Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 24 settembre 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 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.
- 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 l'alfabeto $\Sigma = \{a, b, c\}$. Si considerino le due espressioni regolari R_1 e R_2 seguenti, di alfabeto Σ:

$$\begin{aligned} R_1 &= a\ (\ a\ b\ \mid\ b\ c\ )^* \\ R_2 &= (\ a\ b^*\ \mid\ a\ c^*\ )^* \end{aligned}$$

e i rispettivi linguaggi regolari L_1 e L_2.

Si risponda alle domande seguenti:

(a) Si dica quali sono i digrammi contenuti nelle stringhe dei linguaggi L_1 e L_2, rispettivamente.

(b) Si dica se le espressioni regolari R_1 e R_2 sono ambigue, con motivazione (ossia se l'espressione non è ambigua si spieghi perché, se lo è si dia una stringa ambigua).

(c) Si dica se i linguaggi L_1 e L_2 sono locali, motivando la risposta.

(d) (facoltativa) Si trovi l'espressione regolare R_3 (contenente i soli operatori di concatenamento unione stella croce) che genera il linguaggio regolare intersezione $L_3 = L_1 \cap L_2$, lavorando a scelta in modo intuitivo o algoritmico.

Soluzione

(a) Un modo per trovare i digrammi è di ricorrere al calcolo dei séguiti. Prima si numerano i generatori nelle espressioni regolari R_1 e R_2, come segue:

$$\begin{aligned} R_1 &= a_1\ (\ a_2\ b_3\ \mid\ b_4\ c_5\ )^* \\ R_2 &= (\ a_1\ b_2^*\ \mid\ a_3\ c_4^*\ )^* \end{aligned}$$

I séguiti dei generatori sono i seguenti:

espressione R_1

generatore	séguito
a_1	a_2
a_2	b_3
b_3	a_2
b_3	b_4
b_4	c_5
c_5	a_2
c_5	b_4

espressione R_2

generatore	séguito
a_1	a_1
a_1	a_3
a_1	b_2
b_2	a_1
b_2	a_3
b_2	b_2
a_3	a_1
a_3	a_3
a_3	c_4
c_4	a_1
c_4	a_3
c_4	c_4

Trascurando la numerazione si ottengono i digrammi seguenti:

digrammi di $R_1 = a\,a \quad a\,b \quad b\,a \quad b\,b \quad b\,c \quad c\,a \quad c\,b$

digrammi di $R_2 = a\,a \quad a\,b \quad a\,c \quad b\,a \quad b\,b \quad c\,a \quad c\,b \quad c\,c$

(b) L'espressione regolare R_1 è non ambigua. Osservando la numerazione, si ha che:

- il generatore a_1 compare all'inizio dell'espressione, deve necessariamente derivare il carattere iniziale della stringa e dunque non dà luogo ad ambiguità
- trascurando la a iniziale (già discussa e innocua ai fini dell'ambiguità), le lettere a e c compaiono in posizioni uniche nell'espressione e dunque non possono essere derivate ambiguamente in nessuna stringa
- la lettera b compare in due posizioni differenti nell'espressione, e dunque quando la stringa ne contiene una ci si potrebbe chiedere quale delle due comparse di b la ha derivata; tuttavia i generatori b_3 e b_4 sono sempre distinguibili in quanto b_3 è preceduto da a e b_4 è seguito da c; dunque la lettera b non può essere derivata ambiguamente in nessuna stringa perché si distingue sempre guardando la lettera che precede o segue

In conclusione l'espressione R_1 non può generare la stessa stringa in due o più modi diversi e pertanto non è ambigua.

L'espressione regolare R_2 è ambigua. Osservando la numerazione si vede infatti che la stringa a è generata in due modi come a_1 e a_3. Più in generale le stringhe a^+ sono tutte ambigue in quanto generate come $(\ a_1 \mid a_3\ )^+$, sottoespressione derivata da R_2 e molto ambigua. Si ha infatti che il grado di ambiguità di a^n ($n \geq 1$) vale 2^n ed è addirittura esponenziale (si lascia la verifica al lettore).

(c) Il linguaggio L_1 non è locale: stando ai digrammi $a\,a$ e $a\,b$ di R_1 si dovrebbe avere $a\,a\,a\,b \in L_1$, ma R_1 non genera tale stringa.

Il linguaggio L_2 è locale. Infatti l'espessione R_2 si riscrive raccogliendo così:

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

La forma riscritta è di tipo lineare (ogni generatore compare una sola volta) e pertanto il linguaggio L_2 è locale.

(d) Ricorrendo all'intuizione si trova rapidamente l'intersezione R_3 delle due espressioni R_1 e R_2. Si cominci con il porre la formulazione seguente di R_3:

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

da dove però bisogna eliminare l'operatore di intersezione. Se nel secondo membro dell'intersezione l'elemento c^* viene derivato e genera una o più c, si forma il digramma $a\,c$ che il primo membro non è in grado di produrre. Pertanto nel secondo membro l'elemento c^* deve sempre derivarsi in ε. Si ha dunque la semplificazione seguente:

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

Ora però il secondo membro dell'intersezione non contiene (più) il generatore c e dunque neppure il primo lo può derivare. Giacché c compare nel secondo componente $b\,c$ dell'unione, si può semplicemente rimuovere tale componente e di conseguenza anche l'unione. Si ha dunque la semplificazione seguente:

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

che si riduce ulteriormente come segue (dato che a deriva da $a\,b^*$):

$$R_3 = (\; a\; (\; a\; b\;)^*\;) \;\cap\; (\; a\; b^*\;)^*$$

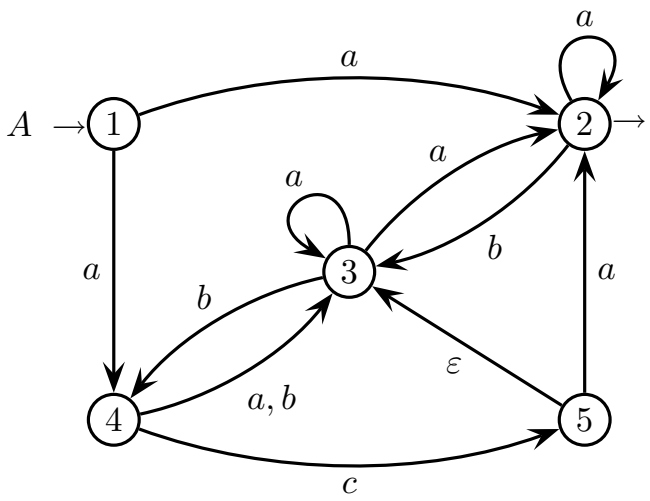
Così l'espressione R_3 si è alquanto ridotta e si possono confrontare direttamente i due membri dell'intersezione. Il primo genera stringhe di tipo $a\,a\,b\,a\,b\,\ldots\,a\,b$ e il secondo di tipo $a\,b\,\ldots\,b\,a\,b\,\ldots\,b\,\ldots\,a\,b\,\ldots\,b$; in effetti il secondo genera il linguaggio universale di alfabeto $\{a, b\}$ dove ogni stringa tranne ε inizia con a. Evidentemente il primo insieme di stringhe è un sottinsieme del secondo e dunque si ha la semplificazione finale seguente:

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

che rappresenta l'espressione regolare R_3 cercata. Si vede pure facilmente che questa formulazione di R_3 è non ambigua.

In alternativa si può ricorrere ai metodi algoritmici: trasformare R_1 e R_2 in automi (det. o anche non det.) tramite l'algoritmo di Berri-Sethi, costruire l'automa intersezione tramite prodotto di automi e ritrasformare l'automa intersezione in espressione regolare tramite il metodo di eliminazione dei nodi (Brozozowski) o altro metodo affine; il procedimento arriva senz'altro in porto e non richiede grandi doti d'intelletto, ma è un po' lungo e tedioso da svolgere.

