Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 07.02.2006 - Parte I: Teoria

CON SOLUZIONI (MOLTO ESTESE)

NOME & COGNOME:

MATRICOLA: FIRMA:

ISTRUZIONI - LEGGERE CON ATTENZIONE:

- L'esame si compone di due parti:
  - I (80%) Teoria:
    1. espressioni regolari e automi finiti
    2. grammatiche e automi a pila
    3. analisi sintattica e parsificatori
    4. traduzione e analisi semantica
  - II (20%) Esercitazioni Flex e Bison
- Per superare l'esame l'allievo deve avere sostenuto con successo entrambe le parti (I e II), in un solo appello oppure in appelli diversi della stessa sessione d'esame.
- Per superare la parte I (teoria) occorre dimostrare di possedere sufficiente conoscenza di tutte le quattro sezioni (1-4).
- È permesso consultare libri e appunti personali.
- Per scrivere si utilizzi lo spazio libero e se occorre anche il tergo del foglio; in fondo a ogni sezione (1-4) c'è un foglio bianco addizionale.
- Tempo: Parte I (teoria): 2h.30m - Parte II (esercitazioni): 30m

1 Espressioni regolari e automi finiti 20%

1. È data l'espressione regolare R seguente:

$$R = a\,(ab^* \mid b^*a^+c)^*$$

Si svolgano i punti seguenti:

(a) Si stabilisca se l'espressione regolare R sia ambigua o no, motivando in breve la risposta.

(b) Si progetti l'automa deterministico (a scelta se in forma minima o no) che riconosca il linguaggio $L(R)$.

(c) (facoltativo) Si progetti un automa indeterministico che riconosca $L(R)$, cercando di limitarne al massimo il numero di stati.

Soluzione

(a) L'espressione regolare R è ambigua. Ecco l'esempio che lo dimostra:

$$R = a\,\left(\underbrace{a\,b^*}_{\alpha} \mid \underbrace{b^*\,a^+\,c}_{\beta}\right)^*$$

e si ha:

$$a\,a\,b\,b\,a\,c = a\,\underbrace{a\,b}_{\alpha}\,\underbrace{b\,a\,c}_{\beta} = a\,\underbrace{a\,b\,b}_{\alpha}\,\underbrace{a\,c}_{\beta}$$

Dunque il concatenamento $\alpha\beta$ dei fattori α e β è ambiguo. Si potrebbe procedere anche numerando i generatori di R, come segue:

$$R = a_1\,\left(a_2\,b_3^* \mid b_4^*\,a_5^+\,c_6\right)^*$$

e osservando che:

$$a\,a\,b\,b\,a\,c = a_1\,a_2\,b_3\,b_4\,a_5\,c_6 = a_1\,a_2\,b_3\,b_3\,a_5\,c_6$$

arrivando così alle stesse conclusioni di prima, benché ora il tipo di ambiguità sia meno visibile e identificabile. Il lettore cerchi altre forme di ambiguità in R, se ve ne sono, oppure ne argomenti l'inesistenza.

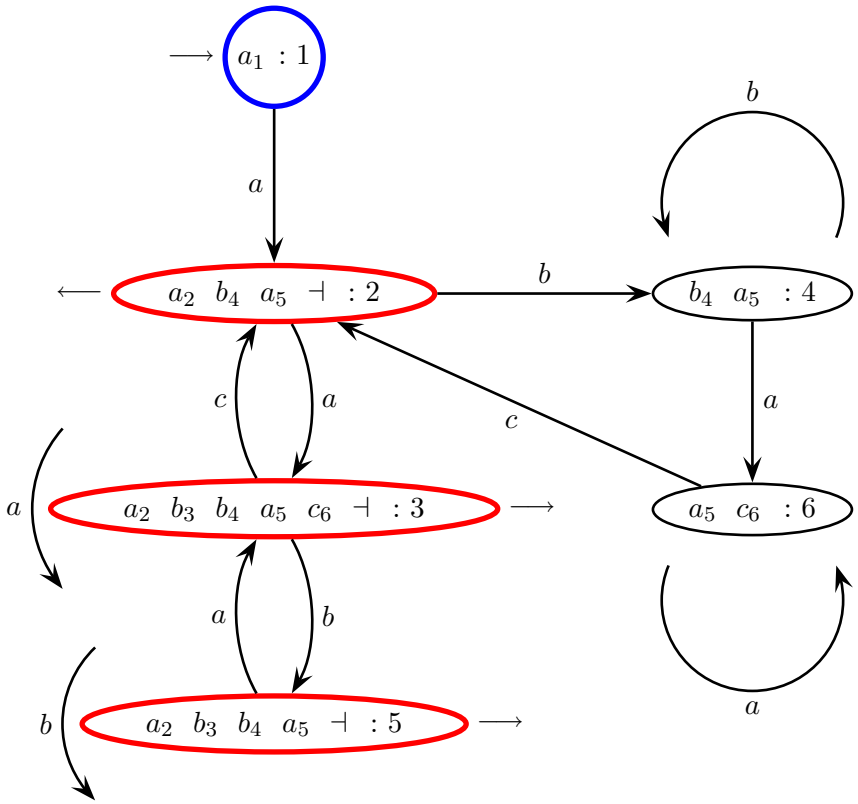
(b) Sembra opportuno applicare il metodo di Berry e Sethi (l'espressione non è di quelle semplicissime sì da potere procedere facilmente in modo intuitivo). Ecco la numerazione dei generatori dell'espressione regolare R:

$$R = a_1 \ \left(a_2 \ b_3^* \ \mid \ b_4^* \ a_5^+ \ c_6\right)^* \ \dashv$$

Poi ecco gli insiemi dei séguiti di R:

gen.	séguiti
a_1	$a_2 \ \ b_4 \ \ a_5 \ \ \dashv$
a_2	$a_2 \ \ b_3 \ \ b_4 \ \ a_5 \ \ \dashv$
b_3	$a_2 \ \ b_3 \ \ b_4 \ \ a_5 \ \ \dashv$
b_4	$b_4 \ \ a_5$
a_5	$a_5 \ \ c_6$
c_6	$a_2 \ \ b_4 \ \ a_5 \ \ \dashv$

E infine ecco l'automa deterministico equivalente a R (con 6 stati):



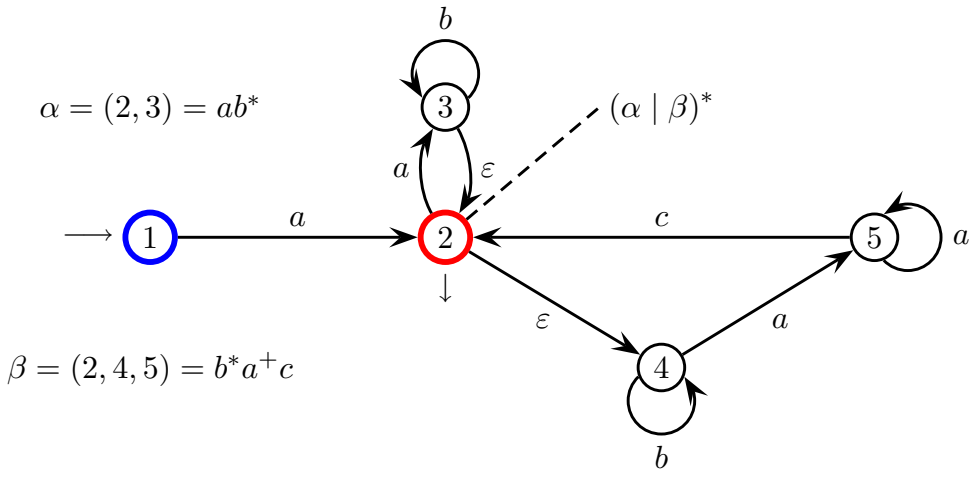
Si osservi che l'automa è in forma pulita: stati tutti raggiungibili e definiti.

Inoltre, ecco la tabella degli stati dell'automa (dedotta dal grafo degli stati):

stato	a	b	c	*finale?*
1	2	–	–	*no*
2	3	4	–	*si*
3	3	5	2	*si*
4	6	4	–	*si*
5	3	5	–	*no*
6	6	–	1	*no*

È facile vedere come l'automa deterministico sia già minimo. Si potrebbe dimostrarlo formalmente, ma qui basta un breve ragionamento (peraltro non richiesto dall'esercizio). Comunque eccolo: presa una coppia qualunque di stati, si trova sempre un ingresso tale che se uno dei due stati va in errore l'altro non lo fà, e ciò significa che i due stati non sono equivalenti giacché l'automa è in forma pulita (se non lo fosse, uno stato indefinito potrebbe equivalere a quello di errore, che è indefinito per definizione); fanno eccezione le coppie $(2,4)$, $(2,5)$ e $(4,5)$: ma l'equivalenza $2 \sim 4$ implica l'eq. $3 \sim 6$, che non vale, dunque $2 \not\sim 4$; e inoltre $2 \not\sim 5$ e $4 \not\sim 5$ perché 2 e 4 sono finali mentre 5 non lo è. Ciò chiude la questione. Si ricordi che il metodo di Berry e Sethi, in generale, non produce automi (deterministici) in forma minima.

(c) Ed ecco un automa indeterministico equivalente a R, che usa solo 5 stati, invece di 6 come l'automa deterministico (minimo). Si lascia al lettore la verifica formale di equivalenza (che si può fare mettendo l'automa prima in forma deterministica, poi in forma minima e infine confrontandolo con quello di prima), oppure la si giustifichi provando con qualche esempio significativo di stringa.



È evidente come i due cicli $(2,3)$ e $(2,4,5)$ corrispondano alle due sottoespressioni ab^* e b^*a^+c di R unite sotto stella, rispettivamente (sono i termini α e β già

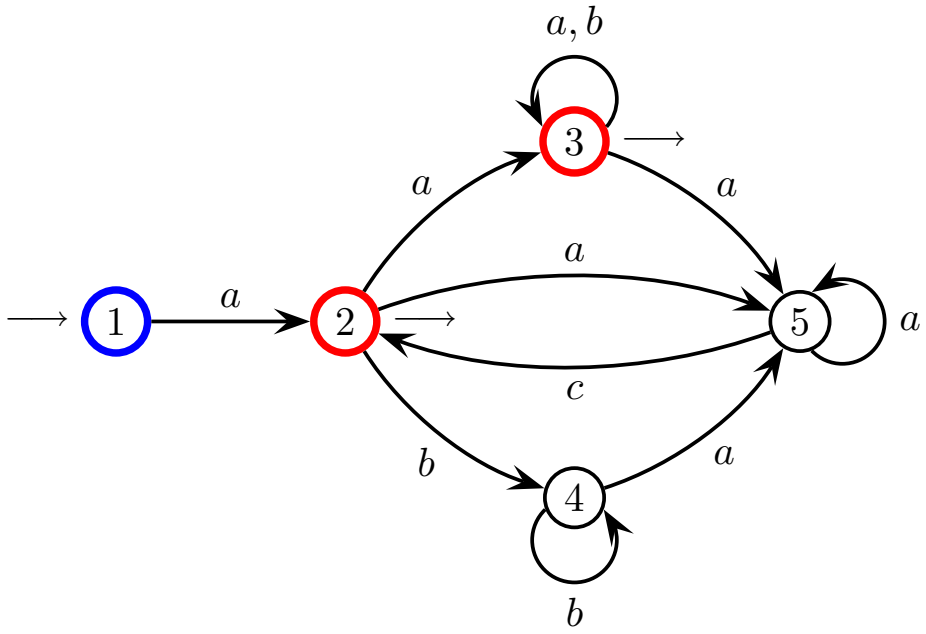
considerati nel punto (a)). In effetti si vede subito che l'espressione regolare R produce stringhe aventi la struttura generica seguente:

$$a\ (\alpha\ |\ \beta)^* = a\,\alpha \ldots \alpha\,\beta \ldots \beta\,\alpha \ldots$$

cioè successioni arbitrarie di fattori α e β, anche vuote, con prefisso a obbligatorio; ne risulta l'automa indeterministico visto sopra.

