Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 27 giugno 2009 - 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 sostenere 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), rispondendo alle domande obbligatorie (si noti che il punteggio pieno può essere ottenuto solo rispondendo alle parti non obbligatorie).
- 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 dato il programma seguente:

```
a: read (i)
b: while (i ≠ 0)
c:     read (i)
d:     if (i > 0)
e:         then
               print (i)
f:         else
               print (−i)
               break
       end if
   end while
```

   Si considerino le etichette come i caratteri dell'alfabeto terminale $\Sigma = \{a, b, c, d, e, f\}$. L'istruzione break provoca l'uscita immediata dal ciclo while.

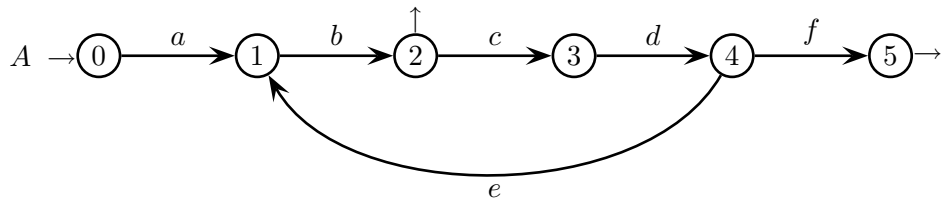
   Si risponda alle domande seguenti:

   (a) Si formalizzi il grafo di flusso del programma come un automa finito A che riconosce l'insieme $L \subseteq \Sigma^+$ delle sequenze (p. es. $a\,b\,c\,d\,f$) che corrispondono a una possibile esecuzione del programma. Si disegni il grafo dell'automa A.

   (b) Si calcoli in modo algoritmico l'espressione regolare R del linguaggio L.

Soluzione

(a) Ecco l'automa A:



(b) Ecco l'espressione regolare R. Si sceglie di risolvere le equazioni insiemistiche del linguaggio.

$$
\begin{aligned}
0 &= a\,1 \\
1 &= b\,2 \mid b \\
2 &= c\,3 \\
3 &= d\,4 \\
4 &= e\,1 \mid f
\end{aligned}
$$

da cui:

$$
\begin{aligned}
0 &= a\,1 \\
1 &= b\,2 \mid b \\
2 &= c\,3 \\
3 &= d\,e\,1 \mid d\,f
\end{aligned}
$$

e poi:

$$
\begin{aligned}
0 &= a\,1 \\
1 &= b\,2 \mid b \\
2 &= c\,d\,e\,1 \mid c\,d\,f
\end{aligned}
$$

e ancora:

$$
\begin{aligned}
0 &= a\,1 \\
1 &= b\,c\,d\,e\,1 \mid b\,c\,d\,f \mid b
\end{aligned}
$$

Con l'identità di Arden si ottiene:

$$1 = (\,b\,c\,d\,e\,)^*\,(\,b\,c\,d\,f \mid b\,)$$

e infine:

$$0 = a\,(\,b\,c\,d\,e\,)^*\,(\,b\,c\,d\,f \mid b\,)$$

che è l'espressione regolare R di L.

2. Per l'alfabeto $\{a, b, c\}$, si prenda il linguaggio regolare L delle stringhe x definite dalle condizioni seguenti:

$$x = u\ a\ v\ b\ w$$

dove si ha

$$u \in \{b, c\}^+ \quad v \in \{a, b, c\}^* \quad w \in \{a, c\}^+ \quad |u| \text{ e } |w| \text{ sono entrambi dispari}$$

Si osservi che la stringa u non contiene la lettera a e la stringa w non contiene la lettera b.

Esempi:

$$b\,a\,c\,b\,a\,c\,a = \underbrace{b}_{u}\ a\ \underbrace{c}_{v}\ b\ \underbrace{a\,c\,a}_{w}$$

$$b\,c\,c\,a\,b\,b\,a\,b\,c = \underbrace{b\,c\,c}_{u}\ a\ \underbrace{b\,b\,a}_{v}\ b\ \underbrace{c}_{w}$$

Controesempi:

$$a\,b \qquad\qquad b\,b\ a\ b\ a\,a$$

Si risponda alle domande seguenti:

(a) Si costruisca un automa finito A riconoscitore di L, a scelta deterministico o non. Si spieghi brevemente il funzionamento dell'automa A.

(b) (facoltativa) Si verifichi se l'automa A sia quello deterministico minimo. Se non lo fosse, si costruisca l'automa deterministico minimo A'.

