Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 10 settembre 2007 - 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 espressioni regolari seguenti:

$$R_1 = ((a\, b)^* \; c)^* \qquad\qquad R_2 = (c^* \; (a\, b)^*)^*$$

Si risponda ai punti seguenti:

(a) Si verifichi intuitivamente se le due espressioni R_1 e R_2 siano equivalenti (se non lo fossero si mostri una stringa appartenente a un'espressione ma non all'altra).

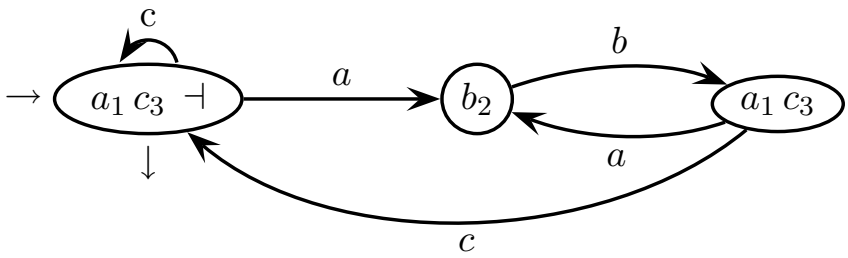
(b) Si verifichi in modo algoritmico se le due espressioni regolari siano equivalenti.

Soluzione

(a) Che le due espressioni R_1 e R_2 *non* siano equivalenti, dovrebbe essere pressoché evidente in modo intuitivo: le stringhe generate da R_1 (tranne ε) terminano necessariamente con almeno una lettera c, mentre quelle generate da R_2 possono terminare con una lettera b (oltre che con c). Per esempio, la stringa $a\, b$ è generata da R_2 ma non da R_1

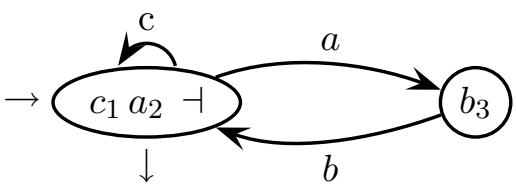
(b) Comunque, qui si chiede di procedere algoritmicamente. Un modo per farlo è di trovare gli automi riconoscitori deterministici minimi e verificare se siano identici o no (giacché la forma minima è unica). Si tracciano i due automi deterministici, tramite l'algoritmo di McNaughton-Yamada.
Automa di $R_1 = ((a_1\, b_2)^* \; c_3)^* \; \dashv$:

inizî	$a_1\; c_3\; \dashv$
generatore	séguiti
a_1	b_2
b_2	$a_1\; c_3$
c_3	$a_1\; c_3\; \dashv$



Questo automa deterministico è palesemente minimo, perché gli stati b_2 e $a_1\, c_3$ sono certamente distinguibili, avendo il secondo arco uscente c e il primo no, e lo stato finale è ovviamente distinguibile dagli altri due che non sono finali.
Automa di $R_2 = (\, c_1^* \; (\, a_2\, b_3 \,)^* \,)^* \; \dashv$:

inizî	$c_1\ a_2\ \dashv$
generatore	séguiti
c_1	$c_1\ a_2\ \dashv$
a_2	b_3
b_3	$c_1\ a_2\ \dashv$



Questo automa deterministico è palesemente minimo, giacché i due stati sono uno finale e l'altro no, dunque necessariamente distinguibili.

Poiché i due automi deterministici minimi sono diversi (hanno tre e due stati, rispettivamente), non sono equivalenti e dunque non lo sono neppure le espressioni regolari R_1 e R_2.

