Linguaggi Formali e Compilatori
Prof. Crespi Reghizzi
Soluzioni Prova scritta[1]
11/03/2004

	punti %	annotazioni	VOTO
1. Espressioni regolari e automi finiti			
2. Grammatiche			
3. Laboratorio Flex Bison			
4. Grammatiche e analisi sintattica			
5. Traduzione e semantica			
VOTO			

Per superare la prova l'allievo deve dimostrare la conoscenza di tutte e 5 le parti.

1 Espressioni regolari e automi finiti 20%

1. Progetto di espr. regolare
   Alfabeto terminale $\{p, \vee, \wedge\}$. Una frase è una proposizione logica che contiene al più due operatori $\wedge$.
   Esempi: p , $p \vee p$, $p \vee p \wedge p$, $p \wedge p \vee p \wedge p$
   Controesempi: $pp\vee$, $p \wedge p \wedge p \wedge p \vee p$

   Soluzione:

   L è l'unione disgiunta di tre ling., aventi risp. 0, 1 e 2 operatori $\wedge$:

   $$L = L_0|L_1|L_2$$

   Il primo è semplicemente una lista di p separati da $\vee$

   $$L_0 = p(\vee p)^*$$

   Il secondo è

   $$L_1 = L_0 \wedge L_0$$

   Il terzo è

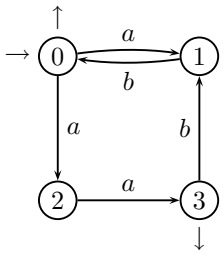
   $$L_1 = L_0 \wedge L_0 \wedge L_0$$

[1]Tempo 2 ore 30'. Libri e appunti personali possono essere consultati. È consentito scrivere a matita. Scrivere il proprio nome sugli eventuali fogli aggiuntivi.

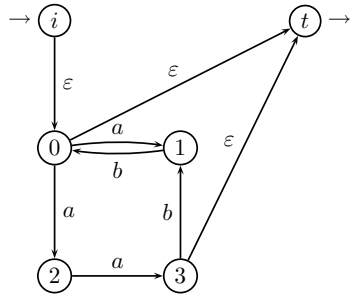
Raccogliendo le sottoespressioni comuni si ha

$$L = L_0|L_1|L_2 = [[L_0\wedge]L_0\wedge]L_0$$

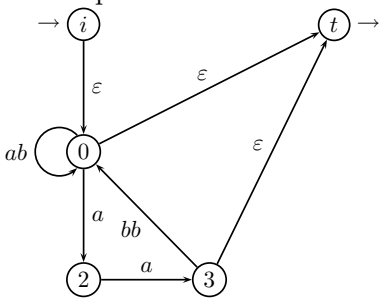
2. Calcolare, mostrando i passaggi, l'espressione regolare del ling. riconosciuto dall'automa seguente.



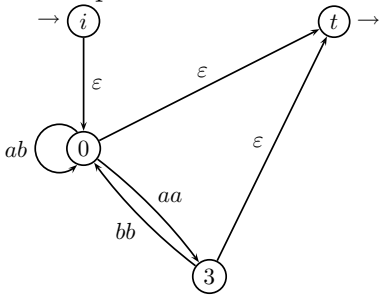
Soluzione. Applicheremo il metodo di eliminazione di Brzozowsky e McCluskey, eliminando i nodi ad es. nell'ordine 1, 2, 3, 0.



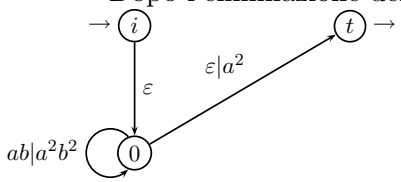
Dopo l'eliminazione del nodo 1:



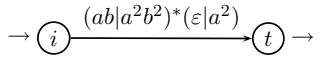
Dopo l'eliminazione del nodo 2:



Dopo l'eliminazione del nodo 3:



Dopo l'eliminazione del nodo 0:



2 Grammatiche 20%

1. Progettare una grammatica per generare il ling. definito dalla formula

$$L_1 = L_{Dyck} \cap ((a|c)^* ca (a|c)^*)$$

dove L_{Dyck} è il ling. di Dyck di alfabeto $\{a, c\}$.