2. È dato l'automa non deterministico A mostrato in figura, di alfabeto $\{a, b, c\}$.



Si risponda alle domande seguenti:

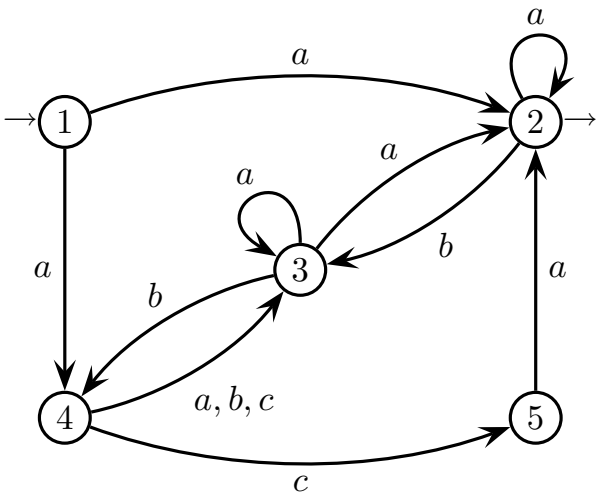
(a) Si costruisca un automa deterministico A' equivalente ad A e se ne tracci il grafo stato-transizione.

(b) Se necessario, si trovi l'automa minimo A'' equivalente ad A' e se ne tracci il grafo stato-transizione.

(c) (facoltativa) Si trovi un'espresione regolare R, preferibilmente non ambigua, che genera il linguaggio $L(A)$.

Soluzione

(a) Si osservi che l'automa A è indeterministico in quanto ha una transizione spontanea e archi uscenti dallo stesso nodo ma con etichette identiche.
Si procede dapprima all'eliminazione della transizione spontanea $5 \xrightarrow{\varepsilon} 3$: gli archi entranti nello stato 5 vanno duplicati nello stato 3. Ecco il risultato:



Ora si può applicare la costruzione classica dei sottinsiemi tramite tabella dei successori:

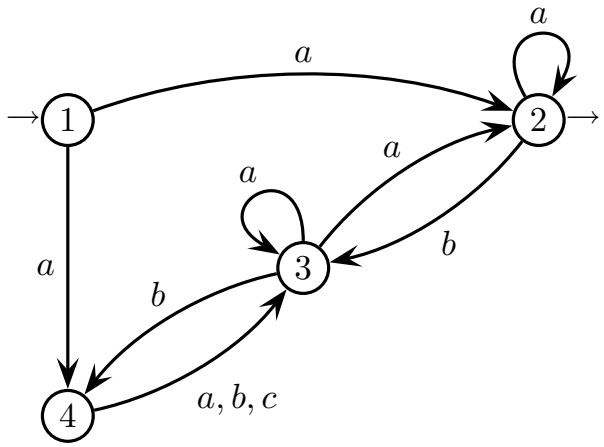
gruppo	a	b	c	finale?
1	2 4	–	–	no
2 4	2 3	3	3	sì
2 3	2 3	3 4	–	sì
3	2 3	4	–	no
3 4	2 3	4	3 5	no
4	3	3	3 5	no
3 5	2 3	4	–	no

Rinominando i gruppi di stati si ottiene la tabella degli stati seguente:

stato	a	b	c	finale?
α	β	–	–	no
β	γ	δ	δ	sì
γ	γ	ε	–	sì
δ	γ	ζ	–	no
ε	γ	ζ	η	no
ζ	δ	δ	η	no
η	γ	ζ	–	no

Questa è la tabella degli stati dell'automa deterministico A' (stato iniziale α). Si lascia al lettore il compito di tracciarne il grafo stato-transizione.

Giova fare un'osservazione: dopo avere eliminato la transizione spontanea (domanda (a)) si ha un automa ancora indeterministico (si riveda la figura), dove però lo stato 5 è facilmente eliminabile in modo intuitivo (cioè senza disporre di un algoritmo formalizzato di minimizzazione indeterministica). Infatti il percorso $4 \xrightarrow{c} 5 \xrightarrow{a} 3$ esiste anche come $4 \xrightarrow{c} 3 \xrightarrow{a} 3$. Pertanto si può togliere lo stato 5 e i due archi che si appoggiano a esso. Ecco il risultato:



Beninteso questo automa è ancora indeterministico perché ha archi uscenti con etichette duplicate, ma ha meno stati del precedente. Si può effettuare la costruzione dei sottinsiemi e ottenere un automa deterministico equivalente, che presumibilmente avrà meno stati del corrispettivo precedente.

Eliminare dall'automa indeterministico lo stato 5, che di per sé è utile (infatti è sia raggiungibile sia definito), a tutti gli effetti è una minimizzazione di stati. Il lettore tenga presente che un algoritmo di mimimizzazione di automi indeterministici esiste, ma è più complesso che nel caso deterministico e in generale il risultato non è unico. Qui si riesce comunque a minimizzare senza bisogno di ricorrere a tale teoria perché il caso è molto semplice.

(b) Riprendendo la tabella degli stati dell'automa deterministico A', si vede subito che le righe δ e η sono identiche e che pertanto gli stati δ e η sono indistinguibili. La tabella si semplifica nel modo seguente:

stato	a	b	c	finale?
α	β	$-$	$-$	no
β	γ	δ	δ	sì
γ	γ	ε	$-$	sì
δ	γ	ζ	$-$	no
ε	γ	ζ	δ	no
ζ	δ	δ	δ	no

Questa tabella degli stati è in forma minima per i motivi seguenti:

- gli stati finali β e γ sono distinguibili in colona c
- lo stato α (non finale) è distinguibile da ogni altro stato non finale (δ ε e ζ) in colonna b e c
- lo stato δ (non finale) è distinguibile dagli stati non finali ε e ζ in colonna c
- gli stati (non finali) ε e ζ sono distinguibili in colonna a, dove l'equivalenza $\gamma \sim \delta$ non sussiste (giacché γ è finale e δ non lo è)

Pertanto quella data sopra con sei stati è la tabella dell'automa minimo A''. Si lascia al lettore il compito di tracciarne il grafo stato-transizione.

Naturalmente si sarebbe potuto minimizzare l'automa partendo dalla versione deterministica con meno stati di cui si è discusso prima (domanda (a)).

(c) Per ottenere l'espressione regolare R, non ambigua ed equivalente all'automa A, si può per esempio procedere in modo algoritmico ricavando R dall'automa deterministico (e minimo) A'' mediante il metodo di eliminazione dei nodi (Brozozowski); si lascia il compito al lettore. In alternativa si può ricorrere al metodo delle equazioni linguistiche lineari a destra.

Contentandosi di un'espressione regolare R qualunque, dunque potenzialmente ambigua, si può partire direttamente dall'automa indeterministico A.

2 Grammatiche libere e automi a pila 20%

1. Le operazioni p (= *producer*) e c (= *consumer*) inseriscono e tolgono, rispettivamente, un elemento da un tampone (buffer) di dimensione illimitata; all'inizio il tampone è vuoto. Una sequenza di operazioni $x \in \{c, p\}^*$ (eventualmente nulla) è valida se non causa "underflow" del tampone, ossia se non tenta di togliere un elemento quando il tampone è vuoto. Ecco alcuni esempi di stringhe valide:

   $$\varepsilon \qquad p \qquad pc \qquad pp \qquad ppc \qquad pcpc \qquad ppcpcc \qquad \ldots$$

   Ed ecco alcuni esempi di stringhe non valide:

   $$c \qquad cc \qquad cp \qquad pcc \qquad \ldots$$

   Il linguaggio L è definito come insieme di tutte le sequenze valide.

   Si risponda alle domande seguenti:

   (a) Si descriva a parole il funzionamento di un automa a pila A che riconosce il linguaggio L.

   (b) Per definire il linguaggio L si scriva una grammatica libera G, in forma non estesa (BNF) e preferibilmente non ambigua.

