Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

prof. S. Crespi Reghizzi, prof.ssa L. Sbattella
(prof. Luca Breveglieri)

Prova scritta - 28 settembre 2006 - Parte I: Teoria

CON SOLUZIONI

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 e automi a pila
    3. analisi sintattica e parsificatori
    4. traduzione 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 sufficiente conoscenza 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. È data l'espressione regolare seguente:

$$R = (a \mid \varepsilon)\ (b \mid c)^*\ (b\ c \mid a\ b)^*$$

Si svolgano i punti seguenti:

(a) Si elenchino tutte le stringhe generate da R di lunghezza ≤ 2, indicando per ognuna l'eventuale grado di ambiguità (in quanti modi diversi viene generata)

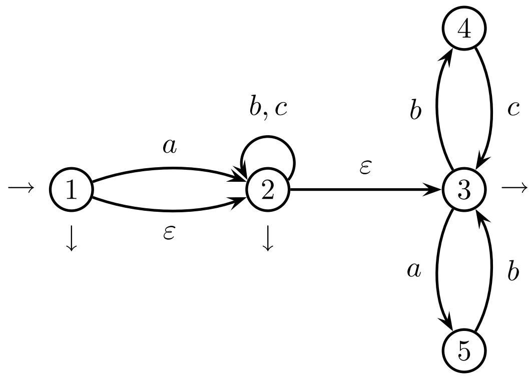
(b) Si tracci l'automa non-deterministico dell'espressione, con un metodo qualunque

(c) Si determinizzi l'automa non-deterministico mediante la costruzione dei sottinsiemi (non è necessario sia in forma minima)

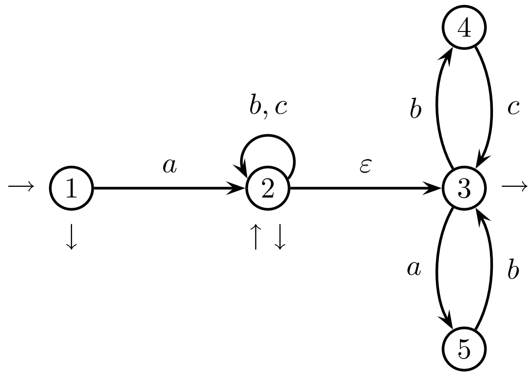
Soluzione

(a) Le stringhe di di R lunghezza ≤ 2 sono: ε, a, b, c, ab, ac, bc, bb, cb, cc. Di queste hanno ambiguità 2 le stringhe ab e bc.

(b) Metodo di Thompson, con qualche scorciatoia presa subito:



(c) Taglio archi ε (con propagazione archi entranti):



Taglio archi ε (con retrazione archi uscenti):

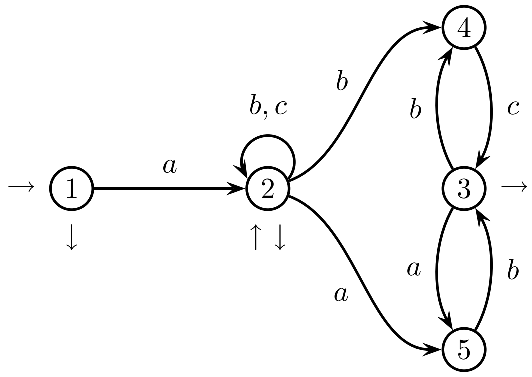


Tabella dei successori:

	a	b	c	finale ?
12	25	24	2	si
25	5	234	2	si
24	5	24	23	si
2	5	24	2	si
5	-	3	-	no
234	5	24	23	si
23	5	24	2	si
3	5	4	-	si
4	-	-	3	no

ed è evidente che l'automa deterministico non è in forma minima, perché nella tabella ci sono righe identiche. Al lettore il compito di minimizzarlo.

2. Si considerino le stringhe di alfabeto $\Sigma = \{a, b, c\}$, e i due vincoli seguenti:

   (a) se nella stringa ci sono lettere a, non sono adiacenti

   (b) se nella stringa ci sono una o più lettere b, almeno una di esse è seguita da almeno una lettera c, a distanza qualunque

   Si svolgano i punti seguenti:

   (a) Si scrivano le due espressioni regolari (non importa se ambigue) che generano le stringhe corrispondenti a ciascun vincolo, usando solo concatenamento, unione e stella (o croce)

   (b) Applicando un metodo algoritmico, si trovi l'automa deterministico che riconosce il linguaggio le cui stringhe soddisfano entrambi i vincoli

   (c) (facoltativo) Si minimizzi il numero di stati dell'automa deterministico ricavato al punto precedente.

