Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 27 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. Si considerino i due linguaggi seguenti (si osservi che gli alfabeti sono differenti):

$$L_1 = \{\, x \in \{\, a, b \,\}^* \mid \text{il secondo carattere di } x \text{ è } a \,\} \tag{1}$$

$$L_2 = \{\, x \in \{\, a, b, c \,\}^* \mid \text{il penultimo carattere di } x \text{ è } a \,\} \tag{2}$$

Spiegando i ragionamenti applicati, si costruiscano gli automi seguenti:

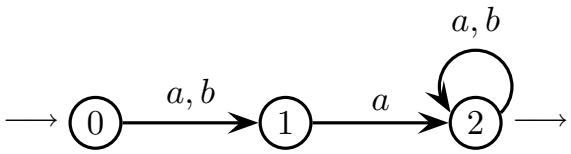
(a) l'automa deterministico minimo che riconosce il linguaggio unione $L_1 \cup L_2$

(b) l'automa deterministico minimo che riconosce il linguaggio unione $L_1 \cup (L_2)^R$

Soluzione

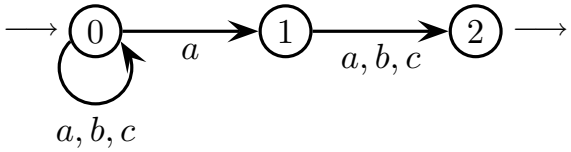
(a) Volendo procedere algoritmicamente, conviene dare prima, in modo intuitivo, i due automi A_1 e A_2 (eventualmente indeterministici) dei linguaggi L_1 e L_2, rispettivamente, poi unirli (indeterministicamente), poi determinizzare l'automa risultante a infine minimizzarlo. Ecco i due automi di partenza:
Automa di $L_1 = (\, a \mid b \,)\; a \;(\, a \mid b \,)^*$:

Automa A_1:



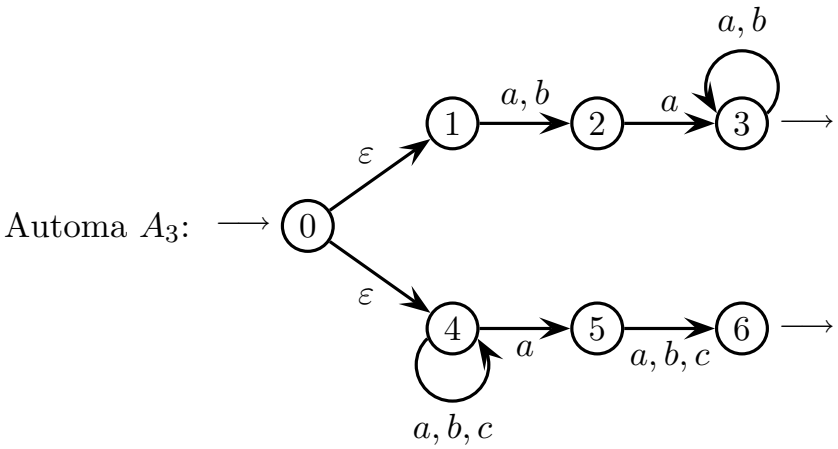
L'automa A_1 risulta già deterministico.
Automa di $L_2 = (\, a \mid b \mid c \,)^*\; a \;(\, a \mid b \mid c \,)$:

Automa A_2:



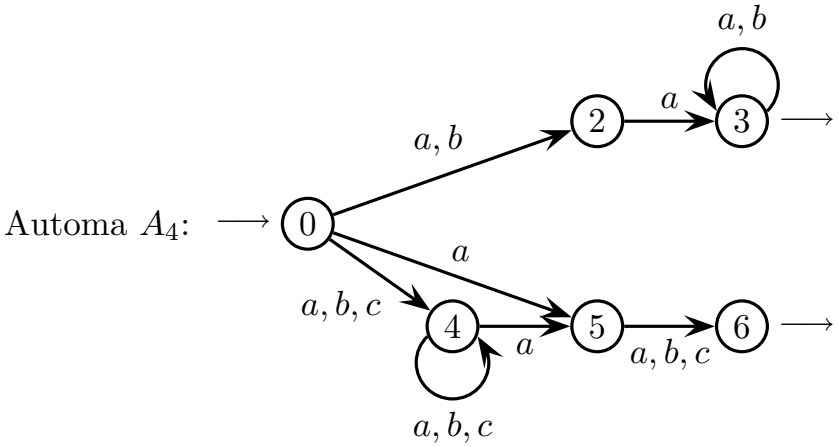
L'automa A_2 risulta indeterministico (nello stato 0).
Volendo, si potrebbero dare algoritmicamente i due automi A_1 e A_2, per esempio mediante i metodi di Berri-Sethi o McNaughton-Yamada (nel secondo caso gli automi sarebbero deterministici). Qui si procede con i due automi intuitivi.

Automa unione (indeterministica) A_3:



L'automa A_3 risulta indeterministico negli stati 0 e 4. Dallo stato 0 escono transizioni spontanee (ε-transizioni).

Eliminazione delle transizioni spontanee (produce l'automa A_4):



L'automa A_4 risulta ancora indeterministico negli stati 0 e 4 (ma senza transizioni spontanee).

Costruzione dei sottinsiemi (produce l'automa A_5):

Automa A_5

gruppo	a	b	c	finale?
0	2 4 5	2 4	4	no
2 4 5	3 4 5 6	4 6	4 6	no
2 4	3 4 5	4	4	no
4	4 5	4	4	no
3 4 5 6	3 4 5 6	3 4 6	4 6	sì
4 6	4 5	4	4	sì
3 4 5	3 4 5 6	3 4 6	4 6	sì
4 5	4 5 6	4 6	4 6	no
3 4 6	3 4 5	3 4	4	sì
4 5 6	4 5 6	4 6	4 6	sì
3 4	3 4 5	3 4	4	sì

stato	a	b	c	finale?
A	B	C	D	no
B	E	F	F	no
C	G	D	D	no
D	H	D	D	no
E	E	I	F	sì
F	H	D	D	sì
G	E	I	F	sì
H	L	F	F	no
I	G	M	D	sì
L	L	F	F	sì
M	G	M	D	sì

Minimizzazione dell'automa A_5. Poiché le righe I e M coincidono, come pure le righe E e G, gli stati rispettivi sono indistinguibili e si hanno subito le equivalenze $I \sim M$ e $E \sim G$. Dunque si ottiene l'automa A_6:

Automa A_6

stato	a	b	c	finale?
A	B	C	D	no
B	E	F	F	no
C	E	D	D	no
D	H	D	D	no
E	E	I	F	sì
F	H	D	D	sì
H	L	F	F	no
I	E	I	D	sì
L	L	F	F	sì

Non sarebbe difficile constatare intuitivamente che gli stati rimanenti sono tutti distinguibili.

Comunque, la verifica formale di minimalità va fatta mediante la tabella triangolare delle implicazioni. Eccola:

B	BE CF DF							
C	BE CD	DF						
D	BH CD	EH FD	EH					
E	×	×	×	×				
F	×	×	×	×	EH DI DF			
H	BL CF DF	EL	EL DF	HL DF	×	×		
I	×	×	×	×	FD	EH DI	×	
L	×	×	×	×	EL FI	HL DF	×	EL FI DF
	A	B	C	D	E	F	H	I

Ora bisogna effettuare la propagazione delle distinguibilità. Nella tabella sotto, in **rosso** sono marcate le coppie di stati distinguibili direttamente.

B	~~BE~~ ~~CF~~ ~~DF~~							
C	~~BE~~ CD	~~DF~~						
D	BH CD	~~EH~~ ~~DF~~	~~EH~~					
E	×	×	×	×				
F	×	×	×	×	~~EH~~ ~~DI~~ ~~DF~~			
H	~~BL~~ ~~CF~~ ~~DF~~	EL	EL ~~DF~~	~~HL~~ ~~DF~~	×	×		
I	×	×	×	×	~~DF~~	~~EH~~ ~~DI~~	×	
L	×	×	×	×	EL FI	~~HL~~ ~~DF~~	×	EL FI ~~DF~~
	A	B	C	D	E	F	H	I

Ci sono ancora tre coppie di stati irrisolte (AC, BH e EL). Nella tabella sotto, sono rimossi i vincoli ormai irrilevanti e in **verde** sono marcate le coppie di stati resi distinguibili nella tabella precedente.

B	×							
C	×	×						
D	BH ~~CD~~	×	×					
E	×	×	×	×				
F	×	×	×	×	×			
H	×	EL	×	×	×	×		
I	×	×	×	×	×	×	×	
L	×	×	×	×	EL ~~FI~~	×	×	×
	A	B	C	D	E	F	H	I

Resta una sola coppia di stati irrisolta (BH). Nella tabella sotto, sono rimossi i vincoli ormai irrilevanti e in **blu** sono marcate le coppie di stati resi distinguibili nella tabella precedente.