   (a) Elencare le tre stringhe più corte appartenenti a $L_{Dyck} - L_1$

   (b) Scrivere la gramm. di L_1, preferibilmente non ambigua

   (c) Disegnare l'albero sintattico di una frase di L_1

   Soluzione

   (a) L'intersezione con l'insieme regolare impone la presenza della sottostringa ca, così escludendo le stringhe di Dyck $\varepsilon, ac, aacc, \ldots$.

   (b) La gramm. di L_1 si ottiene con il seguente ragionamento. La frase più corta è $acac$, e può essere allungata inserendo una stringa del ling. di Dyck, denotato D, tra a e c:
   $S \to aDcaDc$
   $D \to aDcD|\varepsilon$ - - la nota grammatica di Dyck
   Ma questa soluzione difetta, non generando le frasi, come ad es. $acacac$, che contengono 3 o più nidi al primo livello. Per generarle basta concatenare il ling. di Dyck D al ling. precedente:
   $S \to aDcaDcD$
   $D \to aDcD|\varepsilon$
   Mancano ancora le frasi come $aacacc$ in cui le parentesi concatenate $acac$ stanno all'interno di altre parentesi. Per generare queste frasi si aggiunge la regola $S \to aSc$, ottenendo la grammatica
   $S \to aDcaDcD$
   $S \to aSc$
   $D \to aDcD|\varepsilon$

2. Il ling. di comando di un *plotter* contiene le seguenti istruzioni:

   traccia linea spezzata:

   $$\mathrm{TS}'(\text{'spessore,'}\ (\text{'}x_1, y_1\ \text{'})', \ldots, \text{'}\ (\text{'}x_m, y_m\ \text{'})'\ \text{'})'$$

   dove spessore è un numero decimale nell'intervallo $0.1 \ldots 9.9$
   il numero di punti è $m \geq 2$. Le coordinate dei punti sono numeri interi.

   traccia arco di cerchio:

   $$\mathrm{TA}'(\text{'}\mathit{centro}, \mathit{raggio\_e\_coordinate}\text{'})'$$

   dove è lasciato alla vostra scelta come specificare il centro, il raggio e le coordinate dell'arco da tracciare.

   fissa colori delle linee:

   $$\mathrm{FC}'(\text{'COLORE} = \mathit{colore}\text{'})'\text{BEGIN} \ldots \text{END}$$

   dove $\mathit{colore} \in \{g, r, v\}$
   e nel blocco racchiuso tra BEGIN . . . END può stare una serie di istruzioni, TS, TA e anche, nota bene, FC.

   (a) Scrivere la grammatica (consentita la forma EBNF)
   (b) Disegnare, almeno in parte, i diagrammi sintattici delle regole della grammatica
   (c) Disegnare un albero sintattico sufficientemente rappresentativo.

Soluzione.

$S \rightarrow$ F^+
- - F sta per una frase elementare
$F \rightarrow$ $\mathrm{FC}'(\text{'COLORE} = C')'\text{BEGIN}F^+\text{END}$ |
- - C sta per un colore
$\mathrm{TS}'(\text{'}W, P, P(, P)^*\ \text{'})'$ |
- - W spessore, P una coppia di interi tra parentesi
$\mathrm{TA}'(\text{'}P, I, P)')'$
- - P centro, I raggio, P coppia di angoli tra parentesi
$P \rightarrow$ $'(\text{'}I, I')'$
$W \rightarrow$ $(1 \ldots 9) \bullet (0 \ldots 9)$ |
$0 \bullet (1 \ldots 9)$
$C \rightarrow$ $g|r|v$
$I \rightarrow$ $(1 \ldots 9)(0 \ldots 9)^*$ | 0

3 Domanda relativa alle esercitazioni 20%

Considerate l'implementazione del compilatore *Simple* fornita di seguito e con supporto per il costrutto `for` e l'istruzione `break`. Modificatela in modo che venga riconosciuta anche l'istruzione `continue` nei cicli `for`. Assumete per `continue` la stessa semantica che essa ha nel linguaggio C. Il compilatore da voi modificato deve riconoscere il costrutto e generare una traduzione corretta nel linguaggio assembly della macchina *SimpleVM*. Un esempio banale di uso della `continue` segue:

```
for (c:=1; c<20; c:=c+1)
do
  if c=10 then
   continue;
  else
   write c;
  fi;
od;
```

Notate che in *Simple*, è legale annidare i cicli, quindi `continue` deve saltare al punto corretto del ciclo `for` corretto.