Soluzione

(a) Un modo intuitivo per rispondere è di considerare i vincoli imposti alle frasi e ricondurli a casi ben noti di linguaggi. Dall'enunciato si vede che (i) la prima lettera a deve essere in posizione pari (non 0) e (ii) l'ultima lettera b deve esssere in posizione pari (non 0) rispetto alla fine della stringa.

Il vincolo (i) si esprime come:

$$u \in (b \mid c) \left((b \mid c)^2 \right)^* = U$$

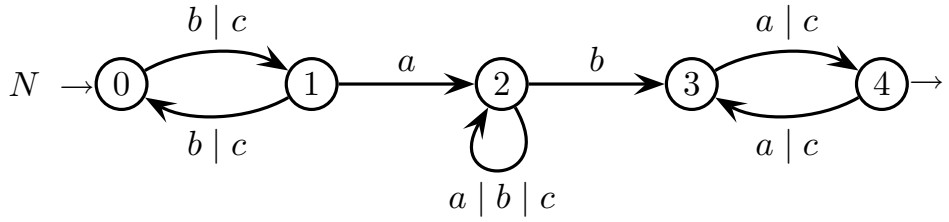
Analogamente, il vincolo (ii) si esprime come:

$$w \in (a \mid c) \left((a \mid c)^2 \right)^* = W$$

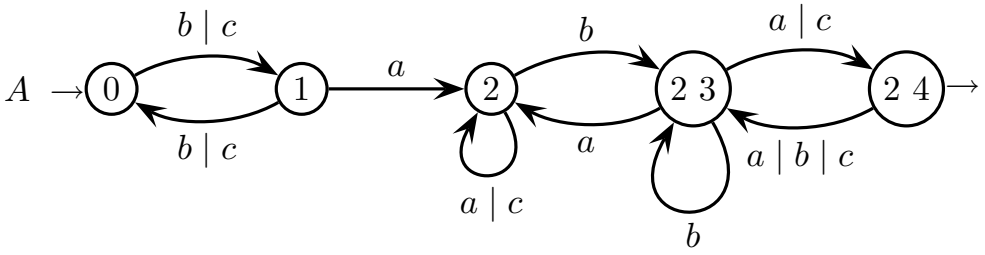
Poiché le stringhe v dell'enunciato sono arbitrarie, il linguaggio è

$$L = U\ a\ \Sigma^*\ b\ W$$

Di qui è facile costruire un automa N indeterministico concatenando le parti:



(b) Tramite la costruzione dei sottinsiemi, dall'automa N si ottiene l'automa deterministico equivalente A. Eccolo:



Facilmente si vede che tutti gli stati sono distinguibili, pertanto l'automa A è minimo.

2 Grammatiche libere e automi a pila 20%

1. Le stringhe di un linguaggio libero di alfabeto $\{a, b\}$ sono costituite da un numero multiplo di quattro di gruppi di lettere a consecutive, ossia ci sono $4\,n$ gruppi per qualche $n > 0$. Ogni gruppo di lettere a è separato da quello successivo da una lettera b.

   Nella prima metà dei gruppi (per i primi $2\,n$ gruppi) ogni gruppo in posizione dispari (il primo, il terzo, ecc) è meno lungo (composto da un numero inferiore di a) del gruppo immediatamente successivo (che occupa una posizione pari).

   Nella seconda metà dei gruppi (per gli ultimi $2\,n$ gruppi) ogni gruppo in posizione dispari (il primo, il terzo, ecc) è più lungo (composto da un numero maggiore di a) del gruppo immediatamente successivo (che occupa una posizione pari).

   Ecco un esempio di stringa appartenente al linguaggio (con $n = 2$ ossia 8 gruppi):

   $$\underbrace{\overbrace{a^2}^{\text{gruppo 1}}\; b \;\overbrace{a^5}^{\text{gruppo 2}}\; b\,a^7\,b\,a^9}_{\text{prima metà}}\underbrace{b\,a^6\,b\,a^4\,b\,a^3\,b\,a^2}_{\text{seconda metà}}$$

   Si risponda alle domande seguenti:

   (a) Si definisca una grammatica G, possibilmente non ambigua, che genera il linguaggio sopra descritto.

