Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 6 marzo 2009 - Parte I: Teoria

CON SOLUZIONI - A SCOPO DIDATTICO LE SOLUZIONI SONO MOLTO ESTESE E COMMENTATE VARIAMENTE - NON SI RICHIEDE CHE IL CANDIDATO SVOLGA IL COMPITO IN MODO ALTRETTANTO AMPIO, BENSÌ CHE RISPONDA IN MODO APPROPRIATO E A SUO GIUDIZIO RAGIONEVOLE

NOME:

COGNOME:

MATRICOLA: FIRMA:

ISTRUZIONI - LEGGERE CON ATTENZIONE:

- L'esame si compone di due parti:
  - I (80%) Teoria:
    1. espressioni regolari e automi finiti
    2. grammatiche libere e automi a pila
    3. analisi sintattica e parsificatori
    4. traduzione sintattica e analisi semantica
  - II (20%) Esercitazioni Flex e Bison
- Per superare l'esame l'allievo deve sostenere con successo entrambe le parti (I e II), in un solo appello oppure in appelli diversi, ma entro un anno.
- Per superare la parte I (teoria) occorre dimostrare di possedere conoscenza sufficiente di tutte le quattro sezioni (1-4), rispondendo alle domande obbligatorie (si noti che il punteggio pieno può essere ottenuto solo rispondendo alle parti facoltative).
- 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 binario $\Sigma = \{a, b\}$. Si consideri il linguaggio regolare L, di alfabeto Σ, definito come segue:

$$L = \{ \ w \mid \quad w \text{ non contiene tre lettere } b \text{ consecutive } \}$$

Esempi di stringhe di L:

$$\varepsilon \qquad a \qquad b \qquad a\,b \qquad b\,b \qquad a\,b\,b\,a \qquad a\,b\,a\,b\,b \qquad \ldots$$

Controesempi di stringhe di L:

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

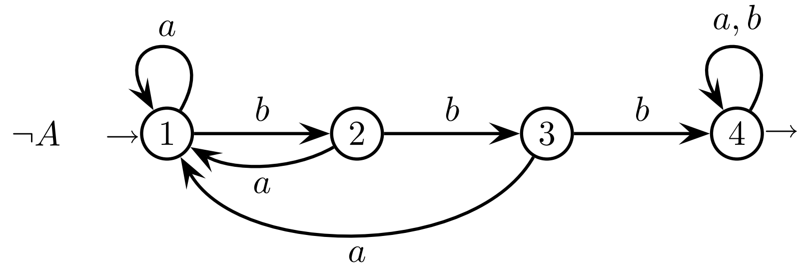
Si risponda alle domande seguenti:

(a) Si costruisca il grafo stato-transizione dell'automa deterministico minimo A che riconosce il linguaggio L.

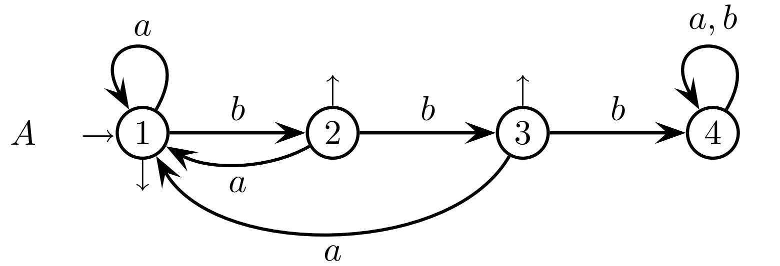
(b) (facoltativa) Partendo dall'automa A si ottenga l'espressione regolare R equivalente.

Soluzione

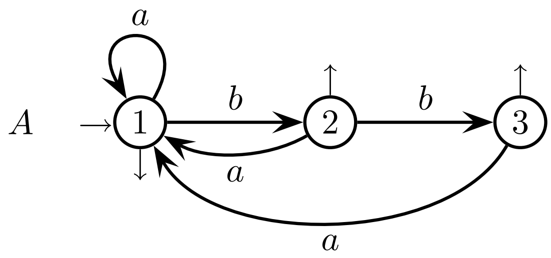
(a) Conviene partire del linguaggio $\neg L$, complemento di L, perché è più semplice. Ecco l'automa deterministico $\neg A$ che riconosce $\neg L$:



L'automa $\neg A$ è in forma naturale completa (ogni stato ha due archi uscenti). Ora si complementa tale automa, ottenendo l'automa A riconoscitore del linguaggio L. Eccolo:

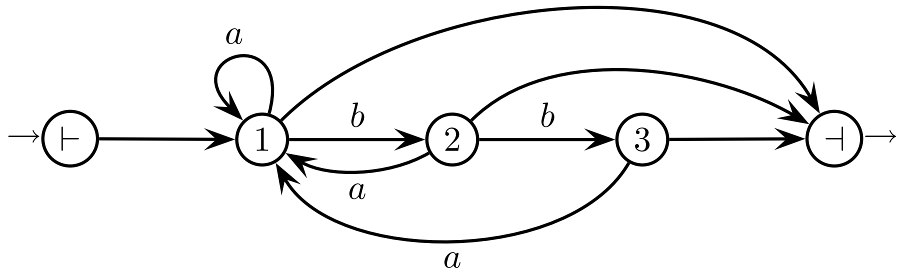


L'automa A non è in forma ridotta (pulita): lo stato 4 è indefinito (non raggiunge alcuno stato finale). Si può ridurre l'automa A nel modo seguente:

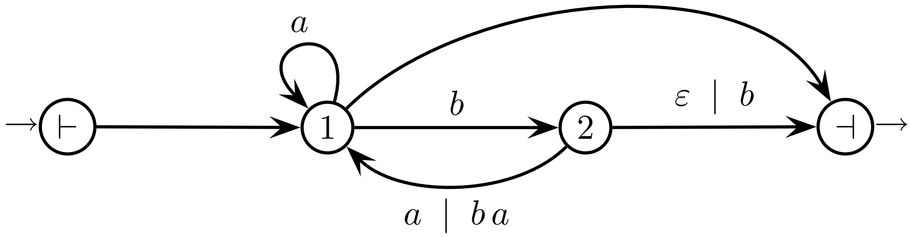


L'automa A ridotto così ottenuto è deterministico e minimo. Infatti lo stato 3 è distinguibile dagli stati 1 e 2 perché non ha arco b uscente, e gli stati 1 e 2 sono distinguibili perché a parità d'ingresso b vanno negli stati 2 e 3, rispettivamente, che come visto prima sono distinguibili. Dunque gli stati dell'automa A ridotto sono tutti distinguibili e in conclusione l'automa è minimo.

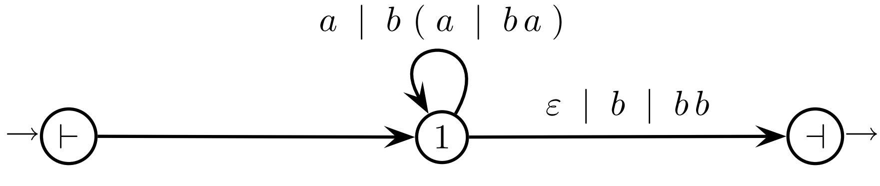
(b) Per ricavare l'espressione regolare R equivalente all'automa A si può ricorrere al metodo di eliminazione dei nodi di A (Brzozowski). Prima di rendono unici gli stati iniziale e finali.



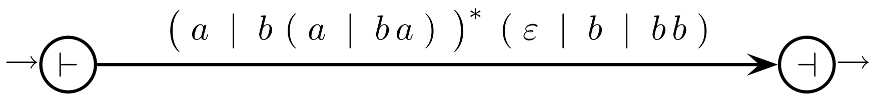
Poi si procede all'eliminazione dei nodi. Elimina nodo 3:



Elimina nodo 2:



Elimina nodo 1:



Così si ha la forma seguente per l'espressione regolare R:

$$R = \big(a \mid b\,(a \mid b\,a) \big)^* \, (\varepsilon \mid b \mid b\,b)$$

Essendo ricavata da automa deterministico, l'espressione R non è ambigua.
Applicando la proprietà distributiva di concatenamento rispetto a unione, si può riscrivere l'espressione R come segue:

$$R = (a \mid b\,a \mid b\,b\,a)^* \, (\varepsilon \mid b \mid b\,b)$$

Ora l'interpretazione è evidente: si forma qualunque stringa di lettere a e b, ma è impossibile che tre o più lettere b compaiano consecutivamente.
Beninteso si può riscrivere in numerosi modi equivalenti, come il seguente:

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

che pure ha interpretazione molto intuitiva, o anche come:

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

di aspetto leggermente più compatto; e altri modi ancora.