Per vostra utilità forniamo un contenitore generico di puntatori, che potete usare sia come pila (push, pop, top) che come array dinamico o coda (add, get-size, get-at). Il contenitore è utile per memorizzare dati di contesto o i punti dove effettuare il backpatching.

- `void * gpc_top (GenericPtrContainer pc);`
  restituisce l'elemento al top della pila;
- `void gpc_push (GenericPtrContainer * pc, void * ptr);`
  impila un nuovo elemento;
- `void gpc_pop (GenericPtrContainer * pc);`
  disimpila un elemento;
- `void gpc_add (GenericPtrContainer * pc, void * ptr);`
  aggiunge un elemento in coda;
- `int gpc_get_size (GenericPtrContainer pc);`
  restituisce la lunghezza della coda;
- `void * gpc_get_at (GenericPtrContainer pc, int i);`
  restituisce l'elemento i-esimo;
- `GenericPtrContainer var = {NULL,0,0};`
  dichiarazione di un contenitore inizialmente vuoto di nome *var*;

Buon lavoro.

4 Grammatiche e analisi sintattica 20%

1. Data la grammatica EBNF

$$G_1 : \left| \begin{array}{l} S \to (B|bc)^* \\ B \to bBa|\varepsilon \end{array} \right.$$

   - Calcolare gli insiemi guida di G_1 verificando se essa risulta ELL(1)
   - Scrivere una grammatica G_2, in forma non EBNF, equivalente a G_1 adatta all'anlisi deterministica discendente.
   - Verificare che G_2 sia LL(k) calcolandone gli insiemi guida.

   Soluzione

   - Verifichiamo la condizione ELL(1) in tutte le sottoespressioni dove vi sono scelte.

$$\underbrace{(B|bc)}_{e} : \ (Ini(B) \cup Seg_e(B)) \cap Ini(bc) = \{b, \dashv\} \cap \{b\} \neq \emptyset$$

     dove i seguiti di B vanno calcolati limitatamente alla sottoespressione e considerata, quindi non contengono il carattere a.

$$(B|bc)^* : \ Ini(B|bc) \cap Seg((B|bc)^*) = \{b\} \cap \{\dashv\} = \emptyset$$

$$Ini(bBa) \cap Seg(B) = \{b\} \cap \{a, b, \dashv\} \neq \emptyset$$

     La condizione ELL(1) è violata due volte.
     Si nota anche che G_1 è ambigua:

$$S \Rightarrow \varepsilon \text{ opp. } S \Rightarrow B \Rightarrow \varepsilon$$

     È semplice descrivere il linguaggio definito

$$L(G_1) = (u|bc)^* \text{ dove } u \in \{b^n a^n | n \geq 1\}$$

   - Scriviamo la grammatica in forma non estesa, eliminando al contempo l'ambiguità:

		Condizione LL(2)
$S \to$	$DS \mid$	$Ini_2(D) = \{ba, bb\}$
	$bcS \mid$	$Ini_2(bc) = \{bc\}$
	ε	$Seg(S) = \dashv$
$D \to$	$bDa \mid$	$Ini_2(bDa) = \{bb\}$
	ba	$Ini_2(ba) = \{ba\}$

- La condizione LL(2) è soddisfatta.

