specificando i digrammi ammissibili e le lettere iniziali e finali, si devono permettere sia il digramma $a\,a$ sia la lettera finale a a motivo delle stringhe di L_1 e di L_2, rispettivamente. Ma allora inevitabilmente le stringhe di L_2 potrebbero avere più di una lettera a finale, e tali stringhe aggiuntive sono estranee a L_U. Pertanto non esiste un automa locale che riconosca il linguaggio L_U, il quale dunque non è un linguaggio locale. Naturalmente L_U è pur sempre un linguaggio della famiglia regolare, ma non della sottofamiglia locale.

2 Grammatiche libere e automi a pila 20%

1. È dato il linguaggio L seguente, unione di due linguaggi componenti L_1 e L_2, entrambi di alfabeto $\{a, b\}$:

$$L = \underbrace{\{(a^n\, b^n)^* \mid \quad n \geq 1\}}_{L_1} \cup \underbrace{\left\{\left(a\, b^+\right)^*\right\}}_{L_2}$$

Ecco come esempio alcune stringhe del linguaggio L:

$$\varepsilon \qquad a\,b \qquad a\,b^+ \qquad a^2\,b^2 \qquad a^2\,b^2\,a\,b\,a\,b \qquad a\,b^2 \qquad a\,b^2\,a\,b$$

Si risponda alle domande seguenti:

(a) Si scriva una grammatica G, in forma non estesa (BNF), che generi il linguaggio L (non importa se G è ambigua).

(b) Se necessario, si scriva una grammatica G', non ambigua, che generi il linguaggio L, motivandone in breve la non ambiguità.

Soluzione

(a) Separatamente le due grammatiche BNF G_1 e G_2 (assiomi S_1 e S_2) dei componenti L_1 e L_2 del linguaggio L sono le seguenti:

$$G_1 \begin{cases} S_1 & \to & a\ B_1\ S_1 \\ S_1 & \to & \varepsilon \\ B_1 & \to & a\ B_1\ b \\ B_1 & \to & b \end{cases} \qquad G_2 \begin{cases} S_2 & \to & a\ B_2\ S_2 \\ S_2 & \to & \varepsilon \\ B_2 & \to & B_2\ b \\ B_2 & \to & b \end{cases}$$

Volutamente le due grammatiche G_1 e G_2 sono quanto più possibile simili strutturalmente. Prese separatamente G_1 e G_2 sono non ambigue. Una grammatica G, ambigua, che genera il linguaggio L si ottiene nel modo usuale unendo G_1 e G_2 (le quali hanno entrambe alfabeti nonterminali disgiunti), e aggiungendo la regola assiomatica $S \to S_1 \mid S_2$. La grammatica G è ambigua perché, per esempio, le stringhe ε e $a\,b$ sono generate sia da G_1 sia da G_2.

(b) I due componenti L_1 e L_2 del linguaggio L non sono disgiunti e danno origine ad ambiguità di unione. Le stringhe comuni ai due componenti sono le seguenti:

$$\varepsilon \quad a\,b \quad a\,b\,a\,b \quad \ldots \qquad \text{in generale } \left(a\,b\right)^*$$

Tali stringhe comuni vanno escluse da uno dei componenti. Si sceglie di escluderle dal secondo componente L_2, perché l'operazione è più semplice essendo questo puramente regolare. Basta imporre che la grammatica G_2 non possa generare ε e che in una stringa diversa da ε non possa sempre ripetere solo il fattore $a\,b$,

ma che ci debba essere almeno un fattore $a\,b^k$ $(k \geq 2)$. Ecco come ristrutturare G_2 in G_2' (assioma S_2') (mentre la grammatica G_1 resta invariata):

$$G_1 \begin{cases} S_1 & \to & a\ B_1\ S_1 \\ S_1 & \to & \varepsilon \\ B_1 & \to & a\ B_1\ b \\ B_1 & \to & b \end{cases} \qquad G_2' \begin{cases} S_2' & \to & a\ b\ S_2' \\ S_2' & \to & S_2'' \\ S_2'' & \to & a\ b\ B_2\ S_2''' \\ S_2''' & \to & a\ B_2\ S_2''' \\ S_2''' & \to & \varepsilon \\ B_2 & \to & B_2\ b \\ B_2 & \to & b \end{cases}$$

La grammatica G_2' funziona così: partendo dall'assioma S_2' genera prima i fattori $a\,b$ ripetuti (o li salta del tutto), poi genera almeno un fattore $a\,b^k$ $(k \geq 2)$ e infine genera liberamente fattori di tipo $a\,b^+$ (tra i quali potrebbero anche esserci altri fattori $a\,b$). Chiaramente G_2' non può generare $(a\,b)^*$ (e dunque neppure ε), mentre genera ogni stringa contenente quanti si vogliano fattori $a\,b$, distribuiti in modo del tutto libero, ma necessariamente contenente anche almeno un fattore di tipo $a\,b^k$ $(k \geq 2)$. Si noti che G_2', presa isolatamente, è non ambigua.

Unendo le grammatiche G_1 e G_2', e aggiungendo la regola assiomatica $S \to S_1 \mid S_2'$, si ha una grammatica G' non ambigua che genera il linguaggio L. È possibile esistano altre soluzioni, più compatte.