Soluzione

(a) Non ambigua:

$$R_a = (b \mid c)^* \left(a \ (b \mid c)^+\right)^* \ (a \mid \varepsilon)$$

Se c'è a mette senz'altro b o c, a meno che la stringa non sia finita.

Fortemente ambigua:

$$R_b = (a \mid c)^* \ (\Sigma^* \ b \ \Sigma^* \ c \ \Sigma^*)^*$$

Se c'è b impone prima o poi c, e ripete 0 o più volte per lasciare b facoltativa. Molto semplice, ovviamente molto ambigua: non dice quale b sarà quella seguita da c (vedi anche sotto).

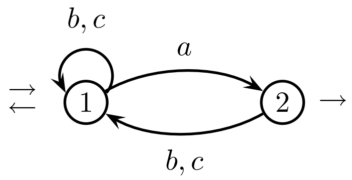
Comunque, se la si vuole non ambigua, eccone una equivalente:

$$R'_b = (a \mid c)^* \ (b \ (a \mid b)^* \ c \ (a \mid b \mid c)^* \mid \varepsilon)$$

la quale dice che se c'è una b seguita da c, si può sempre pensare che sia la prima.

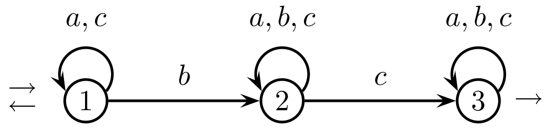
(b) Conviene trovare i due automi det. di L_a e L_b e farne l'intersezione (prodotto cartesiano).

Automa A_a:

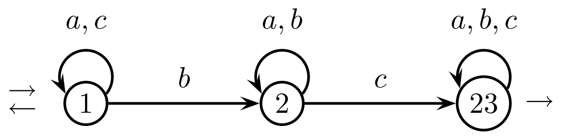


Automa A_b:

Conviene prima farlo non modo non-deterministico, è più naturale



perché non dice quale b sarà seguita da c. Ma se ci si pensa, può sempre essere la prima, dunque determinizzando in modo intuitivo o con i sottinsiemi:



Per inciso, questo automa corrisponde esattamente a R'_b.

Ora basta usare la costruzione del prodotto cartesiano. Vengono 6 stati, è fattibile a mano, ma non è detto sia in forma minima.

(c) Si usi il solito metodo basato sulla tabella delle equivalenze.

2 Grammatiche libere e automi a pila 20%

1. È data la grammatica G seguente:

$$
\begin{array}{lcl}
S & \rightarrow & a\,X \;|\; X \\
X & \rightarrow & a\,X\,b \;|\; b \;|\; \varepsilon
\end{array}
$$

Si svolgano i punti seguenti:

(a) Si descrivano con precisione, a parole o con una formula logica, le frasi del linguaggio $L(G)$.

(b) Si individuino le frasi ambigue, mostrandone gli alberi sintattici.

(c) Si progetti una grammatica equivalente a G, non ambigua.

(d) Si progetti un automa a pila deterministico per riconoscere il linguaggio $L(G)$; se serve si immagini la stringa sia terminata da uno end-marker.

Soluzione

(a) Ecco l'intero linguaggio:

$$L = \left\{ a^h b^k \;|\quad h, k \geq 0 \;\wedge\; 0 \leq |h - k| \leq 1 \right\}$$

Sono nidi di tipo ab, dove il numero di a non differisce dal numero di b di più di un'unità.

(b) L è ambiguo in quanto le frasi bilanciate (tipo $a^n b^n$, con $n \geq 1$) si possono originare in due modi. Esempio con $aabb$:

$$S \Rightarrow a\ X \Rightarrow a\ a\ X\ b \Rightarrow a\ a\ b\ b$$

oppure

$$S \Rightarrow X \Rightarrow a\ X\ b \Rightarrow a\ a\ X\ b\ b \Rightarrow a\ a\ b\ b$$