2. Il linguaggio L di alfabeto $\{a, b\}$ è definito per mezzo delle condizioni seguenti:

   (a) le frasi devono iniziare con la stringa $a\,b$

   and

   (b) le frasi devono finire con il carattere b

   and

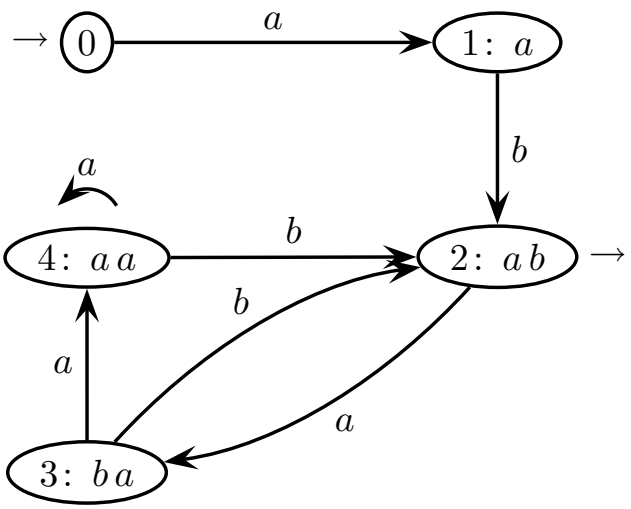
   (c) le frasi non possono contenere la sottostringa $a\,b\,b$

   Si risponda ai punti seguenti:

   (a) Si costruisca in modo sistematico un automa riconoscitore deterministico di L.
   (b) Se occorre si minimizzi il riconoscitore così costruito.

Soluzione

(a) Il linguaggio è di tipo locale, seppure ragionando con terne di caratteri consecutivi e non solo con coppie. L'automa deterministico deve soltanto ricordare gli ultimi due caratteri letti nel nastro di ingresso. Eccolo:



Le lettere registrate negli stati sono (da sinistra verso destra) il penultimo e l'ultimo carattere letto in ingresso; così per esempio arrivando nello stato 2: $a\,b$ il penultimo e l'ultimo carattere letto sono a e b, rispettivamente. Lo stato 0 (iniziale) non ne contiene nessuna e lo stato 1 (secondo stato) solo una, naturalmente.

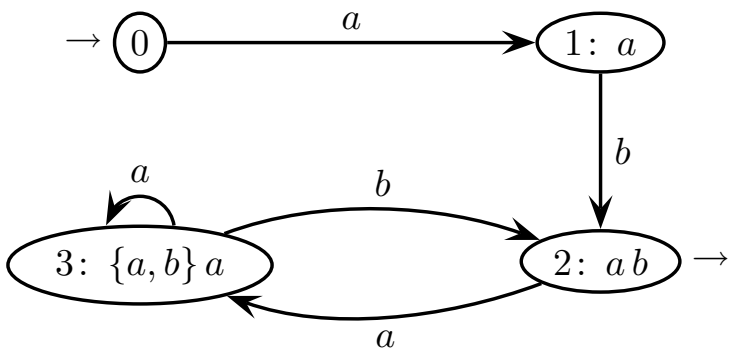
È evidente che lo stato $b\,b$ è indefinito (cioè da esso non si può raggiungere nessuno stato finale): se si entrasse in tale stato, la stringa conterrebbe da qualche parte il fattore $a\,b\,b$ (giacché essa inizia necessariamente con a), che è vietato.

In alternativa, si può costruire l'automa mediante intersezione (prodotto cartesiano) di tre automi più semplici (si ponga $\Sigma = \{\,a, b\,\}$): (1) l'automa di $a\,b\,\Sigma^*$ (stringhe che iniziano con $a\,b$), (2) quello di $\Sigma^*\,b$ (stringhe che finiscono b), oppure direttamente quello di $a\,b\,(\,\varepsilon \mid \Sigma^*\,b\,)$ (stringhe che iniziano con $a\,b$ e finiscono con b), e (3) quello di $\neg\,(\,\Sigma^*\,a\,b\,b\,\Sigma^*\,)$ (stringhe che non contengono il fattore $a\,b\,b$); quest'ultimo richiede di calcolare anche il complemento. Sono tutti e tre (o due) automi molto semplici e con pochi stati. Si lascia al lettore questa costruzione alternativa come esercizio.