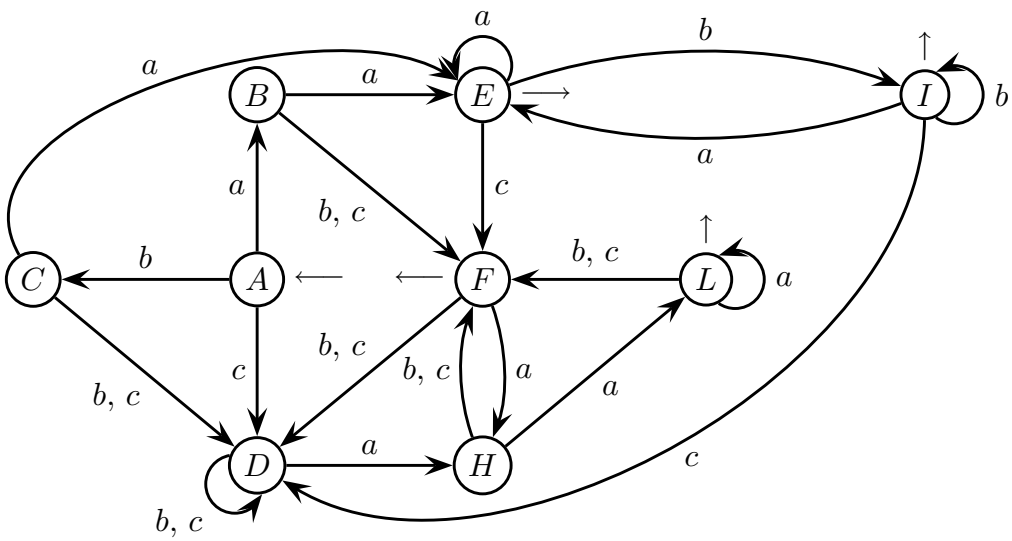
B	×							
C	×	×						
D	×	×	×					
E	×	×	×	×				
F	×	×	×	×	×			
H	×	~~EL~~	×	×	×	×		
I	×	×	×	×	×	×	×	
L	×	×	×	×	×	×	×	×
	A	B	C	D	E	F	H	I

Tabella finale

B	×							
C	×	×						
D	×	×	×					
E	×	×	×	×				
F	×	×	×	×	×			
H	×	×	×	×	×	×		
I	×	×	×	×	×	×	×	
L	×	×	×	×	×	×	×	×
	A	B	C	D	E	F	H	I

A destra è riportata la tabella finale. Non ci sono più coppie di stati irrisolte e tutte le coppie risultano distinguibili. Pertanto l'automa A_6 è già in forma minima. Eccone il grafo:

Automa A_6:

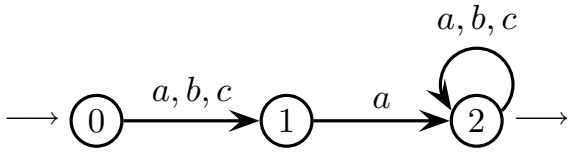


L'automa A_6 ha nove stati, è deterministico e minimo, e pertanto è anche unico. Il numero elevato di stati, più numerosi della somma degli stati dei due automi di partenza A_1 e A_2 (nonché la topologia un po' intricata del grafo), non deve sorprendere: è necessario per avere determinismo. Infatti l'unione dei due linguaggi originari L_1 e L_2 è ambigua, perché essi hanno stringhe in comune (per esempio la stringa $b\,a\,b$). Inoltre i due rispettivi alfabeti sono diversi, e pertanto una stringa contenente la lettera c appartiene necessariamente solo a L_2 benché un suo prefisso senza c possa appartenere solo a L_1 (per esempio la stringa $b\,a\,c$ appartiene solo a L_2 ma il suo prefisso $b\,a$ appartiene solo a L_1). Questi due comportamenti, più o meno indeterministici, sono senz'altro determinizzabili (ciò si può sempre fare con un automa a stati finiti), ma al costo di usare numerosi stati. Ciò risponde alla domanda.

(b) Se si prende l'immagine riflessa di una stringa, il secondo e il penultimo carattere si scambiano di posizione (o la mantengono se già hanno la stessa). L'effetto della riflessione è pertanto di scambiare i due predicati caratteristici dei linguaggi L_1 e L_2 (si ricordi però che i due alfabeti sono diversi). Dunque si ha:

$$L_2^R = \left\{ x \in \{ a, b, c \}^* \mid \underbrace{\text{il secondo carattere di } x \text{ è } a}_{\text{predicato di } L_1} \right\}$$

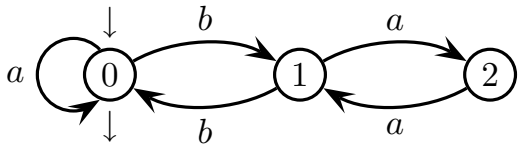
Così la differenza tra L_2^R e L_1 si riduce alla sola diversità di alfabeto. Dato che l'alfabeto di L_2 contiene strettamente quello di L_1, ne segue che L_1 è contenuto strettamente nell'immagine riflessa di L_2, cioè si ha $L_1 \subsetneq L_2^R$; pertanto si conclude che unire L_1 e L_2^R dà ancora L_2^R, cioè si ha $L_1 \cup L_2^R = L_2^R$. L'automa deterministico minimo cercato è allora il seguente:



come si verifica facilmente in modo intuitivo. Ciò risponde alla domanda.

2. È dato l'automa A (deterministico) seguente:

Automa A:



Si risponda alle domande seguenti:

(a) Si trasformi l'automa A in uno equivalente "normalizzato" A_N, che deve avere le proprietà seguenti:

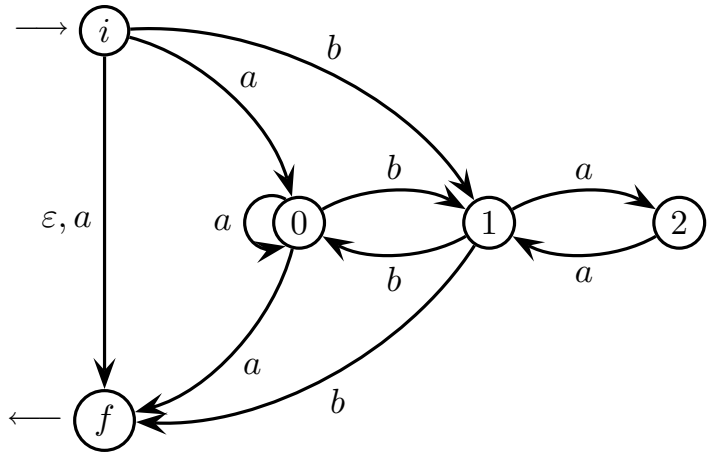
- ha un solo stato iniziale i
- ha un solo stato finale f
- lo stato iniziale differisce da quello finale, cioè $i \neq f$
- nessun arco del grafo entra nello stato i
- nessun arco del grafo esce dallo stato f

(b) Si dica, motivando la risposta, se l'automa A_N sia deterministico e, nel caso non lo sia, se sia ambiguo (cioè se contenga due o più percorsi diversi che riconoscono la medesima stringa).

Soluzione

(a) Per costruire l'automa "normalizzato" A_N si aggiungono i nuovi stati iniziale e finale a quelli già dati, e li si collega in modo da preservare il linguaggio:

Automa A_N:



L'automa A_N ha due stati in più rispetto ad A. Ciò risponde alla domanda.

(b) L'automa A originale è deterministico e pertanto è certamente non ambiguo. Evidentemente però l'automa A_N è indeterministico, negli stati i, 0 e 1, e di conseguenza potrebbe essere ambiguo. Un automa si dice ambiguo se una frase è riconosciuta con due calcoli (cioè percorsi) diversi. Si esamina dunque se possa esistere una frase accettata da A_N con due calcoli diversi. Ci si può limitare a considerare i calcoli che toccano gli stati indeterministici i, 0 e 1.

Evidentemente la stringa vuota ε è riconosciuta solo lungo il percorso (che si riduce a un solo arco) $i \to f$, pertanto non dà luogo ad ambiguità.