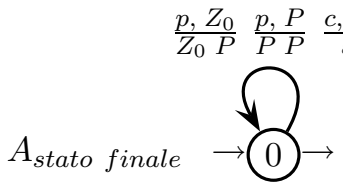
   (c) (facoltativa) Ci sono due tamponi (buffer), e due coppie produttore-consumatore p_1, c_1 e p_2, c_2: la coppia p_1, c_1 opera sul primo tampone e la coppia p_2, c_2 sul secondo; all'inizio ciascuno dei due tamponi è vuoto. Una sequenza di operazioni di entrambe le coppie è valida (come per esempio $p_1\, p_2\, c_1\, c_2$) se non causa "underflow" in nessuno dei due tamponi. Si riconsideri la domanda (a) per il linguaggio delle sequenze valide.

Soluzione

(a) Un automa a pila A riconosce facilmente il linguaggio L simulando il tampone per mezzo della pila. Il calcolo inizia con pila vuota. Conviene scegliere la modalità di accettazione a stato finale, e non a pila vuota, poiché il linguaggio L ha la proprietà di prefisso: data una generica stringa x appartenente a L, ogni prefisso di x (compreso ε) è anch'esso una stringa appartenente a L. Di conseguenza per riconoscere a pila vuota si dovrebbe programmare lo svuotamento della pila.

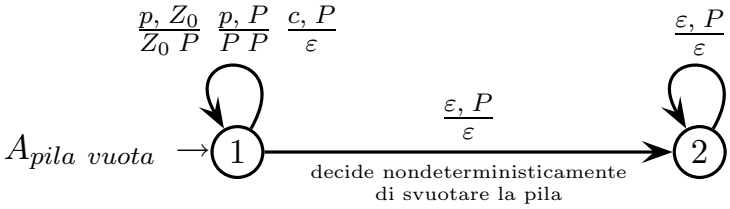
L'automa A funziona così: se la pila è vuota o in cima c'è il simbolo P, l'automa legge la lettera p e impila P; se in cima c'è il simbolo P, l'automa legge la lettera c e spila P; se la pila è vuota e in ingresso c'è la lettera c, l'automa va in stato di errore; e ogni stato è di riconoscimento (tranne quello di errore).

Benché non sia richiesto ecco il grafo stato-transizione dell'automa a pila A, con un solo stato sia iniziale sia finale (giacché l'automa riconosce ogni prefisso):



L'automa A è deterministico e chiaramente riconosce il linguaggio L, giacché accetta qualunque stringa purché in ingresso non ci sia una lettera c quando la pila (buffer) è vuota, dato che la transizione $c,\ Z_0$ è l'unica mancante (ovvero è presente ma porta nello stato di errore qui non messo in evidenza).

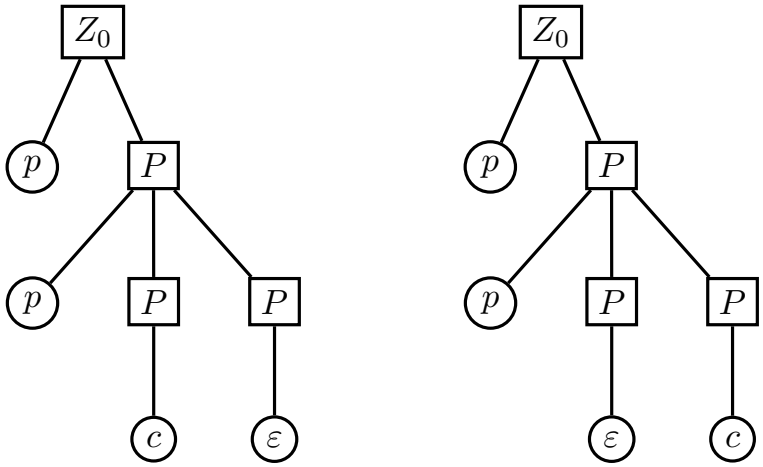
(b) Per trovare rapidamente una grammatica G che genera il linguaggio L, basta emulare il funzionamento dell'automa A modificato in modo da riconoscere a pila vuota. È sufficiente immaginare che l'automa A decida nondeterministicamente di svuotare la pila in un momento qualunque. Eccolo:



Naturalmente l'automa $A_{pila\ vuota}$ è nondeterministico. Dato che ora A funziona a pila vuota e che gli stati finiti qui non hanno alcun effetto sulla lettura dell'ingresso, ma servono solo per svuotare pila, è facile trasformarlo in grammatica: basta immaginare di mappare le transizioni in regole grammaticali ignorando gli stati. Ecco la grammatica G così costruita (assioma Z_0):

$$G \begin{cases} Z_0 \to p\,P \mid \varepsilon & \text{- inizio legge } p \text{ e impila } P \\ P \to p\,P\,P & \text{- legge } p \text{ e impila } P \\ P \to c & \text{- legge } c \text{ e spila } P \\ P \to \varepsilon & \text{- svuota nondet. la pila} \end{cases}$$

Tanto basta per rispondere alla domanda. Tuttavia la grammatica G è molto ambigua, perché se le lettere c sono meno numerose delle lettere p ci sono molti gradi di libertà nello scegliere tra le regole alternative $P \to c$ e $P \to \varepsilon$; ciò corrisponde a decidere nondeterministicamente se leggere o svuotare la pila. Per esempio si consideri la stringa $p\,p\,c$, che ha due alberi sintattici differenti:



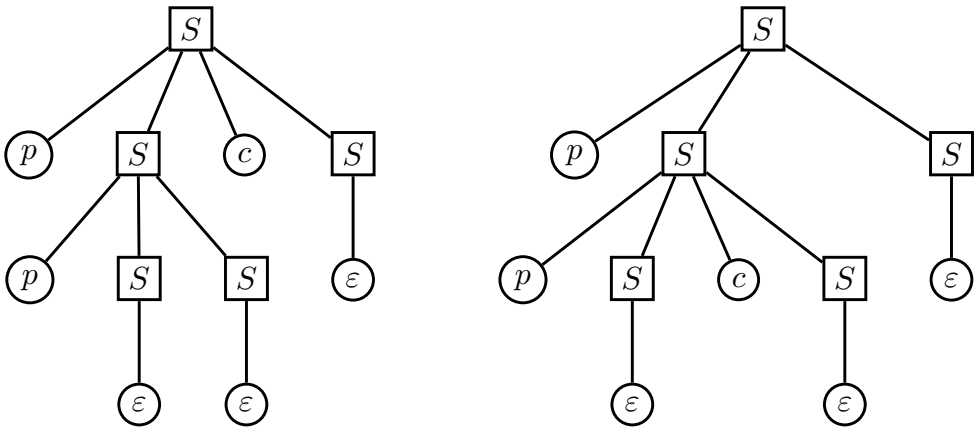
Tuttavia l'automa A iniziale (a stato finale) riconosce il linguaggio L ed è deterministico, pertanto L non è inerentemente ambiguo e l'esistenza di una versione

non ambigua della grammatica G è certa; il lettore può provare da sé a trovarne una ispirata alla struttura dell'automa A. Però tale approccio non sembra agevole perché l'automa deterministico A riconosce con stato finale e non a pila vuota, e d'altro canto una versione deterministica dell'automa a pila vuota è impossibile perché L è un linguaggio con prefissi.