2. È dato l'alfabeto binario $\Sigma = \{a, b\}$. Si considerino i linguaggi regolari L_1 e L_2 seguenti, di alfabeto Σ:

$$L_1 = \{\ w \mid \quad |w| \geq 2\ \wedge\ \text{il penultimo carattere di } w \text{ è } b\ \}$$

$$L_2 = \{\ w \mid \quad \text{in ogni posizione dispari di } w \text{ il carattere è } b\ \}$$

Ecco alcuni esempi di stringhe del linguaggio L_2:

$$\varepsilon \qquad b \qquad b\,a \qquad b\,b \qquad \ldots$$

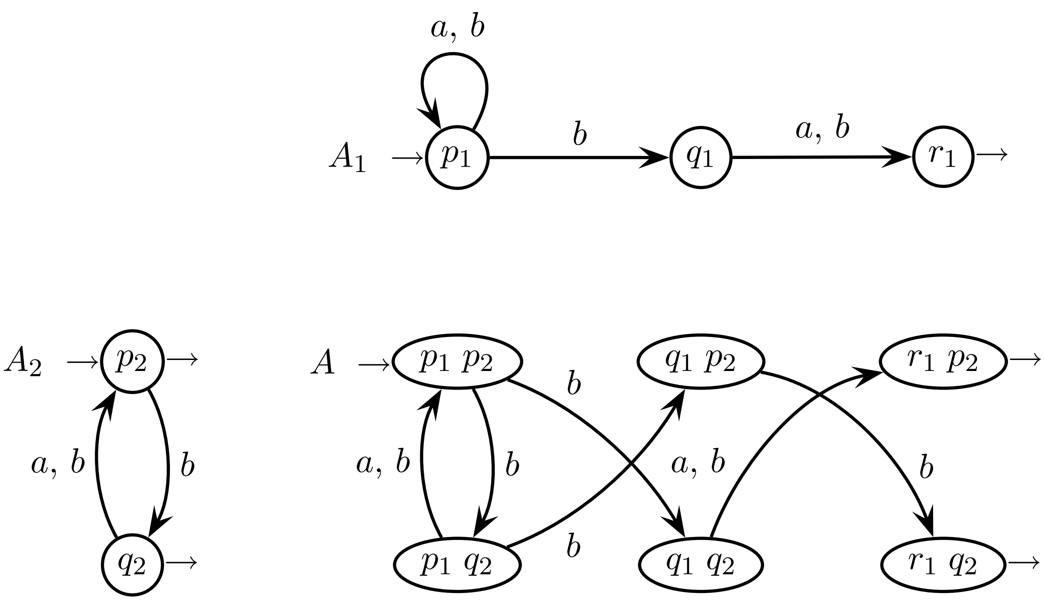
Si risponda alle domande seguenti:

(a) Si costruisca in modo sistematico un automa A, deterministico o non, che riconosce il linguaggio intersezione $L = L_1 \cap L_2$.

(b) (facoltativa) Si costruisca in modo sistematico un automa A' che riconosce il linguaggio differenza $L' = L_1 \setminus L_2$ (che si scrive anche $L' = L_1 - L_2$).

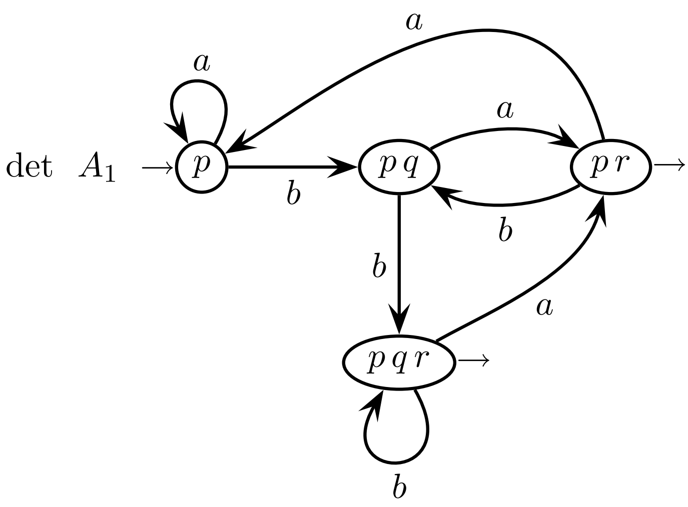
Soluzione

(a) Si disegnano facilmente gli automi A_1 e A_2 dei due linguaggi L_1 e L_2, rispettivamente, e si costruisce la macchina prodotto cartesiano $A = A_1 \times A_2$ per riconoscere il linguaggio intersezione L. Ecco il tutto:



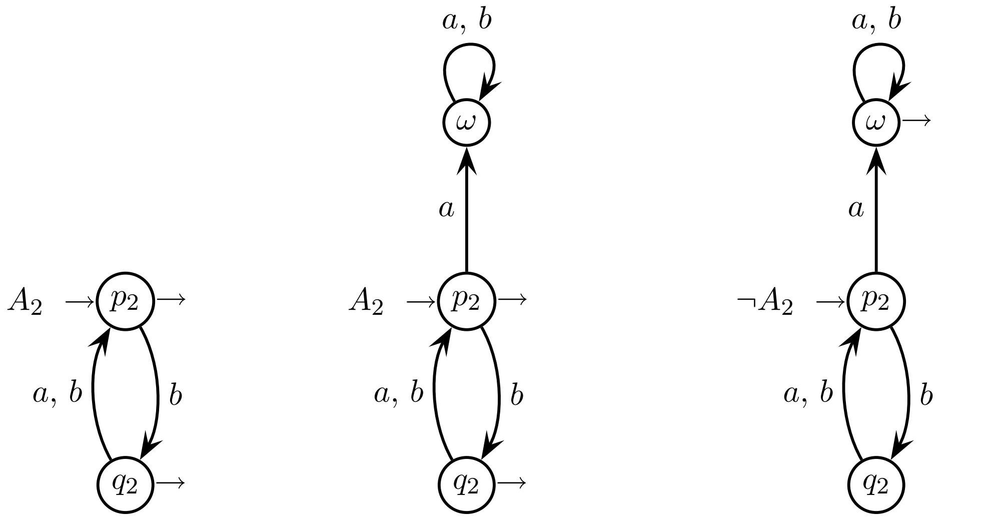
L'automa prodotto A è in forma ridotta (pulita): tutti gli stati sono utili, ossia raggiungibili e definiti (o post-raggiungibili). Non essendo deterministico il riconoscitore A_1 del linguaggio L_1, non lo è neanche l'automa A.

Si noti che se l'automa A_1 fosse deterministico, avrebbe quattro stati. Eccone il grafo stato-transizione, ottenuto mediante la costruzione dei sottinsiemi (per brevità la numerazione è omessa):



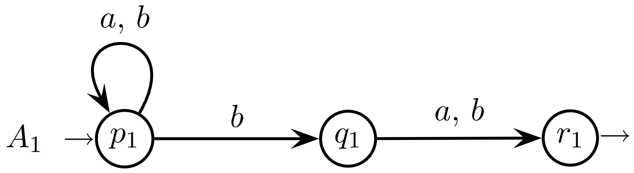
Utilizzando tale versione deterministica dell'automa A_2, il prodotto cartesiano A risulta pure deterministico, ma il numero di stati è potenzialmente maggiore, salvo poi eventualmente pulire l'automa e minimizzarlo.

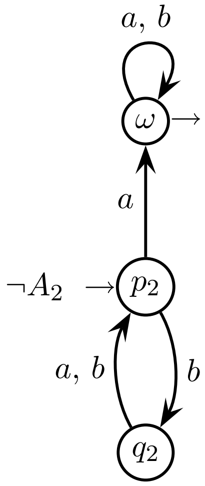
(b) Si ha l'identità insiemistica $L' = L_1 \setminus L_2 = L_1 \cap \neg L_2$. Pertanto il riconoscitore A' del linguaggio differenza L' è pure ottenibile come prodotto cartesiano $A' = A_1 \times \neg A_2$, dove $\neg A_2$ è l'automa complemento di A_2. Bisogna pertanto complementare il riconoscitore A_2 del linguaggio L_2. Essendo il riconoscitore A_2 deterministico, basta completare in modo naturale la funzione di transizione (con stato di errore ω) e poi scambiare stati finali e non, ottenendo così l'automa $\neg A_2$ seguente (deterministico e palesemente minimo):

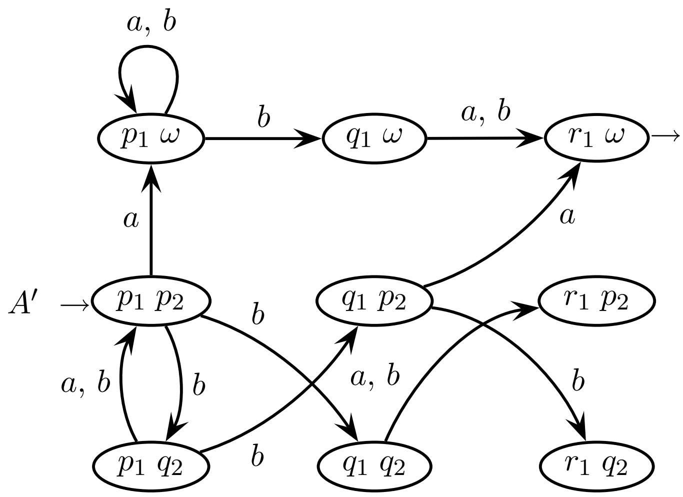


automa — completamento naturale — complementazione

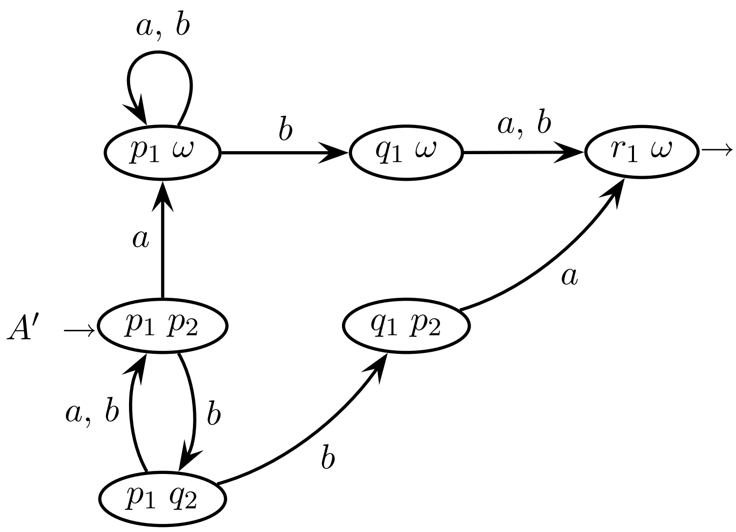
Ora si può costruire l'automa prodotto A'. Eccolo:







Gli stati dell'automa A' sono tutti raggiungibili, ma parecchi sono indefiniti (ossia non sono post-raggiungibili). Ecco la forma ridotta (pulita) di A', dove tutti gli stati sono utili:



Anche dopo la pulizia, l'automa A' risulta essere indeterministico.

È facile verificare intutivamente che l'automa A' riconosce effettivamente il linguaggio L', giacché questo ha la caratterizzazione seguente:

$$\begin{aligned} L' &= L_1 \setminus L_2 = L_1 \cap \neg L_2 \\ &= \left\{ w \;\middle|\; \begin{array}{l} \text{la lunghezza di } w \text{ è maggiore o uguale a 2, e} \\ \text{nella penultima posizione di } w \text{ c'è } b\text{, e} \\ \text{in almeno una posizione dispari di } w \text{ c'è } a \end{array} \right\} \end{aligned}$$

Osservando che la seconda e terza clausola del predicato implicano la stringa w abbia lunghezza non minore di 3, si può riscrivere la caratterizzazione così:

$$L' = \left\{ w \;\middle|\; \begin{array}{l} \text{la lunghezza di } w \text{ è maggiore o uguale a 3, e} \\ \text{nella penultima posizione di } w \text{ c'è } b\text{, e} \\ \text{in almeno una posizione dispari di } w \text{ c'è } a \end{array} \right\}$$

Si riscontra facilmente tale caratterizzazione sul grafo stato-transizione. Comunque l'automa A' è indeterministico. In linea di principio lo si potrebbe determinizzare e, ciò fatto, se necessario minimizzare.

2 Grammatiche libere e automi a pila 20%

1. Con riferimento al linguaggio di Dyck con parentesi tonde e quadre, si considerino i due linguaggi L_1 e L_2 seguenti, entrambi sottinsiemi di Dyck. Eccone la caratterizzazione:

   - Le stringhe di L_1 contengono solo parentesi tonde e hanno le parentesi di livello più interno a profondità pari.
   - Nelle stringhe di L_2 le parentesi <u>tonde</u> sono annidate <u>tra loro</u> fino a un livello di profondità pari, indipendentemente dalla presenza di eventuali parentesi quadre.

     Nota bene: cancellando le parentesi quadre dalle stringhe di L_2 si ottiene L_1.

   Ecco alcuni esempi di stringhe di L_1:

   $$\varepsilon \qquad (\,(\,)\,) \qquad (\,(\,)\,(\,)\,) \qquad (\,(\,)\,)\,(\,(\,)\,) \qquad (\,(\,(\,(\,)\,)\,)\,)$$

   Ed ecco alcuni esempi di stringhe di L_2:

   $$\varepsilon \qquad [\,] \qquad (\,(\,)\,) \qquad (\,[\,(\,)\,]\,) \qquad [\,(\,[\,[\,(\,)\,]\,(\,[\,]\,)\,]\,)\,]$$

   Si risponda alle domande seguenti:

   (a) Si definisca le grammatica G_1, preferibilmente non ambigua, che genera il linguaggio L_1.

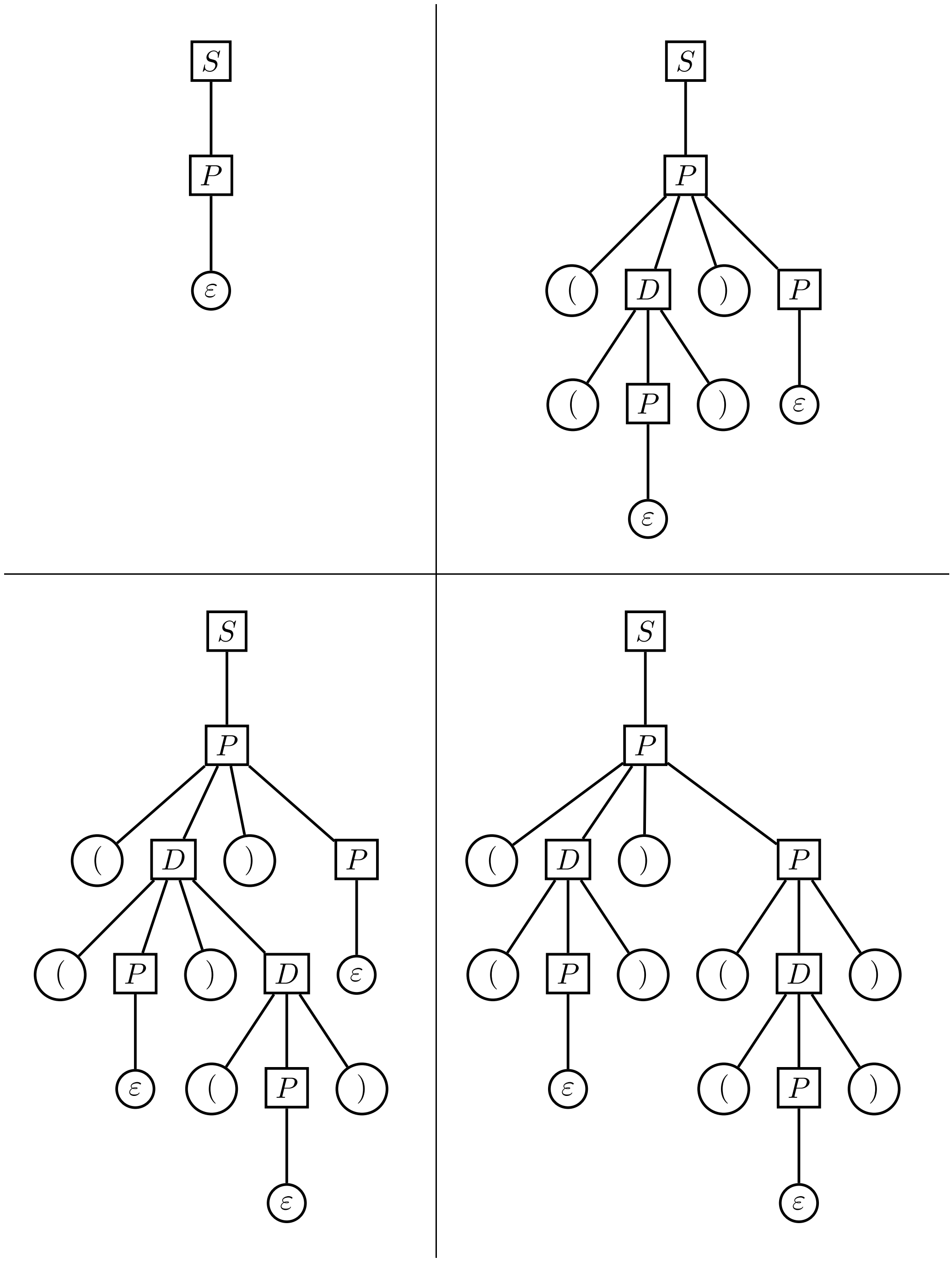
   (b) (facoltativa) Si definisca le grammatica G_2, preferibilmente non ambigua, che genera il linguaggio L_2.