Se un calcolo parte dallo stato i e legge il carattere a, si danno solo due possibilità, mutuamente esclusive. Eccole:

- o il calcolo termina in f riconoscendo la stringa a e nessun'altra
- o il calcolo va nello stato 0, non finale, e pertanto può riconoscere una stringa sicuramente più lunga di a, dunque diversa da a

Se un calcolo etichettato con una stringa y raggiunge lo stato 0 e lì legge il carattere a, può proseguire nei due modi seguenti: andare nello stato f o restare nello stato 0. Ecco le conseguenze che ne derivano:

- nel primo caso il calcolo riconosce la stringa $y\,a$ e nessun'altra
- nel secondo il calcolo ha la possibilità di proseguire ancora e così potrà riconoscere una stringa $y\,a\,\ldots$ certamente più lunga di $y\,a$

Pertanto i due calcoli non possono riconoscere la stessa stringa. Lo stesso ragionamento vale per i calcoli che toccano lo stato 1 (con la lettera b).

In conclusione l'automa A_N non è ambiguo, e ciò risponde alla domanda. In effetti si può dimostrare in generale come la costruzione qui illustrata che rende unici gli stati iniziale e finale, benché di per sé sia indeterministica, preservi però sempre la non ambiguità dell'automa.

2 Grammatiche libere e automi a pila 20%

1. Dati i due linguaggi seguenti:

$$L_1 = \{ a^m b^n \mid 1 \leq m \leq n \}$$

$$L_2 = \{ b^m a^n \mid 1 \leq m \leq n \}$$

si prenda il linguaggio seguente (concatenamento di L_1 e L_2):

$$L_3 = L_1 L_2$$

Si risponda alle domande seguenti:

(a) Si elenchino le frasi del linguaggio L_3 in ordine di lunghezza crescente, fino alla lunghezza 5 compresa.

(b) Si scriva una grammatica non ambigua in forma non estesa per il linguaggio L_3, e si disegni l'albero sintattico di una a scelta delle stringhe più lunghe tra quelle elencate al punto precedente.

Soluzione

(a) Ecco le frasi di L_3 richieste, di lunghezza ≤ 5 (se ne mostra anche la scomposizione in fattori di L_1 e L_2):

$$\underbrace{a\,b}_{L_1}\,\underbrace{b\,a}_{L_2} \qquad \underbrace{a\,b\,b}_{L_1}\,\underbrace{b\,a}_{L_2} \qquad \underbrace{a\,b}_{L_1}\,\underbrace{b\,a\,a}_{L_2}$$

Non ce ne sono altre di lunghezza ≤ 5. Si noti che si scompongono in fattori di L_1 e L_2 in modo unico, dunque non ambiguo.

(b) Si mostra dapprima che la grammatica di L_3 ottenuta nel modo più diretto, ovvero concatenando gli assiomi S_1 e S_2 delle grammatiche che generano i linguaggi componenti L_1 e L_2, soffre di ambiguità.

Grammatica di L_1	Grammatica di L_2
$S_1 \to a\,S_1\,b$	$S_2 \to b\,S_2\,a$
$S_1 \to a\,b\,B$	$S_2 \to b\,a\,A$
$B \to b\,B$	$A \to a\,A$
$B \to \varepsilon$	$A \to \varepsilon$

Grammatica di L_3 (assioma (S_3)
$S_3 \to S_1\,S_2$

Una frase ambigua, di lunghezza 7, è la seguente (qui è analizzata in due modi differenti):

$$\overbrace{a\,b}^{S_1}\overbrace{b\,b\,a\,a\,a}^{S_2} \qquad\qquad \overbrace{a\,b\,b}^{S_1}\overbrace{b\,a\,a\,a}^{S_2}$$

Osservando meglio i due linguaggi dati, l'ambiguità è facilmente prevedibile: si tratta di un caso classico di ambiguità indotta dal concatenamento di due linguaggi tali che un prefisso del secondo può essere interpretato anche come suffisso al primo. Nell'esempio la seconda lettera b (da sinistra) può passare dal prefisso di L_2 al suffisso di L_1 e viceversa.

Per togliere l'ambiguità, basta imporre una delle due scelte seguenti: che le stringhe funzionanti sia come prefisso sia come suffisso, le quali sono tutte del tipo b^*, siano da intendere come suffisso del primo linguaggio, non come prefisso del secondo; oppure come prefisso del secondo, non come suffisso del primo; qui si opta per la prima scelta. Si osservi che:

$$L_1 = \{\ a^m\,b^n\ \mid\quad 1 \le m \le n\ \} = \{\ a^m\,b^m\ \mid\quad m \ge 1\ \} \cdot b^*$$

I due linguaggi da concatenare diventano allora i seguenti:

$$K_1 \;=\; L_1 \cdot b^* = \{\ a^m\,b^m\ \mid\quad m \ge 1\ \} \cdot b^* \cdot b^* = L_1$$

$$K_2 \;=\; \{\ b\,a^n\ \mid\quad n \ge 1\ \} = b\,a^+$$

e risulta facilmente:

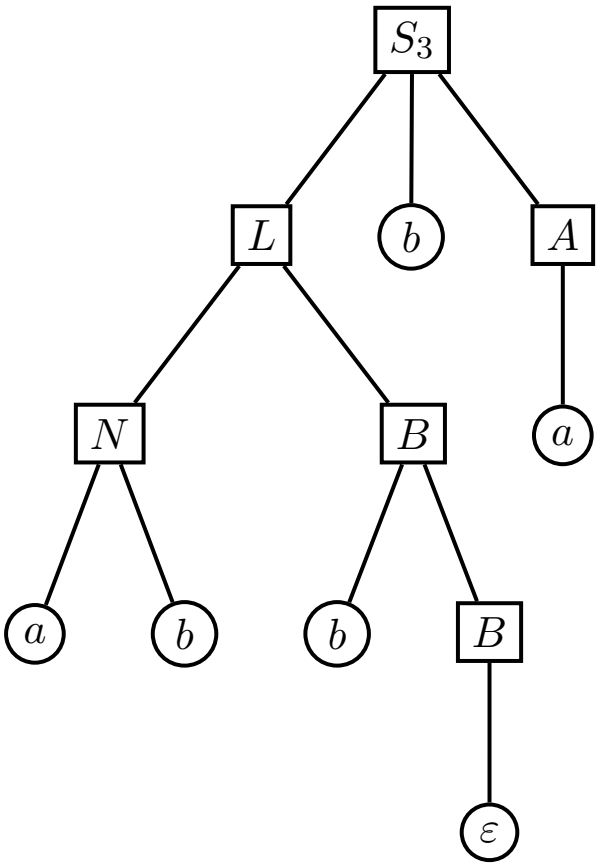
$$L_3 = L_1\,b\,a^+$$

La grammatica G_3 non ambigua di L_3 è allora di scrittura quasi immediata. Eccola (assioma S_3):

$$G_3 \begin{cases} S_3 \;\to\; L\,b\,A \\ L \;\to\; N\,B \\ N \;\to\; a\,N\,b \\ N \;\to\; a\,b \\ B \;\to\; b\,B \\ B \;\to\; \varepsilon \\ A \;\to\; a\,A \\ A \;\to\; a \end{cases}$$

I nonterminali N, B e A generano i linguaggi $a^m\,b^m$ $(m \ge 1)$, b^* e a^+, rispettivamente. Il nonterminale L genera il linguaggio L_1, espandendosi in $N\,B$ e dunque in $a^m\,b^m\,b^* = L_1$ $(m \ge 1)$. Complessivamente S_3 genera il linguaggio L_3 in modo non ambiguo. Naturalmente tale soluzione non è unica.

Ecco l'albero sintattico della stringa (di lunghezza 5) $a\,b\,b\,b\,a$:



L'albero rende piuttosto evidente che nella grammatica G_3 la generazione della sequenza di lettere b interne alla stringa non soffre di ambiguità, giacchè le b che uguagliano in numero le a iniziali sono generate tutte e solo dal nonterminale N mediante una regola autoinclusiva del tipo $N \to a\,N\,b \mid a\,b$, e quelle eventualmente eccedenti sono generate tutte e solo dal nonterminale B (tranne l'ultima che è fissa ed è generata direttamente da S_3) mediante una regola lineare a destra del tipo $B \to b\,B \mid \varepsilon$, dunque puramente regolare.

2. Si vuole progettare una grammatica non ambigua che genera espressioni logiche con le variabili, gli operatori binari di prodotto logico `and`, di somma logica `or` e di implicazione "⇒", con l'operatore unario `not` e con le parentesi "(" e ")". La grammatica deve rispettare i requisiti seguenti:

   - Le variabili sono schematizzate con il terminale v.
   - Le precedenze tra operatori sono le seguenti: `not` precede `and`, che precede `or`, che precede "⇒".
   - L'operatore "⇒" è considerato non associativo, ossia se vi sono due o più operatori "⇒" di séguito, l'ordine di applicazione va precisato con le parentesi.
   - L'operatore `or` è considerato associativo a sinistra.
   - L'operatore `and` è considerato associativo a destra.