Ricavare questo automa è dunque pressoché immediato, giacché in sostanza si applica la costruzione modulare di Thomson per passare da espressione regolare ad automa non-deterministico, con qualche scorciatoia del tutto intuitiva per risparmiare subito sul numero di stati, sì da averne meno di 6.

Se si preferisce una versione senza transizioni spontanee (ε-transizioni), eccola (questa è stata ottenuta per via puramente intuitiva):

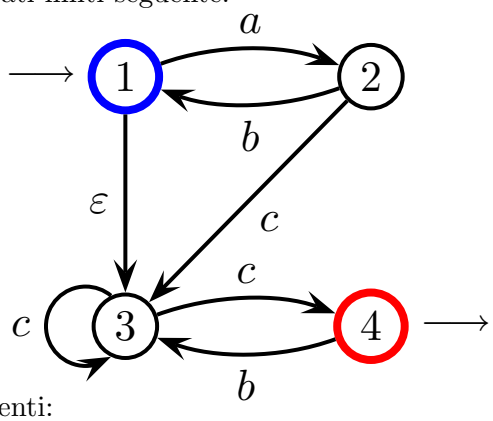


Una versione senza transizioni spontanee si otterrebbe comunque eliminandole dalla prima versione (ma non è detto venga proprio questa). Infatti, tagliando le ε-transizioni il numero di stati resta invariato, o addirittura cala quando si può applicare la regola di collasso degli ε-cicli.

Beninteso ci potrebbero essere altre soluzioni, anche migliori quanto a numero di stati. Si badi bene che, in forma indeterministica, l'automa mimimo non è unico, in generale (benché in qualche caso possa valere comunque l'unicità).

Un'aggiunta culturale: esistono algoritmi per trovare le forme non-deterministiche minime, quanto a numero di stati, di un automa, ma sono complessi; ovviamente si ha sempre che: (# di stati in forma min. non-det.) $\leq$ (# di stati in forma min. det.); inoltre si dimostra (curiosamente) che se ci sono due o più forme min. non-det. dello stesso automa, tra esse ce n'è una (e una sola) che è massima quanto a numero di archi, sì che tutte le altre sono ottenute togliendo qualche arco a questa; detto altrimenti, si ha unicità anche in caso di non-det., ma imponendo simultaneamente il min per il numero di stati e il max per il numero di archi ...

2. È dato l'automa M a stati finiti seguente:



Si svolgano i punti seguenti:

(a) Si ricavi l'automa deterministico (non necessariamente in forma minima) equivalente all'automa M.

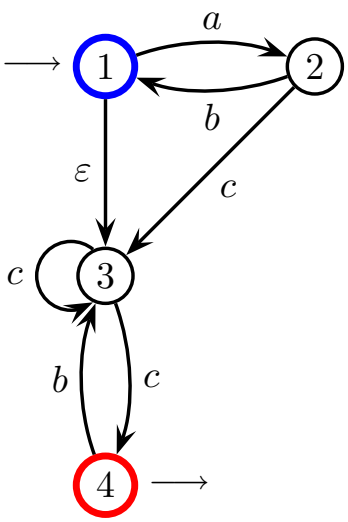
(b) Si minimizzi il numero di stati dell'automa deterministico (o si giustifichi se sia già in forma minima).

(c) (facoltativo) Si ricavi un'espressione regolare (ambigua o meno) equivalente all'automa M.

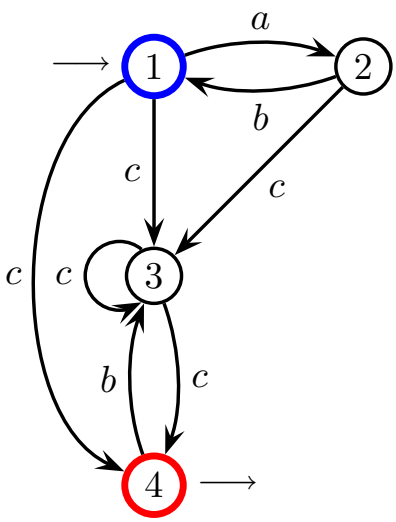
Soluzione

(a) Si osservi subito che l'automa è già in forma pulita (tutti gli stati sono raggiungibili e definiti). Si procede all'eliminazione della transizione spontanea $1 \xrightarrow{\varepsilon} 3$ mediante la regola di retrazione degli archi (corrispondente a quella di eliminazione delle produzioni di copia, come per esempio $A \to B$):

prima



dopo

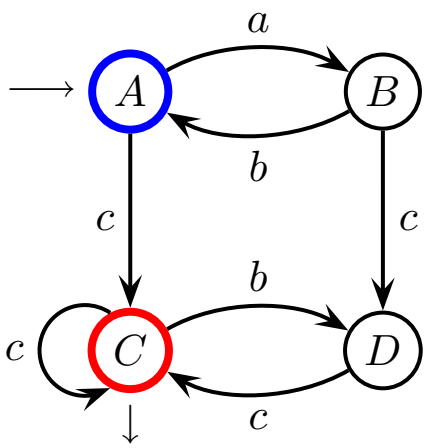


Beninteso, si potrebbe anche applicare la regola di taglio con propagazione in avanti, qui plausibilmente la complessità del procedimento è circa la stessa.

Si può ora procedere alla costruzione dei sottoinsiemi ($*$ indica gli stati finali), secondo il metodo della tabella dei successori:

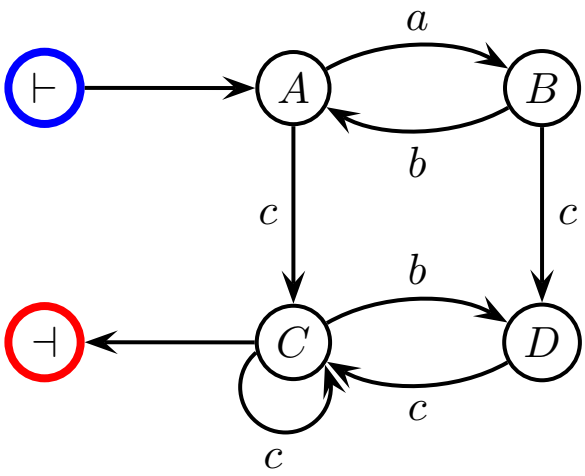
stati	a	b	c	*rinomina*
1	2	–	34	A
2	–	1	3	B
34*	–	3	34	C^*
3	–	–	34	D

(b) Rinominando gli stati si ottiene l'automa det. in forma minima:

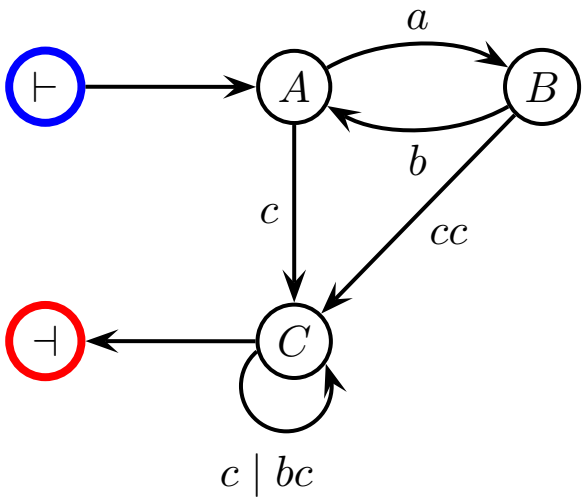


L'automa è in forma pulita: tutti gli stati sono raggiungibili da A (stato iniziale) e raggiungono C (stato finale), cioè sono definiti; ciò del resto è garantito dal metodo dei successori.
È facile constatare come l'automa deterministico sia già in forma minima: la sola equivalenza possibile sarebbe $B = 2 \sim 34^* = C$ (le altre righe della tabella differiscono tra sé nello stato di errore e dunque non possono essere equivalenti), ma non vale perché C è finale mentre B non lo è. Si ricordi che il metodo dei successori, in generale, non produce automi in forma minima.

(c) Per ricavare l'espressione regolare equivalente a M, si sceglie di eliminare i nodi (Brozosowsky) dall'automa deterministico costruito prima[1]:

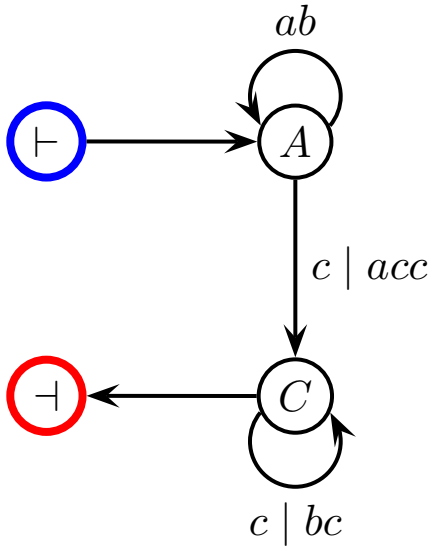


Elimina il nodo D:

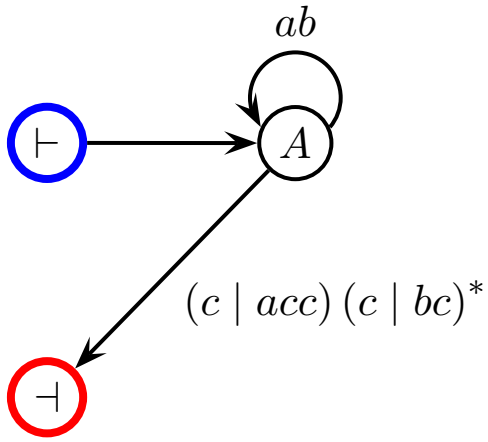


[1]Il metodo delle equazioni linguistiche lineari andrebbe altrattanto bene, probabilmente qui avrebbe lo stesso livello di difficoltà; non è detto però dìa la stessa soluzione.