Soluzione

(a) Ecco la grammatica G_1, non ambigua, che genera il linguaggio L_1 (assioma S):

$$G_1 \begin{cases} S \rightarrow P \\ P \rightarrow \text{‘(’}\ D\ \text{‘)’}\ P \mid \varepsilon \\ D \rightarrow \text{‘(’}\ P\ \text{‘)’}\ D \mid \text{‘(’}\ P\ \text{‘)’} \end{cases}$$

La grammatica G_1 è ottenuta da quella non ambigua di Dyck $S \rightarrow (\ S\ )\ S \mid \varepsilon$, distinguendo tra strutture parentetiche di profondità pari o dispari (mediante i nonterminali P e D, rispettivamente). Pertanto anche G_1 non è ambigua. Eccone alcuni alberi sintattici, per le prime quattro stringhe di esempio:

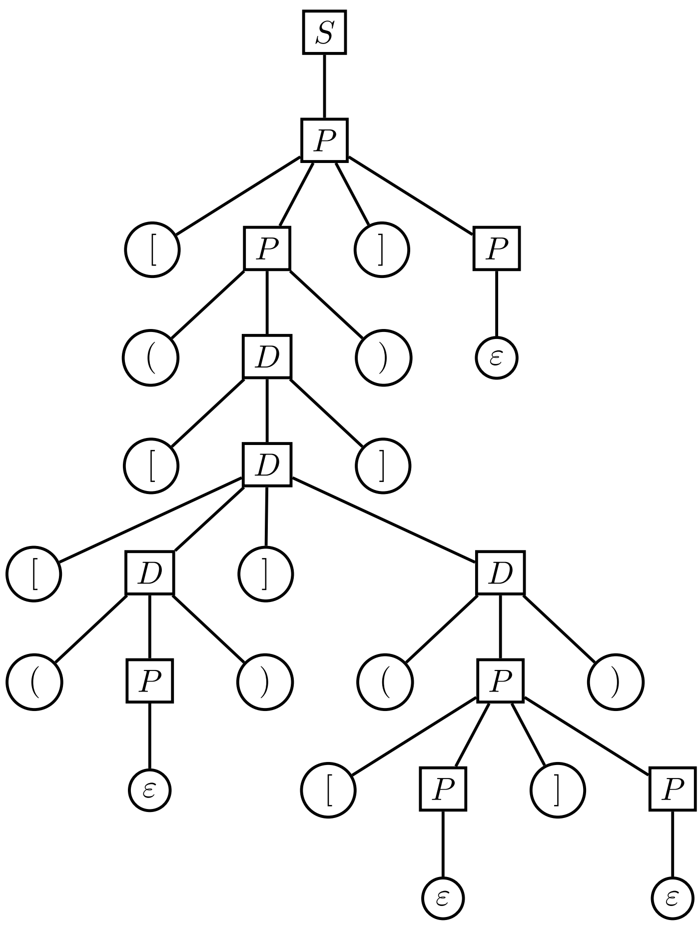


Si vede chiaramente che il conteggio pari-dispari scatta ogniqualvolta una coppia di parentesi viene innestata nella struttura in costruzione. Similmente per la quinta stringa di esempio, qui omessa (il lettore tracci da sé l'albero sintattico corrispondente).

(b) Ed ecco la grammatica G_2 che genera il linguaggio L_2, ottenuta dalla grammatica G_1 mediante inserimento di parentesi quadre (assioma S):

$$G_2 \left\{ \begin{array}{lcl} S & \rightarrow & P \\ P & \rightarrow & \text{‘(’}\ D\ \text{‘)’}\ P \ \mid\ \text{‘[’}\ P\ \text{‘]’}\ P \ \mid\ \varepsilon \\ D & \rightarrow & \text{‘(’}\ P\ \text{‘)’}\ D \ \mid\ \text{‘(’}\ P\ \text{‘)’} \\ D & \rightarrow & \text{‘[’}\ D\ \text{‘]’}\ D \ \mid\ \text{‘[’}\ D\ \text{‘]’} \end{array} \right.$$

Le regole della grammatica G_2 sono le stesse della grammatica G_1, ma "sdoppiate" sostituendo le parentesi tonde con le quadre e aggiustando di conseguenza il conteggio di parità delle tonde. Similmente, anche G_2 non è ambigua. Ecco l'albero sintattico della quinta stringa di esempio, la più complessa:



Si vede chiaramente che il conteggio pari-dispari scatta ogniqualvolta una coppia di parentesi tonde (non quadre) viene innestata nella struttura in costruzione. Similmente per le prime quattro stringhe di esempio, qui omesse (il lettore tracci da sé gli alberi sintattici corrispondenti).

2. Si consideri il linguaggio delle espressioni aritmetiche intere con gli aspetti lessicali e sintattici seguenti:

   - il linguaggio ha l'identificatore alfanumerico, in stile simile al linguaggio C, come

     `a` `alfa` `alfa12` `beta_1` `alfa_omega`

   - il linguaggio ha la costante intera decimale, in stile simile al linguaggio C, come

     `0` `1` `1234` `98012`

   - l'espressione contiene variabili, il cui nome è un generico identificatore, e costanti numeriche intere decimali
   - l'espressione contiene funzioni, con la sintassi seguente:

     ```
     nome_funzione ( lista_di_parametri )
     ```

     dove il nome della funzione è un identificatore
   - la lista di parametri non è vuota, i parametri sono separati da ',' (virgola) e in generale il parametro è un'espressione
   - l'espressione contiene gli operatori aritmetici '+' e '*' (addizione e moltiplicazione), di tipo <u>infisso</u>, e la moltiplicazione ha precedenza sull'addizione
   - l'espressione contiene l'operatore <u>binario</u> 'max', di tipo <u>prefisso</u>, che ha precedenza <u>inferiore</u> sia all'addizione sia alla moltiplicazione, come

     ```
     max a b + c
     ```

     dove prima si calcola la somma `b + c` e poi il massimo tra `a` e somma
   - l'espressione contiene parentesi tonde '(' e ')'

   Ecco un esempio di espressione:

   ```
   12 + (max b1 + c_2 alfa * (d + fun (a, omega))) * beta
   ```

   Si scriva una grammatica G, non ambigua e in forma estesa (EBNF) che genera il linguaggio così descritto.

Soluzione

Ecco la grammatica G (assioma EXPR):

$$G \left\{ \begin{array}{rcll}
 & & & \text{prec.} \\
\langle \mathsf{EXPR} \rangle & \rightarrow & \text{`max'}\ \langle \mathsf{EXPR} \rangle\ \langle \mathsf{EXPR} \rangle \ \mid \ \langle \mathsf{TERM} \rangle_1 & \text{min} \\
\hline
\langle \mathsf{TERM} \rangle_1 & \rightarrow & \langle \mathsf{TERM} \rangle_2 \ (\ \text{`+'}\ \langle \mathsf{TERM} \rangle_2\ )^* & \text{med} \\
\hline
\langle \mathsf{TERM} \rangle_2 & \rightarrow & \langle \mathsf{FACT} \rangle \ (\ \text{`}*\text{'}\ \langle \mathsf{FACT} \rangle\ )^* & \text{max} \\
\hline
\langle \mathsf{FACT} \rangle & \rightarrow & \text{`('}\ \langle \mathsf{EXPR} \rangle\ \text{`)'} & \\
\langle \mathsf{FACT} \rangle & \rightarrow & \langle \mathsf{ID} \rangle\ \text{`('}\ \langle \mathsf{PAR\_LIST} \rangle\ \text{`)'} & \\
\langle \mathsf{FACT} \rangle & \rightarrow & \langle \mathsf{ID} \rangle \ \mid \ \langle \mathsf{NUM} \rangle & \\
\hline
\langle \mathsf{PAR_LIST} \rangle & \rightarrow & \langle \mathsf{EXPR} \rangle \ (\ \text{`,'}\ \langle \mathsf{EXPR} \rangle\ )^* & \\
\hline
\langle \mathsf{ID} \rangle & \rightarrow & \langle \mathsf{CHAR} \rangle \ (\ \langle \mathsf{CHAR} \rangle \ \mid \ \langle \mathsf{DIGIT0} \rangle \ \mid \ \text{`_'}\ )^* & \\
\langle \mathsf{NUM} \rangle & \rightarrow & \langle \mathsf{DIGIT1} \rangle\ \langle \mathsf{DIGIT0} \rangle^* \ \mid \ \text{`0'} & \\
\hline
\langle \mathsf{CHAR} \rangle & \rightarrow & [\, A - Z \,] \ \mid \ [\, a - z \,] & \\
\langle \mathsf{DIGIT0} \rangle & \rightarrow & \langle \mathsf{DIGIT1} \rangle \ \mid \ \text{`0'} & \\
\langle \mathsf{DIGIT1} \rangle & \rightarrow & [\, 1 - 9 \,] &
\end{array} \right.$$

La grammatica G ha struttura modulare e i componenti sono non ambigui (liste, grammatica standard EBNF delle espressioni, ecc), pertanto non è ambigua. A destra sono indicati i livelli di precedenza degli operatori. Ecco alcune osservazioni:

- l'operatore "max" ha precedenza minima, in quanto è generato per primo partendo dall'assioma, ossia figura più vicino alla radice dell'albero sintattico
- la precedenza tra operatori "+" e '$*$" è quella usuale, ossia l'addizione ha precedenza inferiore alla moltiplicazione, pertanto l'addizione è generata prima della moltiplicazione, cioè nell'albero sintattico l'addizione compare sopra la moltiplicazione
- le regole che espandono caratteri e numeri sono espressioni regolari semplici
- il resto della grammatica è essenzialmente l'insieme standard di regole estese (EBNF) per le espressioni aritmetiche

Si lascia al lettore il compito di tracciare l'albero sintattico per l'espressione di esempio data prima, eventualmente compattandone le parti meno significative.