2. Si consideri un linguaggio che modella un programma semplificato, consistente in una lista di istruzioni di assegnamento, e che comprende gli aspetti sintattici seguenti:

   - Il programma è una lista (non vuota) di istruzioni di assegnamento, separate da ";" (punto e virgola); anche l'ultima istruzione ha il separatore.
   - L'assegnamento è denotato in forma prefissa, sul modello seguente:

     ```
     :=  obj  espressione
     ```

     dove ":=" (due punti uguale) è l'operatore (prefisso) di assegnamento e "espressione" è il valore da assegnare all'oggetto "obj" (vedi sotto).
   - L'espressione è a due livelli del tipo somma di prodotti e usa gli operatori "add" e "mul" (addizione e moltiplicazione), è associativa a sinistra ed è anch'essa denotata in forma prefissa, sul modello seguente (a destra come commento è data la forma infissa equivalente):

     ```
     Forma prefissa:                Forma infissa di commento:
       add obj obj                    obj + obj
       add add obj obj obj            obj + obj + obj
       add obj mul obj obj            obj + obj * obj
     ```

     dove "obj" rappresenta un oggetto (vedi sotto).
   - Gli oggetti "obj" che possono figurare nell'assegnamento (dove ha senso) e nell'espressione sono i seguenti:
     - valore costante, schematizzato tramite il simbolo terminale "c"
     - variabile nominale, schematizzata tramite il simbolo terminale "v"
     - elemento di array (a una o più dimensioni), con la sintassi seguente:

       ```
       a ( lista_di_espressioni )
       ```

       dove le espressioni della lista (non vuota) sono separate da "," (virgola), e il nome dell'array è schematizzato con il simbolo terminale "a"

   Ecco un breve esempio di programma (per aiutare a capire a destra come commento si dà la notazione infissa usuale):

   ```
   Programma:                          Commento (forma infissa):
     :=  v  c ;                          v := c ;
     :=  v  add v c ;                    v := v + c ;
     :=  v  add add c v v ;              v := c + v + v ;
     :=  v  add mul v v v                v := v * v + v ;
     :=  v  a(mul v v) ;                 v := a(v * v) ;
     :=  a(c, add v v)  c ;              a(c, v + v) := c ;
   ```

   Si risponda alle domande seguenti:

   (a) Si scriva una grammatica G, non ambigua e in forma estesa (EBNF), che genera il linguaggio descritto sopra.

   (b) Si dica in breve quali aspetti semantici del programma così descritto non sono modellabili in modo puramente sintattico.

Soluzione

(a) Ecco la grammatica G richiesta (assioma $\langle \mathsf{PROG} \rangle$), in forma estesa:

$$
\begin{array}{rcl}
\langle \mathsf{PROG} \rangle & \rightarrow & (\langle \mathsf{INSTR} \rangle \ \text{';'})^{+} \\
\langle \mathsf{INSTR} \rangle & \rightarrow & \text{':='} \ (\langle \mathsf{VAR} \rangle \mid \langle \mathsf{VECT} \rangle) \ \langle \mathsf{EXPR} \rangle \\
\hline
\langle \mathsf{EXPR} \rangle & \rightarrow & \text{'add'} \ \langle \mathsf{EXPR} \rangle \ \langle \mathsf{TERM} \rangle \mid \langle \mathsf{TERM} \rangle \\
\langle \mathsf{TERM} \rangle & \rightarrow & \text{'mul'} \ \langle \mathsf{TERM} \rangle \ \langle \mathsf{FACT} \rangle \mid \langle \mathsf{FACT} \rangle \\
\langle \mathsf{FACT} \rangle & \rightarrow & \langle \mathsf{CONST} \rangle \mid \langle \mathsf{VAR} \rangle \mid \langle \mathsf{VECT} \rangle \\
\hline
\langle \mathsf{CONST} \rangle & \rightarrow & \text{'c'} \\
\langle \mathsf{VAR} \rangle & \rightarrow & \text{'v'} \\
\langle \mathsf{VECT} \rangle & \rightarrow & \text{'a'} \ \text{'('} \ \langle \mathsf{EXPR} \rangle \ (\text{','} \ \langle \mathsf{EXPR} \rangle)^{*} \ \text{')'}
\end{array}
$$

Siccome la grammatica G è la composizione modulare di forme non ambigue (liste, espressioni, ecc), anch'essa è non ambigua. L'espressione è a due livelli di tipo somma di prodotti ed è associativa a sinistra (come richiesto dall'esercizio), giacché le regole che espandono l'espressione sono ricorsive a sinistra.

(b) Il linguaggio è molto semplificato, e resta poco di semantico da verificare. Essenzialmente non si può garantire la coerenza dei tipi nell'espressione e nell'assegnamento, e che un vettore abbia un numero costante di dimensioni.

3 Analisi sintattica e parsificatori 20%

1. È data la grammatica G seguente (assioma S), in forma estesa (EBNF):

$$G \left\{ \; S \;\; \rightarrow \;\; (a \;\mid\; c)\; (b \;\mid\; S^{+})\; c\, d \right.$$

definita sull'alfabeto terminale $\{a, b, c, d\}$.

Si risponda alle domande seguenti:

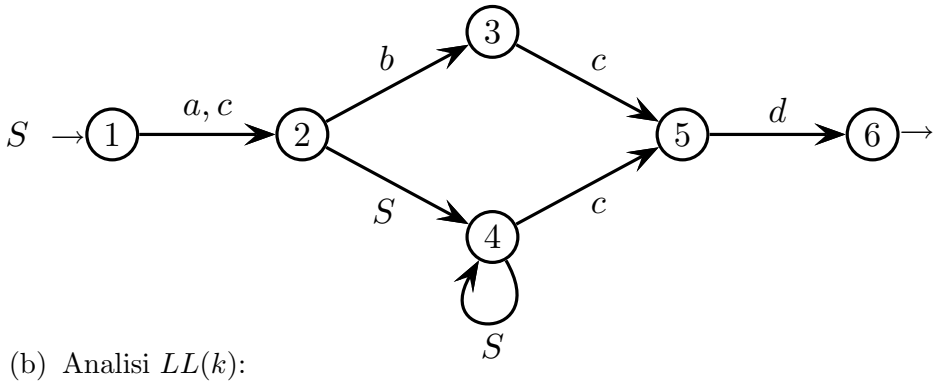
(a) Si rappresenti la grammatica G in forma di rete di automi ricorsivi a stati finiti, sull'alfabeto totale (unione di terminali e nonterminali).