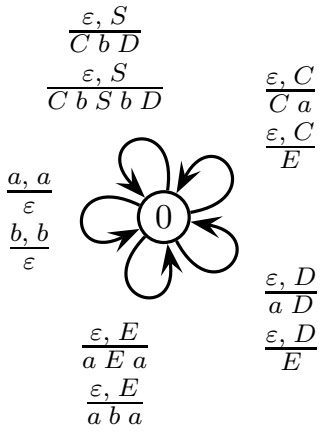
   (b) (facoltativa) Dalla grammatica G si ricavi in modo sistematico un automa a pila A che riconosce il linguaggio. Si dica se l'automa A è deterministico, giustificando la risposta.

Soluzione

(a) Ecco la grammatica G (assioma S):

$$G \begin{cases} S & \to & C\ b\ D \mid C\ b\ S\ b\ D \\ C & \to & C\ a \mid E \\ D & \to & a\ D \mid E \\ E & \to & a\ E\ a \mid a\ b\ a \end{cases}$$

(b) Ecco l'automa a pila A. Applicando il procedimento illustrato nel paragrafo 4.1.2 del testo si ottiene l'usuale automa a pila nondeterministico "a margherita" sull'alfabeto di pila $\{S, C, D, E, a, b\}$, di cui si riporta la parte rilevante.



L'automa A riconosce a pila vuota ed è (pesantemente) indeterministico.

2. Si scriva una grammatica G EBNF non ambigua del seguente linguaggio di script per il calcolo numerico:

   - le variabili non sono dichiarate e non hanno tipo esplicito, ma hanno tutte un valore numerico; esse sono scalari o vettori con uno o più indici, racchiusi tra una coppia di parentesi quadre
   - gli identificatori delle variabili sono indicati con il simbolo terminale *id*; esempi:

     ```
     id    id[1]    id[2, 3, 4]
     ```

   - ci sono due tipi di espressione:
     - aritmetica, composta da costanti, variabili e operatori aritmetici
     - logica (condizione), composta da espressioni aritmetiche con operatori di relazione e logici
   - le costanti numeriche sono indicate dal simbolo terminale const
   - gli operatori aritmetici, relazionali e logici sono elencati nel seguito per gruppi di priorità decrescente; la priorità deve essere resa dalla grammatica
     - $\times$ e $/$, prodotto e divisione
     - $+$ e $-$, addizione e sottrazione
     - $>$ e $\geq$, maggiore in senso stretto e lato
     - not, negazione logica
     - and, prodotto logico
     - or, somma logica
   - le espressioni possono contenere parentesi tonde, con il significato usuale
   - ci sono i seguenti tipi d'istruzione:
     - assegnamento, come

       ```
       a := b + c ;
       ```

       le istruzioni di assegnamento sono seguite da punto e virgola e contengono solo espressioni aritmetiche (le altre istruzioni non hanno punto e virgola)
     - ciclo for
     - ciclo while
     - condizionale if-then-else
   - le istruzioni strutturate si possono annidare in modo generale, tranne che il ciclo for non può contenere il ciclo while, a qualunque livello di profondità
   - struttura del ciclo for:

     ```
     for var in start : step : end do
         lista di istruzioni
     end
     ```

     dove start, step ed end sono il valore iniziale, il passo di incremento e il valore finale della variabile di conteggio var, e sono espressioni aritmetiche
   - strutture del ciclo while e del condizionale if-then-else (il ramo else è facoltativo):

     ```
     while condizione do
         lista di istruzioni
     end
     ```

     ```
     if condizione
         then lista di istruzioni
         else lista di istruzioni
     end
     ```

Soluzione

Nella grammatica G che segue (assioma INSTR_LIST) i nonterminali hanno nomi autoesplicativi. Il suffisso _NW denota le istruzioni strutturate che non possono contenere cicli while. Le classi sintattiche AR_EXP e COND_EXP sono le espression aritmentiche e le condizioni. Le classi sintattiche AE1 e AE2 (rispettivamente CE1, CE2 e CE3) indicano espressioni aritmetiche (rispettivamente condizioni) con priorità crescente.