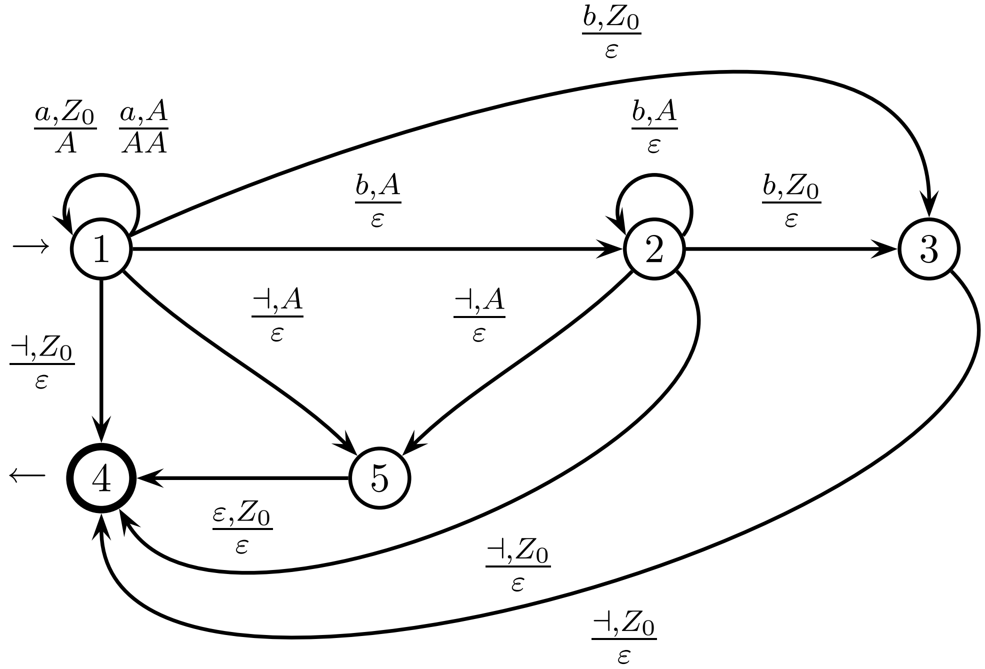
Alberi sintattici, ovvi.

(c) Basta separare i tre casi: $h = k$, $h = k + 1$ e $h + 1 = k$ (con $h, k \geq 0$). Eccoli:

$$S \rightarrow a\ S\ b \;|\; a \;|\; b \;|\; \varepsilon$$

Ora le stringhe bilanciate si formano in un solo modo. Ci saranno pure altre soluzioni.

(d) Riconoscimento a stato finale, end-marker $\dashv$.



Verifica di corettezza:

- riconosce $\varepsilon \dashv$ con il percorso $\xrightarrow[Z_0]{\dashv} 4$
- riconosce $a \dashv$ con il percorso $1 \xrightarrow[Z_0]{a} 1 \xrightarrow[\substack{A\\Z_0}]{\dashv} 5 \xrightarrow[Z_0]{\varepsilon} 4$
- riconosce $b \dashv$ con il percorso $1 \xrightarrow[Z_0]{b} 3 \xrightarrow[Z_0]{\dashv} 4$
- riconosce $ab \dashv$ con il percorso $1 \xrightarrow[Z_0]{a} 1 \xrightarrow[\substack{A\\Z_0}]{b} 2 \xrightarrow[Z_0]{\dashv} 5 \xrightarrow[Z_0]{\varepsilon} 4$
- riconosce $aab \dashv$ con il percorso $1 \xrightarrow{a} 1 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{\dashv} 5 \xrightarrow{\varepsilon} 4$
- riconosce $abb \dashv$ con il percorso $1 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{\dashv} 4$
- riconosce $aabb \dashv$ con il percorso $1 \xrightarrow{a} 1 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 2 \xrightarrow{\dashv} 5 \xrightarrow{\varepsilon} 4$

e ormai ben si vede come generalizzare: si passa per 3 se il numero di b eccede quello di a; si passa per 5 se il numero di b uguaglia o è inferiore a quello di a; il caso iniziale fà eccezione.

Può darsi si possa fare con riconoscimento a pila vuota, bisogna pensarci.

2. Si progetti la grammatica EBNF non ambigua che modella il costrutto `if-then-else` del linguaggio C, un po' semplificato. Sono presenti i concetti seguenti:

   - ci sono variabili e numeri interi, denotati come si fa in C
   - ci sono espressioni, contenenti variabili, numeri interi positivi o negativi, gli operatori "+" e "∗" in forma infissa, ed eventuali sottoespressioni racchiuse tra parentesi tonde "(" e ")"; la moltiplicazione precede l'addizione
   - c'è l'assegnamento:

     `var. = espr.;`

     sempre terminato da ";"
   - c'è il blocco (lista) di assegnamenti, non vuoto e racchiuso tra parentesi graffe "{" e "}"; le graffe sono facoltative se il blocco contiene un solo assegnamento
   - ci sono gli operatori relazionali "==", "! =", "<" e ">"
   - la condizione di `if` è un'espressione relazionale:

     `(espr. op.rel. espr.)`

     tra parentesi tonde
   - il ramo `then` dello `if` è obbligatorio e introduce un blocco di assegamenti, come descritto sopra
   - il ramo `else` dello `if` è facoltativo, e se è presente introduce o un blocco di assegamenti, come descritto sopra, o un costrutto `if-then-else`

   Esempio:

```
if (a + 2 < 10 * b) {
  a = c + 1;
} else if (b == 5)
  a = b;
else {
  c = 0;
}
```

   Si scriva la grammatica G in questione (in forma EBNF non ambigua).