In alternativa, per trovare altrettanto rapidamente una soluzione basta rendersi conto che L è il linguaggio standard di Dyck con parentesi aperta p e chiusa c, modificato premettendo alla parentesi chiusa c di mancare. Infatti in ogni prefisso di una stringa di Dyck il numero di parentesi aperte p non è mai inferiore a quello di parentesi chiuse c (ossia il tampone non va mai in underflow) e ciò è vero a maggior ragione se si permette alla parentesi chiusa di mancare. La grammatica standard di Dyck è ben nota, per esempio $S \to p\ S\ c\ S \mid \varepsilon$. Ecco una grammatica G che genera il linguaggio L, ispirata a Dyck (assioma S):

$$G \begin{cases} S \to p\ S\ c\ S & \text{- coppia } p\,c \\ S \to p\ S\ S & \text{- } c \text{ mancante} \\ S \to \varepsilon & \text{- fine} \end{cases}$$

La grammatica G non è altro che quella standard di Dyck data prima, dove però la regola $S \to p\ S\ c\ S$ ha un'alternativa con terminale c mancante. Tanto basta per rispondere alla domanda e non è più complicato di prima. Per chiarezza ecco sotto a sinistra l'albero sintattico della stringa $p\,p\,c$, ottenuto dalla stringa di Dyck $p\,p\,c\,c$ eliminando la prima c:



Si noti tuttavia che pure questa versione della grammatica G è ambigua, in quanto la stringa $p\,p\,c$ è ottenibile dalla stringa di Dyck $p\,p\,c\,c$ eliminando la seconda c (non la prima); l'albero sintattico sopra a destra mostra tale possibilità. Può anche succedere che due stringhe di Dyck differenti diano per cancellazione la stessa stringa di L: per esempio $p\,p$ deriva sia da $p\,p\,c\,c$ sia da $p\,c\,p\,c$.

Non è difficile disambiguare questa versione della grammatica G, tuttavia occorre analizzare meglio la struttura del linguaggio L. Si indichino con $D_{null}\,(p,c)$ e $D_{non\ null}\,(p,c)$ i linguaggi di Dyck di alfabeto $\{p,c\}$ con e senza ε, rispettivamente. Il linguaggio L ammette allora la rappresentazione seguente:

$$L = D_{null}\,(p,c)\ \left(\ p^{+}\ D_{non\ null}\,(p,c)\ \right)^{*}\ p^{*}$$

Si giustifica facilmente tale rappresentazione così. Una frase x del linguaggio L è una sequenza arbitraria di lettere p e c con la condizione che x e ogni suo prefisso contengano un numero di c non superiore a quello di p (ovvero uguale o inferiore o anche nullo). È noto che una generica frase del linguaggio $D_{null}\,(p,c)$ contiene uguale numero di parentesi aperte p e chiuse c, e che ogni suo prefisso contiene un numero di c non superiore a quello di p; pertanto $D_{null}\,(p,c)$ è contenuto in L. Tuttavia L è più ampio di $D_{null}\,(p,c)$, perché una generica frase x di L potrebbe globalmente contenere un numero di lettere p superiore a quello di c. Nel caso generale la frase x di L è dunque modellabile come una sequenza di frasi di Dyck intercalate da lettere p aggiuntive; come casi particolari la frase x può ridursi a una sola frase di Dyck ed eventualmente avere lettere p in testa e / o in fondo. Espandendo la scomposizione data sopra si ha infatti quanto segue:

$$x = \underbrace{D\,(p,c)}_{\text{Dyck con } \varepsilon}\ \underbrace{\underbrace{p\ \ldots}_{p^+}\ \underbrace{D\,(p,c)}_{\text{Dyck senza } \varepsilon}\ \ \ldots\ \underbrace{p\ \ldots}_{p^+}\ \underbrace{D\,(p,c)}_{\text{Dyck senza } \varepsilon}}_{\text{può mancare}}\ \underbrace{\ldots}_{p^*}$$

Tale scomposizione del linguaggio L è non ambigua: si tratta di una lista di elementi del tipo $p^+\,D_{non\ null}\,(p,c)$ (con elemento iniziale e finale diverso), che può essere generata in modo non ambiguo, contenente il linguaggio di Dyck standard, che ha una nota grammatica non ambigua. Per inciso, si noti che se internamente ci fosse Dyck con ε si formerebbero strutture ambigue del tipo $p^+\,\varepsilon\,p^+$: per esempio p^3 sarebbe generabile in due modi diversi come $p^2\,p$ o $p\,p^2$. Osservando che vale l'identità $D_{non\ null}\,(p,c) = (\ p\ D_{null}\,(p,c)\ \ c\ )^+$, la scomposizione si riscrive in forma più uniforme così (ed è sempre non ambigua):

$$L = D_{null}\,(p,c)\ \ \big(\ p^*\ \ (\ p\ D_{null}\,(p,c)\ \ c\ )^+\ \big)^*\ \ p^*$$

Inoltre vale l'identità $D_{null}\,(p,c) = \varepsilon\ \ \mid\ \ (\ p\ D_{null}\,(p,c)\ \ c\ )^+$. Un attimo di riflessione basta allora per convincersi che la scomposizione si può semplificare ulteriormente così (e resta sempre non ambigua):

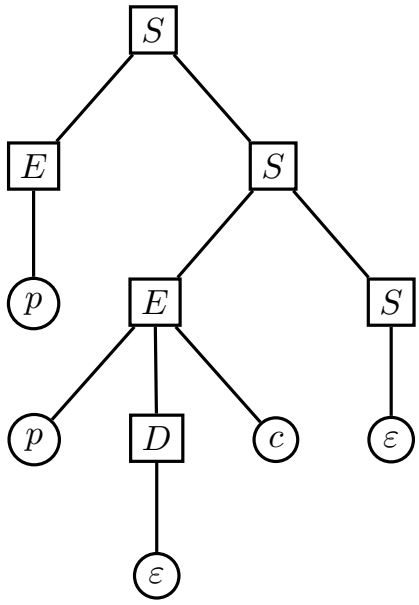
$$L = (\ p\ D_{null}\,(p,c)\ \ c\ \mid\ p\ )^*$$

In definitiva la formulazione asserisce che una generica frase del linguaggio L è una sequenza di nidi di Dyck tipo $p\ \ldots\ c$ eventualmente intercalati da lettere p in eccesso (anche in testa e in coda). Si può pure dire che L è il linguaggio universale costruito sui fattori p e $p\,D\,(p,c)\ c$, ossia di alfabeto p e "nidi di Dyck". Tale scomposizione è semplicissima e del tutto non ambigua, benché forse non immediata da immaginare ancorché illuminante una volta vista.

Ed ecco infine la versione non ambigua della grammatica G che genera il linguaggio L, ricavata proprio dall'ultima scomposizione (assioma S):

$$G\begin{cases} S \ \to\ E\ S & \text{- lista di elementi } E \\ S \ \to\ \varepsilon & \text{- fine di lista} \\ E \ \to\ p\ D\ c & \text{- elemento di lista } (p\ D_{null}\,(p,c)\ \ c) \\ E \ \to\ p & \text{- elemento di lista } (p) \\ D \ \to\ p\ D\ c\ D & \text{- Dyck standard (con } \varepsilon) \\ D \ \to\ \varepsilon & \text{- fine di Dyck} \end{cases}$$

Quest'ultima versione della grammatica G che riconosce il linguaggio L, è non ambigua per costruzione. Per esempio ecco l'albero sintattico della stringa $p\,p\,c$:



È facile convincersi che l'albero sintattico è unico, per questa e ogni altra stringa del linguaggio L, mentre nelle versioni precedenti della grammatica G ce n'è più d'uno. Per esempio la stringa $p\,p$, che si ottiene cancellando le lettere c dalle due stringe di Dyck differenti $p\,c\,p\,c$ e $p\,p\,c\,c$, ora è modellabile univocamente come due sole lettere p in eccesso (e nessun nido di Dyck); mentre la stringa $p\,p\,c$, che si ottiene cancellando una sola lettera c dalle due stesse stringe di Dyck, ora è modellabile univocamente come:

$$p \quad \underbrace{p\,c}_{\text{nido di Dyck}}$$

ossia come una sola lettera p in eccesso seguita da un solo nido di Dyck; e si immagina facilmente come funziona per stringhe di L via via più articolate, tipo:

$$p\,p\,p\,c\,p\,c\,c\,p\,c\,p\,p\,p\,p\,c\,c\,p \quad \Rightarrow \quad p\,\underbrace{p\,p\,c\,p\,c\,c}_{\text{nido}}\,\underbrace{p\,c}_{\text{nido}}\,p\,p\,\underbrace{p\,p\,c\,c}_{\text{nido}}\,p$$

dove il modo di prendere i nidi di Dyck è chiaramente univoco e dove di conseguenza le lettere p in eccesso (sono 4) sono pure univocamente determinate.