(b) Si stabilisca se la grammatica G sia di tipo $LL(k)$, per un qualche $k \geq 1$.

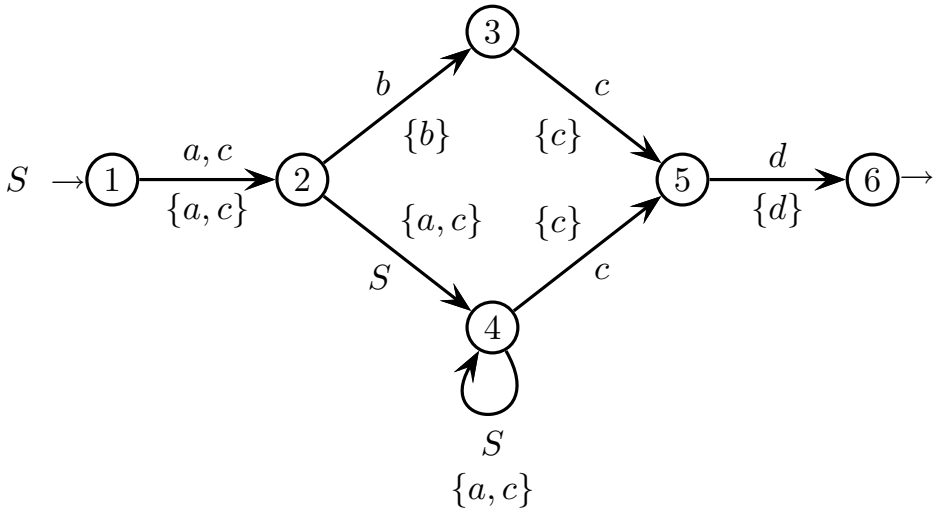
(c) (facoltativo) Si scriva la procedura sintattica dell'assioma S dell'analizzatore sintattico LL della grammatica G (per il valore di k trovato prima).

Soluzione

(a) Ecco la rete di automi ricorsivi che modella la grammatica G (c'è un solo automa perché c'è un solo nonterminale):

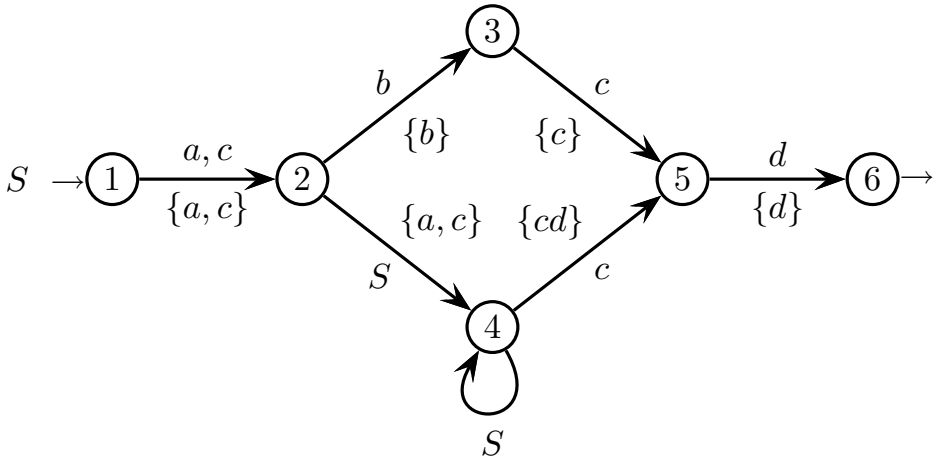


(b) Analisi $LL(k)$:
Per $k = 1$:



La grammatica G non è $LL(1)$.

Per $k = 2$:



$\{ab, cb, aa, ac, ca, cc\}$

La grammatica G è $LL(2)$.

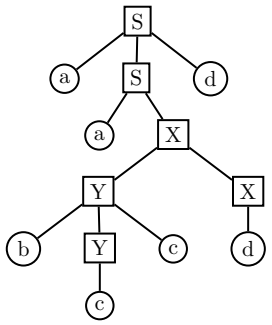
(c) L'analizzatore sintattico $LL(2)$ fa riferimento al secondo automa con prospezione. Esso fa uso di una procedura ausiliaria "shift" per fare scorrere la finestra di prospezione di ampiezza $k = 2$ sulla stringa di ingresso, se è possibile (cioè se non si è raggiunta la fine della stringa). L'analizzatore utilizza la prospezione con $k = 2$ solo dove è indispensabile (cioè nello stato 4), altrimenti si limita alla prospezione con $k = 1$. Eccolo, codificato in pseudocodice tipo Pascal:

```
char cur, next                  – variabili globali: car. corrente e prossimo

procedure shift                 – scorrimento finestra di prospezione
        cur = next
        if (cur ≠⊣) then
                read (next)
        end if
end procedure

program SYNTAX_ANALYSER
        read (next)             – posizionamento iniziale finestra di prospezione
        shift
        call S
        if (cur ≠⊣) then        – verifica di fine stringa
                error
        end if
end program

procedure S                     – procedura sintattica di S
        – stato 1 (iniziale)
        if (cur = a or cur = c) then
                shift
        else error
        – stato 2
        if (cur = b) then
                shift
                – stato 3
                if (cur = c) then
                        shift
                else error
        else if (cur = a or cur = c) then
                repeat
                        call S
                        – stato 4
                until (cur, next ≠ ab, cb, aa, ac, ca, cc)
                – stato 4
                if (cur, next = cd) then
                        shift
                else error
        else error
        – stato 5
        if (cur = d) then
                shift
        else error
        – stato 6 (finale)
end procedure
```

Si noti il ciclo "repeat-until" che modella la transizione $2 \xrightarrow{S} 4$ seguita dall'autoanello $4 \xrightarrow{S} 4$; insieme infatti queste due transizioni realizzano un ciclo iterabile una o più volte.