3 Analisi sintattica e parsificatori 20%

1. L'alfabeto $c = call$, $r = return$ e $s = statement$ idealizza le istruzioni `call`, `return` e i generici `statement` di un linguaggio di programmazione L. Il linguaggio L è definito dalla grammatica G seguente, in forma estesa (EBNF) con assioma S:

$$G \begin{cases} S \rightarrow (s \mid X)^* (s \mid X) \\ X \rightarrow c (s \mid X) r \end{cases}$$

Il carattere c è sempre seguito (non immediatamente) da un carattere r corrispondente, e i caratteri s possono trovarsi dappertutto, in numero qualunque.

Si risponda alle domande seguenti:

(a) Si disegni una rete di due macchine equivalenti alla grammatica G data.

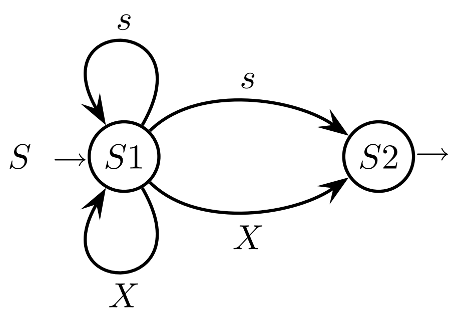
(b) Per tale rete si calcolino gli insiemi guida nei punti dov'è necessario e s'indichi se la grammatica è $LL(1)$ o $LL(k)$, per $k > 1$.

(c) (facoltativa) Si estenda il linguaggio L consentendo chiamate `call` prive del corrispondente `return`, per esempio $s\,c\,s\,c\,s\,r$. Si esamini se il linguaggio così esteso risulta $LL(k)$.

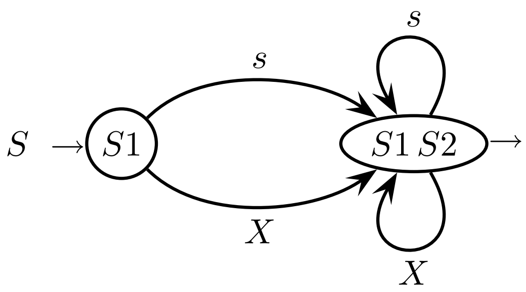
Soluzione

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

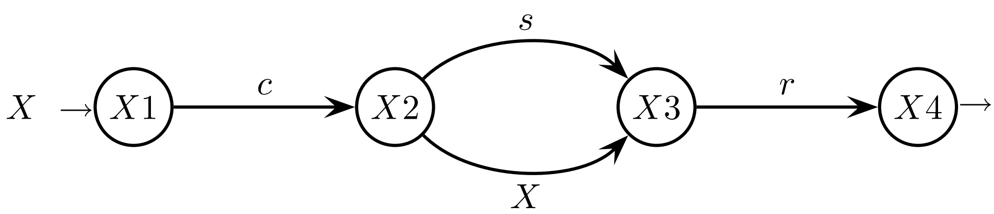
La macchina che espande l'assioma, strutturalmente corrispondente alla regola assiomatica estesa, è la seguente:



La macchina risulta indeterministica. Per l'analisi LL va determinizzata (rispetto all'alfabeto totale), come segue:

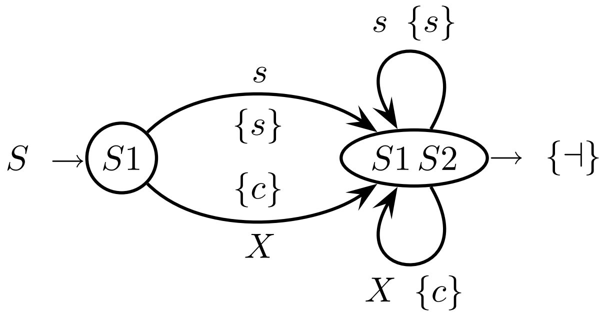


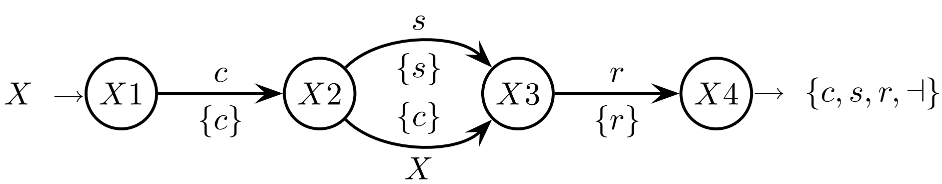
La macchina che espande il nonterminale X, strutturalmente corrispondente alla regola estesa, è la seguente:



La macchina risulta direttamente deterministica, pronta per l'analisi LL.

(b) Ecco l'analisi $LL(1)$ completa (anche dove non serve) della grammatica G.





Gli insiemi guida per $k = 1$ sono riportati sulle figure. Nelle biforcazioni tali insiemi sono disgiunti, pertanto la grammatica G è $LL(1)$.

(c) Si possono trascurare i caratteri s che sono permessi ovunque nelle frasi. Il linguaggio esteso contiene tra le altre le stringhe dei due casi seguenti:

$$c^n\ r^n\ c^+ \qquad\qquad n \geq 1$$

$$c^n\ r^n\ c^m\ r^m \qquad\qquad n, m \geq 1$$

Al termine della scansione del prefisso $c^n\ r^n$ $(n \geq 1)$ comune ai due casi, il parsificatore (automa a pila) deterministico di tipo discendente si troverà necessariamente nella stessa configurazione. Ma per ogni valore di $k \geq 1$ fissato i prossimi k caratteri possono essere gli stessi per c^+ e $c^m\ r^m$, mentre le regole per generare il componente regolare c^+ o la struttura parentetica $c^m\ r^m$ sono necessariamente diverse. Pertanto il linguaggio esteso non è $LL(k)$. Si potrebbe però verificare che esso è deterministico.

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

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

Si risponda alle domande seguenti:

(a) Si dimostri che la grammatica G è ambigua, trovando la stringa più corta che ha due alberi sintattici.

(b) Si dimostri che la grammatica non è di tipo $LR(1)$, costruendo l'automa pilota fino a ottenere almeno uno stato inadeguato.

(c) (facoltativa) Con il metodo di Early si analizzi la stringa seguente:

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

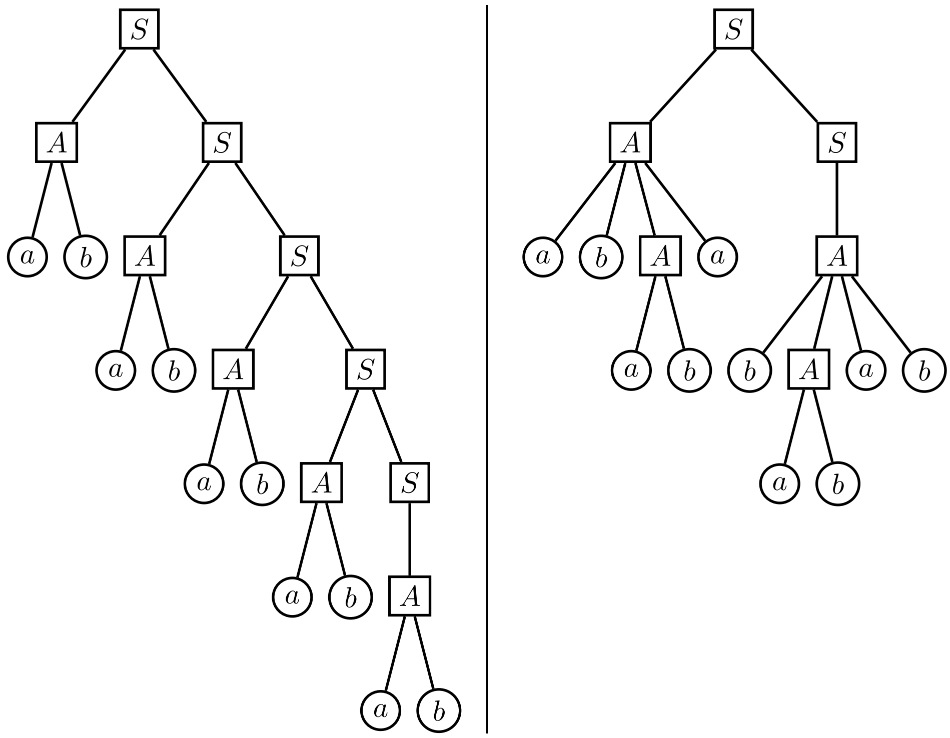
mostrando quali prefissi sono accettati in quanto pure appartenenti al linguaggio (si utilizzi la tabella predisposta di seguito con regole riportate per comodità).