$$
\begin{array}{rcl}
\langle \text{INSTR\_LIST} \rangle & \rightarrow & \langle \text{INSTR} \rangle^{+} \\
\langle \text{INSTR} \rangle & \rightarrow & \langle \text{FOR} \rangle \mid \langle \text{WHILE} \rangle \mid \langle \text{IF} \rangle \mid \langle \text{ASSIGN} \rangle \\
\langle \text{FOR} \rangle & \rightarrow & \text{`for'} \, \langle \text{VAR} \rangle \, \text{`in'} \, \langle \text{AR\_EXP} \rangle \, \text{`:'} \, \langle \text{AR\_EXP} \rangle \, \text{`:'} \, \langle \text{AR\_EXP} \rangle \, \text{`do'} \, \langle \text{I\_LIST\_NW} \rangle \, \text{`end'} \\
\langle \text{WHILE} \rangle & \rightarrow & \text{`while'} \, \langle \text{COND\_EXP} \rangle \, \text{`do'} \, \langle \text{INSTR\_LIST} \rangle \, \text{`end'} \\
\langle \text{IF} \rangle & \rightarrow & \text{`if'} \, \langle \text{COND\_EXP} \rangle \, \text{`then'} \, \langle \text{INSTR\_LIST} \rangle \; [\; \text{`else'} \, \langle \text{INSTR\_LIST} \rangle \; ] \; \text{`end'} \\
\langle \text{ASSIGN} \rangle & \rightarrow & \langle \text{VAR} \rangle \, \text{`:='} \, \langle \text{AR\_EXP} \rangle \, \text{`;'} \\
\hline
\langle \text{I\_LIST\_NW} \rangle & \rightarrow & \langle \text{INSTR\_NW} \rangle^{+} \\
\langle \text{INSTR\_NW} \rangle & \rightarrow & \langle \text{FOR} \rangle \mid \langle \text{IF\_NW} \rangle \mid \langle \text{ASSIGN} \rangle \\
\langle \text{IF\_NW} \rangle & \rightarrow & \text{`if'} \, \langle \text{COND\_EXP} \rangle \, \text{`then'} \, \langle \text{I\_LIST\_NW} \rangle \; [\; \text{`else'} \, \langle \text{I\_LIST\_NW} \rangle \; ] \; \text{`end'} \\
\hline
\langle \text{VAR} \rangle & \rightarrow & \text{`id'} \mid \text{`id'} \; [\; \text{`['} \; \langle \text{AR\_EXP} \rangle \; (\text{`,'} \; \langle \text{AR\_EXP} \rangle \;)^{*} \; \text{`]'} \;] \\
\hline
\langle \text{AR\_EXP} \rangle & \rightarrow & \langle \text{AE1} \rangle \; (\; (\text{`}+\text{'} \mid \text{`}-\text{'}) \, \langle \text{AE1} \rangle \;)^{*} \\
\langle \text{AE1} \rangle & \rightarrow & \langle \text{AE2} \rangle \; (\; (\text{`}\times\text{'} \mid \text{`/'}) \, \langle \text{AE2} \rangle \;)^{*} \\
\langle \text{AE2} \rangle & \rightarrow & \langle \text{VAR} \rangle \mid \text{`const'} \mid \langle \text{AR\_EXP} \rangle \\
\hline
\langle \text{COND\_EXP} \rangle & \rightarrow & \langle \text{CE1} \rangle \, (\, \text{`or'} \, \langle \text{CE1} \rangle \,)^{*} \\
\langle \text{CE1} \rangle & \rightarrow & \langle \text{CE2} \rangle \, (\text{`and'} \, \langle \text{CE2} \rangle \,)^{*} \\
\langle \text{CE2} \rangle & \rightarrow & (\text{`not'} \, \langle \text{CE2} \rangle \,) \mid \langle \text{CE3} \rangle \\
\langle \text{CE3} \rangle & \rightarrow & \text{`('} \, \langle \text{COND\_EXP} \rangle \, \text{`)'} \mid \langle \text{REL\_EXP} \rangle \\
\langle \text{REL\_EXP} \rangle & \rightarrow & \langle \text{AR\_EXP} \rangle \, (\text{`}=\text{'} \mid \text{`}>\text{'} \mid \text{`}\geq\text{'}) \, \langle \text{AR\_EXP} \rangle
\end{array}
$$

La grammatica G ha struttura modulare e i componenti sono non ambigui (liste, grammatica standard EBNF delle espressioni, ecc), pertanto G non è ambigua.