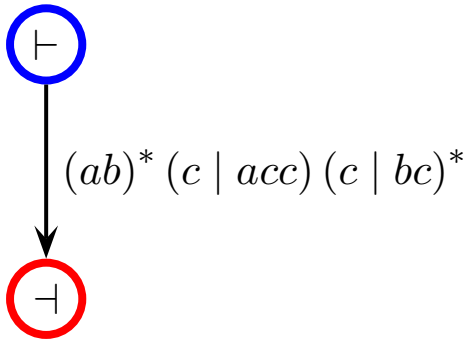
Elimina il nodo B:



Elimina il nodo C:



Elimina il nodo A:



Pertanto si conclude:

$$R = (ab)^* (c \mid acc) (c \mid bc)^*$$

Ovviamente l'espressione regolare R non è ambigua, giacché deriva da un automa deterministico. Volendo perfezionare (non è richiesto), e osservando come valga $(c \mid bc)^* = c^*(bc^+)^*$, l'espressione si compatta intuitivamente come segue[2]:

$$\begin{aligned} R &= (ab)^* (c \mid acc) (c \mid bc)^* = \\ &= (ab)^* (\varepsilon \mid ac) cc^* \left(bc^+\right)^* = \\ &= (ab)^* (\varepsilon \mid ac) c^+ \left(bc^+\right)^* \end{aligned}$$

Si badi bene che, trasformandola, l'espressione potrebbe diventare ambigua; qui resta non ambigua, come si vede facilmente.

[2]Naturalmente vi sono molte altre trasformazioni possibili, anzi infinite.

2 Grammatiche libere e automi a pila 20%

1. Si consideri il linguaggio libero L seguente, di alfabeto $\Sigma = \{a, b\}$:

$$L = \left\{a^{2k}b^{2k} \mid \ k \geq 0\right\} \cup \left\{a^{3k}b^{3k} \mid \ k \geq 0\right\}$$

Esempi: ε $aabb$ $aaabbb$

Controesempi: $aabbb$ $aaabb$

Si svolgano i punti seguenti:

(a) Si progetti una grammatica libera G che generi il linguaggio L, accertandosi non sia ambigua.

(b) Si disegni l'albero sintattico della frase: a^6b^6

Soluzione

(a) Le grammatiche (non ambigue) dei linguaggi $a^{2k}b^{2k}$ e $a^{3k}b^{3k}$ sono pressoché standard. Eccole (X e Y assiomi):

$$X \to aaXbb \mid \varepsilon \qquad\qquad Y \to aaaYbbb \mid \varepsilon$$

Però se le due sottogrammatiche vengono unite con $S \to X \mid Y$, si ha ambiguità di unione per le stringhe a^6b^6, $a^{12}b^{12}$, ecc, cioè quando gli esponenti contengono come fattore sia 2 sia 3, ovvero sono multipli di 6.
Si rimedia facilmente, impedendo alla sottogrammatica che genera gli esponenti di tipo $3k$ di terminare quando $k = 0, 2, 4, 6, \ldots$ (k pari). Ecco come:

$$\begin{array}{rcl} S & \to & X \ \mid \ Y_1 \\ X & \to & a\ a\ X\ b\ b \ \mid \ \varepsilon \\ Y_1 & \to & a\ a\ a\ Y_2\ b\ b\ b \ \mid \ a\ a\ a\ b\ b\ b \\ Y_2 & \to & a\ a\ a\ Y_1\ b\ b\ b \end{array}$$

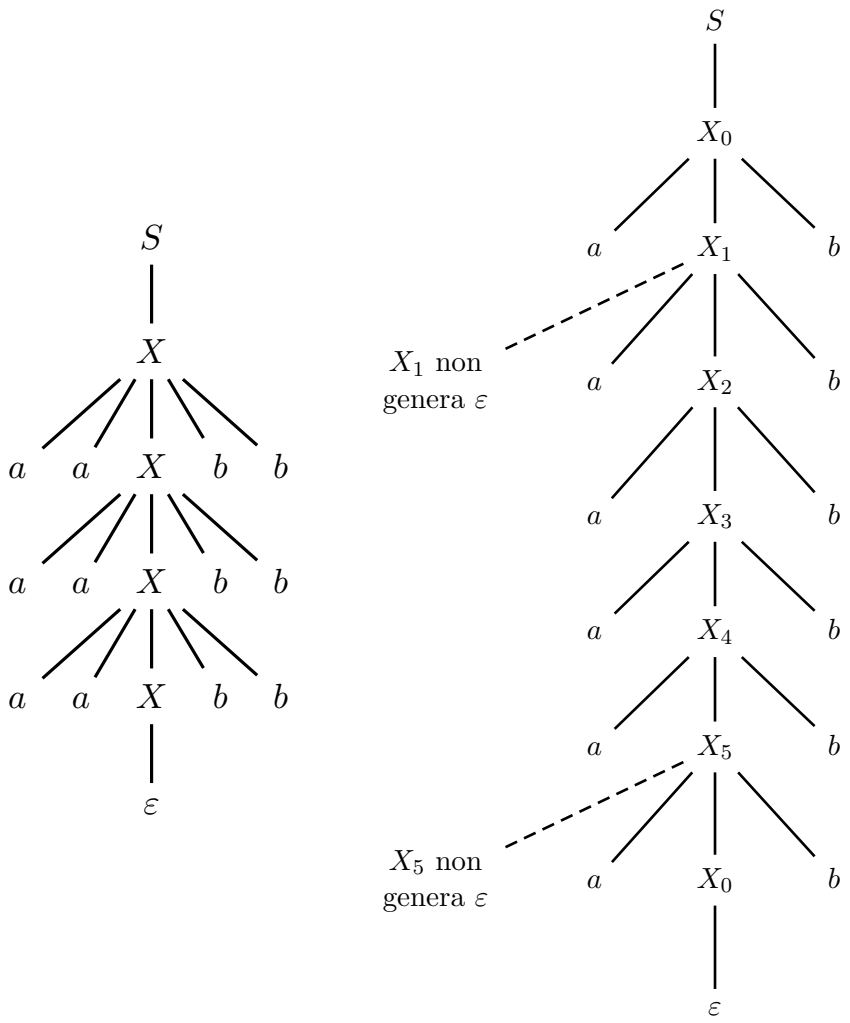
La grammatica riflette la descrizione datane sopra, e dovrebbe essere evidente come funzioni[3]. Ci sono naturalmente altre soluzioni del tutto naturali: impedire alla sottogrammatica $2k$ di terminare quando k contiene 3 come fattore (ma si userebbero più non-terminali), oppure usare produzioni del tipo $X_i \to a\ X_{(i+1 \mod 6)}\ b$ (con $0 \leq i \leq 5$) permettendo solo a X_0, X_2, X_3 e X_4 di terminare con $X_i \to \varepsilon$ ($i = 0, 2, 3, 4$), cioè (si usano parecchi non-terminali):

$$\begin{array}{rcll} & & & \text{gui. } k = 1 \\ S & \to & X_0 & a, \dashv \\ X_i & \to & a\ X_{(i+1 \mod 6)}\ b \qquad (\text{con } 0 \leq i \leq 5) & a \\ X_i & \to & \varepsilon \qquad\qquad\qquad\quad (\text{con } i = 0, 2, 3, 4) & b, \dashv_{\text{se } i = 0} \end{array}$$

oppure magari anche soluzioni meno evidenti o più *ad hoc*.

[3]Si può compattare ulteriormente: $S \to X \mid Y$, $X \to a^2Xb^2 \mid \varepsilon$ e $Y \to a^6Yb^6 \mid a^3b^3$; così ha sostanzialmente la stessa logica ma usa ancora meno non-terminali.

(b) Ecco gli alberi sintattici di a^6b^6 per le due grammatiche date prima:



È evidente come riducendo il numero di non-terminali si aumenti il grado (numero di figli) dei nodi, per compensare la minore capacità di conteggio.

Per inciso, osservando l'albero a destra si comprende bene come mai non vengano generati gli esponenti $1, 5, 7, 11, 13, 17, \ldots$ (ovvero gli esponenti e di $a^e b^e$ tali che $e \bmod 6 = 1, 5$), cioè proprio quelli la cui forma non è esprimibile né come $2k$ né come $3k$, ricordando che la produzione $X_i \to \varepsilon$ manca proprio per $i = 1, 5$.

La prima grammatica è non ambigua per costruzione, giacché si usano solo componenti non ambigui uniti in modo non ambiguo. Quanto alla seconda, uno sguardo rapido mostra che è det. $LL(1)$, e dunque anch'essa certamente non ambigua. Con un po' più di sforzo si vedrebbe come la prima, che non è det. $LL(k)$ per nessun k (perché? il lettore la esamini), sia però det. $LR(1)$ (idem, il lettore ne faccia l'esame) e dunque non ambigua anche per tale ragione (oltre che per costruzione).

La prima grammatica riflette una concezione modulare, dove il problema è scomposto in sottoproblemi risolti (quasi) indipendentemente, la seconda una globale, dove esso è risolto complessivamente esaminandone più a fondo le peculiarità strutturali. È una dicotomia che si verifica spesso, sebbene a rigore non sia un evento necessario.

2. Si progetti la grammatica EBNF non ambigua di un frammento di linguaggio C, che modelli la struttura di controllo `switch`. Sono presenti i concetti seguenti:

   - `switch` può avere uno o più rami, eventualmente anche il ramo `default` (l'ultimo della sequenza)
   - la condizione di `switch` è un'espressione algebrica di tipo intero, tra parentesi
   - le etichette dei rami di `switch` sono costanti di tipo intero, racchiuse tra apici, separate tra loro con ',' e dalla parte esecutiva del ramo con ':'
   - i rami di `switch` consistono in liste (o blocchi) di istruzioni, secondo la consueta notazione del linguaggio C, e possono essere chiusi dalla clausola `break`
   - ogni istruzione è una chiamata a procedura, con possibili parametri, o un assegnamento a variabile, o un altro `switch`
   - vi sono variabili e costanti di tipo intero
   - vi sono espressioni algebriche di tipo intero, come segue
     - contenenti variabili e costanti
     - con gli operatori binari infissi $+$, $-$ e $\times$
     - l'operatore $\times$ precede $+$ e $-$
     - non si usano parentesi
   - variabili (e proc.) e costanti sono schematizzate con *id* e *cost*, rispettivamente

   Esempio:

```
switch (a + b - 1) {
    ‘1’ : {
          c = c + 2;
          proc1 (12, b + 1);
     }
     ‘2’, ‘3’ : d = d * e + 22;
     ‘5’ : {
          proc2 ();
          break;
     }
     default: { ... }
}
```

   Si svolgano i punti seguenti:

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

   (b) Si disegni l'albero sintattico della frase data sopra come esempio.