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	pos. a	stato 5	

$S \to A\ S$ $\qquad$ $S \to A$ $\qquad$ $A \to a\ b\ A\ a$ $\qquad$ $A \to b\ A\ a\ b$ $\qquad$ $A \to a\ b$

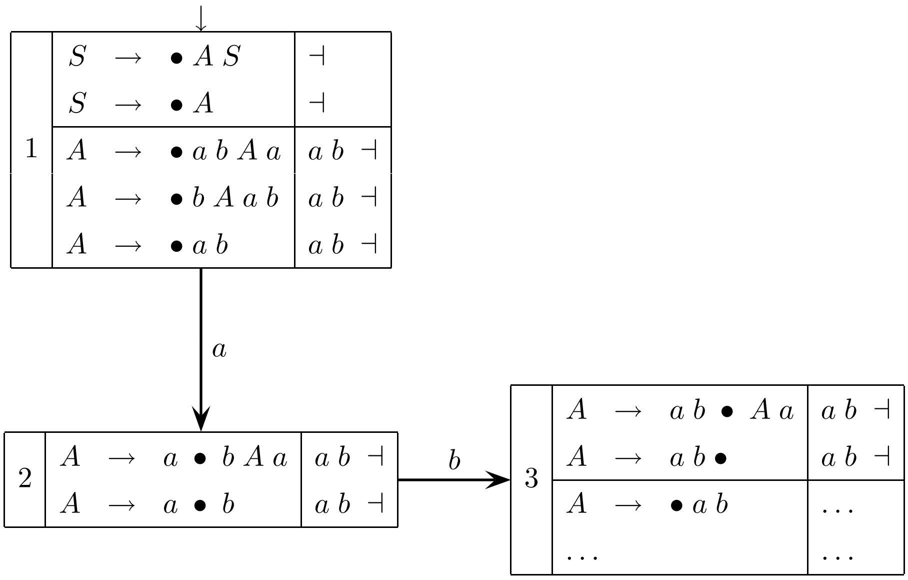
Soluzione

(a) La grammatica è ambigua, come si vede considerando la stringa ambigua più corta $a\,b\,a\,b\,a\,b\,a\,b\,a\,b = (\,a\,b\,)^5$, che ammette i due alberi sintattici seguenti:



I due alberi sono piuttosto diversi e illustrano due meccanismi di formazione della stringa alquanto differenti.

(b) Dal macrostato 1 iniziale dell'automa pilota con due transizioni in sequenza, etichettate a la prima e b la seconda, si arriva al macrostato 3 che presenta un conflitto tra riduzione (candidata $A \to a\ b\ \bullet$ con prospezione a) e spostamento (candidata $A \to \bullet\ a\ b$). Ecco la parte di pilota rilevante allo scopo:



Il macrostato 3 è inadeguato, a motivo della candidata di riduzione $A \to a\ b\ \bullet$ che prospeziona a, e della candidata di spostamento $A \to \bullet\ a\ b$ che legge a, identica alla prospezione.

Ciò corrisponde al fatto che la stringa ambigua $a\,b\,a\,b\,a\,b\,a\,b\,a\,b = (\,a\,b\,)^5$ ammette un albero sintattico dove le prime due lettere $a\,b$ sono le uniche figlie di un nodo A (vedi sopra a sx), e un altro albero dove le stesse due lettere sono le prime due figlie di un nodo A con serie di figli $a\ b\ A\ a$ (vedi sopra a dx).

(c) La stringa $a\,b\,a\,b\,a$ appartiene al linguaggio, come pure i suoi due prefissi $a\,b$ e $a\,b\,a\,b = (\,a\,b\,)^2$. Ecco la simulazione dell'algoritmo di Earley:

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	pos. a	stato 5	
$S \rightarrow \bullet A\, S$	0	$A \rightarrow a \bullet b\, A\, a$	0	$A \rightarrow a\, b \bullet A\, a$	0	$A \rightarrow a \bullet b\, A\, a$	2	$A \rightarrow a\, b \bullet A\, a$	2	$A \rightarrow a\, b\, A\, a \bullet$	0
$S \rightarrow \bullet A$	0	$A \rightarrow a \bullet b$	0	$A \rightarrow a\, b \bullet$	0	$A \rightarrow a \bullet b$	2	$A \rightarrow a\, b \bullet$	2	$A \rightarrow a \bullet b\, A\, a$	4
$A \rightarrow \bullet a\, b\, A\, a$	0			$S \rightarrow A \bullet S$	0			$A \rightarrow a\, b\, A \bullet a$	0	$A \rightarrow a \bullet b$	4
$A \rightarrow \bullet b\, A\, a\, b$	0			$S \rightarrow A \bullet$	0			$S \rightarrow A \bullet S$	2	$S \rightarrow A \bullet S$	0
$A \rightarrow \bullet a\, b$	0			$A \rightarrow \bullet a\, b\, A\, a$	2			$S \rightarrow A \bullet$	2	$S \rightarrow A \bullet$	0
				$A \rightarrow \bullet b\, A\, a\, b$	2			$S \rightarrow A\, S \bullet$	0		
				$A \rightarrow \bullet a\, b$	2			$A \rightarrow \bullet a\, b\, A\, a$	4		
				$S \rightarrow \bullet A\, S$	2			$A \rightarrow \bullet b\, A\, a\, b$	4		
				$S \rightarrow \bullet A$	2			$A \rightarrow \bullet a\, b$	4		
								$S \rightarrow \bullet A\, S$	4		
								$S \rightarrow \bullet A$	4		

$S \rightarrow A\, S$ $\qquad$ $S \rightarrow A$ $\qquad$ $A \rightarrow a\, b\, A\, a$ $\qquad$ $A \rightarrow b\, A\, a\, b$ $\qquad$ $A \rightarrow a\, b$

La candidata di riduzione assiomatica $S \rightarrow A \bullet \mid 0$ nello stato 2 riconosce il prefisso $a\, b$.
La candidata di riduzione assiomatica $S \rightarrow A\, S \bullet \mid 0$ nello stato 4 riconosce il prefisso $a\, b\, a\, b = (\, a\, b\,)^2$.
La candidata di riduzione assiomatica $S \rightarrow A \bullet \mid 0$ nello stato 5 riconosce l'intera stringa $a\, b\, a\, b\, a = (\, a\, b\,)^2\ a$.

4 Traduzione e analisi semantica 20%

1. È dato l'alfabeto $\Sigma = \{a, b, c, d\}$, sia sorgente sia destinazione. Si considerino i linguaggi sorgente e destinazione L_s e L_d seguenti:

$$L_s = a^* \ b^+ \ c^+ \ d \qquad L_d = a^* \ c^+ \ b^+ \ d$$

Su tali linguaggi si definisca la traduzione sintattica τ seguente:

$$\tau \colon L_s \to L_d$$

$$\tau \left(a^p \ b^q \ c^r \ d \right) \mapsto a^p \ c^r \ b^q \ d \qquad p \geq 0 \qquad q, r \geq 1$$

Exempio:

$$\tau \left(a \ a \ b \ b \ b \ b \ c \ c \ d \ \right) = a \ a \ c \ c \ b \ b \ b \ b \ d$$

Si risponda alle domande seguenti:

(a) Si tracci il grafo stato-transizione di un automa trasduttore a pila T che realizza la traduzione τ. Si scelga liberamente se farlo deterministico o indeterministico e se riconoscere a pila vuota o a stato finale.

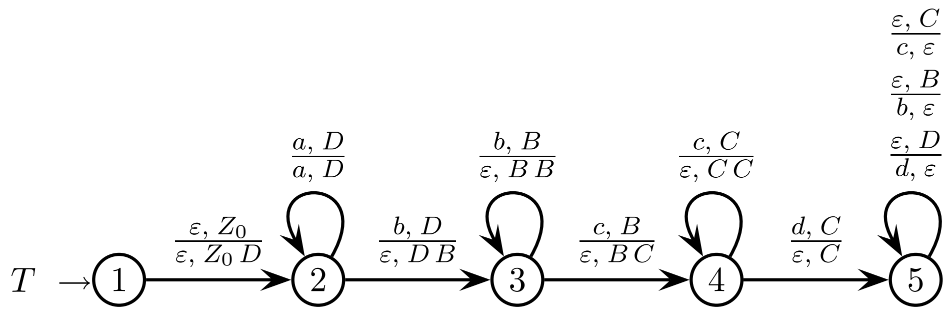
(b) (facoltativa) Si scriva lo schema sintattico di traduzione G_τ (o la grammatica di traduzione) che realizza la traduzione τ. Si suggerisce di completare lo schema sorgente G_s dato di seguito, scrivendo lo schema pozzo (destinazione) G_p.