   Si risponda alle domande seguenti:

   (a) Si scriva la grammatica, non ambigua e in forma non estesa (non EBNF).

   (b) Si disegni l'albero sintattico di una stringa a scelta contenente tutti gli operatori.

Soluzione

(a) Ecco la grammatica, presentata modularmente (assioma ⟨ESPRESSIONE⟩):

⟨ESPRESSIONE⟩	→	'(' ⟨IMPLICAZIONE⟩ ')' '⇒' ⟨ESPRESSIONE⟩
⟨ESPRESSIONE⟩	→	⟨ESPRESSIONE⟩ '⇒' '(' ⟨IMPLICAZIONE⟩ ')'
⟨ESPRESSIONE⟩	→	⟨IMPLICAZIONE⟩
⟨IMPLICAZIONE⟩	→	⟨SOMMA_DI_PROD⟩ '⇒' ⟨SOMMA_DI_PROD⟩
⟨IMPLICAZIONE⟩	→	⟨SOMMA_DI_PROD⟩
⟨SOMMA_DI_PROD⟩	→	⟨SOMMA_DI_PROD⟩ '`or`' ⟨TERMINE⟩
⟨SOMMA_DI_PROD⟩	→	⟨TERMINE⟩
⟨TERMINE⟩	→	⟨FATTORE⟩ '`and`' ⟨TERMINE⟩
⟨TERMINE⟩	→	⟨FATTORE⟩
⟨FATTORE⟩	→	'`not`' '(' ⟨ESPRESSIONE⟩ ')'
⟨FATTORE⟩	→	'(' ⟨ESPRESSIONE⟩ ')'
⟨FATTORE⟩	→	'`not`' v
⟨FATTORE⟩	→	v

Le regole che espandono la classe sintattica ESPRESSIONE generano catene di due o più operatori di implicazione "⇒", precisando d'obbligo con le parentesi se l'ordine di calcolo parta da sinistra o da destra. Infatti è vietato scrivere p. es. $v \Rightarrow v \Rightarrow v$, perché non si comprenderebbe quale sia l'ordine di calcolo; si

deve invece scrivere o $(v \Rightarrow v) \Rightarrow v$ o $v \Rightarrow (v \Rightarrow v)$. Naturalmente scrivere $v \Rightarrow v$ è consentito, perché se si ha un solo operatore l'associatività non entra in gioco.

Le regole che espandono la classe sintattica IMPLICAZIONE generano l'implicazione elementare, con un solo operatore di implicazione "⇒". In questo caso le parentesi non sono d'obbligo, benché comunque siano consentite.

Le regole che espandono la classe sintattica SOMMA_DI_PROD (somma di prodotti) generano la somma logica di termini. La ricorsione figura a sinistra per modellare l'associatività a sinistra dell'operatore or.

Le regole che espandono la classe sintattica TERMINE (prodotto di fattori) generano il prodotto logico di fattori. La ricorsione figura a destra per modellare l'associatività a destra dell'operatore and.

Le regole che espandono la classe sintattica FATTORE generano il prodotto di sottoespressioni parantetizzate e di variabili, eventualmente negate con not.

La stratificazione delle regole (IMPLICAZIONE si espande in SOMMA_DI_PROD, che si espande in TERMINE, che si espande in FATTORE) garantisce l'ordine di precedenza corretto tra operatori, come richiesto dalla specifica.

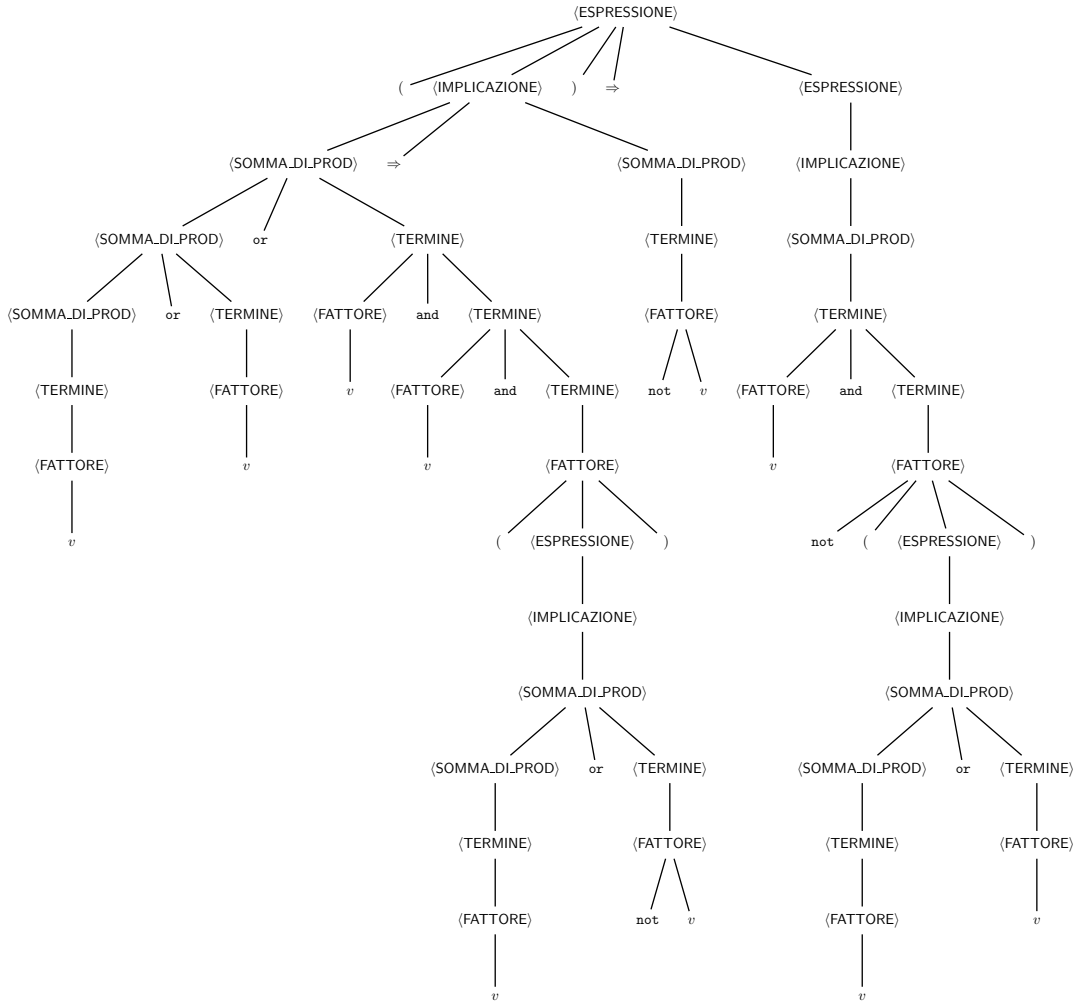
L'intera grammatica è modellata su quella standard in forma non estesa delle espressioni aritmetiche, che è non ambigua. Pertanto anche la grammatica data qui è non ambigua. Ciò risponde alla domanda.

(b) Una stringa, tra le tante, che sollecita gli aspetti principali della grammatica data prima è la seguente:

$$(v \text{ or } v \text{ or } v \text{ and } v \text{ and } (\ v \text{ or not } v\ )\ \Rightarrow\ \text{not } v\ )\ \Rightarrow\ v \text{ and not } (v \text{ or } v\ )$$

Essa mette in gioco l'associatività di `or` e `and`, e contiene una catena di due implicazioni che impone l'uso di parentesi per specificare l'ordine di calcolo.

Ecco l'albero sintattico dell'espressione indicata prima:



Osservando i sottoalberi delle classi sintattiche SOMMA_DI_PROD e TERMINE si possono rilevare le associatività sinistra e destra di `or` e `and`, rispettivamente. L'espressione di esempio contiene una catena di due implicazioni, del tipo $\cdots \Rightarrow \cdots \Rightarrow \ldots$. Alla radice dell'albero si può osservare come l'implicazione a destra abbia la premessa, che costituisce l'implicazione a sinistra, racchiusa tra parentesi, dunque da valutare a valle di questa. Pertanto l'albero precisa un ordine di associazione del tipo $(\cdots \Rightarrow \ldots) \Rightarrow \ldots$. L'implicazione a sinistra è elementare (premessa e conseguenza non contengono altre implicazioni) e dunque non abbisogna di parentesi (benché si possano specificare facoltativamente).

3 Analisi sintattica e parsificatori 20%

1. Per il linguaggio seguente:

$$L = \{\ a^n\, b\, c^n \ \mid \quad n \geq 0\ \} \ \cup\ \{\ a^n\, c^n\, d \ \mid \quad n \geq 0\ \}$$

si deve progettare una grammatica adatta all'analisi sintattica deterministica. La scelta tra il metodo $LL(.)$, $LR(.)$ o $LALR(.)$ è libera.