  Volendo è facile ottenere una grammatica equivalente LL(1), prima espandendo D:

  $$\begin{array}{ll} S \to & bDaS \mid \\ & bcS \mid \\ & \varepsilon \\ D \to & bDa \mid \\ & \varepsilon \end{array}$$

  poi fattorizzando a sin.:

		Condizione LL(1)
$S \to$	$bXS \mid$	b
	ε	$\dashv$
$X \to$	$Da \mid$	a, b
	c	c
$D \to$	$bDa \mid$	b
	ε	a

2. Grammatica LR(1)
   Per la seguente grammatica:

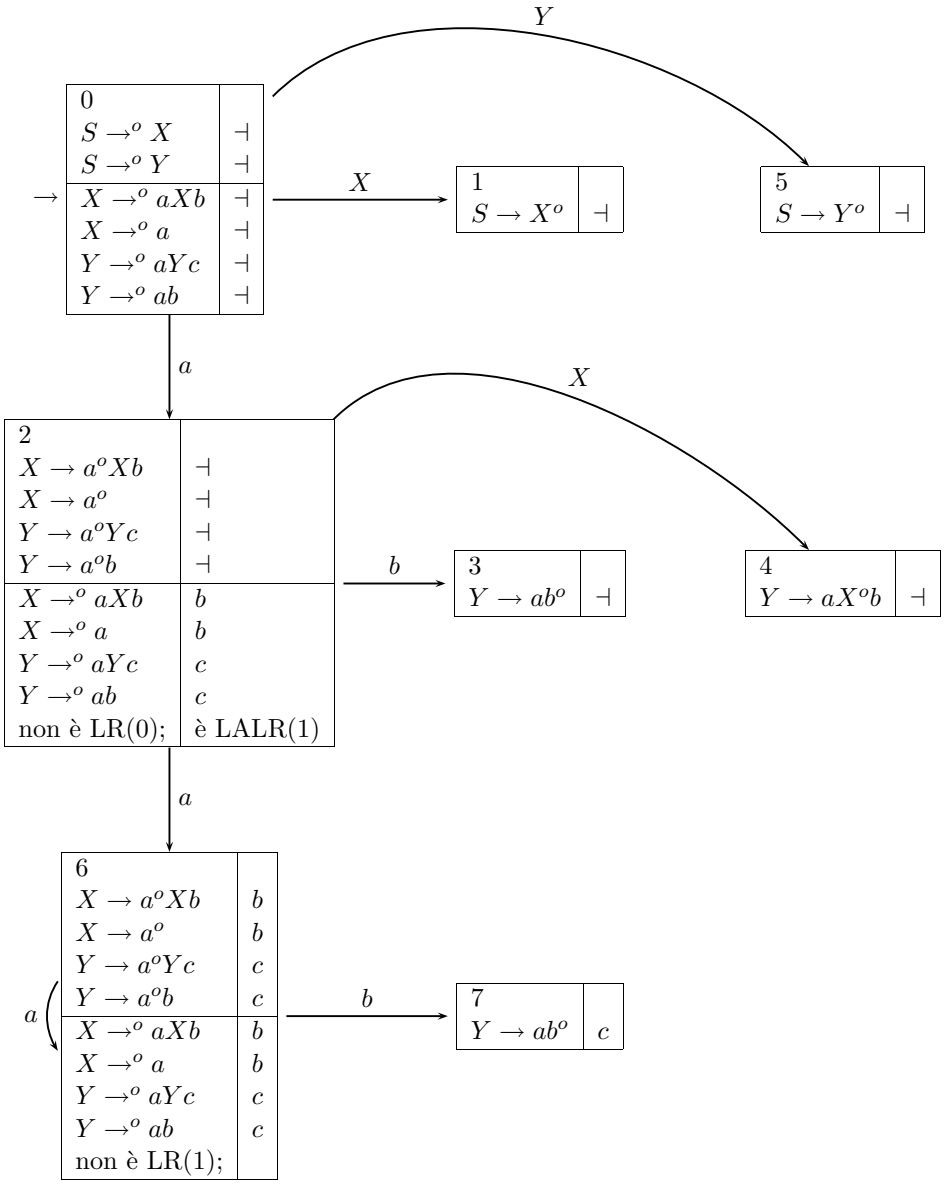
$$G_1 : \left| \begin{array}{l} S \to X \\ S \to Y \\ X \to aXb \\ X \to a \\ Y \to aYc \\ Y \to ab \end{array} \right.$$