2. Si consideri la grammatica G seguente (assioma S), non in forma estesa (BNF), definita sull'alfabeto terminale $\{a, b, c, d\}$:

$$G \begin{cases} S & \rightarrow & a\ S\ d \mid a\ X \\ X & \rightarrow & Y\ X \mid d \\ Y & \rightarrow & b\ Y\ c \mid c \end{cases}$$



stringa di esempio $a\,a\,b\,c\,c\,d\,d$

Si risponda alle domande seguenti:

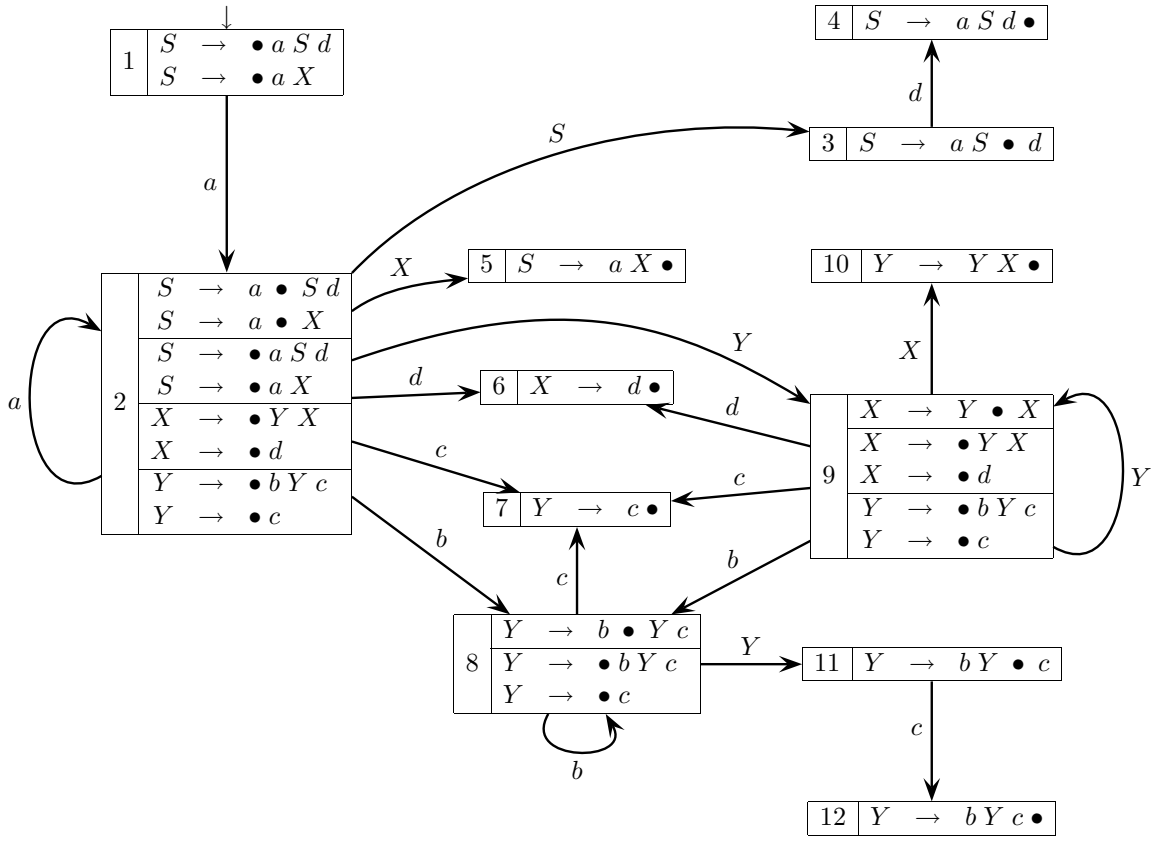
(a) Si tracci il grafo pilota (riconoscitore dei prefissi) di tipo $LR(0)$ della grammatica G e si dimostri che G è di tipo $LR(0)$.

(b) Si scriva la simulazione del riconoscimento della stringa di esempio $a\,a\,b\,c\,c\,d\,d \in L(G)$ (albero sintattico sopra) da parte dell'analizzatore sintattico $LR(0)$ di G; allo scopo si usi la tabella preparata di seguito (il numero di righe non è significativo), dove le due righe iniziali sono già compilate.

#	**transizione**	**operazione di pila**	**operazione di ingresso**	**contenuto della pila**	**contenuto del nastro di ingresso**	**commento**
0		push 1		1	$a\,a\,b\,c\,c\,d\,d\,\dashv$	inizia a macrostato 1
1	$1 \xrightarrow{a} 2$	push 2	shift a	1 2	$a\,b\,c\,c\,d\,d\,\dashv$	
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						

Soluzione

(a) Ecco il grafo pilota $LR(0)$ della grammatica G:



Non ci sono conflitti (le candidate di riduzione figurano tutte isolate), pertanto la grammatica G è di tipo $LR(0)$.

(b) Ecco la simulazione richiesta:

#	trans.	operazione di pila	operazione di ingresso	contenuto della pila	contenuto del nastro di ingresso	commento
0		push 1		1	$a\,a\,b\,c\,c\,d\,d\ \dashv$	inizia a macrostato 1
1	$1 \xrightarrow{a} 2$	push 2	shift a	1 2	$a\,b\,c\,c\,d\,d\ \dashv$	
2	$2 \xrightarrow{a} 2$	push 2	shift a	1 2 2	$b\,c\,c\,d\,d\ \dashv$	
3	$2 \xrightarrow{b} 8$	push 8	shift b	1 2 2 8	$c\,c\,d\,d\ \dashv$	
4	$8 \xrightarrow{c} 7$	push 7	shift c	1 2 2 8 7	$c\,d\,d\ \dashv$	
5		pop		1 2 2 8	$Y\,c\,d\,d\ \dashv$	riduci $Y \to c$
6	$8 \xrightarrow{Y} 11$	push 11	shift Y	1 2 2 8 11	$c\,d\,d\ \dashv$	
7	$11 \xrightarrow{c} 12$	push 12	shift c	1 2 2 8 11 12	$d\,d\ \dashv$	
8		pop, pop, pop		1 2 2	$Y\,d\,d\ \dashv$	riduci $Y \to b\,Y\,c$
9	$2 \xrightarrow{Y} 9$	push 9	shift Y	1 2 2 9	$d\,d\ \dashv$	
10	$9 \xrightarrow{d} 6$	push 6	shift d	1 2 2 9 6	$d\ \dashv$	
11		pop		1 2 2 9	$X\,d\ \dashv$	riduci $X \to d$
12	$9 \xrightarrow{X} 10$	push 10		1 2 2 9 10	$d\ \dashv$	
13		pop, pop		1 2 2	$X\,d\ \dashv$	riduci $X \to Y\ X$
14	$2 \xrightarrow{X} 5$	push 5	shift X	1 2 2 5	$d\ \dashv$	
15		pop, pop		1 2	$S\,d\ \dashv$	riduci $S \to a\ X$
16	$2 \xrightarrow{S} 3$	push 3	shift S	1 2 3	$d\ \dashv$	
17	$3 \xrightarrow{d} 4$	push 4	shift d	1 2 3 4	$\dashv$	
18		pop, pop, pop		1	$S\ \dashv$	riduci $S \to a\ S\ d$
19				1	$S\ \dashv$	termina

La stringa di esempio è riconosciua correttamente in 18 passi (il primo e l'ultimo sono formali), in quanto viene ridotta per intero all'assioma S, come atteso.

Si ricorda che la scrittura di un nonterminale nel nastro di ingresso è un'operazione fittizia, che l'analizzatore non esegue realmente (come del resto non ne esegue la rilettura); tale operazione serve semplicemente come espediente notazionale per capire come avvenga la mossa di shift immediatamente successiva, che "rilegge" (fittiziamente) appunto tale nonterminale.

4 Traduzione e analisi semantica 20%

1. Si vuole trasformare un'espressione aritmetica di tipo somma di prodotti, e con sottoespressioni tra parentesi tonde, come per esempio:

$$a + a \times a \qquad a + a \times a \times a \qquad a + a \times (a + a)$$

nel modo seguente:

(a) il segno infisso "+" diventa l'operatore ancora infisso "add"

(b) il segno infisso "×" diventa l'operatore ancora infisso "div" e il secondo fattore "fact_2" di "×" diventa l'inverso "$(1\ \mathsf{div}\ \mathsf{fact}_2)$", cioè:

$$\mathsf{fact}_1 \times \mathsf{fact}_2 \quad \Rightarrow \quad \mathsf{fact}_1\ \mathsf{div}\ (1\ \mathsf{div}\ \mathsf{fact}_2)$$

Si noti che le parentesi tonde introdotte assicurano che l'operatore "div" sia associativo a destra. Per esempio, le tre espressioni date prima vanno tradotte così (nell'ordine):

$$a\ \mathsf{add}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ a)$$

$$a\ \mathsf{add}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ a))$$

$$a\ \mathsf{add}\ a\ \mathsf{div}\ (1\ \mathsf{div}\ (a\ \mathsf{add}\ a))$$

Il linguaggio sorgente è definito dalla grammatica G seguente:

$$G \begin{cases} E & \rightarrow & E\ \text{`+'}\ T \ \mid\ T \\ T & \rightarrow & T\ \text{`}\times\text{'}\ F \ \mid\ F \\ F & \rightarrow & \text{`('}\ E\ \text{`)'} \ \mid\ \text{`}a\text{'} \end{cases}$$

Si risponda alle domande seguenti:

(a) Si progetti uno schema sintattico di traduzione G_τ per la trasformazione descritta sopra, dove se si preferisce la grammatica sorgente può essere diversa da G, purché a essa equivalente.

(b) Si disegni l'albero sintattico sorgente e destinazione della terza espressione di esempio data sopra.

Soluzione

(a) Dato che l'operatore "div" è chiaramente associativo a destra, prima bisogna rendere associativa a destra la grammatica G data nel testo, la quale ha le ricorsioni a sinistra e pertanto è associativa a sinistra, come segue:

$$G' \begin{cases} E & \rightarrow & T\ \text{`+'}\ E \ \mid\ T \\ T & \rightarrow & F\ \text{`}\times\text{'}\ T \ \mid\ F \\ F & \rightarrow & \text{`('}\ E\ \text{`)'} \ \mid\ \text{`}a\text{'} \end{cases}$$