(b) Intuitivamente, gli stati 3: $b\,a$ e 4: $a\,a$ sono indistinguibili e si possono unificare. Comunque, procedendo algoritmicamente, ecco la tabella degli stati:

stato	a	b	finale ?
0	1	–	no
1	–	2	no
2	3	–	sì
3	4	2	no
4	4	2	no

Le righe 3 e 4 sono identiche, dunque i due stati corrispondenti sono indistinguibili. Gli altri stati sono tutti distinguibili. Ecco l'automa minimizzato:



Lo stato 3: $\{a, b\}$ ricorda, indifferentemente, se il penultimo carattere letto in

ingresso sia a o b. Si noti comunque che gli stati 3 e 1 sono distinguibili, perché il secondo non ha arco a uscente.

2 Grammatiche libere e automi a pila 20%

1. Si considerino inizialmente il linguaggio L_1 di alfabeto $\Sigma = \{a, b, c\}$:

$$L_1 = (\, x\, c\,)^* \quad \text{dove } x \in \Sigma^* \text{ and } |x|_a = |x|_b \geq 0 \text{ and } |x|_c = 0$$

esemplificato dalle stringhe:

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

e il linguaggio regolare R_2 (sempre di alfabeto Σ):

$$R_2 = \left(\, a^+\, b^+\, c\,\right)^*$$

Si guardi ora il linguaggio L_3 definito dall'intersezione

$$L_3 = L_1 \cap R_2$$

Si risponda ai punti seguenti:

(a) Si scrivano tutte le frasi di lunghezza minore o eguale a 6 appartenenti al linguaggio L_3.

(b) Si scriva una grammatica G non ambigua, in forma non estesa, che generi il linguaggio L_3, e si disegni l'albero sintattico di una frase di lunghezza 6.

Soluzione

(a) Ecco le frasi di lunghezza ≤ 6:

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

(b) Si vede facilmente che il linguaggio L_3 definito dall'intersezione è:

$$L_1 \cap R_2 = a^{n_1}\, b^{n_1}\, c\, a^{n_2}\, b^{n_2}\, c \ldots a^{n_k}\, b^{n_k}\, c \qquad k \geq 1 \quad \forall\, 1 \leq j \leq k \;\; n_j \geq 1$$

Ecco la grammatica soluzione G (assioma S):

$$G \begin{cases} S \;\rightarrow\; B\; c\; S \;\mid\; \varepsilon \\ B \;\rightarrow\; a\; B\; b \;\mid\; a\; b \end{cases}$$

Ed ecco l'albero sintattico della stringa $a\,b\,c\,a\,b\,c$:
DA FARE ...

2. Si consideri il linguaggio L dei programmi composti da (almeno) uno o più assegnamenti separati e terminati da punto e virgola, con espressioni contenenti solo addizione e senza parentesi. Ecco un esempio di programma:

$$\begin{array}{l} i = i + i + i + i; \\ i = i; \\ i = i + i + i + i + i; \end{array}$$

Con riferimento a detto linguaggio L si aggiunga la possibilità che il programma contenga un solo errore sintattico. Gli errori da considerare sono di due tipi:

omissione del segno di assegnamento $=$

omissione del segno di addizione $+$

I due programmi seguenti contengono ciascuno un solo errore sintattico:

$$\begin{array}{l} i = i + i + i + i; \\ i = i; \\ i \quad i + i + i + i + i + i; \end{array}$$

$$\begin{array}{l} i = i\ i + i + i; \end{array}$$

Si definisce linguaggio L_F come l'insieme di tutti i programmi corretti e di quelli che contengono esattamente un errore dell'uno o l'altro tipo (ma non tutti e due):

$$L_F = L \cup \text{ programmi contenenti un solo errore}$$

Invece il prossimo programma non appartiene al linguaggio L_F perché contiene più di un errore:

$$\begin{array}{l} i = i + i + i\ i; \\ i = i; \\ i \quad i + i + i + i + i + i; \end{array}$$

Si risponda alle domande seguenti.

(a) Si scriva una grammatica EBNF non ambigua del linguaggio L.

(b) Si scriva una grammatica EBNF non ambigua del linguaggio L_F.

(c) Si spieghi in che modo la grammatica di L_F trovata garantisca che la stringa contenga al massimo un errore sintattico.