(c) Una sola pila non basta per rappresentare lo stato dei due tamponi. Pertanto è necessario un automa di tipo più complesso, come una macchina di Turing o anche una macchina dotata di due contatori. Il linguaggio delle sequenze valide non è dunque libero, bensì dipendente dal contesto (di tipo 1 o 0).

2. Un testo marcato, ispirato a XML, può contenere tabelle e liste. Un esempio di tabella con due righe di tre e due elementi ciascuna, è il seguente:

   ⟨TABLE⟩

   ⟨TR⟩ . . . ⟨TD⟩ . . . ⟨TD⟩ . . .

   ⟨TR⟩ . . . ⟨TD⟩ . . .

   ⟨\ TABLE⟩

   dove TR sta per Table Row, TD per Table Divider (ovvero separatore tra elementi di riga) e i puntini indicano i contenuti del testo. La tabella ha almeno una riga e la riga può anche non contenere TD.

   Un esempio di lista di tre elementi è il seguente (anche qui i puntini indicano i contenuti del testo):

   ⟨LIST⟩

   ⟨ITEM⟩ . . . ⟨ITEM⟩ . . . ⟨ITEM⟩ . . .

   ⟨\ LIST⟩

   La lista non può essere vuota.

   Si impongono le seguenti regole di composizione dei due costrutti:

   - un elemento di tabella può essere di tipo lista o testo
   - un elemento di lista può essere di tipo tabella o testo
   - un elemento di tabella non può essere di tipo tabella
   - un elemento di lista non può essere di tipo lista

   Per modellare il testo si usi il simbolo terminale *pt* (che sta per *plain text*), che non è necessario espandere ulteriormente.

   Si scriva una grammatica in forma estesa (EBNF) per tale linguaggio, verificando che essa non sia ambigua.

Soluzione

Per brevità i caratteri terminali sono ridenominati come $\{t, t', r, d, l, l', i, p_t\}$. Una grammatica G possibile è la seguente (assioma S):

$$
G \begin{cases}
S \rightarrow T \mid L & \text{- tabella o lista} \\
T \rightarrow t\ (\ r\ E_T\ (\ d\ E_T\ )^*\ )^+\ t' & \text{- tabella} \\
E_T \rightarrow L \mid p_t & \text{- elemento di tabella} \\
L \rightarrow l\ (\ i\ E_L\ )^+\ l' & \text{- lista} \\
E_L \rightarrow T \mid p_t & \text{- elemento di lista}
\end{cases}
$$

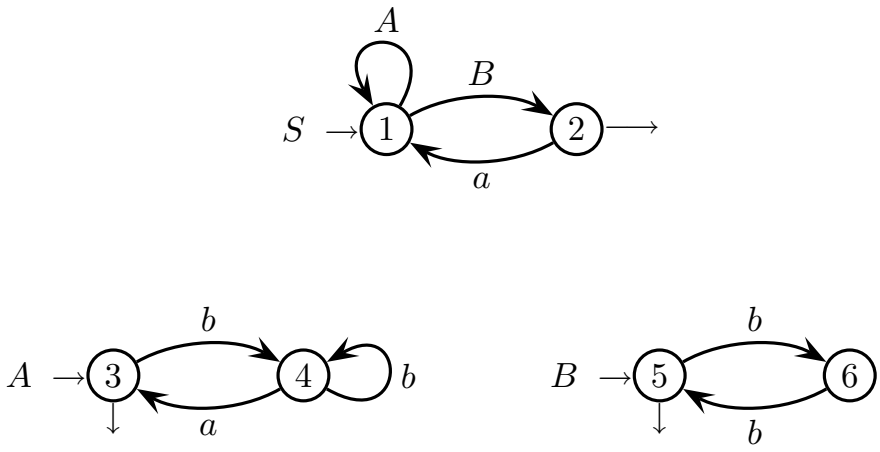
La grammatica G è non ambigua per costruzione. Volendo riscriverla molto formalmente senza abbreviazioni e con classi sintattiche dai nomi autoesplicativi, rieccola (assioma PSEUDO_XML):

$$
\begin{aligned}
\langle \text{PSEUDO\_XML} \rangle &\rightarrow \langle \text{TABLE} \rangle \mid \langle \text{LIST} \rangle \\
\langle \text{TABLE} \rangle &\rightarrow \text{`}\langle \text{TABLE} \rangle\text{'} \left( \text{`}\langle \text{TR} \rangle\text{'} \ \langle \text{T\_ELEM} \rangle \ \left(\text{`}\langle \text{TD} \rangle\text{'} \ \langle \text{T\_ELEM} \rangle \ \right)^{*} \right)^{+} \text{`}\langle \backslash \text{TABLE} \rangle\text{'} \\
\langle \text{T\_ELEM} \rangle &\rightarrow \langle \text{LIST} \rangle \mid \text{`}pt\text{'} \\
\langle \text{LIST} \rangle &\rightarrow \text{`}\langle \text{LIST} \rangle\text{'} \left( \text{`}\langle \text{ITEM} \rangle\text{'} \ \langle \text{L\_ELEM} \rangle \ \right)^{+} \text{`}\langle \backslash \text{LIST} \rangle\text{'} \\
\langle \text{L\_ELEM} \rangle &\rightarrow \langle \text{TABLE} \rangle \mid \text{`}pt\text{'}
\end{aligned}
$$

La nuova versione è strutturalmente analoga alla precedente, ma annotata secondo le convenzioni usuali.

3 Analisi sintattica e parsificatori 20%

1. È data la rete di automi in figura seguente:



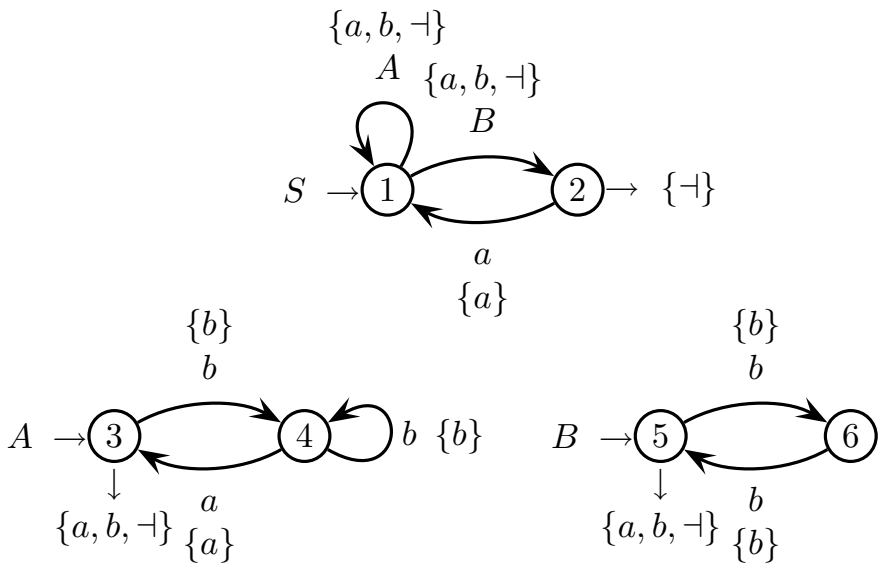
Si risponda alle domande seguenti:

(a) Si calcolino gli insiemi guida $LL(1)$ (scrivendone il valore calcolato nella figura in tutti i punti rilevanti) e si spieghi se la grammatica è $LL(1)$.