   (a) Si costruisca il riconoscitore dei prefissi ascendenti di G_1

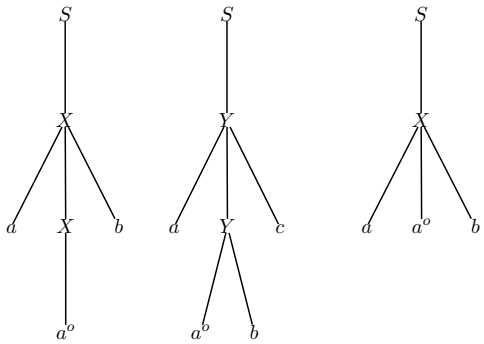
   (b) Si verifichi se essa è LR(0), LALR(1), LR(1)

   (c) Se necessario si trasformi la grammatica per ottenere una grammatica LR(1)

   Soluzione:
   (omessi alcuni stati)

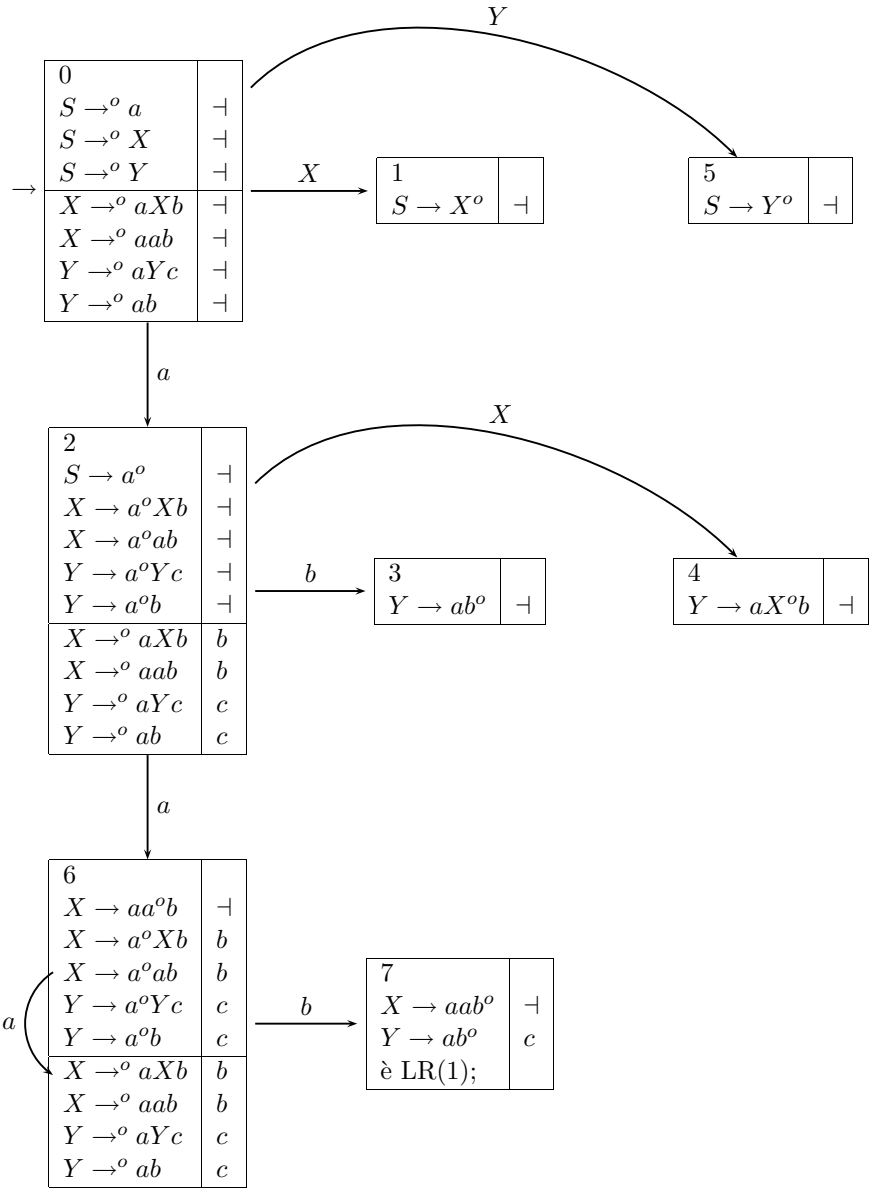

Y
0
$S \to^o X$ ⊣
$S \to^o Y$ ⊣
→
$X \to^o aXb$ ⊣
$X \to^o a$ ⊣
$Y \to^o aYc$ ⊣
$Y \to^o ab$ ⊣
X
1
$S \to X^o$ ⊣
5
$S \to Y^o$ ⊣
a
X
2
$X \to a^oXb$ ⊣
$X \to a^o$ ⊣
$Y \to a^oYc$ ⊣
$Y \to a^ob$ ⊣
$X \to^o aXb$ b
$X \to^o a$ b
$Y \to^o aYc$ c
$Y \to^o ab$ c
non è LR(0); è LALR(1)
b
3
$Y \to ab^o$ ⊣
4
$Y \to aX^ob$ ⊣
a
6
$X \to a^oXb$ b
$X \to a^o$ b
$Y \to a^oYc$ c
$Y \to a^ob$ c
$X \to^o aXb$ b
$X \to^o a$ b
$Y \to^o aYc$ c
$Y \to^o ab$ c
non è LR(1);
a
b
7
$Y \to ab^o$ c

Trasformazione della grammatica. Il conflitto riduzione- spostamento in 6 si può eliminare ritardando la riduzione $X \to a^o$. Infatti provocano conflitto i primi due alberi:



Il terzo albero, equivalente al primo, rimuove il conflitto.

$$G_2 : \left| \begin{array}{l} S \to a \\ S \to X \\ S \to Y \\ X \to aXb \\ X \to aab \\ Y \to aYc \\ Y \to ab \end{array} \right.$$


Y
0
$S \to^o a$ ⊣
$S \to^o X$ ⊣
$S \to^o Y$ ⊣
$X \to^o aXb$ ⊣
$X \to^o aab$ ⊣
$Y \to^o aYc$ ⊣
$Y \to^o ab$ ⊣
→
X
1
$S \to X^o$ ⊣
5
$S \to Y^o$ ⊣
a
X
2
$S \to a^o$ ⊣
$X \to a^oXb$ ⊣
$X \to a^oab$ ⊣
$Y \to a^oYc$ ⊣
$Y \to a^ob$ ⊣
$X \to^o aXb$ b
$X \to^o aab$ b
$Y \to^o aYc$ c
$Y \to^o ab$ c
b