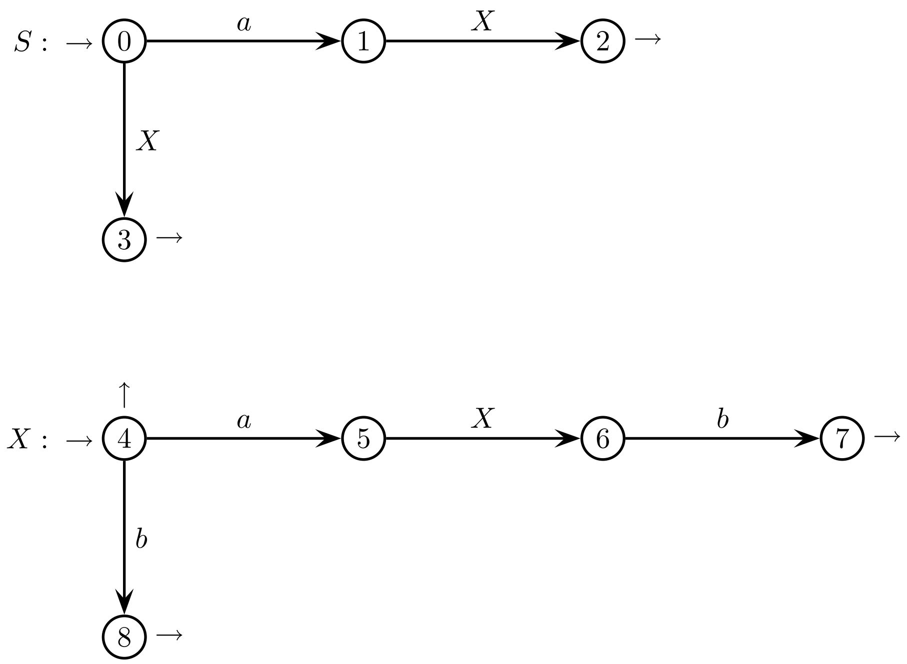
Soluzione

È abbastanza standard. Ecco una soluzione accettabile:

$$
\begin{aligned}
V &\rightarrow [\text{`A'} - \text{`Z'}]\,([\text{`A'} - \text{`Z'}] \mid [\text{`0'} - \text{`9'}])^* \\
N &\rightarrow [\text{`1'} - \text{`9'}]\,[\text{`0'} - \text{`9'}]^* \\
E &\rightarrow T\ (\text{`+'}\ T)^* \\
T &\rightarrow F\ (\text{`}{*}\text{'}\ F)^* \\
F &\rightarrow V \mid N \mid \text{`('}\ E\ \text{`)'} \\
A &\rightarrow V\ \text{`='}\ E\ \text{`;'} \\
B &\rightarrow A \mid \text{`\{'}\ A^+\ \text{`\}'} \\
O &\rightarrow \text{`=='} \mid \text{`!='} \mid \text{`<'} \mid \text{`>'} \\
R &\rightarrow \text{`('}\ E\ O\ E\ \text{`)'} \\
\langle IF \rangle &\rightarrow \text{`}\mathbf{if}\text{'}\ R\ B\ (\varepsilon \mid \text{`}\mathbf{else}\text{'}\,(B \mid \langle IF \rangle))
\end{aligned}
$$

3 Analisi sintattica e parsificatori 20%

1. È data una grammatica mediante la rete di macchine seguente:



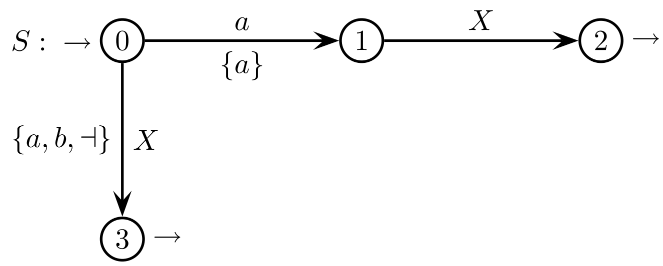
Si svolgano i punti seguenti:

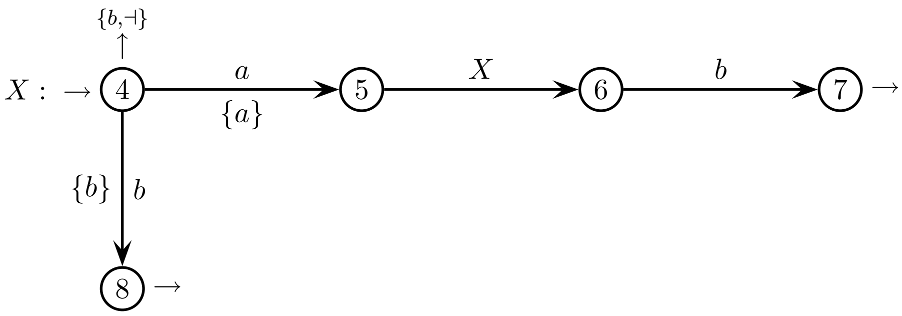
(a) Si calcolino gli insiemi guida nei punti rilevanti della rete e si verifichi se la rete delle macchine sia $LL(1)$.

(b) Se necessario, si verifichi se la grammatica G sia $LL(k)$ per un valore $k > 1$.

Soluzione

(a) Gli insiemi guida rilevanti (dove ci sono biforcazioni) sono sul grafo.





(b) Gli stati 0 e 4 violano la condizione $LL(1)$, il primo a causa della presenza del carattere a negli insiemi guida di entrambi gli archi uscenti da 0, il secondo a causa della presenza di b sull'arco $4 \rightarrow 8$ e sulla freccia finale dello stato 4.

(c) È piuttosto evidente che la violazione in 0 persiste con $k > 1$, perché sia il ramo aX sia il ramo X possono iniziare con una stringa contenente tante a. Lo stesso problema sussiste nello stato 4.

Un altro modo per comprendere l'origine del conflitto, è l'esame dell'ambiguità. La frase ab è ambigua, perché può essere riconosciuta da due calcoli:

$$0 \rightarrow 1 \rightarrow \ (\rightarrow 4 \rightarrow 8 \rightarrow) \ \rightarrow 2$$

$$0 \rightarrow \ (\rightarrow 4 \rightarrow 5 \rightarrow \ (\rightarrow 4 \rightarrow) \ \rightarrow 8 \rightarrow) \ \rightarrow 6 \rightarrow 7 \rightarrow) \ \rightarrow 3$$

Di conseguenza la grammatica non è $LL(k)$ per nessun valore di k.

2. È data la grammatica G seguente:

$$\begin{aligned} S &\rightarrow X\, b \mid X \\ X &\rightarrow a\, X\, b \mid a \mid \varepsilon \end{aligned}$$

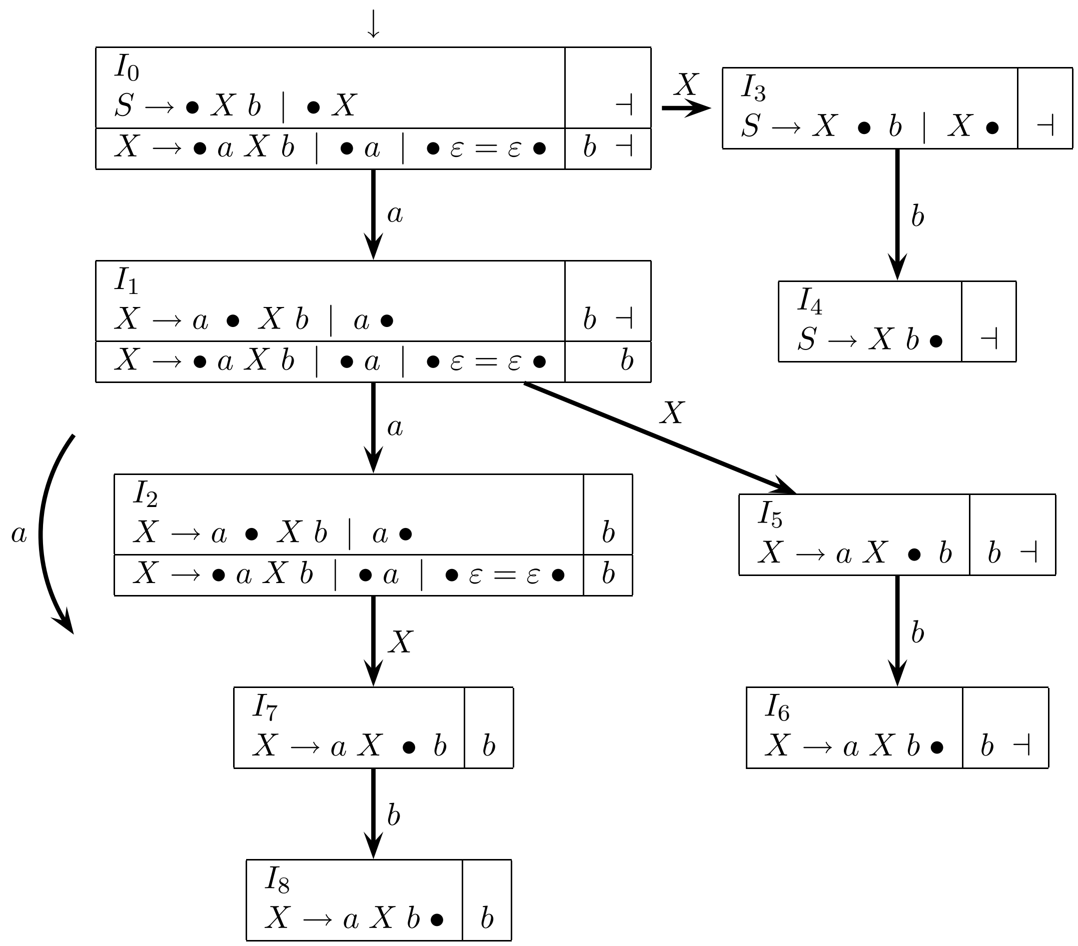
Si svolgano i punti seguenti:

(a) Per G si costruisca l'automa pilota $LR(1)$ (cioè il riconoscitore dei prefissi).

(b) Per ogni macrostato del pilota si indichi se per $k = 1$ vi siano conflitti, spostamento-riduzione o riduzione-riduzione.

Soluzione

(a) Ecco la macchina pilota $LR(1)$:



(b) Analisi dei macrostati. Si esaminano quelli contenenti dueuno spostamento e una riduzione oppure due riduzioni.

Macrostato	Potenziale conflitto
I_0	Tra la riduzione $X \to \varepsilon\ \bullet\ \{b\ \dashv\}$ e lo spostamento di a la scelta è deterministica.
I_1	Tra le riduzioni $X \to a\ \bullet\ \{b\ \dashv\}$ e $X \to \varepsilon\ \bullet\ \{b\}$ la scelta non è deterministica, mentre con lo spostamente di a la scelta è deterministica.
I_2	Tra le riduzioni $X \to a\ \bullet\ \{b\}$ e $X \to \varepsilon\ \bullet\ \{b\}$ la scelta non è deterministica.
I_3	Tra la riduzione $S \to X\ \bullet\ \{\dashv\}$ e lo spostamento di b la scelta è deterministica.

Commento: la grammatica G è palesemente ambigua (le stringhe tipo $a^n b^n$ sono generate in due modi; si veda anche l'esercizio n. 2.1), dunque non può essere $LR(k)$ per nessun $k \geq 0$. La forma non ambigua equivalente data nell'esercizio 2.1 è palesemente $LR(1)$ (ma non è $LR(0)$ perché contiene una ε-produzione).

4 Traduzione e analisi semantica 20%

1. Si consideri il linguaggio sorgente seguente:

$$L = a^+ \; b \; a^+ \; \cup \; a^+ \; c \; a^+$$

e la funzione di traduzione seguente:

$$\begin{array}{rcl} \tau(a^m \; b \; a^n) & = & d^m \\ \tau(a^m \; c \; a^n) & = & d^n \end{array}$$

con $m, n \geq 1$.

Esempi: $\tau(a \; a \; a \; b \; a) = d \; d \; d$ $\tau(a \; a \; a \; c \; a) = d$

Si svolgano i punti seguenti:

(a) Si scriva l'espressione regolare della traduzione τ sopra definita.

(b) Si disegni un automa traduttore finito che calcola la traduzione τ.

(c) Si stabilisca se il traduttore finito dato al punto precedente sia deterministico oppure no.

(d) (Facoltativo) Si argomenti brevemente se sia possibile effettuare la traduzione τ in modo deterministico.