(b) Qualora la grammatica non risulti $LL(1)$, si costruisca una grammatica equivalente $LL(1)$.

Soluzione

(a) Ecco gli insiemi guida (solo dove serve):



La grammatica non è di tipo $LL(1)$: per esempio gli insiemi guida degli archi uscenti dallo stato 1 hanno l'elemento b in comune. Si vede facilmente che la grammatica non è $LL(k)$ per nessun $k \geq 1$, giacché gli insiemi guida di ordine k alla biforcazione sullo stato 1 hanno la stringa b^k in comune.

(b) Come dimostrato prima, effettivamente la grammatica non è $LL(1)$. Il linguaggio però è regolare perché la grammatica non contiene derivazioni ricorsive. Ecco l'espressione regolare del linguaggio per ogni macchina della rete:

$$\begin{aligned} L(S) &= (A \mid B\, a)^* \; B \\ L(A) &= (b^+ \, a)^* \\ L(B) &= (b\, b)^* \end{aligned}$$

Sostituendo nella prima equazione le altre si ottiene la soluzione:

$$L(S) = \big((b^+ \, a)^* \mid (b\, b)^* \; a \big)^* \; (b\, b)^*$$

Ora si può costruire un automa deterministico a stati finiti equivalente con il metodo di Berry-Sethi, da cui poi si ottiene subito la grammatica lineare a destra che risulta $LL(1)$ per costruzione. Si lascia il noioso compito al lettore. Peraltro, dato che b^+ deriva $(b\,b)^+$ si ha l'identità seguente:

$$(b^+ \, a)^* \mid (b\, b)^* \; a = (b^+ \, a)^* \mid a$$

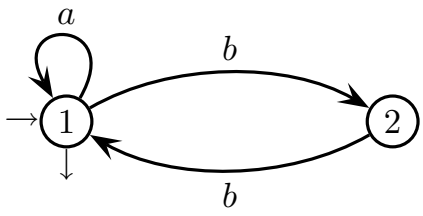
e di conseguenza si ha l'ovvia identità seguente:

$$\big((b^+ \, a)^* \mid a \big)^* = \varepsilon \mid (a \mid b)^* \; a = \varepsilon \mid \Sigma^* \, a$$

che si giustifica subito pensando a come sono fatte le stringhe generate dai due membri (e dove $\Sigma = \{a, b\}$ è l'alfabeto). Dunque segue:

$$L(S) = (\varepsilon \mid \Sigma^* \, a) \; (b\, b)^* = (b\, b)^* \mid \Sigma^* \, a \; (b\, b)^*$$

Ciò si legge dicendo che il linguaggio $L(S)$ è l'insieme di stringhe formate da un numero pari di sole b e di quelle che terminano con a seguita da un numero pari di sole b (zero conta come pari). Intuitivamente ecco l'automa deterministico:



L'automa è minimo. Dall'automa è immediato ricavare la grammatica lineare a destra, di tipo $LL(1)$ per costruzione. Eccola (assioma A):

$$G \begin{cases} A \;\to\; a\, A \mid b\, B \mid \varepsilon \\ B \;\to\; b\, A \end{cases}$$

Presumibilmente è impossibile trovarne una più semplice di questa.

2. È data la seguente grammatica G (assioma S):

$$G \begin{cases} S & \to & a\ X \\ X & \to & Y\ b \\ Y & \to & b\ A \\ Y & \to & B\ a \\ Y & \to & \varepsilon \\ A & \to & a\ B\ a \\ A & \to & a \\ B & \to & b\ A\ b \\ B & \to & b \end{cases}$$

Si risponda alle domande seguenti:

(a) Si analizzi la stringa $a\,b$ con il metodo di Earley, riportando il calcolo nella tabella predisposta.

(b) Si analizzi la stringa $a\,b\,a\,b$ con il metodo di Earley, riportando il calcolo nella tabella predisposta.

(c) Si specifichi il linguaggio generato dalla grammatica G.

(d) (facoltativa) Si dica se la grammatica G sia ambigua, e se lo è si dica se il linguaggio generato da G sia inerentemente ambiguo, spiegandone il motivo; altrimenti si fornisca una grammatica G' equivalente a G ma non ambigua.

Schema di simulazione dell'algoritmo di Earley								
stato 0	pos. a	stato 1	pos. b	stato 2	pos. a	stato 3	pos. b	stato 4	

Soluzione

Per comodità si mostrano subito le tabelle di Earley compilate per le domande (a) e (b); la seconda tabella riprende e completa la prima. Le risposte alle domande seguono.

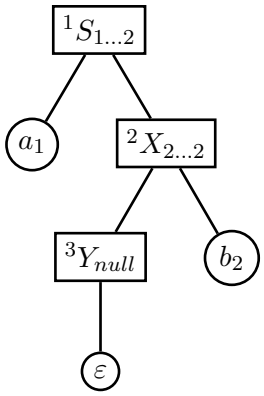
Schema di simulazione dell'algoritmo di Earley - domanda (a)

stato 0	pos. a	stato 1	pos. b	stato 2	pos. a	stato 3	pos. b	stato 4	
$S \rightarrow \bullet\, a\, X$	0	$S \rightarrow a \bullet X$	0	$Y \rightarrow b \bullet A$	1				
		$X \rightarrow \bullet\, Y\, b$	1	$B \rightarrow b \bullet$	1				
		$Y \rightarrow \bullet\, b\, A$	1	$B \rightarrow b \bullet A\, b$	1				
		$Y \rightarrow \bullet\, B\, a$	1	${}^{2}X \rightarrow Y\, b \bullet$	1				
		${}^{3}Y \rightarrow \bullet\, \varepsilon = \varepsilon \bullet$	1	$A \rightarrow \bullet\, a$	2				
		$B \rightarrow \bullet\, b$	1	$A \rightarrow \bullet\, a\, B\, a$	2				
		$B \rightarrow \bullet\, b\, A\, b$	1	$Y \rightarrow B \bullet a$	1				
		$X \rightarrow Y \bullet b$	1	${}^{1}S \rightarrow a\, X \bullet$	0				

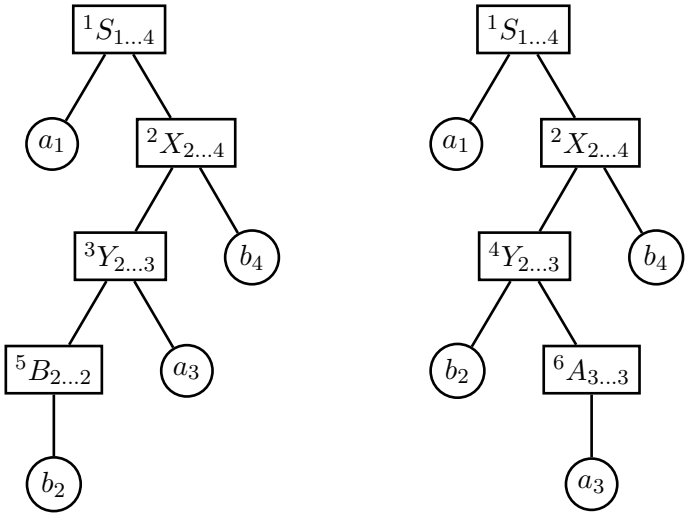
Schema di simulazione dell'algoritmo di Earley - domanda (b)