Soluzione

(a) Si procede modularmente. Per prima si dà la grammatica EBNF che genera solo programmi corretti. Eccola (assioma PROG - il pedice 'corr.' sta per 'corretto'):

$$\begin{aligned} \langle \mathsf{PROG} \rangle_{\text{corr.}} &\rightarrow (\, \langle \mathsf{ASS} \rangle_{\text{corr.}} \text{ ';' })^{+} \\ \langle \mathsf{ASS} \rangle_{\text{corr.}} &\rightarrow i \text{ '=' } \langle \mathsf{ESPR} \rangle_{\text{corr.}} \\ \langle \mathsf{ESPR} \rangle_{\text{corr.}} &\rightarrow i \ (\text{ '+' } i\,)^{*} \end{aligned}$$

Trattandosi più o meno della solita grammatica EBNF delle espressioni (tra l'altro non è neppure ricorsiva), essa è non ambigua.

(b) Poi si dà la grammatica EBNF che omette un solo simbolo di assegnamento (usa parte delle regole della prima, qui non ripetute). Eccola (assioma $\mathsf{PROG}_{\text{senza =}}$):

$$\begin{aligned} \langle \mathsf{PROG} \rangle_{\text{senza =}} &\rightarrow (\, \langle \mathsf{ASS} \rangle_{\text{corr.}} \text{ ';' })^{*} \ \langle \mathsf{ASS} \rangle_{\text{senza =}} \text{ ';' } (\, \langle \mathsf{ASS} \rangle_{\text{corr.}} \text{ ';' })^{*} \\ \langle \mathsf{ASS} \rangle_{\text{senza =}} &\rightarrow i \ \langle \mathsf{ESPR} \rangle_{\text{corr.}} \end{aligned}$$

La regola che espande la classe sintattica di assegnamento $\mathsf{ASS}_{\text{senza =}}$ non contiene il simbolo '='. Essendo questa grammatica una semplice aggiunta regolare alla prima, è non ambigua.

Poi si dà la grammatica EBNF che omette un solo simbolo di addizione (usa parte delle regole della prima, qui non ripetute). Eccola (assioma $\mathsf{PROG}_{\text{senza +}}$):

$$\begin{aligned} \langle \mathsf{PROG} \rangle_{\text{senza +}} &\rightarrow (\, \langle \mathsf{ASS} \rangle_{\text{corr.}} \text{ ';' })^{*} \ \langle \mathsf{ASS} \rangle_{\text{senza +}} \text{ ';' } (\, \langle \mathsf{ASS} \rangle_{\text{corr.}} \text{ ';' })^{*} \\ \langle \mathsf{ASS} \rangle_{\text{senza +}} &\rightarrow i \text{ '=' } \langle \mathsf{ESPR} \rangle_{\text{senza +}} \\ \langle \mathsf{ESPR} \rangle_{\text{senza +}} &\rightarrow i \ (\text{ '+' } i\,)^{*} \ i \ (\text{ '+' } i\,)^{*} \end{aligned}$$

La regola che espande la classe sintattica di espressione $\mathsf{ESPR}_{\text{senza +}}$ forza la comparsa di esattamente due variabili i senza simbolo '+' infisso. Anche questa grammatica è non ambigua (per lo stesso motivo di prima).

Infine basta unire le tre grammatiche (assioma $\langle \mathsf{PROG} \rangle$). Essendo i tre linguaggi disgiunti, quest'unione è non ambigua:

$$\langle \mathsf{PROG} \rangle \rightarrow \langle \mathsf{PROG} \rangle_{\text{corr.}} \mid \langle \mathsf{PROG} \rangle_{\text{senza =}} \mid \langle \mathsf{PROG} \rangle_{\text{senza +}}$$

Per inciso, si può notare che il linguaggio L_F è puramente regolare.

È possibile che esistano soluzioni più compatte, unificando le regole.

(c) La grammatica complessiva di L_F è non ambigua per costruzione. La spiegazione è già sostanzialmente implicita nel ragionamento precedente (si parte da una grammatica base non ambigua e si applicano solo passaggi non ambigui).

3 Analisi sintattica e parsificatori 20%

1. Si trasformi la grammatica data G in modo da ottenere una grammatica equivalente G' adatta all'analisi sintattica discendente deterministica.

$$G \begin{cases} S \rightarrow S\ A \mid A \\ A \rightarrow a\ A\ b \mid \varepsilon \end{cases}$$

Si risponda alle seguenti domande.