3 Analisi sintattica e parsificatori 20%

1. È data la grammatica G seguente, di alfabeto terminale $\{a, b, c, d\}$ (assioma S):

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

   Si risponda alle domande seguenti:

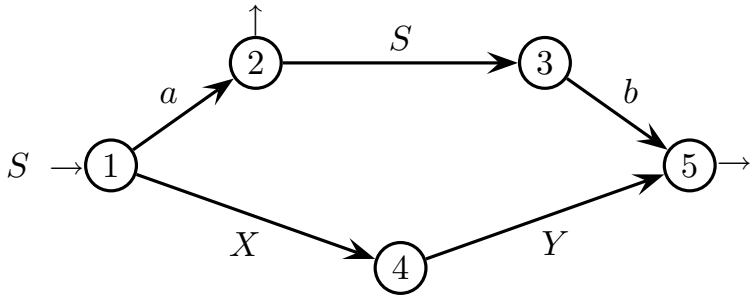
   (a) Si disegni la rete di macchine a stati finiti, sull'alfabeto totale $\{a, b, S, X, Y\}$, che rappresenta la grammatica G.

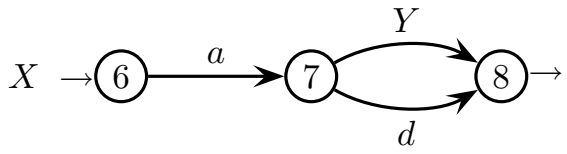
   (b) Si trovi il valore di $k \geq 1$ per il quale la grammatica G è di tipo $LL(k)$, calcolando gli insiemi guida tu tutti gli archi della rete di macchine per $k = 1$, e solo se e dove serve per $k > 1$.

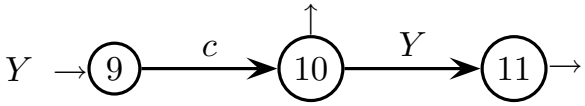
   (c) (facoltativa) Per il valore di k trovato, si scriva l'analizzatore sintattico a discesa ricorsiva della grammatica G.

Soluzione

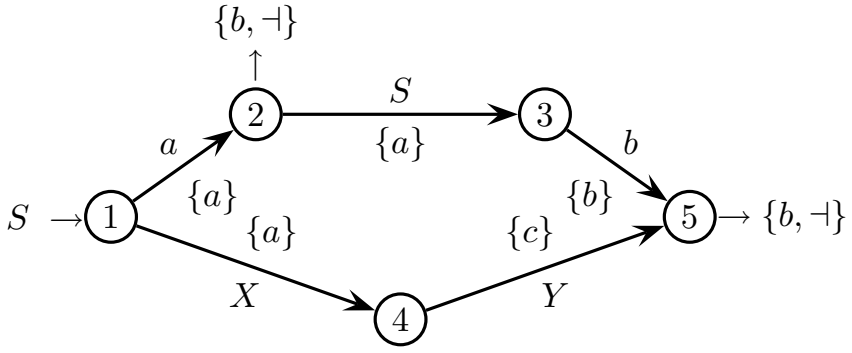
(a) Ecco la rete di macchine che rappresenta la grammatica G:

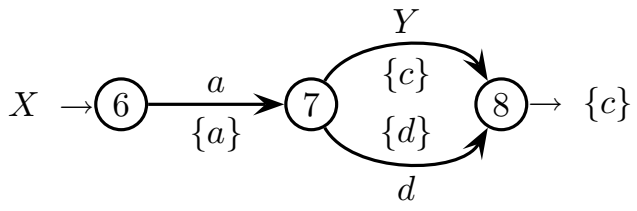


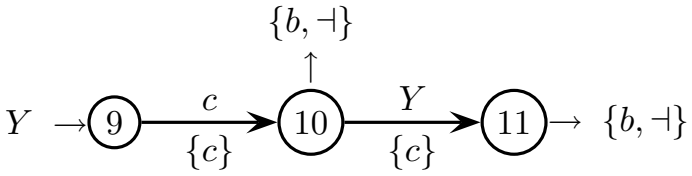




(b) Ecco gli insiemi guida per $k = 1$:

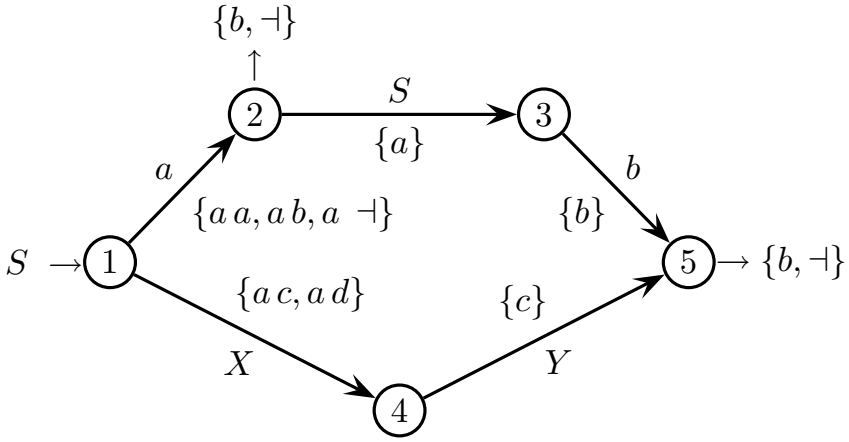






C'è un conflitto (causato dalla lettera a) al nodo 1, pertanto la grammatica G non è di tipo $LL(1)$.

Ecco gli insiemi guida per $k = 2$ (solo dove serve):



Il conflitto al nodo 1 è scomparso, pertanto la grammatica G è di tipo $LL(2)$.

(c) L'analizzatore sintattico a discesa ricorsiva si scrive facilmente per $k = 1$, esteso con $k = 2$ solo dove serve (nodo 1 della procedura ricorsiva di S).

2. È data la grammatica G seguente, di alfabeto terminale $\{a, b, c\}$ (assioma S):

$$G \begin{cases} S & \rightarrow & a\ S\ B \ \mid \ a\ S\ C \ \mid \ a \\ B & \rightarrow & b \\ C & \rightarrow & C\ c \ \mid \ b\ c \end{cases}$$

Si risponda alle domande seguenti:

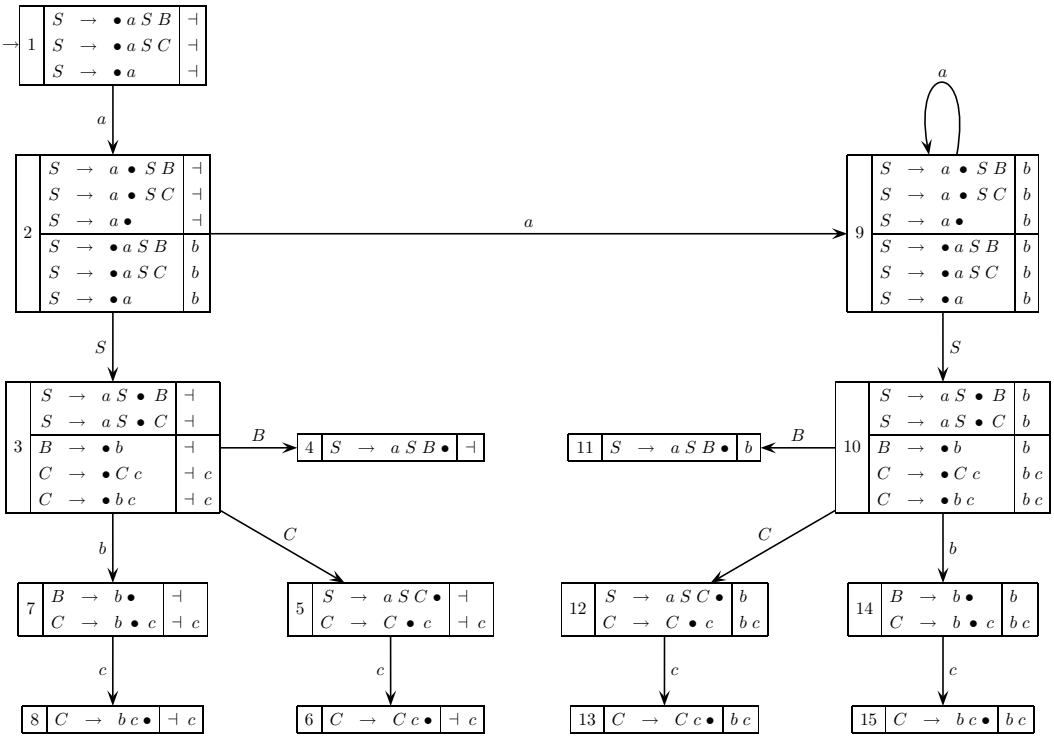
(a) Si giustifichi nel modo più rapido possibile perché la grammatica G non è $LR(0)$.

(b) Si dimostri che la grammatica G è di tipo $LR(1)$, costruendo l'automa pilota.