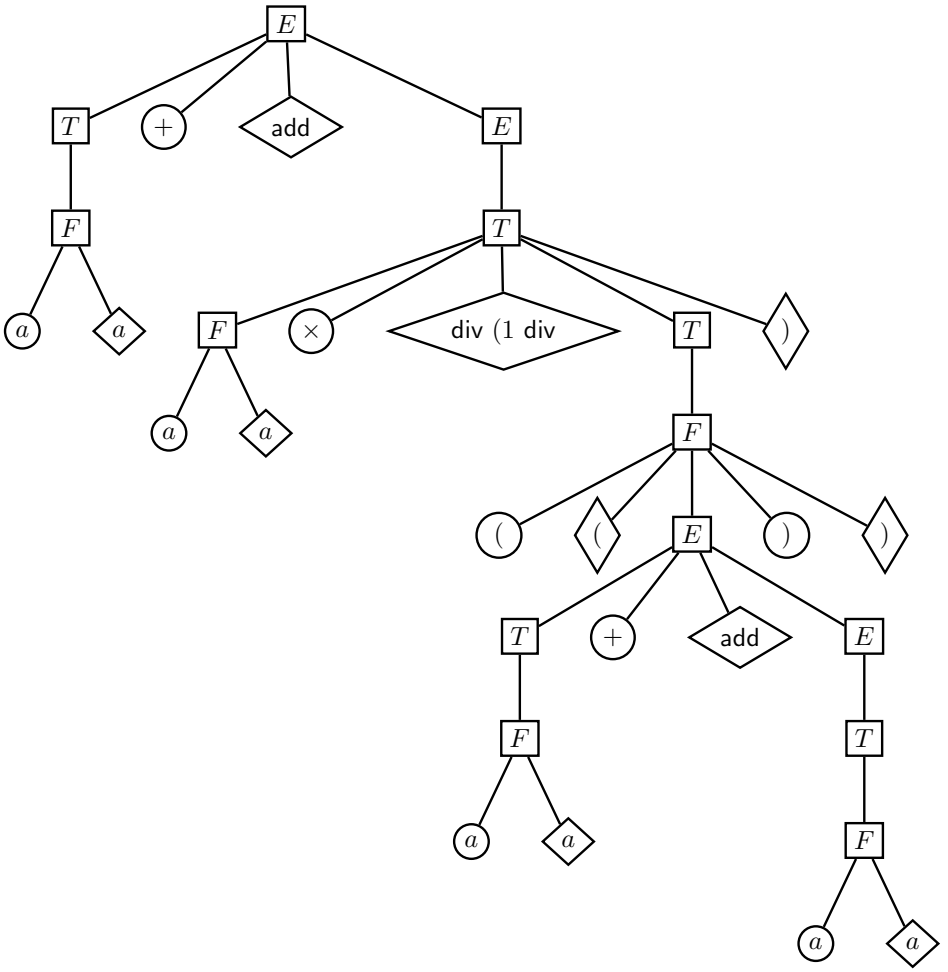
Ovviamente G' è equivalente a G, ma ha la ricorsione a destra delle regole e dunque è associativa a destra. Ed ecco lo schema sintattico G_τ richiesto (in forma combinata), ottenuto dalla grammatica sorgente modificata G':

$$G_\tau \begin{cases} E & \rightarrow & T \text{ `+'} \ \{\text{`add'}\} \ E \ \mid \ T \\ T & \rightarrow & F \text{ `}\times\text{'} \ \{\text{`div (1 div'}\} \ T \ \{\text{`)'}\} \ \mid \ F \\ F & \rightarrow & \text{`('} \ \{\text{`('}\} \ E \text{ `)'} \ \{\text{`)'}\} \ \mid \ \text{`}a\text{'} \ \{\text{`}a\text{'}\} \end{cases}$$

o separatamente:

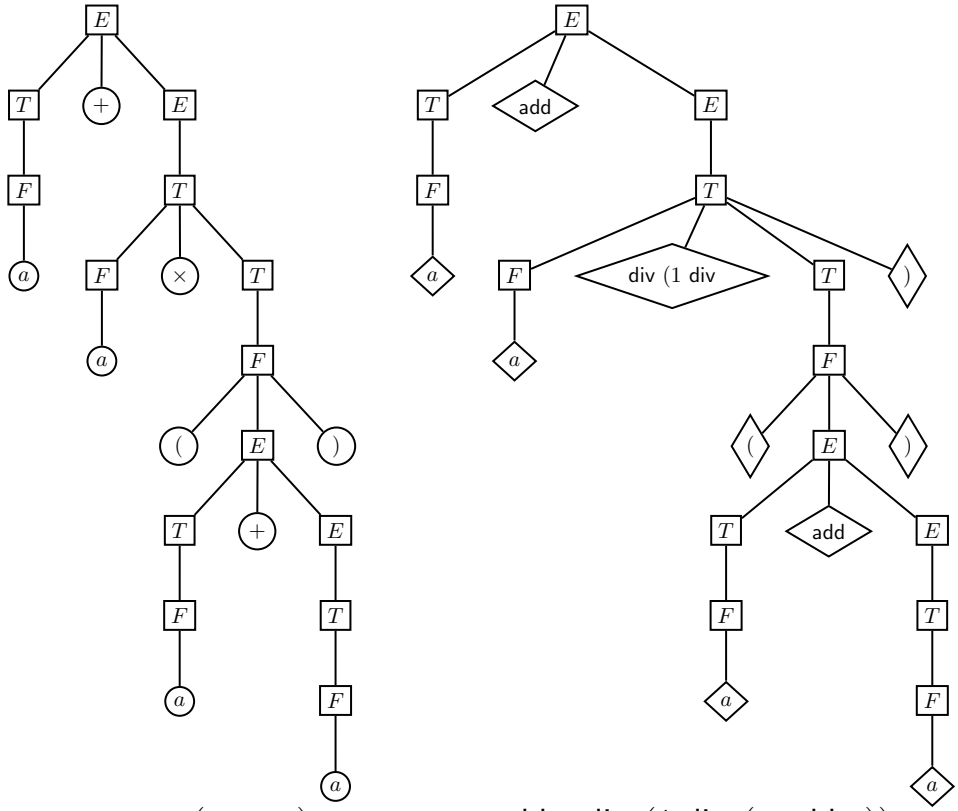
$$G_\tau \begin{cases} E & \rightarrow & T \text{ `+'} \ E \ \mid \ T & \qquad E & \rightarrow & T \text{ `add'} \ E \ \mid \ T \\ T & \rightarrow & F \text{ `}\times\text{'} \ T \ \mid \ F & \qquad T & \rightarrow & F \text{ `div (1 div'} \ T \text{ `)'} \ \mid \ F \\ F & \rightarrow & \text{`('} \ E \text{ `)'} \ \mid \ \text{`}a\text{'} & \qquad F & \rightarrow & \text{`('} \ E \text{ `)'} \ \mid \ \text{`}a\text{'} \end{cases}$$

(b) L'albero sintattico della traduzione richiesta (in forma combinata) è il seguente:



$$a + a \times (a + a) \qquad \Rightarrow \qquad a \text{ add } a \text{ div } (1 \text{ div } (a \text{ add } a))$$

o separatamente:



$a + a \times (a + a)$ $\qquad$ a add a div $(1$ div $(a$ add $a))$

Pertanto la traduzione richiesta è modellata correttamente in modo puramente sintattico.