(a) Si scriva la grammatica G', giustificandone l'equivalenza.

(b) Si calcolino gli insiemi guida della grammatica G', verificando che essa sia $LL(1)$ o $LL(k)$ per qualche $k > 1$.

Soluzione

(a) La grammatica G ha due difetti che causano la perdita della proprietà $LL(k)$: essa è ricorsiva a sinistra e ambigua (due modi diversi di produrre ε sono $S \Rightarrow A \Rightarrow \varepsilon$ e $S \Rightarrow S\ A \overset{2}{\Rightarrow} \varepsilon\ \varepsilon$). Poiché S genera la forma di frase A^+, e A genera il linguaggio $L_A = \{\, a^n\, b^n \mid n \geq 0 \,\}$, il linguaggio generato è il seguente:

$$L(G) = x^+ \quad \text{dove } x \in L_A$$

(b) Ecco una soluzione in forma BNF non estesa. È facile scrivere una grammatica G_1 equivalente a G, non ricorsiva a sinistra e non ambigua:

G_1:			*Insieme guida*
S	$\rightarrow$	$B\ S$	a
S	$\rightarrow$	ε	$\dashv$
B	$\rightarrow$	$a\ B\ b$	a
B	$\rightarrow$	$a\ b$	a

Si noti che, per togliere l'ambiguità, la stringa vuota è prodotta direttamente ed esclusivamente dall'assioma. Poiché le alternative di B hanno un prefisso comune, violano la condizione $LL(1)$ (però la grammatica G_1 è $LL(2)$, si vede subito giacché gli insiemi guida di livello 2 delle regole di B sono $a\,a$ e $a\,b$). Fattorizzando a sinistra si ottiene la grammatica equivalente G_2, che è $LL(1)$:

G_2:			*Insieme guida*
	$\ldots$		
B	$\rightarrow$	$a\ X$	
X	$\rightarrow$	$B\ b$	a
X	$\rightarrow$	b	b

La grammatica G_2 è pertanto la grammatica G' che si sta cercando.

2. È data la grammatica seguente:

$$G \begin{cases} S \rightarrow a\ S\ A \mid \varepsilon \\ A \rightarrow b\ A \mid a \end{cases}$$