Si risponda alle domande seguenti:

(a) Si scriva la grammatica richiesta.

(b) Si verifichi, riportando i calcoli necessari, che la grammatica soddisfi la condizione di determinismo scelta.

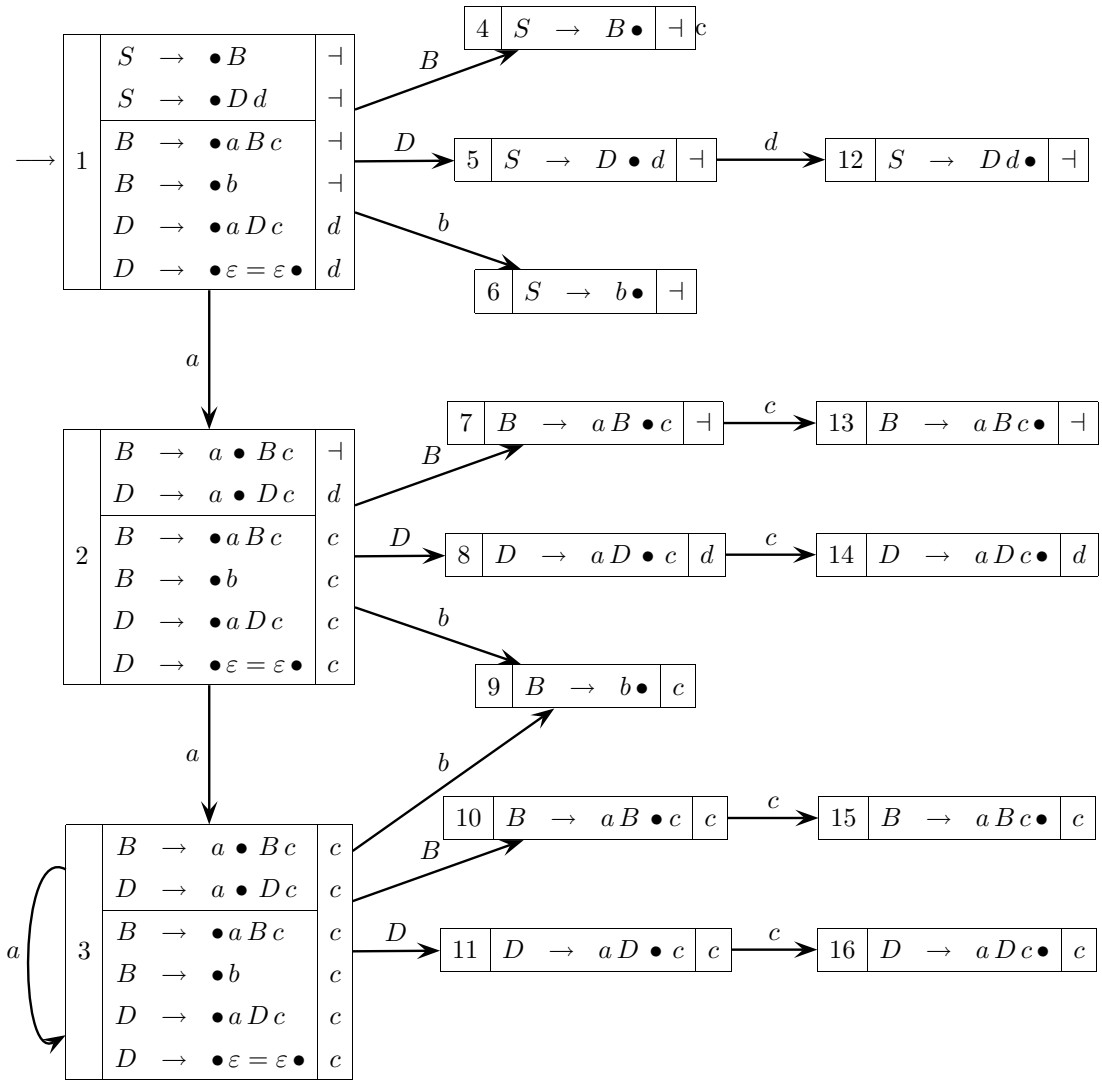
Soluzione

(a) Si sceglie di dare una grammatica G che sia $LR(1)$. Eccola (assioma S):

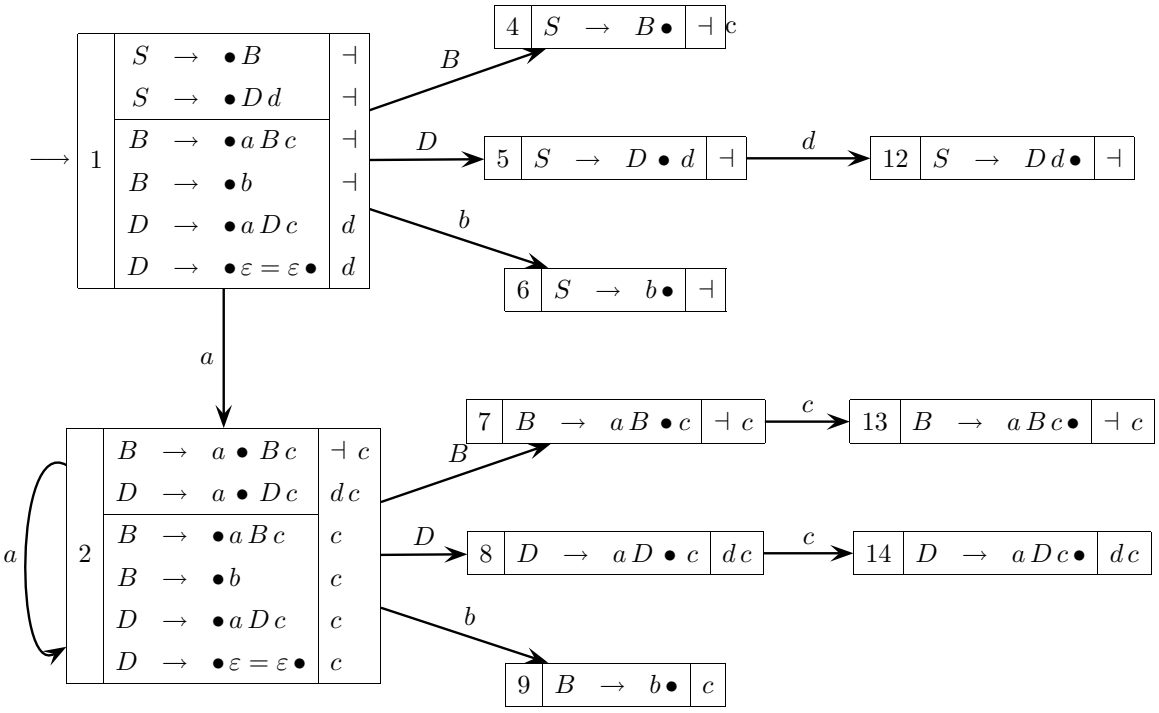
$$G \begin{cases} S \ \to\ B & \mathcal{G}'_1 = \{a, b\} & \mathcal{G}'_2 = \{aa, ab, b\,\dashv\} & \dots \\ S \ \to\ D\,d & \mathcal{G}''_1 = \{a, d\} & \mathcal{G}''_2 = \{aa, ac, d\,\dashv\} & \dots \\ B \ \to\ a\,B\,c & \mathcal{G}'_1 = \{a\} & & \\ B \ \to\ b & \mathcal{G}''_1 = \{b\} & & \\ D \ \to\ a\,D\,c & \mathcal{G}'_1 = \{a\} & & \\ D \ \to\ \varepsilon & \mathcal{G}''_1 = \{c\} & & \end{cases}$$

Intuitivamente la grammatica G genera, mediante la prima e seconda regola assiomatica, il primo o secondo componente del linguaggio L, rispettivamente.
Tuttavia la grammatica G non è $LL(k)$, per nessun $k \geq 1$: le due regole alternative che espandono l'assioma S hanno insiemi guida $\mathcal{G}'_k$ e $\mathcal{G}''_k$ (di ordine $k \geq 1$) contenenti entrambi la stringa a^k, per qualunque $k \geq 1$; essi pertanto non sono disgiunti. Per esempio per tali due regole si hanno $\mathcal{G}'_1 = \{a, b\}$ e $\mathcal{G}''_1 = \{a, d\}$, $\mathcal{G}'_2 = \{aa, ab, b\,\dashv\}$ e $\mathcal{G}''_2 = \{aa, ac, d\,\dashv\}$, ecc. Le altre regole non danno problemi.
Non sembra possibile trovare una grammatica che sia $LL(k)$ per un qualche $k \geq 1$. Infatti l'analizzatore sintattico a discesa ricorsiva dovrebbe leggere un numero arbitrario di lettere a prima di capire se la struttura autoinclusiva $a^n\, c^n$ contenga una lettera b oppure sia seguìta da una lettera d, e d'altro canto tali due scelte sono mutuamente esclusive e fanno capo a regole necessariamente diverse, dunque non unificabili.
La grammatica G risulta però facilmente $LR(1)$ (ma non $LR(0)$ a motivo della regola nulla $D \to \varepsilon$). Intuitivamente, prima si impilano le lettere a e poi le si spilano per contare le lettere c; se le lettere a e c sono separate dalla lettera b, la si legge e si entra in uno stato che non prevede la lettera d in fondo; altrimenti si entra in uno stato che prevede di trovare d in fondo. Un comportamento siffatto sembra ben rientrare tra le capacità di un analizzatore sintattico deterministico $LR(1)$, che può usare gli stati con una certa libertà.