Soluzione

(a) Ecco una grammatica ragionevole e abbastanza semplice (con 13 non-terminali):

$$
\begin{array}{rcll}
S & \rightarrow & \texttt{switch}\ \text{`('}\ E\ \text{`)'}\ \text{`\{'}\ (B^{+}\ [\ D\ ]\ \mid\ D)\ \text{`\}'} & \text{- - } LL(1) \\
B & \rightarrow & L\ \text{`: '}\ X & \\
D & \rightarrow & \texttt{default}\ \text{`: '}\ X & \\
L & \rightarrow & \text{` '}\ cost\ \text{` '}\ (\text{`,'}\ \text{` '}\ cost\ \text{` '})^{*} & \text{- - } LL(1) \\
X & \rightarrow & \text{`\{'}\ Y\ [\texttt{break}\ \text{`;'}]\ \text{`\}'}\ \mid\ I & \text{- - } LL(1) \\
Y & \rightarrow & (\text{`\{'}\ Y\ \text{`\}'}\ \mid\ I)^{*} & \text{- - } LL(1) \\
I & \rightarrow & (P\ \mid\ A)\ \text{`;'}\ \mid\ S & \text{- - } LL(2) \\
P & \rightarrow & id\ \text{`('}\ [\ Q\ ]\ \text{`)'} & \text{- - } LL(1) \\
A & \rightarrow & id\ \text{`='}\ E & \\
Q & \rightarrow & E\ (\text{`,'}\ E)^{*} & \text{- - } LL(1) \\
E & \rightarrow & T\ ((\text{`+'}\ \mid\ \text{`}-\text{'})\ T)^{*} & \text{- - } LL(1) \\
T & \rightarrow & F\ (\text{`}\times\text{'}\ F)^{*} & \text{- - } LL(1) \\
F & \rightarrow & id\ \mid\ cost & \text{- - } LL(1)
\end{array}
$$

dove le classi sintattiche hanno i significati seguenti:

- S struttura `switch` completa (condizione e insieme non vuoto di rami)
- B ramo generico dello `switch`, non vuoto
- D ramo finale dello `switch` (opzionale), non vuoto
- L lista di etichette numeriche, non vuota
- X blocco di istruzioni, con eventuale `break` finale, o istruzione singola
- Y lista di blocchi annidati e istruzioni, anche vuota
- I istruzione elementare (proc., ass.), con terminatore, o `switch` interno
- P chiamata a procedura con parametri attuali (opzionali)
- A assegnamento a variabile (nome di variabile, espressione)
- Q lista di parametri attuali (espressioni), non vuota, con separatore
- E espressione intera (somma algebrica di termini, senza sottoespressioni)
- T termine intero (prodotto di fattori, senza sottoespressioni)
- F fattore intero (variabile o costante)

Non si confonda tra ( ) e [ ] come metasimboli o terminali. La grammatica è non ambigua per costruzione, ma è anche facile constatare come sia det. $LL(2)$ (*id* e *cost* sono supposti terminali), dunque di certo non ambigua. Nei dettagli non espliciti nel testo si è cercato di seguire la sintassi C (come suggerito dal testo stesso): sono ammessi blocchi e istruzioni vuote; il ramo di `switch` può consistere anche di una sola istruzione elementare; per semplicità `break` è ammesso solo alla fine del ramo di `switch` (ma C lo ammetterebbe anche all'interno). Beninteso ci sono altre soluzioni, magari differenti qua e là in dettagli di poco conto (se ammettere blocchi o istruzioni elementari vuoti, se permettere ';' dopo '}', ecc).

(b) Si lascia al lettore, basta anche una versione semplificata.

3 Analisi sintattica e parsificatori 20%

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

produzione	insieme guida
$S \to A\ c\ S$	
$S \to \varepsilon$	
$A \to a\ A\ b$	
$A \to a$	

Si svolgano i punti seguenti:

(a) Si calcolino gli insiemi guida della grammatica G e si verifichi se essa sia $LL(k)$ per qualche k.

(b) Si scriva la procedura sintattica per l'analisi del nonterminale A.

Soluzione

Vedi a pagina seguente.

(a) Ecco il calcolo, molto semplice, degli insiemi guida $LL(k)$, per $k = 1$ e $k = 2$:

produzione	insieme guida	
	$k = 1$	$k = 2$
$S \to A\ c\ S$	a	$aa \quad ac$
$S \to \varepsilon$	$\dashv$	$\dashv\dashv$
$A \to a\ A\ b$	a	aa
$A \to a$	a	$ab \quad ac$

Il calcolo si può fare in modo intuitivo, oppure applicando le regole ricorsive basate su inizî e séguiti. È evidente che la grammatica è $LL(2)$ ma non $LL(1)$.

Ecco il calcolo intuitivo (non si chiede necessariamente di esplicitarlo in forma scritta, ma per rispondere alla domanda va fatto almeno mentalmente). Per $k = 1$: S è nullabile e A non lo è; le guide delle due produzioni alternative di A (cioè $A \to aAb$ e $A \to a$) sono la stringa a, pertanto si ha l'insieme $\{a\}$ per entrambe; la guida di $S \to \varepsilon$ è il séguito di S, pertanto si ha l'insieme $\{\dashv\}$ per definizione[4]; infine la guida di $S \to AcS$ è l'inizio di A, pertanto si ha l'insieme $\{a\}$ (da prima). Per $k = 2$: S è nullabile e A non lo è, ma può generare una stringa di lunghezza 1, segnatamente a; la guida di $A \to aAb$ è la stringa a concatenata con l'inizio, di livello 1, di A, ovvero a (da prima), pertanto si ha l'insieme $\{aa\}$; le guide di $A \to a$ sono la stringa a concatenata con i séguiti, di livello 1, di A, i quali, esaminando le produzioni nel cui membro destro figura A, si vedono essere b e c, pertanto si ha l'insieme $\{ab, ac\}$; la guida di $S \to \varepsilon$ è il séguito, di livello 2, di S, pertanto si ha l'insieme $\{\dashv\dashv\}$ per definizione[4]; infine le guide di $S \to AcS$ sono l'inizio, di livello 2, di A, ovvero aa (da prima), e l'inizio, di livello 1, di A, ovvero a (da prima), concatenato con l'inizio, di livello 1, di cS, ovvero c come si vede subito, dunque giustapponendo ac, pertanto in conclusione si ha l'insieme $\{aa, ac\}$. Qui si chiude l'analisi intuitiva (ma non troppo, il ragionamento è sì informale ma anche rigoroso).

Si può anche procedere scrivendo derivazioni sulla falsariga della definizione stessa di insieme guida, curando però di non trascurarne nessuna significativa, ciò che non sempre è facile perché le derivazioni sono infinite (in generale).

[4]Il non-terminale S non figura in nessuna produzione, tranne che a destra di una assiomatica, dunque senza aggiungere altri séguiti.

Per completezza comunque, ecco il calcolo svolto mediante le regole ricorsive (non è richiesto dall'esercizio). Siano $\mathcal{I}, \mathcal{S}, \mathcal{G} \subseteq \Sigma \cup \dashv$ gli insiemi degli inizî, dei séguiti e guida, rispettivamente. Si osservi che S è nullabile, mentre A non lo è. Per $k = 1$ si ha quanto segue:

$$\begin{aligned}
\mathcal{G}(S \to AcS) &= \mathcal{I}(AcS) = \\
&= \mathcal{I}(A) = \\
&= \mathcal{I}(aAb) \cup \mathcal{I}(a) = \\
&= \mathcal{I}\{a\} \cup \{a\} = \\
&= \{a\} \cup \{a\} = \\
&= \{a\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{G}(S \to \varepsilon) &= \mathcal{S}(S) = \\
&= \{\dashv\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{G}(A \to aAb) &= \mathcal{I}(aAb) = \\
&= \mathcal{I}(a) = \\
&= \{a\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{G}(A \to a) &= \mathcal{I}(a) = \\
&= \{a\}
\end{aligned}$$

Siano $\mathcal{I}_1, \mathcal{S}_1, \mathcal{G}_1 \subseteq \Sigma \cup \dashv$ gli insiemi degli inizî, dei séguiti e guida, rispettivamente, di livello 1, come prima (cioè $\mathcal{I}_1 \equiv \mathcal{I}$, $\mathcal{S}_1 \equiv \mathcal{S}$ e $\mathcal{G}_1 \equiv \mathcal{G}$); siano $\mathcal{I}_2, \mathcal{S}_2, \mathcal{G}_2 \subseteq (\Sigma \cup \dashv)^2$ gli insiemi degli inizî, dei séguiti e guida, rispettivamente, di livello 2. Si osservi che S è nullabile (o, se si preferisce, può generare la stringa di lunghezza 0, vale a dire ε), mentre A non lo è ma può generare una stringa di lunghezza 1, segnatamente la stringa a, oltre a stringhe di lunghezza ≥ 2. Per $k = 2$ si ha quanto segue (il modo di procedere dovrebbe essere evidente a chiunque abbia almeno vagamente intuìto che cosa sia l'insieme guida, se non è così se ne ristudi con attenzione la definizione nel libro di testo):

$$\begin{aligned}
\mathcal{G}_2(S \to AcS) &= \mathcal{I}_2(AcS) = \\
&= \mathcal{I}_2(A) \cup \mathcal{I}_1(A)\mathcal{I}_1(cS) = \\
&= (\mathcal{I}_2(aAb) \cup \mathcal{I}_2(a)) \cup (\mathcal{I}_1(aAb) \cup \mathcal{I}_1(a))\,\mathcal{I}_1(c) = \\
&= \mathcal{I}_1(a)\mathcal{I}_1(Ab) \cup \emptyset \cup (\mathcal{I}_1(a) \cup \{a\})\,\{c\} = \\
&= \{a\}\mathcal{I}_1(a) \cup (\{a\} \cup \{a\})\,\{c\} = \\
&= \{a\}\{a\} \cup \{a\}\{c\} = \\
&= \{aa\} \cup \{ac\} = \\
&= \{aa, ac\}
\end{aligned}$$

$$
\begin{aligned}
\mathcal{G}_2(S \to \varepsilon) &= \mathcal{S}_2(S) = \{\dashv\dashv\} \\
\\
\mathcal{G}_2(A \to aAb) &= \mathcal{I}_2(aAb) = \\
&= \mathcal{I}_2(a) \cup \mathcal{I}_1(a)\mathcal{I}_1(Ab) = \\
&= \emptyset \cup \{a\}\mathcal{I}_1(A) = \\
&= \{a\}\left(\mathcal{I}_1(a) \cup \mathcal{I}_1(aAb)\right) = \\
&= \{a\}\left(\{a\} \cup \mathcal{I}_1(a)\right) = \\
&= \{a\}\left(\{a\} \cup \{a\}\right) = \\
&= \{a\}\{a\} = \\
&= \{aa\} \\
\\
\mathcal{G}_2(A \to a) &= \mathcal{I}_2(a) \cup \mathcal{I}_1(a)\mathcal{S}_1(A) = \\
&= \emptyset \cup \{a\}\left(\mathcal{I}_1(cS) \cup \mathcal{I}_1(b)\right) = \\
&= \{a\}\left(\mathcal{I}_1(c) \cup \{b\}\right) = \\
&= \{a\}\left(\{c\} \cup \{b\}\right) = \\
&= \{a\}\{b, c\} = \\
&= \{ab, ac\}
\end{aligned}
$$