Schema da completare per la domanda (b) facoltativa:

$$G_s \begin{cases} S \rightarrow A \ d \\ A \rightarrow a \ A \\ A \rightarrow b \ X \ c \\ X \rightarrow b \ X \ c \\ X \rightarrow \varepsilon \\ X \rightarrow b \ B \\ X \rightarrow C \ c \\ B \rightarrow b \ B \\ B \rightarrow \varepsilon \\ C \rightarrow C \ c \\ C \rightarrow \varepsilon \end{cases} \qquad G_p \begin{cases} S \rightarrow \\ A \rightarrow \\ A \rightarrow \\ X \rightarrow \\ X \rightarrow \\ X \rightarrow \\ X \rightarrow \\ B \rightarrow \\ B \rightarrow \\ C \rightarrow \\ C \rightarrow \end{cases}$$

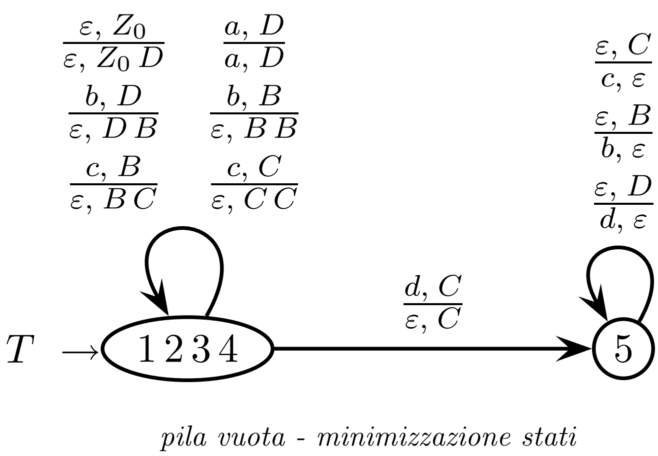
Soluzione

(a) Il trasduttore a pila T è facile da progettare intuitivamente in modo deterministico. La semplice idea iniziale è d'impilare le lettere da scambiare (b e c) e di spilarle in ordine riflesso. Si suppone pertanto di riconoscere a pila vuota. I simboli di memoria B, C e D codificano le lettere d'ingresso b, c e d, rispettivamente. Il trasduttore funziona così: impila una lettera d a futura memoria; legge ed emette le lettere a (se ce ne sono), senza usare la pila; legge e impila le lettere b (almeno una); legge e impila le lettere c (almeno una); legge la lettera d, senza usare la pila; e infine spila ed emette le lettere c, b e la lettera d. Eccone il grafo stato-transizione:



riconoscimento a pila vuota - versione iniziale

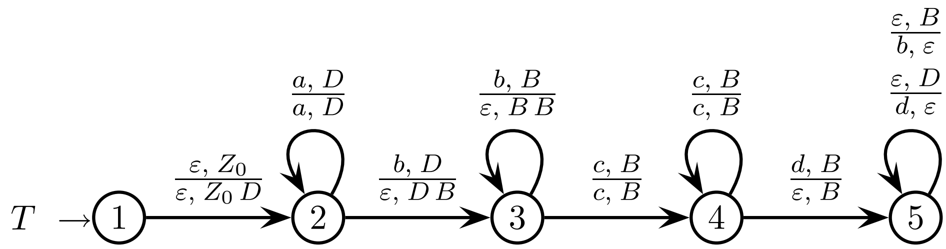
Svuotamento della pila e pertanto riconoscimento e convalida della traduzione emessa avvengono nello stato 5 quando scatta la mossa che spila il simbolo D. Gli stati finiti del trasduttore deterministico T non sono tutti indispensabili, in alcuni passaggi potendosi usare il simbolo in cima alla pila per distinguere tra mosse ammissibili e non. In pratica il simbolo in cima alla pila codifica lo stato. Infatti i simboli Z_0, D, B e C codificano gli stati 1, 2, 3 e 4, rispettivamente; si guardino le transizioni uscenti da tali stati (autoanelli compresi). Ecco pertanto una versione deterministica di T, sempre con riconoscimento a pila vuota ma meno stati della versione precedente, ossia con minimizzazione degli stati:



pila vuota - minimizzazione stati

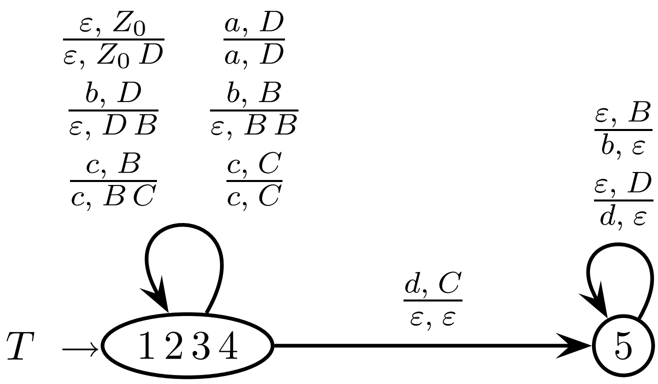
In sostanza gli stati 1, 2, 3 e 4 sono raggruppati in uno solo (costruzione simile ai sottinsiemi), perché come detto prima il simbolo in cima alla pila permette comunque di distinguerli.

Inoltre si può ottimizzare l'uso di memoria, facendo crescere meno la pila. In concreto si osserva che non è necessario impilare le lettere c, ma basta leggerle ed emetterle. Nella versione con più stati si può allora eliminare il simbolo C, come segue:



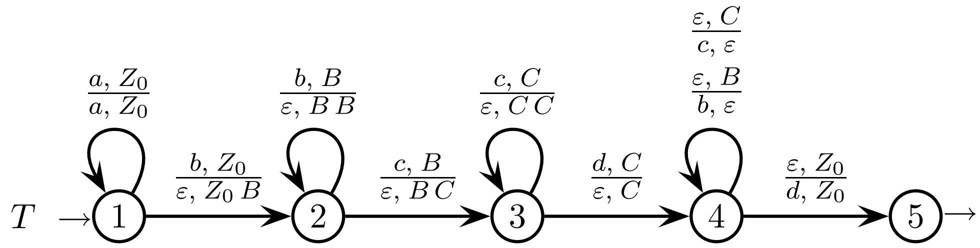
pila vuota - ottimizzazione memoria

Si possono combinare minimizzazione degli stati e ottimizzazione della memoria. Tuttavia nella versione con minimizzazione degli stati non si può eliminare del tutto il simbolo C, in quanto serve per codificare lo ex-stato 4; ora però C non serve più come contatore delle lettere c e se ne usa una sola copia. Ecco il trasduttore:



pila vuota - minimizzazione stati e ottimizzazione memoria

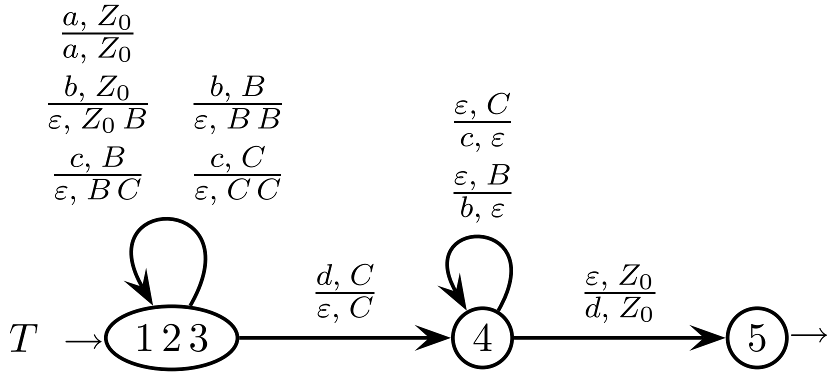
Naturalmente si può sempre riconoscere a stato finale. Si rimane deterministici e si elimina il simbolo D. Ecco il trasduttore iniziale (senza minimizzazione stati e ottimizzazione memoria), modificato così da funzionare a stato finale:



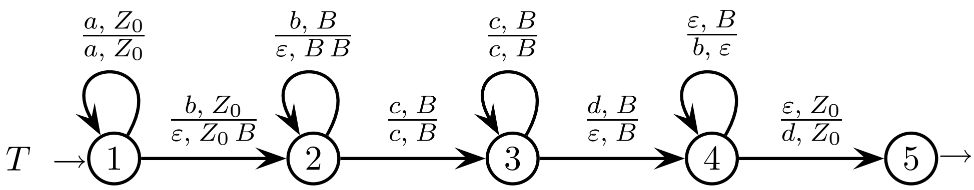
riconoscimento a stato finale - versione iniziale

Si noti che la pila si svuota già nello stato 4, ma riconoscimento e dunque convalida della traduzione emessa avvengono nello stato finale 5.

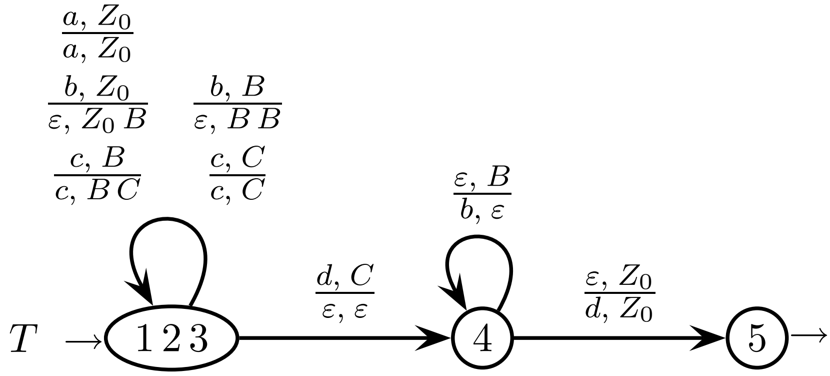
Si possono ancora minimizzare gli stati (raggruppando 1, 2 e 3) od ottimizzare la memoria (eliminando il simbolo C), o entrambe le trasformazioni. Eccole in serie, senza commento (il lettore le esamini da sé):



stato finale - minimizzazione stati



stato finale - ottimizzazione memoria



stato finale - minimizzazione stati e ottimizzazione memoria

Le soluzioni deterministiche testé illustrate sono tutte ugualmente valide ai fini del problema proposto, a prescindere dal grado di ottimizzazione.