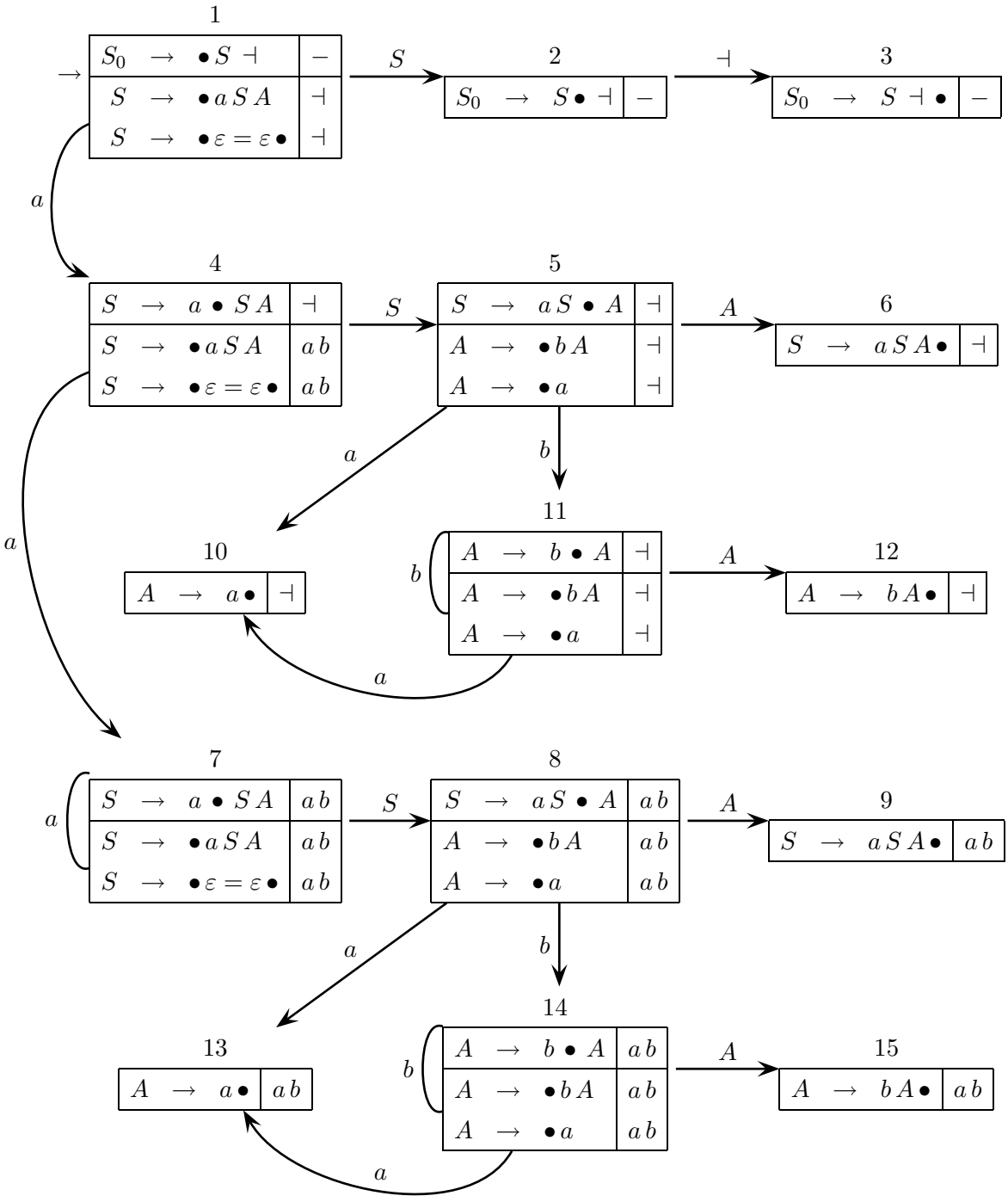
Si risponda alle domande seguenti:

(a) Si costruisca l'automa pilota con il metodo $LR(1)$.

(b) Si verifichi se l'automa pilota soddisfi le condizioni $LR(1)$, $LALR(1)$ e $LR(0)$, giustificando le risposte.

Soluzione

(a) Ecco il grafo pilota $LR(1)$ (completando la grammatica con il terminatore):



(b) A priori la grammatica non può essere $LR(0)$, giacché contiene una regola nulla. Si vede comunque che il grafo pilota non soddisfa la condizione $LR(1)$, giacché contiene due stati (4 e 7) con conflitto spostamento-riduzione. Pertanto la grammatica non è $LR(1)$ e dunque neppure $LALR(1)$.

4 Traduzione e analisi semantica 20%

1. Il linguaggio sorgente contiene espressioni aritmetiche con l'usuale operatore infisso di addizione e le parentesi. Inoltre a sinistra di un gruppo parentesizzato si trova un campo marcatore che specifica se la traduzione del gruppo sia quella prefissa o postfissa. Il linguaggio sorgente è definito dalla grammatica seguente:

$$G \begin{cases} S & \rightarrow & (\text{`prefix'} \mid \text{`postfix'})\ \text{`('}\ E\ \text{`)'} \\ E & \rightarrow & T\ (\text{`+'}\ T)^* \\ T & \rightarrow & a \mid (\text{`prefix'} \mid \text{`postfix'})\ \text{`('}\ E\ \text{`)'} \end{cases}$$

Un esempio della traduzione da produrre è il seguente (per maggiore chiarezza qui i nomi di variabili sono differenziati e gli operatori numerati, ma nel sorgente le variabili si chiamano tutte a egli operatori non hanno numerazione):

sorgente:	postfix (a $+_1$ b $+_2$ prefix (c $+_3$ prefix (d)))
destinazione:	$a\ b\ +_1\ +_3\ c\ d\ +_2$

Si noti che nella traduzione *non* si conserva il campo marcatore.