Come scomporre gli insiemi di livello 2 in concatenamenti di insiemi di livello 1 dovrebbe essere evidente, sapendo quali siano le lunghezze delle stringhe generabili da S e A (lunghezza 0, cioè il non-terminale è nullabile, 1 o ≥ 2). Ecco qualche osservazione sparsa, utile per capire il tutto:

- $\mathcal{G}_2(S \to AcS)$ si riduce a $\mathcal{I}_2(AcS)$ perché deve dare le guide di livello esatto 2 e, in base a quanto detto prima a proposito di S e A, la forma sentenziale AcS genera solo stringhe di lunghezza ≥ 2, e pertanto per trovare le guide di $S \to AcS$ basta esaminare gli inizî di AcS.
- $\mathcal{I}_2(AcS)$ si scompone in $\mathcal{I}_2(A) \cup \mathcal{I}_1(A)\mathcal{I}_1(cS)$ perché A può generare stringhe di lunghezza ≥ 2, e dunque inizî di livello 2, ma anche stringhe di lunghezza 1, e dunque inizî di livello 1, e pertanto occorre concatenare a questi ultimi il contributo dell'inizio (di livello 1) di cS.
- $\mathcal{I}_2(A)$ si scompone in $\mathcal{I}_2(aAb) \cup \mathcal{I}_2(a)$ (e similmente per $\mathcal{I}_1$) applicando l'espansione $A \to aAb \mid a$.
- $\mathcal{I}_2(a)$ si riduce a $\emptyset$ perché per definizione deve dare gli inizî di livello esatto 2, non di livello < 2 (si ricordi che $\mathcal{I}$ può essere vuoto, mentre $\mathcal{S}$ e $\mathcal{G}$ non lo sono mai); similmente si avrebbe $\mathcal{I}_1(\varepsilon) = \emptyset$.
- $\mathcal{G}_2(A \to a)$ si scompone in $\mathcal{I}_2(a) \cup \mathcal{I}_1(a)\mathcal{S}_1(A)$ perché deve dare le guide di livello esatto 2, e dunque formalmente si deve prevedere il contributo $\mathcal{I}_2(a)$ (ancorché si dimostri essere vuoto), ma poiché a genera (anzi, coincide con) una stringa consistente in un solo carattere, bisogna aggiungere i contributi ottenibili concatenando ad a i séguiti (di livello 1) di A, cioè bisogna andare a vedere che cosa segua A nei membri destri di produzione dove A figura.

Il resto dovrebbe ora essere chiaro. Il calcolo intuitivo, comunque, è molto più semplice (se si studia) e rapido (se ci si allena), e infinitamente meno noioso.

(b) La procedura sintattica $LL(2)$ per l'analisi del nonterminale A è la seguente:

```
char  cc                          - - carattere corrente
char  cp                          - - carattere di prospezione
procedure  A
      if  (cc == 'a' and (cp == 'b' or cp == 'c'))  then
            cc = cp               - - caso A → a
            read (cp)
      else  if (cc == 'a' and cp == 'a')  then
            cc = cp               - - caso A → a A b
            read (cp)
            call  A
            if  (cc == 'b')  then
                  cc = cp
                  read (cp)
            else
                  errore          - - caso di errore in A → a A b
            end  if
      else
            errore                - - caso di errore in A → a | a A b
      end  if
end  procedure
```

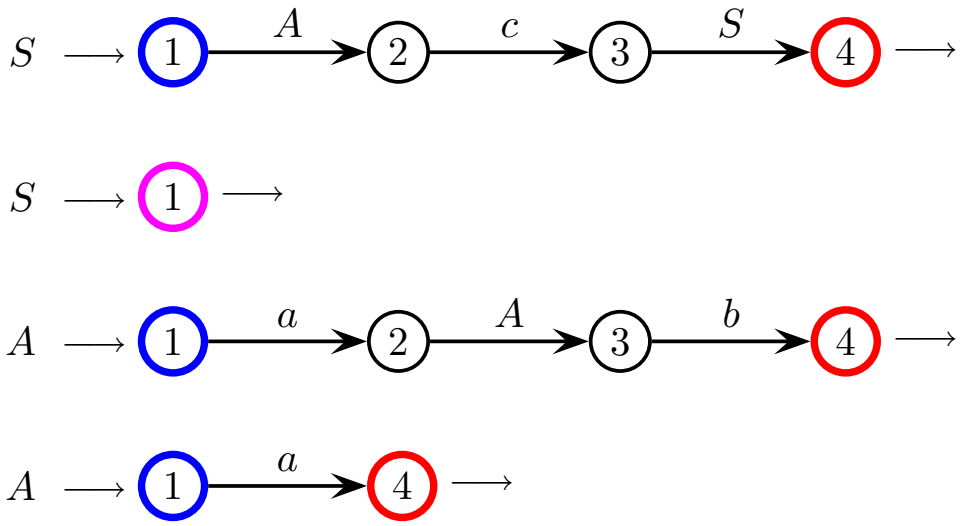
Si tenga presente che cc contiene il carattere corrente, cp quello di prospezione. Essi sono posizionati come segue:

$$a_1 a_2 \ldots \underbrace{a_i}_{cc} \underbrace{a_{i+1}}_{cp} \ldots$$

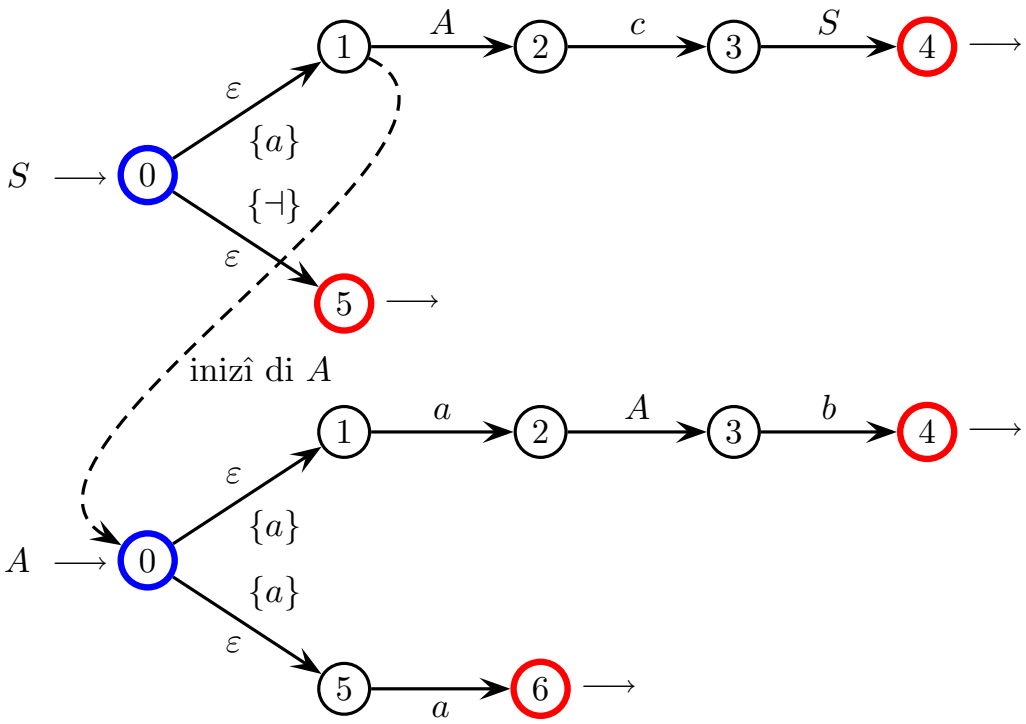
Per leggere e consumare un carattere appartenente a una produzione, occorre spostare cp in cc e leggere cp ex-novo dal nastro in ingresso. Subito prima che la procedura sintattica di una produzione $A \to \alpha$ inizi, si suppone che $cc \in \text{Inizî}(\alpha)$; subito dopo che la procedura sintattica di una produzione $A \to \alpha$ è terminata, si ha che $cc \in \text{Séguiti}(A)$. Ci sono varie altre ottimizzazioni evidenti per qualunque programmatore competente.

Si sarebbe anche potuto procedere prima modellando le produzioni tramite automi e poi applicando il metodo descritto sul sito del corso; è da notare che, in tale modo, la grammatica risulterebbe deterministica $LL(1)$, giacché la procedura di messa in forma deterministica dell'automa associato a produzioni alternative identificherebbe il prefisso a comune alle alternative di A, e lo metterebbe a fattore comune (in sostanza applicherebbe in modo automatico una delle regole di riduzione da $LL(k)$ a $LL(k-1)$), ritardando la scelta e rendendola deterministica. Anche tale soluzione è perfettamente accettabile, anzi perfino più efficiente, ed è pertanto illustrata di seguito.

Soluzione alternativa Si procede per gradi. Prima si dìano gli automi lineari modellanti ciascuna delle quattro produzioni separatamente. Eccoli:

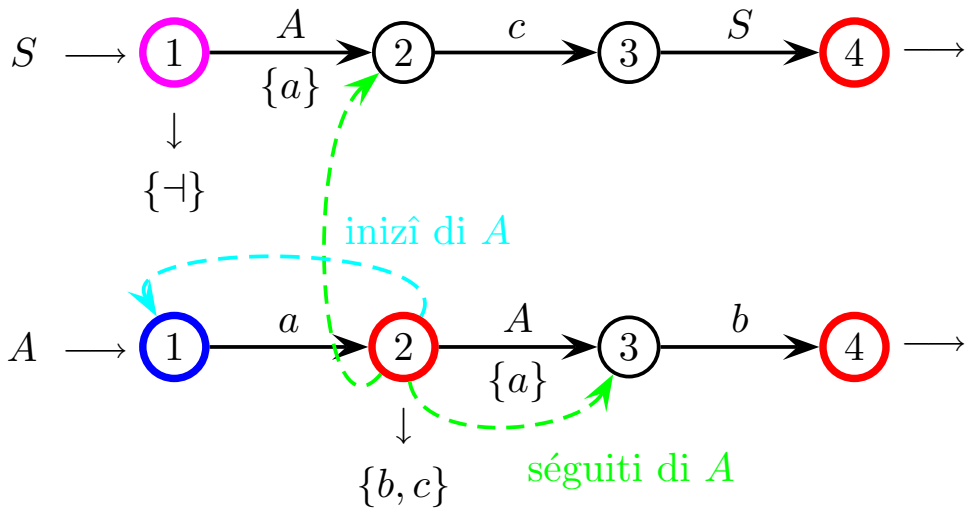


Poi si uniscano in modo indeterministico gli automi corrispondenti ad alternative per lo stesso non-terminale e, nei punti di biforcazione, si calcolino gli insiemi guida sugli archi. Le frecce tratteggiate indicano la parte meno ovvia di tale calcolo. Ecco il risultato dell'unione:



Per ora la grammatica non risulta $LL(1)$, a motivo degli insiemi guida non disgiunti alla biforcazione nello stato 0 dell'automa di A. Comunque si ricordi che il modello ad automi si può mettere in forma deterministica, e tale trasformazione

potrebbe rendere la grammatica $LL(1)$. Pertanto, mettendo in forma det. e ricalcolando gli insiemi guida nei punti di biforcazione, si ha quanto segue:

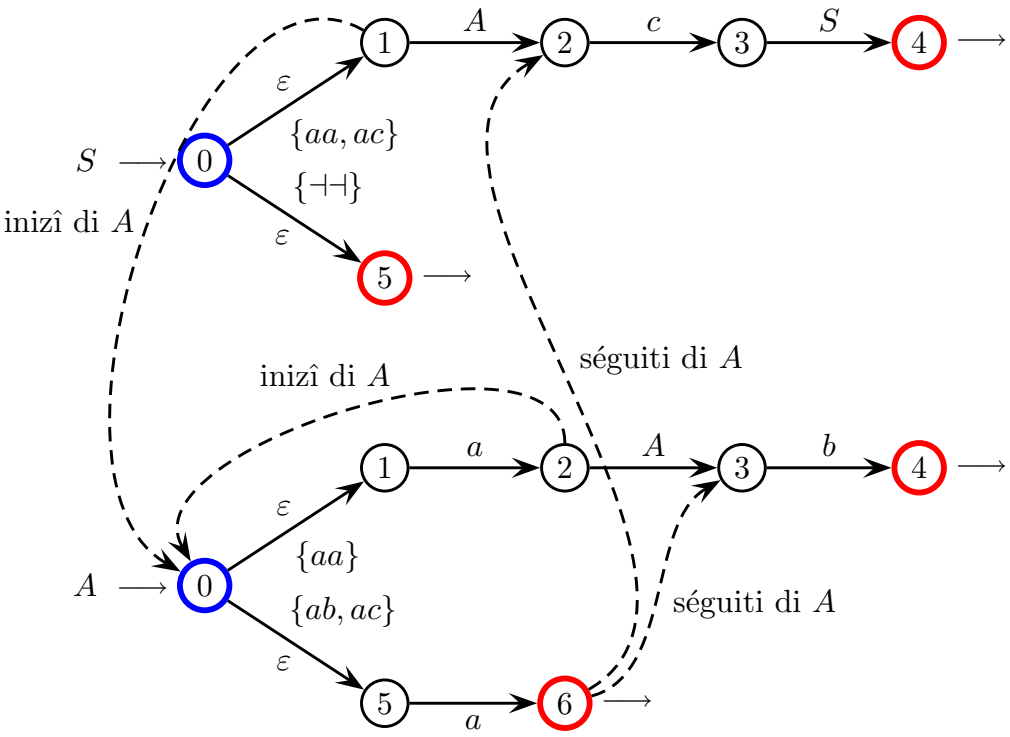


Dopo la trasformazione la grammatica risulta deterministica $LL(1)$, perché il prefisso a delle alternative di A è stato raccolto a fattore comune. La procedura sintattica $LL(1)$ di A deve semplicemente simulare l'automa deterministico corrispondente. Eccola, sulla falsariga dell'altra (ma non identica a quella):

```
char cc                                          - - carattere corrente
procedure A                                      - - parte in stato 1
      if (cc == 'a') then                        - - va in stato 2
            read (cc)
            if (cc == 'b' or cc == 'c') then     - - termina
                  null
            else if (cc == 'a') then             - - va in stato 3
                  call A
                  if (cc == 'b') then            - - va in stato 4
                        read (cc)
                  else
                        errore                   - - errore in stato 3
                  end if
            else                                 - - errore in stato 2
                  errore
            end if
      else                                       - - errore in stato 1
            errore
      end if
end procedure                                    - - termina
```

Beninteso, calcolando gli insiemi guida di profondità 2 sulle versioni indeterministiche degli automi, si sarebbe comunque concluso che la grammatica è

deterministica $LL(2)$, ritrovando così lo stesso risultato di prima. Giusto per completezza, eccoli (non sono richiesti dall'esercizio):



Le linee tratteggiate illustrano il calcolo del secondo simbolo degli insiemi guida (tranne che per $\dashv\dashv$, del tutto ovvio). Si ritrova la conclusione che la grammatica, così com'è, è di tipo $LL(2)$, anche se solo per quanto attiene il non-terminale A (il non-terminale S sarebbe $LL(1)$, preso isolatamente).

Infine si dà la grammatica trasformata per fattorizzazione sinistra, sì da renderla $LL(1)$ (non è richiesta dall'esercizio). Eccola:

produzione			ins. guida ($k = 1$)
S	$\rightarrow$	$A\ c\ S$	a
S	$\rightarrow$	ε	$\dashv$
A	$\rightarrow$	$a\ B$	a
B	$\rightarrow$	$A\ b$	a
B	$\rightarrow$	ε	$b,\ c$

Il non-terminale aggiuntivo B fattorizza a sinistra il prefisso a comune alle alternative di A nella grammatica originale. Si vede bene come la trasformazione sia analoga a mettere in forma deterministica gli automi delle produzioni.

2. È data la grammatica G seguente (assioma S_0):

$$S_0 \to S \ \dashv$$

$$S \to A\ S$$

$$S \to \varepsilon$$

$$A \to a\ A\ b$$

$$A \to a$$

Si svolga a scelta almeno uno dei due punti seguenti (e facoltativamente l'altro, se si ha tempo bastante):

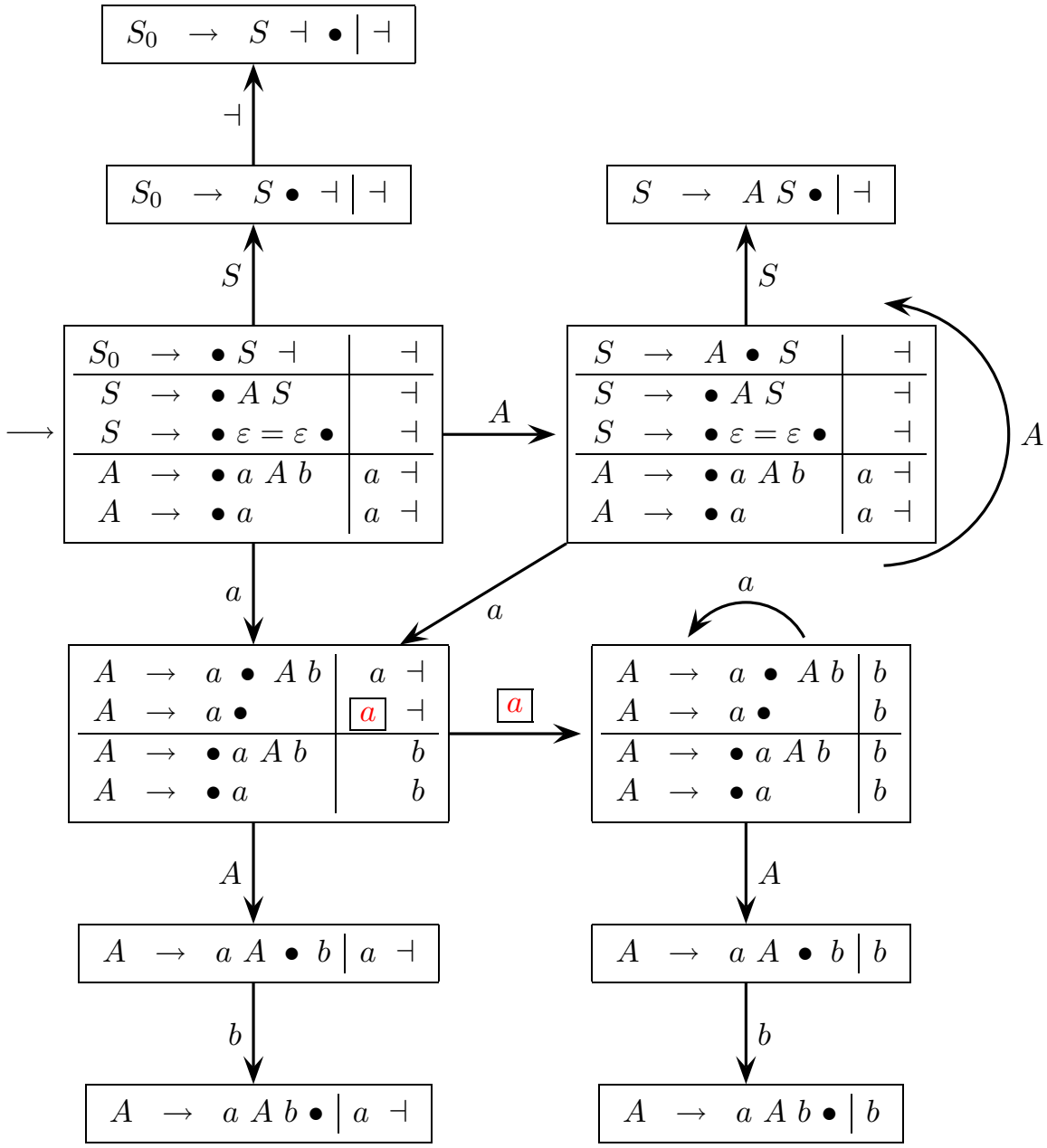
(a) Si tracci il grafo pilota $LR(1)$ (riconoscitore dei prefissi) completo della grammatica G e si stabilisca se essa sia det. $LR\,(0)$, $LALR\,(1)$, $LR\,(1)$, o non-det.

(b) Si effettui il riconoscimento della stringa $aab \dashv \in L(G)$ simulando l'algoritmo di Earley per G (più avanti è già pronto lo schema), e si ricostruisca l'albero sintattico della stringa.

Soluzione

Vedi a pagina seguente.

(a) Ecco il grafo pilota $LR(1)$ (o riconoscitore dei prefissi) della grammatica:



Si vede subito che c'è conflitto riduzione-spostamento (si guardino i caratteri in rosso inquadrati), e che pertanto la grammatica risulta non-deterministica $LR(1)$ (*a fortiori*, dunque, non è né det. $LALR$ né det. $LR(0)$).

Si sarebbe potuto procedere similmente modellando la grammatica mediante gli automi di produzione (vedi sito del corso), ottenendone così il grafo pilota in funzione degli stati degli automi che ne esprimono le produzioni.