3
$Y \to ab^o$ ⊣
4
$Y \to aX^ob$ ⊣
a
6
$X \to aa^ob$ ⊣
$X \to a^oXb$ b
$X \to a^oab$ b
$Y \to a^oYc$ c
$Y \to a^ob$ c
$X \to^o aXb$ b
$X \to^o aab$ b
$Y \to^o aYc$ c
$Y \to^o ab$ c
a
b
7
$X \to aab^o$ ⊣
$Y \to ab^o$ c
è LR(1);

5 Traduzione e semantica 20%

1. Il ling. sorgente L_1 (in forma astratta) contiene una lista di sequenze di numeri di una cifra, con l'eventuale segno meno:
   $S \rightarrow seq\ (\sharp seq\ )^*$
   $seq\ \rightarrow num\ (, num\ )^*$
   $num\ \rightarrow (-|\varepsilon)cifra$

   Esso va tradotto in un ling. pozzo L_2 di eguale alfabeto secondo le seguenti regole:

   (a) Numerate $1, 2, \ldots, i, \ldots$ le sequenze presenti nella stringa sorgente. Se una sequenza ha posto i dispari, i numeri sono ricopiati. Se una sequenza ha posto pari, i numeri sono cambiati di segno.

   (b) Se però la stringa sorgente contiene soltanto numeri negativi, essi sono tutti trasformati in 0

   Esempi:

sorgente	pozzo
$3, -2, -1\sharp 3, -5$	$3, -2, -1\sharp -3, 5$
$-3, -2, -1\sharp -3, -5$	$0, 0, 0\sharp 0, 0$

   È lasciata libertà di scegliere, in base alla convenienza, uno dei seguenti modi di progettare il traduttore:

   - Con uno schema di traduzione sintattica (senza attributi)
   - Con una grammatica a attributi

   (a) Si progetti il traduttore nel modo prescelto

   (b) Si disegnino gli alberi che calcolano la traduzione per i due esempi precedenti

   (c) Si verifichi se la traduzione può essere integrata con l'analisi sintattica.