Si risponda ai punti seguenti:

(a) Modificando se necessario la grammatica sorgente, si scriva uno schema sintattico di traduzione (senza attributi semantici) che traduca il linguaggio sorgente producendo localmente la forma prefissa o postfissa, in accordo con la prescrizione.

(b) Si discuta se lo schema di traduzione sia deterministico.

Soluzione

(a) Si può procedere dividendo la grammatica sorgente e in due schemi di traduzione, prefisso e postfisso secondo il campo marcatore trovato, e poi collegando i due schemi quando si passa da prefisso a postfisso, e viceversa (assioma S). Ecco lo schema, presentato modularmente:

$$G_{trad} \begin{cases} S & \rightarrow & S_{pre} \mid S_{post} \\ & & \\ G_{pre} \begin{cases} S_{pre} & \rightarrow & \text{`prefix'}\ \text{`('}\ E_{pre}\ \text{`)'} \\ E_{pre} & \rightarrow & (\{\text{`+'}\}\ T_{pre}\ \text{`+'})^*\ T_{pre} \\ T_{pre} & \rightarrow & a \mid \text{`prefix'}\ \text{`('}\ E_{pre}\ \text{`)'} \mid \text{`postfix'}\ \text{`('}\ E_{post}\ \text{`)'} \end{cases} \\ & & \\ G_{post} \begin{cases} S_{post} & \rightarrow & \text{`postfix'}\ \text{`('}\ E_{post}\ \text{`)'} \\ E_{post} & \rightarrow & T_{post}\ (\text{`+'}\ T_{post}\ \{\text{`+'}\})^* \\ T_{post} & \rightarrow & a \mid \text{`prefix'}\ \text{`('}\ E_{pre}\ \text{`)'} \mid \text{`postfix'}\ \text{`('}\ E_{post}\ \text{`)'} \end{cases} \end{cases}$$

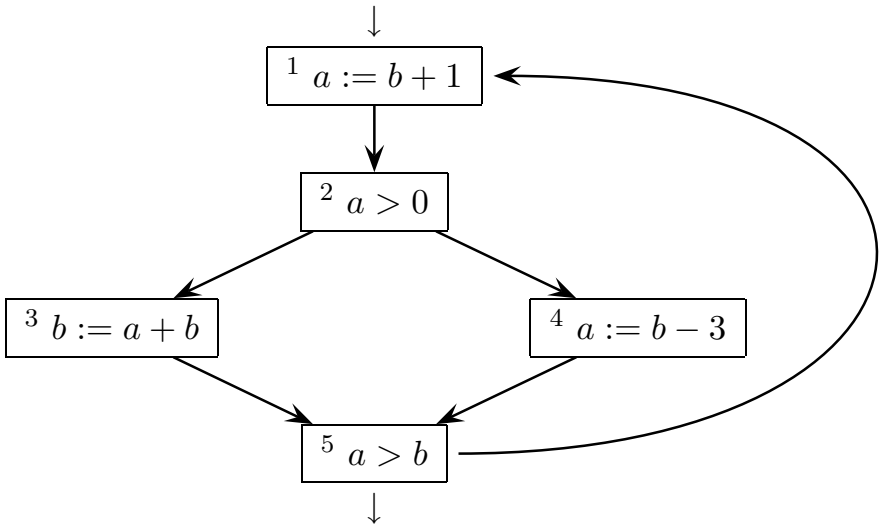
Qui lo schema è combinato e le parentesi graffe '{' e '}' racchiudono i simboli terminali da emettere in traduzione. Volendo si possono separare le parti sorgente e destinazione.

Si noti che la regola che espande E_{pre} cambia l'associatività (traduce $a + b + c$ in $+a + bc$); ciò peraltro è indifferente rispetto al testo dell'esercizio, che non precisa nulla al riguardo.

(b) Esaminando la parte sorgente di G_{pre}, si vede facilmente che non è $LL(k)$ per nessun k, giacché la produzione estesa che espande E_{pre} non permette di capire quando bisogna smettere di espandere l'operatore stella. Così com'è, lo schema G_{trad} non è deterministico.

Comunque, formulazioni diverse dello schema potrebbero essere deterministiche.