Soluzione

(a) Espressione regolare di traduzione:

$$\left(\frac{a}{d}\right)^+ \; \frac{b}{\varepsilon} \; \left(\frac{a}{\varepsilon}\right)^+ \; \cup \; \left(\frac{a}{\varepsilon}\right)^+ \; \frac{c}{\varepsilon} \; \left(\frac{a}{d}\right)^+$$

Chi non avesse ricordato (o non avese studiato) che cosa sia un'espressione regolare di traduzione, avrebbe potuto derivare la grammatica lineare a destra equivalente all'espressione regolare sorgente, e trasformare la grammatica in uno schema sintattico di traduzione (sempre lineare a destra) per poi ricostruire l'espressione regolare di traduzione. Esempio:

$$\begin{array}{rcl} S & \rightarrow & B \mid C \\ B & \rightarrow & a \; B \mid a \; b \; A' \\ C & \rightarrow & a \; C \mid a \; c \; A'' \\ A' & \rightarrow & a \; A' \mid a \\ A'' & \rightarrow & a \; A'' \mid a \end{array}$$

distinguendo tra A' e A'' per preparare il terreno a due traduzioni diverse di a, da cui:

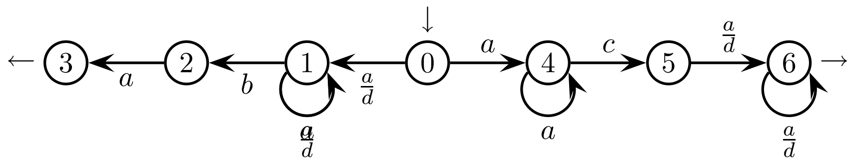
$$\begin{array}{rcl} S & \rightarrow & B \mid C \\ B & \rightarrow & a \; \{ \; d \; \} \; B \mid a \; \{ \; d \; \} \; b \; A' \\ C & \rightarrow & a \; C \mid a \; c \; A'' \\ A' & \rightarrow & a \; A' \mid a \\ A'' & \rightarrow & a \; \{ \; d \; \} \; A'' \mid \{ \; d \; \} \; a \end{array}$$

e tornando indietro:

$$(a\ \{\ d\ \})^+\ \ b\ a^+\ \ |\ \ a^+\ c\ \ (a\ \{\ d\ \})^+$$

ovvero l'espressione di traduzione già data prima, seppure scritta in modo notazionalmente diverso.

(b) Trasduttore finito:



(c) L'automa soggiacente al traduttore non è deterministico nello stato 0, dunque neanche il trasduttore.

(d) Sì, si può fare in modo det. con un automa trasduttore a pila. Il traduttore a pila deterministico opera con la strategia seguente.

Data la stringa sorgente $a^m e a^m$ con $e \in \{b, c\}$, esso legge e impila le m lettere a. Poi, leggendo il carattere e entra nello stato B o C a seconda che sia $e = b$ o $e = c$. Nello stato B la macchina legge le n lettere a e al termine emette una lettera d per ogni simbolo a che toglie dalla pila (in totale ne toglie m).

D'altra parte, nello stato C l'automa legge le n lettere a emettendo una d per ciascuna.

Per maggiore precisione, la macchina deve controllare che vi sia almeno una lettera a sia prima di e sia dopo e.

Osservazione: è dubbio che si possa fare con uno schema di traduzione deterministico $LL(k)$; probabilmente no.
Invece, esprimendo la traduzione nella forma normale postfissa, si può ottenere uno schema di traduzione la cui grammatica sorgente è $LR(1)$. Eccolo, la parte pozzo come al solito è espressa racchiudendola tra parentesi graffe:

$$\begin{array}{rcl}
S & \rightarrow & X\ \mid\ Y \\
X & \rightarrow & a\,X\ A\ \mid\ a\,b\,W\ A \\
W & \rightarrow & a\,W\ \mid\ a \\
A & \rightarrow & \varepsilon\,\{d\} \\
Y & \rightarrow & a\,Y\ \mid\ a\ c\ Z \\
Z & \rightarrow & a\ Z\ \varepsilon\,\{d\}\ \mid\ a\,\{d\}
\end{array}$$

Occorre costruire l'automa pilota della grammatica sorgente, per verificare che sia $LR(1)$. Sembra ragionevole supporre che il pilota $LR(1)$ si possa comportare come l'automa a pila det. descritto sopra (comunque bisogna verificare).

2. Si consideri una tabella contenente una lista di numeri interi positivi tutti distinti e, accanto a ogni numero, un insieme di numeri che danno come somma il numero. In altre parole, si tratta di una base dati relazionale il cui primo attributo è la chiave NUM e il secondo attributo ADD è un insieme (non vuoto) di numeri.