   Soluzione

   - Mediante schema di traduzione puramente sintattico
     Occorre cambiare la sintassi in modo che traduzioni diverse siano associate a nonterminali diversi. Questa strada è più complicata della successiva.

sorgente G_1	pozzo G_2
$S \to N$	$S \to N$
	- - tutti negativi
$N \to seq_N(\sharp\ seq_N)^*$	$N \to seq_N(\sharp\ seq_N)^*$
$seq_N \to num_N(,\ num_N)^*$	$seq_N \to num_N(,\ num_N)^*$
$num_N \to -cifra$	$num_N \to 0$
$S \to M$	$S \to M$
	- - non tutti negativi
$M \to (seq_1\sharp\ seq_2\sharp)^* seq_{1+}(\sharp seq_2\sharp\ seq_1)^*$	$M \to (seq_1\sharp\ seq_2\sharp)^* seq_{1+}(\sharp seq_2\sharp\ seq_1)^*$
	- - seq_1 dispari, seq_2 pari
	- - seq_{1+} dispari e non tutta neg.
$M \to \ldots$	$\ldots$
	altre regole simili
$seq_1 \to num_1(,\ num_1)^*$	$seq_1 \to num_1(,\ num_1)^*$
$num_1 \to cifra$	$num_N \to cifra$
$num_1 \to -cifra$	$num_N \to -cifra$
$seq_{1+} \to (num_1\ ,)^* num_{1+}(,\ num_1)^*$	$seq_{1+} \to (num_1\ ,)^* num_{1+}(,\ num_1)^*$
$num_{1+} \to cifra$	$num_{1+} \to cifra$
$\ldots$	$\ldots$
$seq_2 \to num_2(,\ num_2)^*$	$seq_2 \to num_2(,\ num_2)^*$
$num_2 \to -cifra$	$num_N \to cifra$
$num_2 \to cifra$	$num_N \to -cifra$

- Mediante una grammatica a attributi.

Attr.	Commento
neg	sint.; è vero se tutti i num. sono negativi
eneg	come *neg* ma ereditato
pari	ered.; vero se una seq. è di posto pari
t	sint.; la traduzione
val	il valore di una cifra, attr. lessicale

La seguente grammatica opera su alberi astratti e trascura i separatori.

1.	$\overline{S}_0 \rightarrow S_1$	
	$eneg_1 \leftarrow neg_1$	$pari_1 \leftarrow true$
	$t_0 \leftarrow t_1$	
2.	$S_0 \rightarrow seq_1 S_2$	
	$eneg_1 \leftarrow eneg_0$	$eneg_2 \leftarrow eneg_0$
	$pari_1 \leftarrow pari_0$	$pari_1 \leftarrow \neg pari_0$
	$neg_0 \leftarrow neg_1 \wedge neg_2$	$t_0 \leftarrow t_1 CAT t_2$
3.	$S_0 \rightarrow seq_1$	
	$eneg_1 \leftarrow eneg_0$	
	$pari_1 \leftarrow pari_0$	
	$neg_0 \leftarrow neg_1$	$t_0 \leftarrow t_1$
4.	$seq_0 \rightarrow cifra_1 seq_2$	
	$neg_0 \leftarrow false$	$t_0 \leftarrow f(eneg_0, pari_0, val(cifra_1))CAT t_2$
5.	$seq_0 \rightarrow -cifra_1 seq_2$	
	$neg_0 \leftarrow neg_2$	$t_0 \leftarrow f(eneg_0, pari_0, val(-cifra_1))CAT t_2$
6.	$seq_0 \rightarrow cifra_1$	
	$neg_0 \leftarrow false$	$t_0 \leftarrow f(eneg_0, pari_0, val(cifra_1))CAT t_2$
7.	$seq_0 \rightarrow -cifra_1$	
	$neg_0 \leftarrow true$	$t_0 \leftarrow f(eneg_0, pari_0, val(-cifra_1))CAT t_2$

La funzione f è così definita:

$$f(negativita, parita, valore) \leftarrow \quad if(negativita = true)$$
$$0$$
$$else\ if(parita = false) valore\ else - valore$$