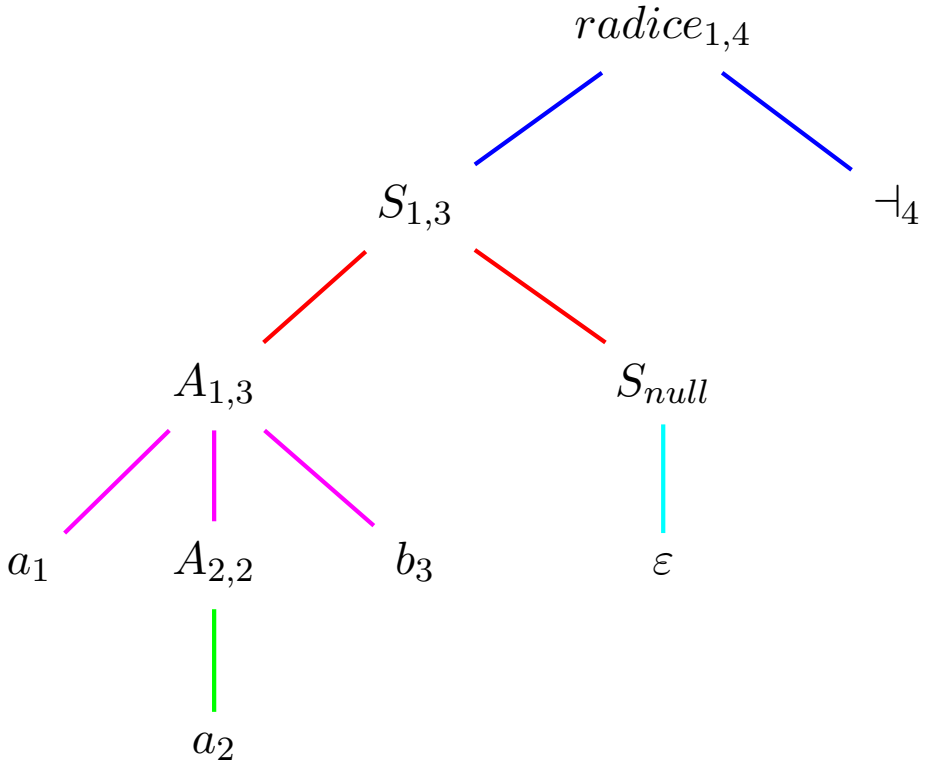
Volendo, si sarebbe potuta trascurare la produzione assiomatica $S_0 \to S \dashv$, che serve solo a esplicitare il terminatore '$\dashv$', partendo direttamente dal non-terminale S; ciò avrebbe ridotto anche più la taglia del grafo pilota, già modesta.

Simulazione dell'algoritmo di Earley

stato 0	p. 1 a	stato 1	p. 2 a	stato 2	p. 3 b	stato 3	p. 4 $\dashv$	stato 4
$S_0 \rightarrow \bullet\ S\ \dashv$	0	$A \rightarrow a \bullet A\ b$	0	$A \rightarrow a \bullet A\ b$	1	$A \rightarrow a\ A\ b \bullet$	0	$S_0 \rightarrow S\ \dashv \bullet$
$S \rightarrow \bullet\ A\ S$	0	$A \rightarrow a \bullet$	0	$A \rightarrow a \bullet$	1	$S \rightarrow A \bullet S$	0	1 candidata
$S \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	0	$A \rightarrow \bullet\ a\ A\ b$	1	$A \rightarrow \bullet\ a\ A\ b$	2	$S \rightarrow \bullet\ A\ S$	3	
$A \rightarrow \bullet\ a\ A\ b$	0	$A \rightarrow \bullet\ a$	1	$A \rightarrow \bullet\ a$	2	$S \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	3	
$A \rightarrow \bullet\ a$	0	$S \rightarrow A \bullet S$	0	$A \rightarrow a\ A \bullet\ b$	0	$A \rightarrow \bullet\ a\ A\ b$	3	
$S \rightarrow S \bullet\ \dashv$	0	$S \rightarrow \bullet\ A\ S$	1	$S \rightarrow A \bullet S$	1	$A \rightarrow \bullet\ a$	3	
6 candidate		$S \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	1	$S \rightarrow \bullet\ A\ S$	2	$S \rightarrow A\ S \bullet$	0	
		$S \rightarrow A\ S \bullet$	0	$S \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	2	$S_0 \rightarrow S \bullet\ \dashv$	0	
		$S_0 \rightarrow S \bullet\ \dashv$	0	$S \rightarrow A\ S \bullet$	1	8 candidate		
		9 candidate		$S \rightarrow A\ S \bullet$	0			
				$S_0 \rightarrow S \bullet\ \dashv$	0			
				11 candidate				

Le sottoliste indicano gli stadi progressivi di espansione e completamento delle varie candidate.
Per l'interpretazione dei colori si veda l'albero sintattico alla pagina seguente.
La candidata assiomatica di completamento nello stato numero 4 assicura l'avvenuto riconoscimento.
Complessivamente sono state esaminate 35 candidate.

(b) La ricostruzione dell'albero sintattico della stringa "$aab \dashv$" è la seguente:



Le candidate di completamento che danno luogo all'albero sono facilmente rintracciabili nella tabella precedente, cercando negli stati indicati dal pedice di destra di ciascun nodo interno (cioè non-terminale) dell'albero. Esse sono inquadrate e in colore corrispondente ai lati dell'albero. Per esempio, il sottoalbero elementare $A_{1,3} \to a_1\ A_{2,2}\ b_3$ (in rosa) è dato dalla candidata di completamento $A \to a\ A\ b\ \bullet \mid 0$ che figura nello stato 3, perché il non-terminale $A_{1,3}$ ha:

- pedice dx di valore 3 e la candidata completata $A \to a\ A\ b\ \bullet \mid 0$ si trova nello stato 3, cioè nello stato immediatamente seguente alla posizione 3 che corrisponde appunto al pedice dx
- pedice sx di valore 1 e il puntatore associato alla candidata ha valore 0, cioè indica lo stato immediatamente precedente alla posizione 1 che corrisponde appunto al pedice sx, stato dove si trova la candidata $A \to \bullet\ a\ A\ b \mid 0$ appena espansa e pronta per iniziare a generare

Si ricorda che il significato dei pedici di un nodo $X_{h,k}$ è che il non-terminale X genera la sottostringa dalla lettera di posizione h (compresa) a quella di posizione k (compresa), con $1 \leq h \leq k \leq n$. Infatti $A_{1,3}$ genera $a_1a_2b_3$.

La ricostruzione è evidentemente unica, giacché la grammatica non è ambigua e dunque la stringa ammette un solo albero sintattico.

4 Traduzione e analisi semantica 20%

1. Si consideri la funzione di traduzione τ seguente:

$$\begin{aligned} \tau(x) &= h_a(x) && \text{se } |x|_a \text{ è pari} \\ \tau(x) &= h_b(x) && \text{se } |x|_a \text{ è dispari} \end{aligned}$$

dove h_a e h_b sono le proiezioni che cancellano b e a, rispettivamente (per esempio $h_a(aba) = aa$ e $h_b(bbabb) = bbbb$), e x è una stringa qualunque di alfabeto $\{a, b\}$.

Esempi:

$$\tau(\varepsilon) = \varepsilon \qquad \tau(a) = \varepsilon \qquad \tau(b) = \varepsilon \qquad \tau(abaa) = b \qquad \tau(bbaa) = aa$$

Si svolgano i punti seguenti:

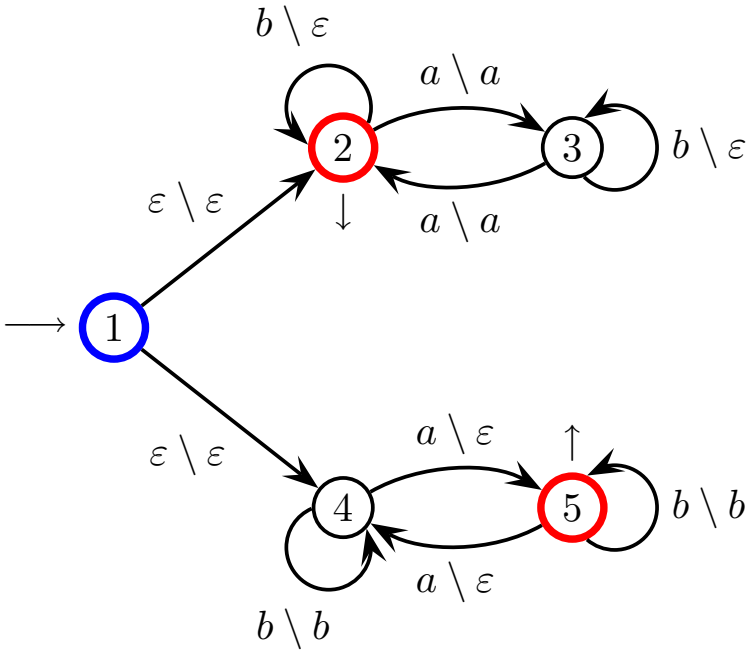
(a) Si progetti un trasduttore a stati finiti (automa I/O), anche indeterministico, che realizzi la traduzione τ.

(b) Si progetti un trasduttore a pila deterministico che realizzi la stessa traduzione (volendo, si può supporre che la stringa sorgente x abbia terminatore).

Soluzione

Vedi a pagina seguente.

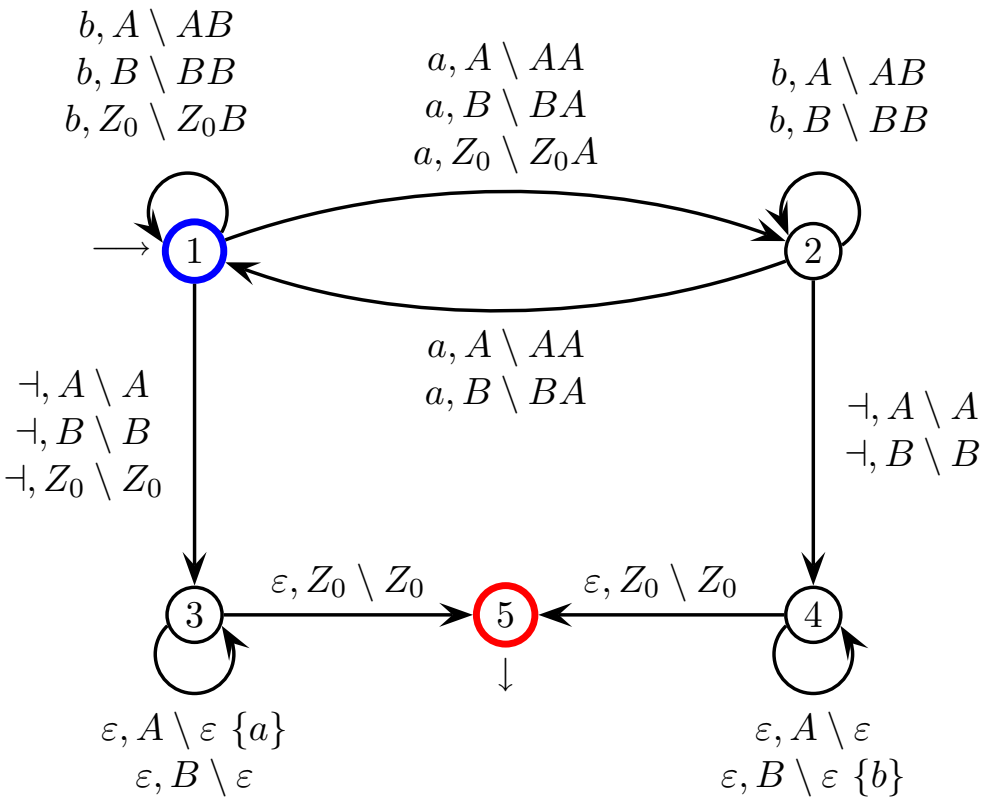
(a) Trasduttore a stati finiti (con riconoscimento a stato finale):



Dato che è ammesso indeterminismo, tanto vale servirsene. L'automa funziona così: parte in stato 1; indeterministicamente sceglie se il numero di lettere a in ingresso si debba considerare pari o dispari passando in stato 2 o 4, rispettivamente; legge e proietta la stringa di ingresso su a o b secondo la scelta fatta, emettendola sul nastro in uscita; esprime la validità della traduzione ogniqualvolta passa per gli stati finali 2 e 5, che contano modulo 2 il numero di lettere a in ingresso, rispettivamente pari o dispari.