(b) Comunque la verifica formale va fatta tracciando il grafo pilota $LR(1)$ di G. Eccolo (si considera implicitamente presente la regola $S_0 \to S \dashv$):



Si vede subito che non ci sono stati conflittuali, dunque la grammatica G è effettivamente $LR(1)$. È anche facile constatare che G è $LALR(1)$: ripiegando il grafo pilota continuano a non esserci conflitti (tuttavia si ricordi da prima che G non è $LR(0)$). Ecco il grafo $LALR(0)$:



Come si vede, non ci sono conflitti. Topologicamente il grafo $LALR(0)$ coincide con quello $LR(0)$, ma se si togliessero gli insiemi di prospezione (così riducendosi al puro grafo $LR(0)$) gli stati 1 e 2 conterrebbero conflitti di tipo riduzione-spostamento (dovuti essenzialmente alla regola nulla $D \to \varepsilon$).

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

$$G \begin{cases} S & \rightarrow & a\,S\,B\,b\,B \\ S & \rightarrow & c \\ B & \rightarrow & b\,B \\ B & \rightarrow & \varepsilon \end{cases}$$

Si risponda alle domande seguenti:

(a) Mediante l'algoritmo di Earley si effettui la simulazione del riconoscimento della stringa seguente:

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

Si utilizzi la tabella all'uopo predisposta nella prossima pagina.

(b) Si disegnino tutti gli alberi sintattici della stringa data al punto precedente, mostrando la corrispondenza tra nodi e candidate di riduzione nella tabella di simulazione prima costruita.

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

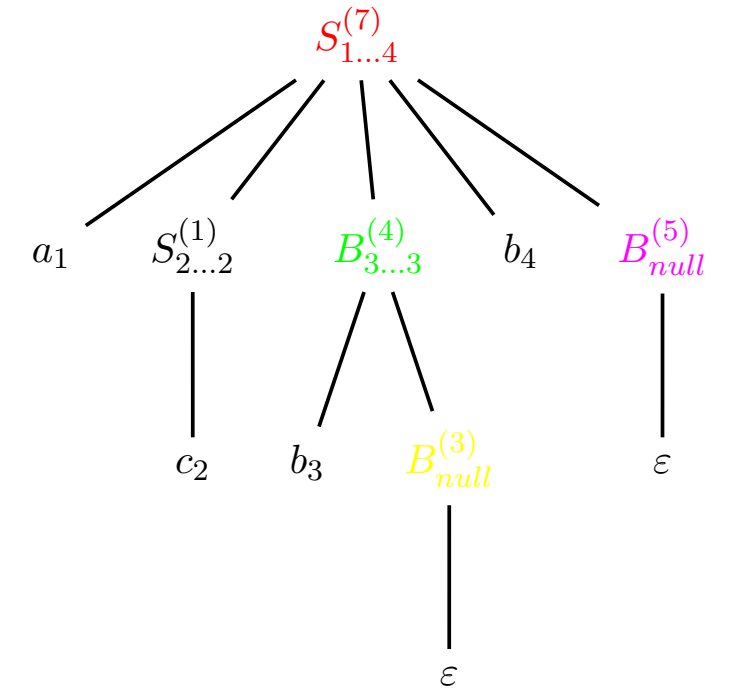
Soluzione

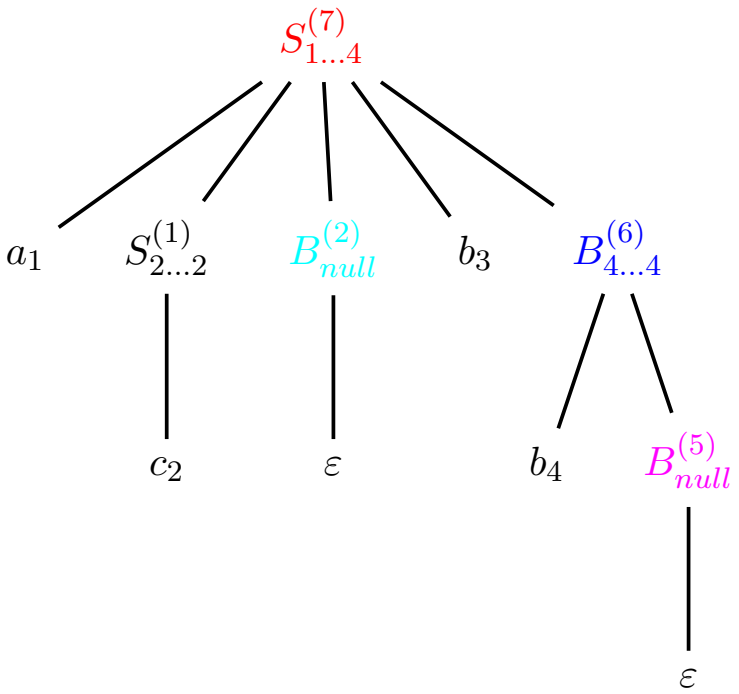
(a) Ecco la simulazione di riconoscimento della stringa $a\,c\,b\,b$:

Schema di simulazione dell'algoritmo di Earley									
stato 0	pos. a	stato 1	pos. c	stato 2	pos. b	stato 3	pos. b	stato 4	pos. $\dashv$
$S \rightarrow \bullet\, a\, S\, B\, b\, B$	0	$S \rightarrow a \bullet S\, B\, b\, B$	0	$\mathbf{S} \rightarrow c \bullet^{(1)}$	1	$B \rightarrow b \bullet B$	2	$B \rightarrow b \bullet B$	3
$S \rightarrow \bullet\, c$	0	$S \rightarrow \bullet\, a\, S\, B\, b\, B$	1	$S \rightarrow a\, S \bullet B\, b\, B$	0	$S \rightarrow a\, S\, B\, b \bullet B$	0	$S \rightarrow a\, S\, B\, b \bullet B$	0
		$S \rightarrow \bullet\, c$	1	$B \rightarrow \bullet\, b\, B$	2	$B \rightarrow \bullet\, b\, B$	3	$B \rightarrow \bullet\, b\, B$	4
				$\mathbf{B} \rightarrow \bullet\, \varepsilon = \varepsilon \bullet^{(2)}$	2	$\mathbf{B} \rightarrow \bullet\, \varepsilon = \varepsilon \bullet^{(3)}$	3	$\mathbf{B} \rightarrow \bullet\, \varepsilon = \varepsilon \bullet^{(5)}$	4
				$S \rightarrow a\, S\, B \bullet b\, B$	0	$\mathbf{B} \rightarrow b\, B \bullet^{(4)}$	2	$\mathbf{B} \rightarrow b\, B \bullet^{(6)}$	3
						$S \rightarrow a\, S\, B\, b\, B \bullet$	0	$\mathbf{S} \rightarrow a\, S\, B\, b\, B \bullet^{(7)}$	0
						$S \rightarrow a\, S\, B \bullet b\, B$	0	$B \rightarrow b\, B \bullet$	2
								$S \rightarrow a\, S\, B \bullet b\, B$	0

Nello stato 4 è presente una candidata di riduzione assiomatica (numero 7 colorata in rosso), con stato di partenza 0.
Tale candidata dunque dà luogo a riconoscimento della stringa $a\, c\, b\, b$.
Colori e numerazione di alcune candidate di riduzione fanno riferimento ai nodi degli alberi sintattici (vedi punto seguente).

(b) La grammatica è palesemente ambigua e la stringa di esempio è generabile in due modi diversi. Ci sono pertanto due alberi sintattici. Eccoli:





La corrispondenza tra nodi e candidate di riduzione presenti nella tabella di simulazione è piuttosto evidente, e comunque è messa in evidenza colorando e numerando i nodi e le candidate di riduzione cui essi fanno riferimento.