Tale traduzione potrebbe servire, per esempio, per convertire un'espressione di tipo somma di prodotti, rendendola adatta all'esecuzione su un processore privo di istruzione di moltiplicazione ma dotato di istruzione di divisione.

2. È data la regola seguente, di una grammatica con attributi G (le altre regole della grammatica qui non interessano):

$$G \begin{cases} \ldots \\ A \ \rightarrow \ X \ A \ Z \\ \ldots \end{cases}$$

La grammatica G ha gli attributi seguenti: α, β e γ. Supponendo la regola numerata nel modo standard, cioè $A_0 \rightarrow X_1 \ A_2 \ Z_3$, sono date le funzioni semantiche seguenti:

$$\begin{aligned} \alpha_0 &= f_1(\beta_0, \alpha_1, \alpha_2) \\ \beta_1 &= f_2(\alpha_3, \beta_3) \\ \beta_2 &= f_3(\beta_0, \alpha_1) \\ \gamma_2 &= f_4(\beta_2) \\ \gamma_3 &= f_5(\gamma_0) \\ \beta_3 &= f_6(\gamma_3) \end{aligned}$$

Qui non interessa conoscere nei dettagli la forma funzionale di $f_1, \ldots, f_6$.

Si risponda alle domande seguenti:

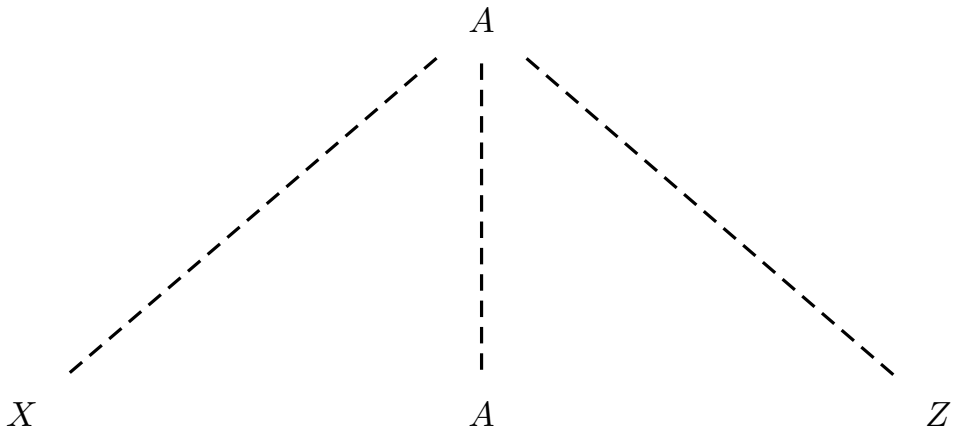
(a) Si dica quali attributi sono sintetizzati e quali ereditati, e poi si tracci il grafo delle dipendenze della regola data; allo scopo si usi l'albero sintattico della regola, preparato di seguito.

(b) Si verifichi se la regola data soddisfi la condizione a una scansione (one-sweep) ed eventualmente se sia di tipo L, in entrambi i casi spiegandone il motivo.

(c) Si scriva la procedura dell'analizzatore semantico per la regola data.

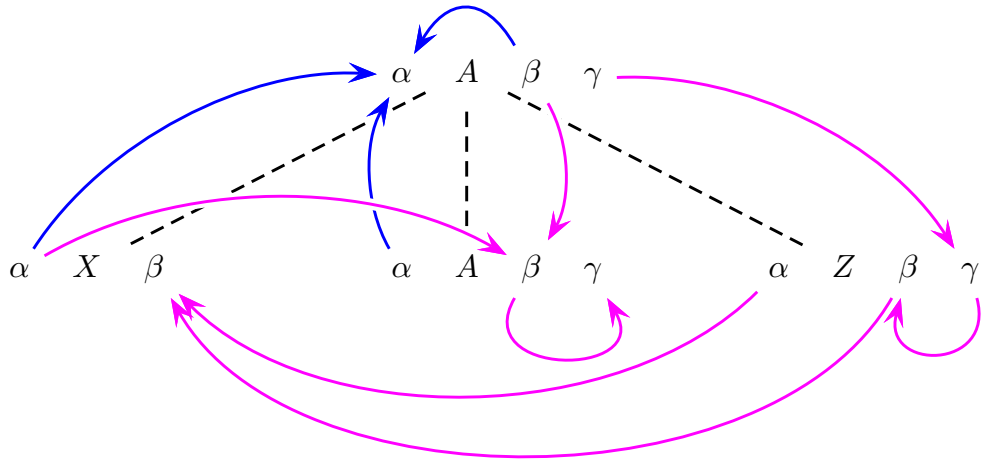
---

supporto sintattico da decorare con le dipendenze funzionali tra attributi



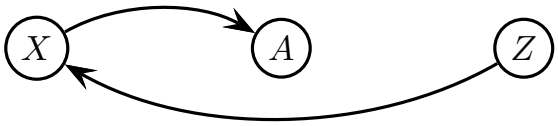
Soluzione

(a) L'attributo α è sintetizzato, gli attributi β e γ sono ereditati. Gli attributi α e β sono associati con i nonterminali A, X e Z, l'attributo γ solo con A e Z (non con X). Ecco il grafo delle dipendenze funzionali tra attributi:



Come d'uso, gli attributi sintetizzati ed ereditati sono indicati a sinistra e destra, rispettivamente, del nodo cui si riferiscono. In blu sono indicate le dipendenze dell'attributo sintetizzato, in magenta quelle degli attributi ereditati.