Soluzione

(a) È immediato vedere che con lo spostamento sul terminale a dal macrostato iniziale si va in un macrostato inadeguato per l'analisi $LR(0)$, in quanto contenente la candidata di riduzione $S \to a \bullet$ e le due candidate di spostamento $S \to a \bullet S\ B$ e $S \to a\ \bullet\ S\ C$.

(b) La dimostrazione rigorosa che la grammatica G è di tipo $LR(1)$ si ottiene costruendo il grafo pilota $LR(1)$. Eccolo:



Non ci sono conflitti, tutti i macrostati sono adeguati e pertanto la grammatica G è di tipo $LR(1)$.

4 Traduzione e analisi semantica 20%

1. Si consideri il linguaggio sorgente L_s costituito dalle liste di espressioni aritmetiche. Un esempio di frase di L_s è il seguente:

   $$a + a + a\,;\; a\,;\; a + a + a\,;\; a + a$$

   Si noti che il separatore è il carattere punto e virgola ';'.

   Il linguaggio sorgente L_s è definito dalla grammatica G_s seguente (assioma S):

   $$G_s \begin{cases} S & \rightarrow & \langle \mathsf{espr} \rangle \mid \langle \mathsf{espr} \rangle \;;\; S \\ \langle \mathsf{espr} \rangle & \rightarrow & a \mid a + \langle \mathsf{espr} \rangle \end{cases}$$

   Si devono definire le traduzioni che operano le trasformazioni specificate di seguito:

   c1 la lista prodotta elenca le espressioni nell'ordine da destra a sinistra

   c2 ogni espressione è scritta nella forma polacca postfissa

   c1 and c2

   Per esempio la traduzione della stringa precedente, con la prescrizione **c1** risulta così:

   $$a + a\,;\; a + a + a\,;\; a\,;\; a + a + a$$

   e con la prescrizione **c2** risulta così:

   $$a\,a + a +\,;\;\; a\,;\; a\,a + a +\,;\; a\,a +$$

   e infine, con la prescrizione **c1 and c2**:

   $$a\,a +\,;\; a\,a + a +\,;\; a\,;\; a\,a + a +$$

   Si noti che la grammatica sorgente degli schemi di traduzione può essere modificata, se necessario.

   Si risponda alle domande seguenti:

   (a) Si scriva lo schema di traduzione (puramente sintattico) che calcola la traduzione specificata dalla prescrizione **c1**. Si disegnino gli alberi sintattici sorgente e pozzo dell'esempio.

   (b) Similmente, si scriva lo schema di traduzione che calcola la traduzione specificata dalla prescrizione **c2**.

   (c) (facoltativa) Si progetti uno schema di traduzione che calcola la traduzione specificata dalla prescrizione **c1 and c2**.

Soluzione

(a) Dovendo produrre la lista in ordine inverso, si vede che essa non è altro che la stringa sorgente riflessa. Ricordando la nota traduzione che riflette una stringa, è facile scrivere lo schema di traduzione:

G_s	G_p
$S \to a\ +\ S$	$S \to S\ +\ a$
$S \to a$	$S \to a$
$S \to a\ ;\ S$	$S \to S\ ;\ a$

Gli alberi sintattici sono semplici e lasciati al lettore.

(b) Si può applicare la classica trasformazione da scrittura infissa a scrittura postfisa direttamente alla grammatica data nel problema.

G_s	G_p
$S \to \langle\mathsf{espr}\rangle$	$S \to \langle\mathsf{espr}\rangle$
$S \to \langle\mathsf{espr}\rangle\ ;\ S$	$S \to \langle\mathsf{espr}\rangle\ ;\ S$
$\langle\mathsf{espr}\rangle \to a\ +\ \langle\mathsf{espr}\rangle$	$\langle\mathsf{espr}\rangle \to a\ \langle\mathsf{espr}\rangle\ +$
$\langle\mathsf{espr}\rangle \to a$	$\langle\mathsf{espr}\rangle \to a$

(c) Si devono combinare le due precedenti soluzioni. Si noti però sarebbe errato scrivere una coppia di regole del tipo

G_s	G_p
$S \to \langle\mathsf{espr}\rangle\ ;\ S$	$S \to S\ ;\ \langle\mathsf{espr}\rangle$

allo scopo di invertire la lista di espressioni: infatti i nonterminali S e $\langle\mathsf{espr}\rangle$ devono essere nello stesso ordine nelle regole che si corrispondono.