4 Traduzione e analisi semantica 20%

1. È dato il linguaggio L seguente, di alfabeto $\{a, b, c, d\}$, unione di due componenti:

$$L = \underbrace{\{\, a^n\, b^n\, c \mid \quad n \geq 0 \,\}}_{\text{componente 1}} \cup \underbrace{\{\, a^n\, b^n\, d \mid \quad n \geq 1 \,\}}_{\text{componente 2}}$$

Si consideri la traduzione τ seguente:

$$\underbrace{\tau\,(\, a^n\, b^n\, c\,) = a^{2n} b^n}_{\text{comportamento 1}} \quad n \geq 0 \qquad \underbrace{\tau\,(\, a^n\, b^n\, d\,) = (a\, b)^n}_{\text{comportamento 2}} \quad n \geq 1$$

Si risponda alle domande seguenti:

(a) Si progetti un trasdutture a pila (eventualmente indeterministico), con riconoscimento a stato finale, che realizzi la traduzione τ indicata sopra.

(b) Si progetti uno schema (o grammatica) di traduzione che realizzi la traduzione τ indicata sopra.

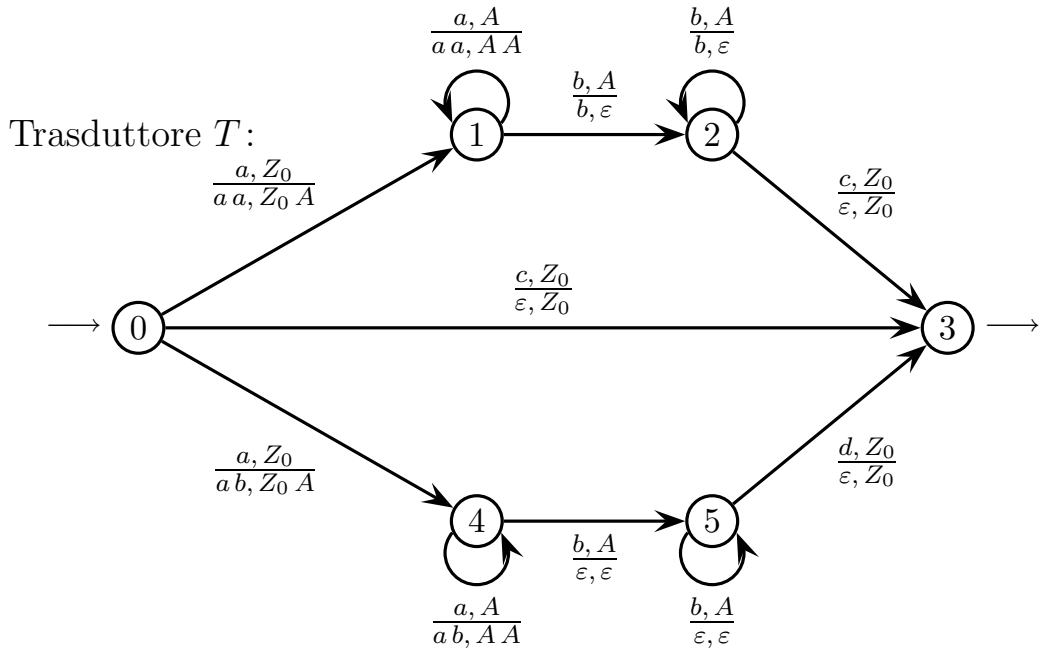
Soluzione

(a) Benché il puro riconoscimento del linguaggio sorgente si possa fare con un automa a pila deterministico (a stato finale), la trasduzione τ è inerentemente indeterministica e va fatta con un automa trasduttore a pila (a stato finale) indeterministico. L'automa trasduttore impila le lettere a in ingresso (codificate in modo opportuno) e agisce in modo diverso secondo i due comportamenti traduttivi (1 o 2) possibili:

Comportamento 1 mentre impila le a, emette le a in uscita raddoppiandole; poi spila le a per contare le lettere b in ingresso ed emettere le b in uscita

Comportamento 2 mentre impila le a, emette le coppie $a\, b$ in uscita; poi spila le a per contare le lettere b in ingresso

All'inizio va fatta una scelta indeterministica su quale tra i due comportamenti traduttivi adottare; la scelta iniziale viene convalidata alla fine, quando si leggono le lettere in ingresso c o d. Nel caso 1 la traduzione può essere la stringa vuota, nel caso 2 no. Ecco il grafo stato-transizione di tale trasduttore:



L'aspetto indeterministico è concentrato solo nello stato 0, il resto del grafo è interamente deterministico. Non sembra del resto possibile eliminare tale pur limitata forma di indeterminismo, giacché il comportamento in uscita è diverso nei due casi e non si vede come si potrebbe unificarli. Il riconoscimento è a stato finale, benché succeda comunque che nello stato finale 3 la pila si svuoti. Ecco la simulazione di due calcoli rappresentativi, tra i tanti possibili:

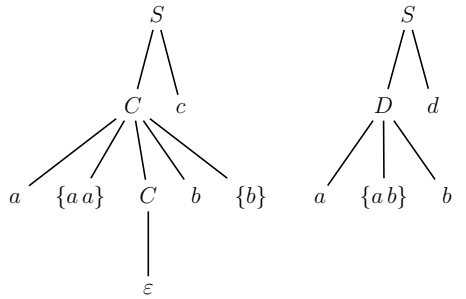
$$\left\langle \tfrac{abc}{\varepsilon},\, 0,\, Z_0 \right\rangle \overset{\frac{a,\,Z_0}{a\,a,\,Z_0\,A}}{\Longrightarrow} \left\langle \tfrac{bc}{aa},\, 1,\, Z_0A \right\rangle \overset{\frac{b,\,A}{b,\,\varepsilon}}{\Longrightarrow} \left\langle \tfrac{c}{aab},\, 2,\, Z_0 \right\rangle \overset{\frac{c,\,Z_0}{\varepsilon,\,Z_0}}{\Longrightarrow} \left\langle \tfrac{\varepsilon}{aab},\, 3,\, Z_0 \right\rangle$$

$$\left\langle \tfrac{abd}{\varepsilon},\, 0,\, Z_0 \right\rangle \overset{\frac{a,\,Z_0}{a\,b,\,Z_0\,A}}{\Longrightarrow} \left\langle \tfrac{bd}{ab},\, 4,\, Z_0A \right\rangle \overset{\frac{b,\,A}{\varepsilon,\,\varepsilon}}{\Longrightarrow} \left\langle \tfrac{d}{ab},\, 5,\, Z_0 \right\rangle \overset{\frac{d,\,Z_0}{\varepsilon,\,Z_0}}{\Longrightarrow} \left\langle \tfrac{\varepsilon}{ab},\, 3,\, Z_0 \right\rangle$$

Funziona anche con calcoli che mettano in gioco gli autoanelli. Queste due simulazioni giustificano a sufficienza la correttezza dell'automa progettato.

(b) Lo schema di traduzione che realizza τ non presenta particolari difficoltà. Eccolo, in forma combinata (assioma S):

$$\begin{aligned}
S &\rightarrow C\ c \\
S &\rightarrow D\ d \\
C &\rightarrow a\ \{\ a\ a\ \}\ C\ b\ \{\ b\ \} \\
C &\rightarrow \varepsilon \\
D &\rightarrow a\ \{\ a\ b\ \}\ D\ b \\
D &\rightarrow a\ \{\ a\ b\ \}\ b
\end{aligned}$$



Esso corrisponde in modo evidente all'automa trasduttore T (a lato sono mostrati gli alberi sintattici delle traduzioni $\tau(abc) = aab$ e $\tau(abd) = ab$ simulate sopra). È facile osservare che tale schema non è $LL(k)$, per nessun $k \geq 1$, a motivo delle due regole alternative che espandono S (i rispettivi insiemi guida di ordine k hanno in comune la stringa a^k). Per quanto detto prima, non sembra possibile trovare uno schema deterministico per la trasduzione τ.

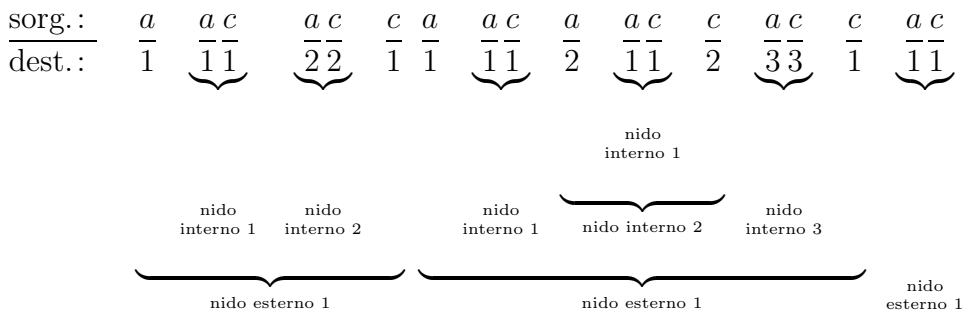
2. Si consideri il linguaggio di Dyck di alfabeto $\{ a, c \}$. Data una frase del linguaggio, si devono numerare progressivamente i nidi di "parentesi", cominciando da 1. Quando si scende di un livello in profondità di annidamento, la numerazione riparte da 1. I nidi esterni (non contenuti in nessun altro nido) sono tutti numerati con 1.