stato 0	pos. a	stato 1	pos. b	stato 2	pos. a	stato 3	pos. b	stato 4	
$S \rightarrow \bullet\, a\, X$	0	$S \rightarrow a \bullet X$	0	$Y \rightarrow b \bullet A$	1	${}^{6}A \rightarrow a \bullet$	2	$B \rightarrow b \bullet$	3
		$X \rightarrow \bullet\, Y\, b$	1	${}^{5}B \rightarrow b \bullet$	1	$A \rightarrow a \bullet B\, a$	2	$B \rightarrow b \bullet A\, b$	3
		$Y \rightarrow \bullet\, b\, A$	1	$B \rightarrow b \bullet A\, b$	1	${}^{3}Y \rightarrow B\, a \bullet$	1	${}^{2}X \rightarrow Y\, b \bullet$	1
		$Y \rightarrow \bullet\, B\, a$	1	$X \rightarrow Y\, b \bullet$	1	$B \rightarrow \bullet\, b$	3	$A \rightarrow \bullet\, a\, B\, a$	4
		$Y \rightarrow \bullet\, \varepsilon = \varepsilon \bullet$	1	$A \rightarrow \bullet\, a$	2	$B \rightarrow \bullet\, b\, A\, b$	3	$A \rightarrow \bullet\, a$	4
		$B \rightarrow \bullet\, b$	1	$A \rightarrow \bullet\, a\, B\, a$	2	${}^{4}Y \rightarrow b\, A \bullet$	1	$A \rightarrow a\, B \bullet a$	2
		$B \rightarrow \bullet\, b\, A\, b$	1	$Y \rightarrow B \bullet a$	1	$B \rightarrow b\, A \bullet b$	1	$Y \rightarrow B \bullet a$	1
		$X \rightarrow Y \bullet b$	1	$S \rightarrow a\, X \bullet$	0	$X \rightarrow Y \bullet b$	1	${}^{1}S \rightarrow a\, X \bullet$	0

(a) La stringa $a\,b$ è l'unica non ambigua e pertanto viene costruito un solo albero sintattico. La tabella è a pagina precedente. Benché non richiesto, ecco l'albero sintattico con la corrispondenza tra nodi e candidate di riduzione:



(b) L'analisi parte dalla precedente e la completa: la stringa $a\,b\,a\,b$ è ambigua e ha due alberi sintattici. La tabella è a pagina precedente. Benché non richiesti, ecco i due alberi sintattici con la corrispondenza tra nodi e candidate di riduzione:



(c) Il linguaggio $L = L(G)$ è regolare e si formula facilmente come insieme così. Per iniziare, sostituendo ripetutamente si trova in breve quanto segue:

$$L(A) = a \mid a\,b\,a \mid a\,b\,a\,b\,a \mid \cdots = (\,a\,b\,)^*\;a$$

e

$$L(B) = b \mid b\,a\,b \mid b\,a\,b\,a\,b \mid \cdots = (\,b\,a\,)^*\;b$$

Sostituendo ancora si ha:

$$L(Y) = b\,L(A) \mid L(B)\,a \mid \varepsilon = b\;(\,a\,b\,)^*\;a \mid (\,b\,a\,)^*\;b\,a \mid \varepsilon$$

ossia semplificando:

$$L(Y) = (\,b\,a\,)^+ \mid (\,b\,a\,)^+ \mid \varepsilon = (\,b\,a\,)^*$$

E infine, sempre sostituendo, si ha:

$$L(X) = L(Y)\,b = (\,b\,a\,)^*\;b$$

e

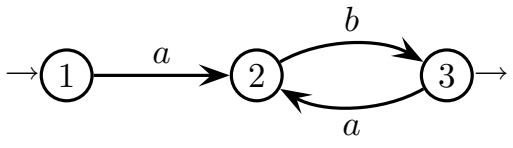
$$L(S) = a\,L(X) = a\;(\,b\,a\,)^*\;b = (\,a\,b\,)^+$$

Riformulando il tutto come insieme si ha dunque:

$$L = (\,a\,b\,)^+ = \{\,(\,a\,b\,)^n \mid \quad n \geq 1\,\}$$

Si noti che sebbene in generale le derivazioni di G siano autoinclusive, qui tuttavia il linguaggio L riesce comunque regolare.

(d) Come visto alla domanda (b) la grammatica G è ambigua. Tuttavia il linguaggio L non è inerentemente ambiguo, come si dimostra dandone una grammatica G' non ambigua: per esempio basta sfruttare il fatto, noto dalla domanda (c), che L è regolare. Ecco l'automa deterministico (e minimo) che riconosce L e la grammatica G' lineare a destra corrispondente (assioma A):



$$G' \begin{cases} A \;\to\; a\,B \\ B \;\to\; b\,C \\ C \;\to\; a\,B \mid \varepsilon \end{cases}$$

Poiché l'automa è deterministico, la grammatica G' corrispondente lineare a destra è di tipo $LL(1)$ e pertanto certamente non ambigua.

Va detto che, inuitivamente, si può anche rispondere più speditamente alla domanda con la grammatica G' seguente: $S \to a\,b\,S \mid a\,b$ (quasi lineare a destra); essa è di tipo $LL(3)$ (ma non di tipo $LL(1)$), dunque certamente non ambigua.

4 Traduzione e analisi semantica 20%

1. Dato il linguaggio $\{\, a^n\, b \mid \quad n \geq 0\}$, che rappresenta l'intero $n \geq 0$, si considerino le due funzioni di traduzione seguenti, che calcolano il quoziente intero e il resto della divisione per tre del numero dato:

$$\begin{aligned} \tau_1\,(\,a^n\,b\,) &= q^{(n \div 3)}\; r^{(n \mod 3)} \\ \tau_2\,(\,a^n\,b\,) &= r^{(n \mod 3)}\; q^{(n \div 3)} \end{aligned}$$

dove gli operatori "$\div$" e "mod" calcolano il quoziente intero e il resto della divisione, rispettivamente.

Esempio:

$$\begin{aligned} \tau_1\,(\,a\,a\,a\,a\,a\,b\,) &= q\,r\,r \\ \tau_2\,(\,a\,a\,a\,a\,a\,b\,) &= r\,r\,q \end{aligned}$$

Si risponda alle domande seguenti:

(a) Si scriva uno schema di traduzione sintattico G' (o grammatica di traduzione) più semplice possibile per calcolare la funzione τ_1.

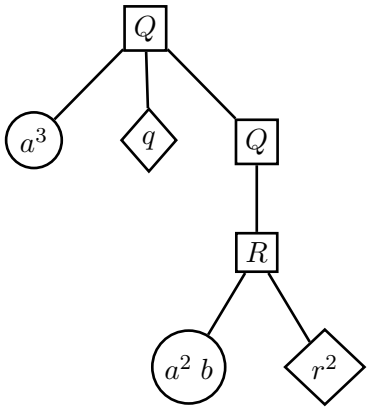
(b) Si scriva uno schema di traduzione sintattico G'' (o grammatica di traduzione) più semplice possibile per calcolare la funzione τ_2.

Soluzione

(a) La traduzione τ_1 è razionale. Ecco lo schema G', con regole di tipo lineare a destra (assioma Q):

$$G' \begin{cases} Q \;\to\; a\;a\;a\;\{\,q\,\}\;Q \\ Q \;\to\; R \\ R \;\to\; b \\ R \;\to\; a\;b\;\{\,r\,\} \\ R \;\to\; a\;a\;b\;\{\,r\,r\,\} \end{cases}$$

Eccone un albero sintattico, per la traduzione $\tau_1\left(a^5\,b\right) = q\,r^2$:

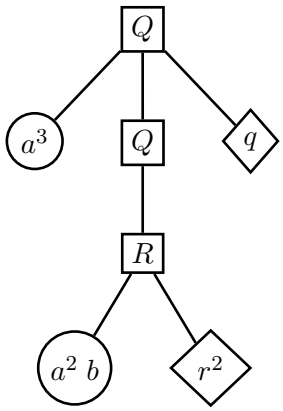


I nonterminali Q e R calcolano quoziente e resto della divisione, rispettivamente.