   Si deve progettare una grammatica con attributi che, per ogni coppia della relazione, verifichi se i numeri della lista ADD danno NUM come somma, e per le tuple che superano il controllo, costruisca la relazione in prima forma normale; vedi sotto:

   Tabella sorgente

NUM	ADD
12	$\{2, 10\}$
6	$\{2, 4\}$
9	$\{2\}$
30	$\{2, 3, 25\}$

   $\Longrightarrow$

   Tabella pozzo

NUM	$FATT$
12	2
12	10
6	2
6	4
30	2
30	3
30	25

   È dato un supporto sintattico G che genera la tabella di partenza. Gli elementi NUM sono numeri interi, qui trattati come terminali.

$$G \begin{cases} S & \rightarrow & T\ S \\ S & \rightarrow & \varepsilon \\ T & \rightarrow & NUM \text{ ``\{'' } ADD \text{ ``\}''} \\ ADD & \rightarrow & NUM\ ADD \\ ADD & \rightarrow & NUM \end{cases}$$

   Ci devono essere almeno gli attributi seguenti:

   - il valore v di NUM, attributo disponibile inizialmente
   - un predicato di correttezza "φ", vero se, e solo se, la somma è corretta
   - una stringa di traduzione "τ", associata alla radice dell'albero, che contiene la tabella normalizzata nella forma illustrata sotto:

     $12, 2/12, 10/6, 2/6, 4/30, 2/30, 3/30, 25/$

   Si possono aggiungere altri attributi, come sembra necessario od opportuno. Ecco i punti da svolgere:

   (a) Elencare gli attributi, con il rispettivo tipo e significato.

   (b) Scrivere le funzioni semantiche che calcolano gli attributi (alle pagine successive sono già pronti gli schemi da compilare).

   (c) Disegnare i grafi delle dipendenze funzionali tra attributi, per ciascuna produzione separatamente.

   (d) Stabilire se la grammatica sia di tipo a una sola scansione.

   (e) Stabilire se la grammatica sia di tipo L.

tipo	nome	nonterminali	dominio	significato
		già dati		
sx	v	NUM	numero int.	valore di NUM
	φ		V / F	tupla corretta
sx	τ	S	stringa	tabella in forma norm.
		da aggiungere o estendere		
dx	φ	ADD	V / F	tupla corretta
sx	τ	S, T, ADD	stringa	tabella in forma norm.
sx	σ	ADD	numero int.	somma addendi
dx	χ	ADD	numero int.	valore chiave NUM

Funzioni semantiche:

sintassi	*funzioni semantiche*
$S_0 \rightarrow T_1\, S_2$	$\tau_0 = CAT\,(\tau_1, \tau_2)$
$S_0 \rightarrow \varepsilon$	$\tau_0 = \varepsilon$
$T_0 \rightarrow NUM_1$ "{" ADD_2 "}"	$\varphi_2 = (\textbf{if } (\sigma_2 == v_1) \textbf{ then} \text{ true } \textbf{else} \text{ false})$ $\chi_2 = v_1$ $\tau_0 = \tau_2$

sintassi	*funzioni semantiche*
$ADD_0 \to NUM_1\ ADD_2$	$\sigma_0 = v_1 + \sigma_2$ $\varphi_2 = \varphi_0$ $\chi_2 = \chi_0$ **if** $(\varphi_0 == \text{true})$ **then** $\tau_0 = CAT\,(\chi_0, \text{`,'}\, v_1, \tau_2, \text{`/'}\,)$ **else** $\tau_0 = \varepsilon$
$ADD_0 \to NUM_1$	$\sigma_0 = v_1$ **if** $(\varphi_0 == \text{true})$ **then** $\tau_0 = CAT\,(\chi_0, \text{`,'}\, v_1, \text{`/'})$ **else** $\tau_0 = \varepsilon$

Soluzione

(a) Concettualmente il processo di traduzione può essere organizzato in due fasi.

Fase di controllo: Verifica tupla per tupla che l'insieme degli addendi sia corretto, marcando le tuple corrette con l'attributo φ.

Fase di scrittura: Visita l'albero, inserendo nell'attributo stringa τ i valori delle sole tuple corrette, nel formato prescritto.

Introduciamo l'attributo sinistro σ per calcolare la somma degli addendi.
Scegliamo di usare φ come attributo destro, al fine di propagarlo a tutti gli addendi da stampare.
Introduciamo l'attributo χ, destro, per propagare il valore del numero, allo scopo di stamparlo accanto a ogni addendo.

(b) Da fare ... (facile).

(c) Non è a una sola scansione, perché per calcolare φ, ereditato, e propagarlo verso il basso, bisogna prima avere calcolato σ, sintetizzato; chiaramente occorre più di una passata (il lettore trovi quante ne servono).

(d) A maggior ragione (punto precedente), non è di tipo L.