2. Si consideri il seguente grafo di controllo di un programma:



Inoltre si precisa che al termine del programma nessuna variabile è viva.

Si risponda alle domande seguenti:

(a) Si scrivano le equazioni di flusso per calcolare gli intervalli di vita delle variabili.

(b) Si calcolino e si scrivano nell'apposita tabella gli insiemi delle variabili vive in ogni punto del programma.

Soluzione

(a) Calcolo dei termini costanti (definizione e uso di variabile):

#	**def**	**use**
1	a	b
2	-	a
3	b	a, b
4	a	b
5	-	a, b

- a viene assegnata in 1, dunque lì è definita, b figura nell'espressione in 1, dunque lì è usata
- a figura nell'espressione in 2, dunque lì è usata
- b viene assegnata in 3, dunque lì è definita, a e b figurano nell'espressione in 3, dunque lì sono usate
- a viene assegnata in 4, dunque lì è definita, b figura nell'espressione in 4, dunque lì è usata
- a e b figurano nell'espressione in 5, dunque lì sono usate

Scrittura delle equazioni di flusso per le variabili vive ai nodi:

$$
\begin{array}{rcll}
in(1) & = & use(1) \cup (out(1) - def(1)) = & \text{– definizione di vitalità in ingresso} \\
 & = & \{b\} \cup (out(1) - \{a\}) = & \\
 & = & \{b\} & \\
out(1) & = & in(2) & \text{– il nodo 1 ha una sola via d'uscita} \\
\hline
in(2) & = & use(2) \cup (out(2) - def(2)) = & \text{– definizione di vitalità in ingresso} \\
 & = & \{a\} \cup (out(2) - \emptyset) = & \\
 & = & \{a\} \cup out(2) & \\
out(2) & = & in(3) \cup in(4) & \text{– il nodo 2 ha due vie d'uscita} \\
\hline
in(3) & = & use(3) \cup (out(3) - def(3)) = & \text{– definizione di vitalità in ingresso} \\
 & = & \{a, b\} \cup (out(3) - \{b\}) = & \\
 & = & \{a, b\} & \\
out(3) & = & in(5) & \text{– il nodo 3 ha una sola via d'uscita} \\
\hline
in(4) & = & use(4) \cup (out(4) - def(4)) = & \text{– definizione di vitalità in ingresso} \\
 & = & \{b\} \cup (out(4) - \{a\}) = & \\
 & = & \{b\} & \\
out(4) & = & in(5) & \text{– il nodo 4 ha una sola via d'uscita} \\
\hline
in(5) & = & use(5) \cup (out(5) - def(5)) = & \text{– definizione di vitalità in ingresso} \\
 & = & \{a, b\} \cup (out(5) - \emptyset) = & \\
 & = & \{a, b\} & \\
out(5) & = & \emptyset \cup in(1) = & \text{– il nodo 5 ha due vie d'uscita} \\
 & = & in(1) &
\end{array}
$$

Numerose equazioni si risolvono subito in assegnamenti a costante.

(b) Calcolo iterativo della soluzione alle equazioni di flusso:

	passo 1		**passo** 2		**passo** 3		**passo** 4					
	stato 0		**stato** 1		**stato** 2		**stato** 3		**stato** 4		**stato** 5	
#	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*
1	$\emptyset$	$\emptyset$	b	$\emptyset$	b	a	b	a	b	a, b	b	a, b
2	$\emptyset$	$\emptyset$	a	$\emptyset$	a	a, b	a, b	a, b	a, b	a, b	a, b	a, b
3	$\emptyset$	$\emptyset$	a, b	$\emptyset$	a, b	a, b	a, b	a, b	a, b	a, b	a, b	a, b
4	$\emptyset$	$\emptyset$	b	$\emptyset$	b	a, b	b	a, b	b	a, b	b	a, b
5	$\emptyset$	$\emptyset$	a, b	$\emptyset$	a, b	b	a, b	b	a, b	b	a, b	b

La convergenza è raggiunta in 4 passi. Entrambe le variabili sono vive a tutti i nodi (cioè ai rispettivi ingressi), tranne ai nodi 1 e 4 dove è viva solo b.