(b) Si realizza facilmente la traduzione τ_2 come schema sintattico libero (context-free). Ecco lo schema G'' (assioma Q):

$$G'' \begin{cases} Q \rightarrow a\ a\ a\ Q\ \{\, q \,\} \\ Q \rightarrow R \\ R \rightarrow b \\ R \rightarrow a\ b\ \{\, r \,\} \\ R \rightarrow a\ a\ b\ \{\, r\ r \,\} \end{cases}$$

Eccone un albero sintattico, per la traduzione $\tau_2\left(a^5\, b\right) = r^2\, q$:



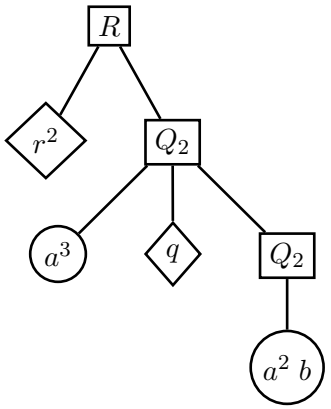
Il nonterminale R calcola il resto della divisione. Il nonterminale Q calcola il quoziente della divisione mediante derivazione autoinclusiva, la quale alla fine genera il resto come prefisso.

La trasduzione τ_2 è realizzabile anche come schema razionale, sia pure in modo un po' involuto facendo appello al nondeterminismo. Infatti si può immaginare un automa a stati finiti nondeterministico che inizialmente, senza leggere niente in ingresso, decide nondeterministicamente quale resto emettere dei tre possibili ε, r e r^2, poi legge l'ingresso mantenendo memoria della scelta indeterministica effettuata ed emette il quoziente (ogni tre lettere a emette una lettera q), e alla fine (quando legge la lettera b) stabilisce qual è il resto effettivo ed entra in uno

stato finale solo se il resto effettivo coincide con la scelta nondeterministica fatta all'inizio (e di cui ha tenuto memoria). Il metodo è un po' macchinoso e usa molti stati, ma funziona. Ecco lo schema sintattico G'' equivalente, ovviamente di tipo lineare a destra (assioma R):

$$
G'' \left\{ \begin{array}{lcl}
R & \rightarrow & Q_0 \\
R & \rightarrow & \{\, r \,\}\ Q_1 \\
R & \rightarrow & \{\, r\, r \,\}\ Q_2 \\
Q_0 & \rightarrow & a\ a\ a\ \{\, q \,\}\ Q_0 \\
Q_0 & \rightarrow & b \\
Q_1 & \rightarrow & a\ a\ a\ \{\, q \,\}\ Q_1 \\
Q_1 & \rightarrow & a\ b \\
Q_2 & \rightarrow & a\ a\ a\ \{\, q \,\}\ Q_2 \\
Q_2 & \rightarrow & a\ a\ b
\end{array} \right.
$$

Lo schema sceglie nondeterministicamente quale resto emettere e memorizza tale scelta nei nonterminali Q_i (il pedice $i = 0, 1, 2$ codifica il resto), calcola ed emette il quoziente (una lettera q ogni tre lettere a) tenendo memoria della scelta, e termina con un numero di lettere a finali (modulo 3) corrispondente al resto scelto all'inizio. Ecco l'albero sintattico della traduzione $\tau_2\left(a^5\, b\right) = r^2\, q$:



Se si interpretano le regole lineari a destra dello schema G'' come mosse di un trasduttore a stati finiti (dove i nonterminali sono gli stati), si ottiene proprio il trasduttore descritto informalmente prima. Si noti che questa versione dello schema G'' è un po' più prolissa della precedente.

2. Tramite una grammatica con attributi, si modelli la traduzione della struttura di controllo "repeat ... until" in istruzioni di salto condizionato a etichette del programma. È data la sintassi G seguente:

$$G \begin{cases} S_0 \rightarrow L & \\ L \rightarrow F \text{ ';' } L & L \text{ è una lista di frasi o istruzioni} \\ L \rightarrow F & F \text{ è una frase o istruzione} \\ F \rightarrow A & A \text{ è un assegnamento} \\ F \rightarrow \mathsf{repeat}\ L\ \mathsf{until}\ C & C \text{ è una condizione} \end{cases}$$

Si usino allo scopo gli attributi seguenti:

tr	codice risultante dalla traduzione di un costrutto - si possono concatenare traduzioni di diversi costrutti mediante l'operatore "‖"
n	attributo intero delle istruzioni utilizzabile per generare le etichette bersaglio dei salti
rc	attributo della condizione C - indica il registro dove porre il risultato della valutazione della condizione

Si utilizzi l'istruzione seguente:

jump-if-false r, en

per codificare il salto all'istruzione di etichetta "en" (dove n è un numero intero positivo) se il valore nel registro "r" è falso. Si faccia uso della funzione *nuovo*, che a ogni invocazione restituisce un diverso valore intero positivo per generare le etichette.

Si risponda alle domande seguenti:

(a) Si definiscano compiutamente gli attributi necessari e si scrivano le funzioni semantiche per il calcolo degli attributi (si vedano le tabelle predisposte).

(b) Si mostri il risultato della traduzione sull'esempio seguente:

sorgente	traduzione da scrivere
$a := 1$;	
$i := 0$;	
repeat	
$\quad a := a * 2$;	
$\quad i := i + 1$	
until $2 * a > k$	

attributi da usare per la grammatica

tipo	nome	(non)terminali	dominio	significato

già dati nel testo dell'esercizio

	tr		stringa	codice risultante dalla traduzione di un costrutto
	n		intero	attributo per generare le etichette dei salti
	rc		stringa	nome di registro

eventualmente da estendere e / o aggiungerne di nuovi

sintassi	*funzioni semantiche*
$S_0 \rightarrow L_1$	
$L_0 \rightarrow F_1$ ';' L_2	
$L_0 \rightarrow F_1$	
$F_0 \rightarrow A_1$	
$F_0 \rightarrow$ repeat L_1 until C_2	

Soluzione

(a) Ecco gli attributi, non ne occorrono di nuovi basta completare quelle già dati:

attributi da usare per la grammatica

tipo	nome	(non)terminali	dominio	significato
già dati nel testo dell'esercizio				
sx	tr	S, L, F	stringa	codice risultante dalla traduzione di un costrutto
dx	n	F	intero	attributo per generare le etichette dei salti
dx	rc	C	stringa	nome di registro dove valutare la condizione di fine ciclo

I due attributi n e rc, che in pratica sono delle funzioni, sono posti di tipo destro come d'uso per gli attributi calcolati tramite funzioni esterne.

Ecco le funzioni semantiche, che riprendono modificandole quelle date nel libro di testo per la generazione del codice del ciclo while:

regola	funzione semantica
$S_0 \rightarrow L_1$	$tr_0 \leftarrow tr_1$
$L_0 \rightarrow F_1$ ';' L_2	$tr_0 \leftarrow tr_1 \cdot tr_2$
$L_0 \rightarrow F_1$	$tr_0 \leftarrow tr_1$
$F_0 \rightarrow$ repeat L_1 until C_2	$tr_0 \leftarrow$ 'e' $\cdot\, n_1 \cdot$ ':' $\cdot\, tr_1 \cdot$ 'jmp-if-false' $\cdot\, rc_2 \cdot$ ',' $\cdot$ 'e' $\cdot\, n_1$
	$n_1 \leftarrow nuovo$

Si ricordi (benché sia un dettaglio minimo) che in Pascal il ciclo repeat tremina quando la condizione diventa vera (contrariamente a quanto fa il ciclo do di C).

(b) Ecco la traduzione. Si suppone che qui le funzioni *nuovo* e *rc* restituiscano l'intero 1 e il nome di registro r, rispettivamente:

sorgente	traduzione da scrivere
$a := 1;$	$tr(a := 1)$
$i := 0;$	$tr(i := 0)$
repeat	e1 :
$\quad a := a * 2;$	$\quad tr(a := a * 2)$
$\quad i := i + 1$	$\quad tr(i := i + 1)$
until $2 * a > k$	jump-if-false r, e1

La grammatica con attributi è sostanzialmente di tipo sintetizzato, i due attributi destri avendo ruolo marginale. Per altre osservazioni si veda il libro di testo.