(b) La grammatica G è corretta in generale in quanto le dipendenze tra attributi sono acicliche. Pertanto gli attributi sono calcolabili, sia pure in più passate.
Per vedere se la grammatica G sia a una scansione (one-sweep), occorre verificare alcune condizioni aggiuntive. Ecco il grafo dei fratelli:

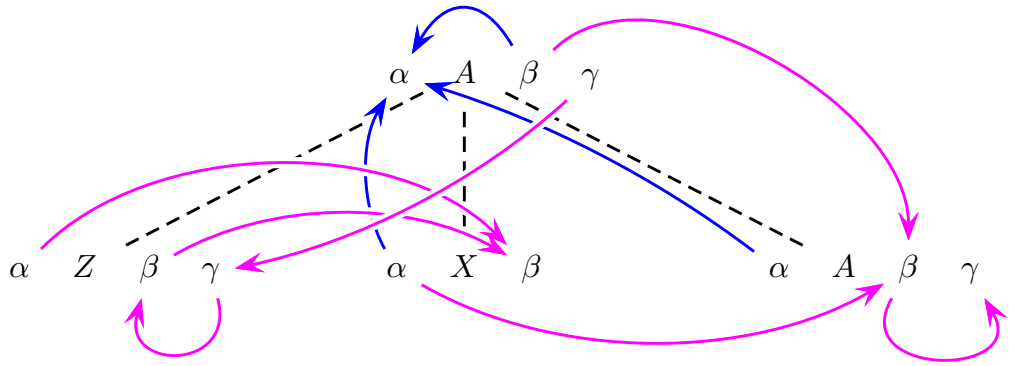


Esso è aciclico e si può ordinare topologicamente come Z, X, A (qui tale ordine è unico). Si vede che la grammatica G è a una scansione (one-sweep), perché la condizione seguente (divisa in tre parti) è soddisfatta:

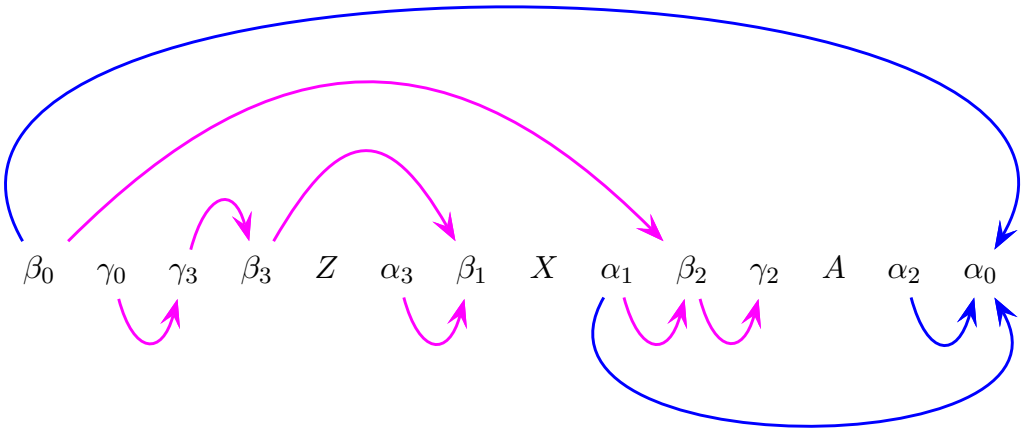
- l'attributo sintetizzato α del nodo padre A dipende dall'attributo sintetizzato α dei nodi figli e dall'attributo ereditato β del padre stesso, mentre non dipende dagli attributi ereditati dei figli
- gli attributi ereditati β e γ dei nodi figli dipendono degli attributi ereditati β e γ del nodo padre, mentre non dipendono dall'attributo sintetizzato α del padre né dall'attributo sintetizzato α del nodo stesso dove β e γ sono definiti
- il grafo dei fratelli è aciclico e si può dunque ordinare linearmente, in modo che le dipendenze tra attributi dei nodi fratelli siano tutte orientate da

sinistra verso destra (si ricordi che l'acicilicità delle dipendenze da sola non implica che il grafo dei fratelli sia anch'esso neecssariamente aciclico)

Ecco l'albero sintatico della regola $A \to X\ A\ Z$ con i nodi figli riordinati:



Ora il flusso di calcolo da sinistra verso destra è evidente (nonché dal basso verso l'alto). Ed ecco un ordine di calcolo lineare tra attributi valido:



Chiaramente però la grammatica G non è di tipo L, perché per valutarne gli attributi in una sola passata occorre cambiare l'ordine dei nodi figli del nodo padre A rispetto all'ordine di scansione lineare da sinistra verso destra implicito nel supporto sintattico.

(c) La procedura semantica del nonterminale A richiede di ordinare la visita dei sottoalberi figli come: Z, X, A. Eccola, preceduta delle intestazioni delle procedure semantiche dei figli X e Z:

procedure X (**in**: β_0; **in**: $tree$; **out**: α_0) – solo intestazione

procedure Z (**in**: β_0, γ_0; **in**: $tree$; **out**: α_0) – solo intestazione

procedure A (**in**: β_0, γ_0; **in**: $tree$; **out**: α_0) – intestazione e corpo

var $\alpha_3, \beta_3, \gamma_3, \alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_2$ – variabili locali attributo

$\gamma_3 = f_5(\gamma_0)$

$\beta_3 = f_6(\gamma_3)$

call Z (α_3, $tree \rightarrow Z$, β_3, γ_3) – valuta figlio Z

$\beta_1 = f_2(\alpha_3, \beta_3)$

call X (α_1, $tree \rightarrow X$, β_1) – valuta figlio X

$\beta_2 = f_3(\beta_0, \alpha_1)$

$\gamma_2 = f_4(\beta_2)$

call A (α_2, $tree \rightarrow A$, β_2, γ_2) – valuta figlio A

$\alpha_0 = f_1(\beta_0, \alpha_1, \alpha_2)$

end procedure

Si noti che la procedura semantica data segue precisamente l'ordine lineare di valutazione degli attributi illustrato sopra.