   Il linguaggio sorgente è definito dalla sintassi seguente (assioma S):

$$G \begin{cases} S \rightarrow N \\ N \rightarrow A\,N\,C\,N \\ N \rightarrow \varepsilon \\ A \rightarrow a \\ C \rightarrow c \end{cases}$$

   che non può essere modificata.

   Ecco un esempio di numerazione (i nidi di parentesi sono messi in evidenza):



   Il numero n da emettere in corrispondenza di ogni lettera in ingresso, va calcolato mediante una grammatica con attributi. Esso deve essere un attributo destro (cioè ereditato) dei nonterminali A e C.

   Si risponda alle domande seguenti:

   (a) Si elenchino gli attributi impiegati nel progetto, e si scrivano le funzioni semantiche necessarie per il calcolo dell'attributo n (in entrambi i casi si compilino gli schemi all'uopo predisposti alle pagine successive).

   (b) Si esamini la traduzione dal linguaggio di Dyck alla stringa dei numeri n, per esempio dalla stringa sorgente $a\,a\,c\,a\,c\,c\,a\,a\,c\,a\,a\,c\,c\,a\,c\,c\,a\,c$ alla stringa destinazione 1 1 1 2 2 1 1 1 1 2 1 1 2 3 3 1 1 1 (è l'esempio dato sopra), verificando se tale funzione di traduzione risulti invertibile.

---

attributi da usare per la grammatica

tipo	nome	(non)terminali	dominio	significato
già dati nel testo dell'esercizio				
dx	n	A, C	num. int.	numero d'ordine del nido
eventualmente da modificare e / o aggiungerne di nuovi				

sintassi	*funzioni semantiche*
$S_0 \rightarrow N_1$	
$N_0 \rightarrow A_1\, N_2\, C_3\, N_4$	
$N_0 \rightarrow \varepsilon$	
$A_0 \rightarrow a$	
$C_0 \rightarrow c$	

Soluzione

(a) L'idea è di avere un attributo numerico n per enumerare i nidi di parentesi e un attributo booleano ext per distinguere tra nidi esterni, tutti quanti numerati con 1, e interni (a livello di profondità 2, 3, ecc), numerati progressivamente da 1 in su a partire dal primo (quello più a sinistra). Ecco gli attributi:

attributi da usare per la grammatica

tipo	nome	(non)terminali	dominio	significato
già dati nel testo dell'esercizio				
dx	n	A, C	num. int.	numero d'ordine del nido
eventualmente da modificare e / o aggiungerne di nuovi				
dx	n	N	num. int.	numero d'ordine del nido
dx	ext	N	booleano	se vero indica nido esterno

L'attributo n, già dato per i nonterminali A e C, va esteso associandolo anche al nonterminale N. L'attributo ext va aggiunto, associandolo a N. Gli attributi sono tutti ereditati, infatti qui non servono attributi sintetizzati.

Ed ecco le funzioni semantiche da associare alle regole:

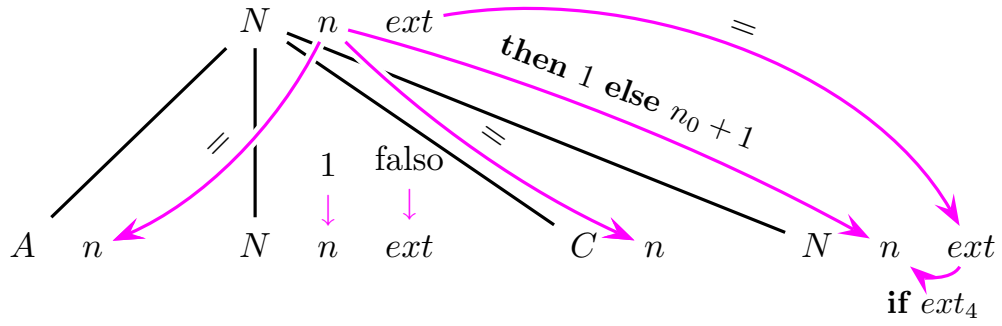
sintassi	*funzioni semantiche*	
$S_0 \rightarrow N_1$	$n_1 = 1$	$ext_1 = \text{vero}$
$N_0 \rightarrow A_1\, N_2\, C_3\, N_4$	$n_1 = n_0$ $n_2 = 1$ $n_3 = n_0$ $n_4 = \textbf{if } ext_4 \textbf{ then } 1 \textbf{ else } n_0 + 1$	$ext_2 = \text{falso}$ $ext_4 = ext_0$
$N_0 \rightarrow \varepsilon$	niente	
$A_0 \rightarrow a$	**stampa** (n_0)	
$C_0 \rightarrow c$	**stampa** (n_0)	

Le funzioni semantiche operano sugli attributi nel modo seguente:

- L'attributo ext è vero solo per i nidi esterni di parentesi, mentre per quelli interni, a qualunque profondità, è sempre falso.
- L'attributo n viene inizializzato a 1 per i nidi esterni, mentre per quelli interni parte da 1 e viene incrementato progressivamente.
- La funzione di stampa emette la traduzione a livello dei nonterminali A e C (o se si preferisce dei terminali a e c che espandono A e C).

Si potrebbe calcolare la traduzione a livello della radice S, concatenando le traduzioni parziali in una stringa da fare risalire a S come attributo sintetizzato.
È facile verificare che la grammatica è a una sola passata (one-sweep). Inoltre, dato che il flusso di calcolo degli attributi ereditati procede tutto dall'alto verso il basso e che il supporto sintattico risulta essere $LL(1)$ (altro non è che la solita grammatica di Dyck), la grammatica risulta anche essere di tipo L.

Come verifica, si dà in aggiunta il grafo delle dipendenze della produzione $N \to A\,N\,C\,N$ (la più importante e complicata della grammatica). Eccolo:



Il flusso di calcolo procede solo dal padre ai figli, non esiste eredità tra nodi fratelli e meno che mai risalita da figlio a padre (non ci sono attributi sintetizzati). Gli archi di dipendenza sono etichettati con le funzioni semantiche rispettive.

(b) Basta osservare che esistono due stringhe sorgente aventi la stessa traduzione:

$$\tau(a\,a\,c\,c) = 1\,1\,1\,1 \qquad\qquad \tau(a\,c\,a\,c) = 1\,1\,1\,1$$

per concludere che la trasduzione τ non è invertibile. L'esempio si generalizza facilmente a stringhe di qualunque lunghezza. Ciò risponde alla domanda.

Volendo esaminare la trasduzione τ in modo completo, si cominci con l'osservare che essa ha come dominio il linguaggio di Dyck e come immagine un insieme di stringhe di numeri interi. Per ogni nonterminale A e C viene emesso un numero ben preciso, dunque ogni stringa di Dyck causa l'emissione di una sola stringa numerica immagine e pertanto la trasduzione τ è una funzione (non una semplice relazione binaria). Tuttavia si tenga presente che non ogni stringa numerica è immagine valida di una stringa di Dyck. Si può modellare tale funzione così:

$$\tau\colon \text{Ling. di Dyck su } \{a, c\} \longrightarrow \text{Stringhe numeriche valide}$$

Valgono poi le considerazioni seguenti:

- non sempre stringhe di Dyck diverse si caratterizzano per numerazioni differenti, per esempio (comprende il caso dato sopra):

$$\tau\left(\left(a^h\,c^h\right)^k\right) = \left(1^h\,1^h\right)^k = 1^{2hk} \qquad h, k \geq 1$$

$$\tau\left(\left(a^k\,c^k\right)^h\right) = \left(1^k\,1^k\right)^h = 1^{2hk} \qquad h, k \geq 1$$

  pertanto la funzione non è iniettiva
- avendo ristretto l'immagine alle sole stringhe di uscita che sono numerazioni valide di stringhe di Dyck, la funzione è surgettiva

In conclusione, non essendo iniettiva, la trasduzione (o funzione) τ non è biunivoca (one-to-one)[1] e dunque non è invertibile.

[1] Una funzione è biunivoca se e solo se è sia iniettiva sia surgettiva.