2. Data la grammatica a attributi

1.	$S_0 \to C_1 C_2$	
	$\varphi_1 \leftarrow 1$	$\varphi_2 \leftarrow f_1(\lambda_2)$
	$\alpha_0 \leftarrow f_2(\alpha_1, \alpha_2)$	
2.	$C_0 \to C_1 B_2$	
	$\varphi_1 \leftarrow f_3(\varphi_0)$	$\varphi_2 \leftarrow \varphi_0$
	$\lambda_0 \leftarrow f_4(\lambda_1)$	
	$\alpha_0 \leftarrow f_5(\alpha_1, \alpha_2)$	
3.	$C_0 \to B_1$	
	$\varphi_1 \leftarrow \varphi_0$	
	$\lambda_0 \leftarrow 8$	
	$\alpha_0 \leftarrow f_6(\alpha_1)$	
4.	$B \to a$	
	$\alpha_0 \leftarrow 5$	
5.	$B \to b$	
	$\alpha_0 \leftarrow f_7(\varphi_0)$	

(a) Si indichi quali attributi sono ereditati e quali sintetizzati

(b) Si verifichi se la grammatica è priva di errori

(c) Si disegnino i grafi delle dipendenze funzionali

(d) Si verifichi, riportando le spiegazioni, se la grammatica è

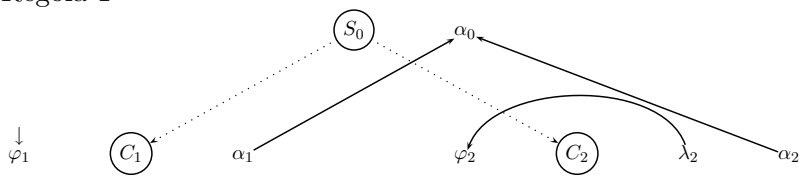
i. del tipo L

ii. valutabile a una scansione

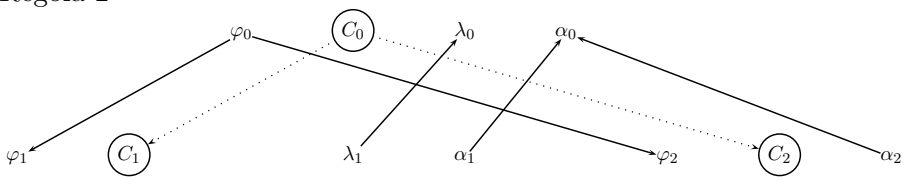
iii. Facoltativo: si scriva una procedura del valutatore semantico.

Soluzione:
I grafi delle dipendenze funzionali delle regole sono:

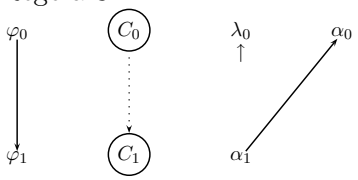
Regola 1



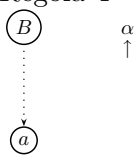
Regola 2



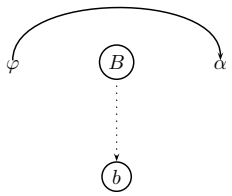
Regola 3



Regola 4



Regola 5



Il grafo 1 viola la condizione per la valutazione a 1 scansione, perché φ_2 dipende da λ_2 un attributo sintetizzato dello stesso nodo. Ciò impedirebbe al valutatore di conoscere gli attributi ereditati della radice C_2 prima di visitare ricorsivamente il sottoalbero di C_2 per calcolare gli attributi sintetizzati di C_2.

A maggiore ragione è violata la condizione L per la valutabilità da sinistra a destra.

Tuttavia è facile vedere che la valutazione può essere fatta con due scansioni, nel modo seguente.

(a) Si valuta l'attributo sint. λ che dipende soltanto da se stesso e è inizializzato nella regola 3.

(b) Calcolato λ, la dipendenza $\varphi_2 \leftarrow \lambda_2$ scompare in quanto λ_2 è un valore noto.

Poiché i rimanenti attributi (α, φ) soddisfano la condizione L, si possono calcolare con una semplice passata.