L'automa è fortemente indeterministico: c'è una secca scelta indeterministica iniziale, da prendere in assenza di qualunque informazione in merito al numero di lettere a che si troveranno effettivamente; si comprende intuitivamente come la traduzione proposta non si possa realizzare a stati finiti in modo deterministico, giacché implicherebbe la conoscenza *a priori* del numero di lettere a in ingresso. La soluzione non è unica, si possono trovare varianti.

(b) Trasduttore a pila (con riconoscimento a stato finale):



La coppia di parentesi graffe (p. es. $\{a\}$) indica l'emissione di un simbolo (p. es. a) sul nastro di uscita. Dove non ci sono graffe, la transizione non emette niente sul nastro di uscita (cioè si sottintende emetta $\{\varepsilon\}$).

Intuitivamente si comprende come, mediante la pila, la traduzione si possa fare in modo deterministico (a stato finale): basta usare le pila come memoria della stringa in ingresso, dunque leggendola tutta e contandone le lettere a prima di cominciare a emetterne la traduzione. L'automa funziona così: parte in stato 1; tramite gli stati 1 e 2 conta, modulo 2, le lettere a in ingresso, e contemporaneamente memorizza in pila l'intera stringa di ingresso, codificandola mediante i simboli di memoria A e B; finita la stringa di ingresso (leggendone il terminatore), passa negli stati 3 o 4, secondo il modulo del conteggio, rilegge dalla pila la stringa e la emette sul nastro di uscita proiettandola su a o b corrispondentemente; infine a pila vuota passa nello stato finale 5. L'automa è palesemente deterministico. La soluzione non è unica, ci sono varianti.

Non sarebbe difficile modificare l'automa a pila ottenendone uno, sempre deterministico, ma con riconoscimento a pila vuota (basta togliere lo stato 5). Si ricordi che il riconoscimento deterministico a pila vuota è meno potente di quello a stato finale (indeterministicamente i due modelli di automa sono equivalenti).

2. È dato un testo alfabetico, di alfabeto italiano a, ..., z, seguito da una chiave di traslitterazione, consistente in un numero intero scritto in notazione unaria (p. es., la chiave 111_{unario} significa 3_{dieci}).

   Il testo va traslitterato, sostituendo a ogni carattere quello che nell'ordine alfabetico lo segue alla distanza della chiave. Per esempio:

testo sorgente con chiave	*testo traslitterato (chiave rimossa)*
`vita111`	`bnzd`

   Il supporto sintattico è il seguente:

$$
\begin{aligned}
S &\rightarrow T\ C \\
T &\rightarrow L\ T \\
T &\rightarrow \varepsilon \\
L &\rightarrow a \mid b \mid \ldots \mid z \\
C &\rightarrow 1\ C \\
C &\rightarrow 1
\end{aligned}
$$

   Si chiede di progettare una grammatica con attributi che ricavi il valore della chiave e lo usi per calcolare l'attributo τ del nonterminale L; tale attributo esprime la traslitterazione del carattere figlio di L (si usino ulteriori attributi se li si ritengono necessari o utili).

   Ecco i punti da svolgere:

   (a) Elencare gli attributi, con il rispettivo tipo e significato.
   (b) Scrivere le funzioni semantiche che calcolano gli attributi (alla pagina successiva è già pronto lo schema da compilare).
   (c) Disegnare i grafi delle dipendenze funzionali tra attributi, per ciascuna produzione separatamente.
   (d) Stabilire se la grammatica sia di tipo a una sola scansione.
   (e) Stabilire se la grammatica sia di tipo L.

Soluzione

(a) L'intuito suggerisce che, per risolvere il problema, si possa usare un attributo sintetizzato per calcolare in salita sull'albero il valore decimale della chiave, un attributo ereditato per propagarlo in discesa verso i terminali L che generano i caratteri del testo, e un attributo (sintetizzato) per esprimere i caratteri tradotti. Non si chiede di definire un attributo contenente l'intero testo traslitterato, ma lo si può benissimo aggiungere, per completezza (non fa parte dell'esercizio); ovviamente sarà di tipo sintetizzato.

Ecco dunque un elenco ragionevole di attributi (ideati a buon senso):

nome	tipo	significato	dominio	non-term. assoc.
τ	sx	traslitt. car.	carattere	L
α	sx	calcolo chiave	num. intero	C
β	dx	propagazione chiave	num. intero	L, T

e volendo si può aggiungere anche

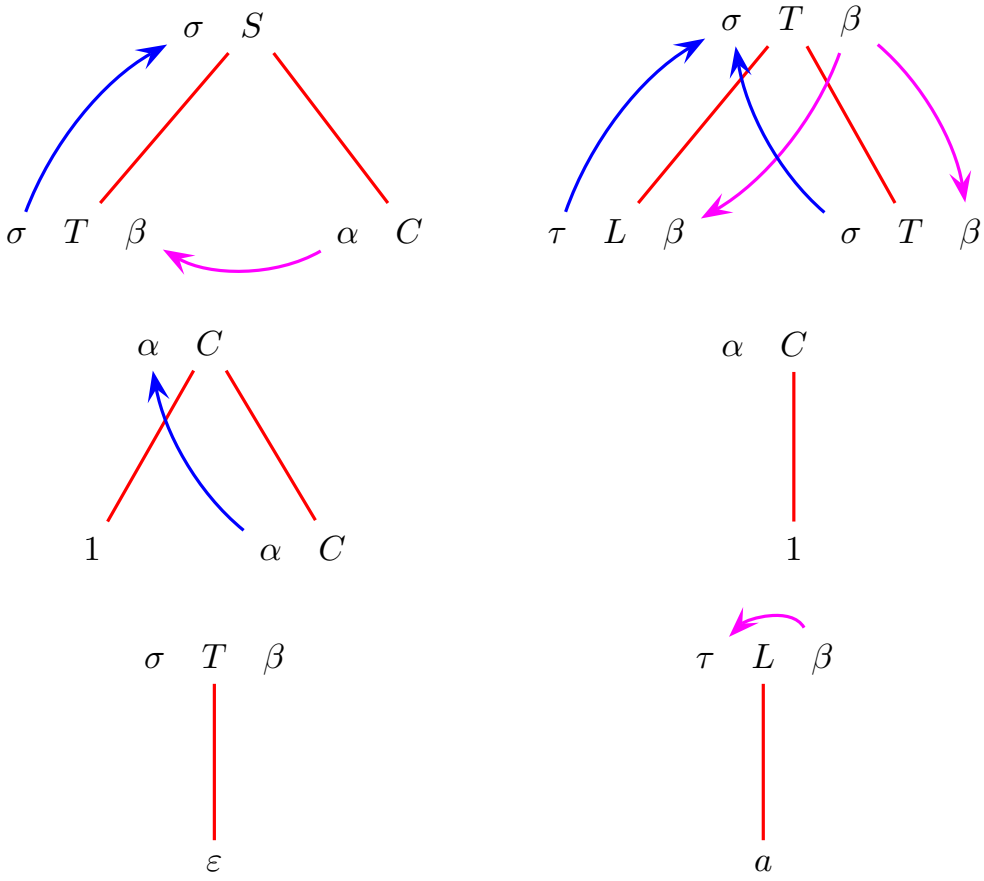
σ	sx	traslitt. testo	stringa	S, T

(b) Ed ecco le funzioni semantiche relative (con un attributo sx σ "in più"):

sintassi	*funzioni semantiche*	*e volendo anche ...*
$S_0 \to T_1\ C_2$	$\beta_1 = \alpha_2$	$\sigma_0 = \sigma_1$
$T_0 \to L_1\ T_2$	$\beta_1 = \beta_0 \qquad \beta_2 = \beta_0$	$\sigma_0 = \tau_1 \bullet \sigma_2$
$T_0 \to \varepsilon$	niente	$\sigma_0 =$ " " (stringa vuota)
$L_0 \to a$	$\tau_0 = \mathit{shift}$ ('a', β_0)	niente
...	$\tau_0 = \ldots$	niente
$L_0 \to z$	$\tau_0 = \mathit{shift}$ ('z', β_0)	niente
$C_0 \to 1\ C_1$	$\alpha_0 = \alpha_1 + 1$	niente
$C_0 \to 1$	$\alpha_0 = 1$	niente

La funzione ausiliaria "*shift*" funziona nel modo ovvio. L'attributo σ esprime la traslitterazione del testo completo (non è richiesto dall'esercizio), tolta la chiave. Il simbolo '$\bullet$' indica il concatenamento di stringhe (o di caratteri).

(c) I grafi delle dipendenze funzionali tra attributi, per ciascuna produzione, sono i seguenti (è mostrato anche l'attributo σ, non richiesto dall'esercizio)[5]:



(d) La grammatica è di tipo a una sola scansione: tutte le condizioni sono verificate, l'ordine di scansione dei fratelli T e C figli di S (ordine dato dal grafo dei fratelli) è prima C e poi T, ovviamente, a motivo della dipendenza di β da α. Infatti, α (sx) viene calcolato subito dopo avere visitato C, β (dx) viene calcolato subito prima di visitare T, β dipende da α, dunque prima si deve visitare C e poi T.

(e) La grammatica non è di tipo L perché, come osservato sopra, pur essendo a una sola scansione (condizione necessaria affinché sia di tipo L), l'ordine di visita dei figli T e C di S non va da sinistra verso destra (come sarebbe richiesto dalla condizione L), ma da destra verso sinistra (a motivo della dip. $\alpha \rightarrow \beta$).

Incidentalmente si può osservare come il supporto sintattico, di per sé, sia di tipo $LL(2)$ (ma non $LL(1)$; comunque si può mettere facilmente in forma $LL(1)$), e come pertanto si presti all'analisi discendente sinistra (ma le dipendenze funzionali tra attributi non la permettono).

Il lettore provi a disegnare l'albero sintattico decorato (con gli attributi e le rispettive dipendenze) della traslitterazione proposta come esempio nell'esercizio.

[5]Come d'uso, gli attributi sx sono posti a sinistra del nodo, quelli dx a destra.