Riprendendo lo schema del caso **c1**, una semplice modifica permette di ottenere la forma postfissa:

G_s	G_p
$S \to a\ +S$	$S \to S\ a\ +$
$S \to a$	$S \to a$
$S \to a\ ;\ S$	$S \to S\ ;\ a$

2. La cosiddetta "regola del nove" fornisce un criterio semplice per verificare la correttezza delle operazioni di moltiplicazione effettuate su numeri in notazione decimale.

   Ecco con un esempio la sua applicazione. Si consideri la moltiplicazione $143 \times 283 = 40469$. Si calcola la somma modulo 9 delle cifre di ognuno dei due fattori e del prodotto, in questo caso ottenendo:

   $$(1 + 4 + 3) \mod 9 = 8 \mod 9 = 8$$

   $$(2 + 8 + 3) \mod 9 = 13 \mod 9 = 4$$

   e

   $$(4 + 0 + 4 + 6 + 9) \mod 9 = 23 \mod 9 = 5$$

   Si calcola il prodotto modulo 9 delle somme ottenute per i due fattori. In questo esempio si ha:

   $$(8 \times 4) \mod 9 = 32 \mod 9 = 5$$

   Quest'ultimo valore deve essere uguale alla somma modulo 9 delle cifre del prodotto (cosa che infatti si verifica nell'esempio).

   Si consideri la grammatica G seguente, che genera le espressioni di uguaglianza tra il prodotto di due fattori e un terzo valore rappresentante il risultato della moltiplicazione (assioma E).

   $$G \begin{cases} E & \rightarrow & T \text{ `=' } C \\ T & \rightarrow & C \text{ `}\times\text{' } C \\ C & \rightarrow & D\ C \\ C & \rightarrow & D \\ D & \rightarrow & \text{`0'} \mid \text{`1'} \mid \ldots \mid \text{`9'} \end{cases}$$

   Si risponda alle domande seguenti:

   (a) Appoggiandosi a questo schema sintattico G, si definisca una semplice grammatica con attributi che realizzi la verifica basata sulla regola del nove, possibilmente usando due soli attributi sintetizzati.

   Suggerimento: si usi un attributo booleano *eq*, vero se e solo se la regola del nove è soddisfatta, e un attributo *val*, intero con valore compreso tra 0 e 8, per rappresentare i valori calcolati nell'applicazione della regola.

   (b) Si verifichi se la grammatica con attributi così ottenuta è a una scansione, giustificando la risposta.

---

attributi da usare per la grammatica

tipo	nome	(non)terminali	dominio	significato
già dati nel testo dell'esercizio				
sx	*eq*		booleano	vero se la regola è soddisfatta, falso altrimenti
sx	*val*		intero modulo 9	valori calcolati nell'applicazione della regola

	sintassi	funzioni semantiche
0:	$D_0 \to 0$	
1:	$D_0 \to 1$	
...	...	
8:	$D_0 \to 8$	
9:	$D_0 \to 9$	
10:	$C_0 \to D_1$	
11:	$C_0 \to D_1\ C_2$	
12:	$T_0 \to C_1\ \times\ C_2$	
13:	$E_0 \to T_1\ =\ C_2$	

Soluzione

(a) Ecco gli attributi:

attributi da usare per la grammatica

tipo	nome	(non)terminali	dominio	significato
sx	*eq*	E	booleano	vero se la regola è soddisfatta, falso altrimenti
sx	*val*	T, C, D	intero modulo 9	valori calcolati nell'applicazione della regola

Ossia:

- *eq* è un attributo sintetizzato del solo nonterminale E
- *val* è un attributo sintetizzato dei nonterminali T, C e D

Ecco le funzioni semantiche:

	sintassi	funzioni semantiche
0:	$D_0 \to 0$	$val_0 = 0$
1:	$D_0 \to 1$	$val_0 = 1$
...	...	...
8:	$D_0 \to 8$	$val_0 = 8$
9:	$D_0 \to 9$	$val_0 = 0$
10:	$C_0 \to D_1$	$val_0 = val_1$
11:	$C_0 \to D_1\ C_2$	$val_0 = val_1 + val_2 \mod 9$
12:	$T_0 \to C_1\ \times\ C_2$	$val_0 = val_1 \times val_2 \mod 9$

13:	$E_0 \to T_1\ =\ C_2$	**if** $(val_1 = val_2)$ **then**
		$eq_0 = \text{true}$
		else
		$eq_0 = \text{false}$
		end if

(b) Utilizzando solo attributi sintetizzati, la grammatica è a una scansione.