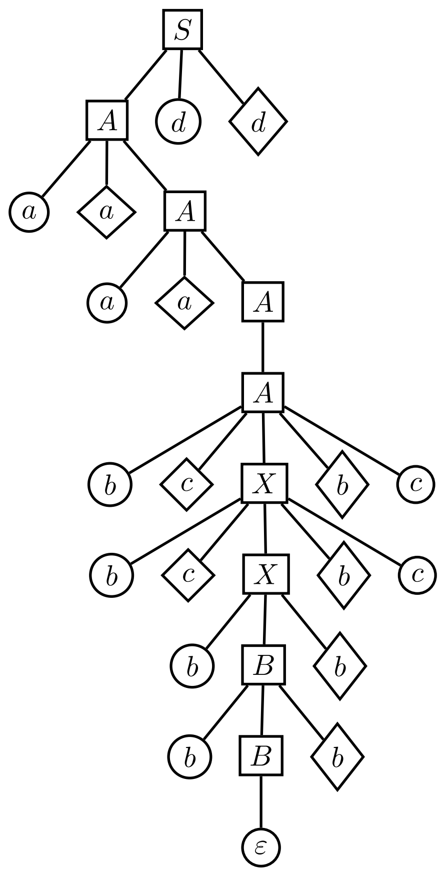
Invece non sembra conveniente progettare in modo indeterministico. Il trasduttore a pila vuota con stati minimizzati ridotti a due soli e memoria ottimizzata con alfabeto di pila di tre soli simboli (oltre a Z_0), è già piuttosto compatto. Sembra ben difficile usare meno simboli di pila e ridurre gli stati a uno solo. Si immagini che cosa succederebbe fondendo i due stati di tale macchina in uno solo: essa tradurrebbe stringhe d'ingresso con alternanza di lettere b e c.

Beninteso si potrebbe prendere la grammatica di traduzione che risponde alla domanda (b) e trasformarla in automa trasduttore a pila con un solo stato, tramite la nota costruzione, così rispondendo alla domanda (a) quasi senza sforzo. Qui si otterrebbe un automa indeterministico perfettamente funzionale, seppure abbastanza complesso e con parecchie mosse. Si lascia l'esercizio al lettore (oppure si veda dopo la domanda (b)).

(b) Ecco la grammatica G_τ (assioma S) che realizza la traduzione τ, con a lato l'albero sintattico della stringa di esempio proposta:

$$
G_\tau \left\{
\begin{array}{lcl}
S & \to & A\ d\ \{d\} \\
A & \to & a\ \{a\}\ A \\
A & \to & b\ \{c\}\ X\ \{b\}\ c \\
X & \to & b\ \{c\}\ X\ \{b\}\ c \\
X & \to & \varepsilon \\
X & \to & b\ B\ \{b\} \\
X & \to & \{c\}\ C\ c \\
B & \to & b\ B\ \{b\} \\
B & \to & \varepsilon \\
C & \to & \{c\}\ C\ c \\
C & \to & \varepsilon
\end{array}
\right.
$$

grammatica di traduzione



albero sintattico di traduzione

Le regole lineari $X, B \to b\ B\ \{b\}$ e $X, C \to \{c\}\ C\ c$ sono equivalenti a $X, B \to b\ \{b\}\ B$ e $X, C \to c\ \{c\}\ C$, sebbene scritte così siano meno simmetriche.

L'albero sintattico si riferisce a un caso con lettere b più numerose delle lettere c, ma è immediato immaginare il caso opposto e quello di uguaglianza.

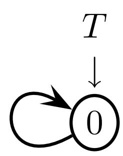
La grammatica sorgente G_s non è ambigua, dunque neppure lo schema completo. Separando le parti sorgente e destinazione si ottiene lo schema sintattico.

Per completezza qui si dà anche il trasduttore T indeterministico con un solo stato, derivato dalla grammatica di traduzione G_τ tramite la nota costruzione. Prima si riscrive G_τ in una grammatica G'_τ equivalente. A tale fine si aggiungono nuovi nonterminali dove serve per avere regole di forma $Z \to s\ \{p\}\ \zeta$, dove s e p sono terminali (sorgente e pozzo rispettivamente) o si riducono a ε, e ζ contiene solo nonterminali o si riduce a ε. Tale forma di regola è immediatamente riscrivibile in mossa come $\frac{s,\ Z}{p,\ \zeta^R}$.

Ecco la grammatica di traduzione G'_τ modificata e a lato il trasduttore T corrispondente, che riconosce a pila vuota e ha un solo stato 0 (ogni mossa è un autoanello):

G'_τ

$S \to A\ D$	$\frac{\varepsilon,\ Z_0}{\varepsilon,\ Z_0\ D\ A}$
$A \to a\ \{a\}\ A \mid b\ \{c\}\ X\ Y$	$\frac{a,\ A}{a,\ A}$ $\frac{b,\ A}{c,\ Y\ X}$
$X \to b\ \{c\}\ X\ Y \mid \varepsilon$	$\frac{b,\ X}{c,\ Y\ X}$ $\frac{\varepsilon,\ X}{\varepsilon,\ \varepsilon}$
$X \to b\ \{b\}\ B \mid c\ \{c\}\ C$	$\frac{b,\ X}{b,\ B}$ $\frac{c,\ X}{c,\ C}$
$B \to b\ \{b\}\ B \mid \varepsilon$	$\frac{b,\ B}{b,\ B}$ $\frac{\varepsilon,\ B}{\varepsilon,\ \varepsilon}$
$C \to c\ \{c\}\ C \mid \varepsilon$	$\frac{c,\ C}{c,\ C}$ $\frac{\varepsilon,\ C}{\varepsilon,\ \varepsilon}$
$Y \to c\ \{b\}$	$\frac{c,\ Y}{b,\ \varepsilon}$
$D \to d\ \{d\}$	$\frac{d,\ D}{d,\ \varepsilon}$



grammatica di traduzione *trasduttore indeterministico*

In G'_τ sono aggiunti i nonterminali D e Y, figurano le regole lineari modificate indicate prima e la regola $Y \to c\ \{b\}$ invece di $Y \to \{b\}\ c$ (sono equivalenti). Le transizioni del trasduttore T sono raggruppate per alternative, come le regole di G'_τ.

Il trasduttore T è pesantemente indeterministico. Il punto cruciale è la mossa $\frac{b,\ X}{c,\ Y\ X}$, l'unica che possa fare crescere la pila illimitatamente (secondo il numero di b in ingresso). Tale mossa, avendo X in cima alla pila, indeterministicamente legge una lettera b, emette subito una lettera c senza ancora sapere quante saranno le lettere c da leggere (nell'ingresso esse si trovano dopo le lettere b), e impila una copia di Y a memoria futura (inoltre tiene sempre una sola copia di X in cima alla pila). In seguito la mossa $\frac{c,\ Y}{b,\ \varepsilon}$ (l'unica capace di spilare Y), avendo Y in cima alla pila, legge c, emette b e spila una copia di Y. Chiaramente la pila si svuota solo se tutte le copie di Y vengono spilate, ossia se le emissioni di c anticipate dalla mossa $\frac{b,\ X}{c,\ Y\ X}$ coincidono in numero con le letture di c posteriori effettuate dalla mossa $\frac{c,\ Y}{b,\ \varepsilon}$. Pertanto il trasduttore T deve avere "indovinato" anticipatamente (ossia indeterministicamente) il numero corretto di esecuzioni della mossa $\frac{b,\ X}{c,\ Y\ X}$.

Per esempio, si consideri il caso con nessuna a (per semplicità) e una lettera b in più rispetto alle lettere c, ossia una stringa d'ingresso di tipo $b^{n+1}\, c^n\, d$ ($n \geq 1$), che si traduce in $c^n\, b^{n+1}\, d$. Il trasduttore T lavora nel modo seguente:

ing.	ε	b	b^{n-1}	b	ε	c^n	d
pila	Z_0	$Z_0\, D\, A$	$Z_0\, D\, Y\, X$	$Z_0\, D\, Y^n\, X$	$Z_0\, D\, Y^n\, B$	$Z_0\, D\, Y^n$	$Z_0\, D$
mos.	$\frac{\varepsilon,\ Z_0}{\varepsilon,\ Z_0\, D\, A}$	$\frac{b,\ A}{c,\ Y\, X}$	$\left(\frac{b,\ X}{c,\ Y\, X}\right)^{n-1}$	$\frac{b,\ X}{b,\ B}$	$\frac{\varepsilon,\ B}{\varepsilon,\ \varepsilon}$	$\left(\frac{c,\ Y}{b,\ \varepsilon}\right)^{n}$	$\frac{d,\ D}{d,\ \varepsilon}$
usc.	ε	c	c^{n-1}	b	ε	b^n	d
pila	$Z_0\, D\, A$	$Z_0\, D\, Y\, X$	$Z_0\, D\, Y^n\, X$	$Z_0\, D\, Y^n\, B$	$Z_0\, D\, Y^n$	$Z_0\, D$	Z_0

Dalla simulazione si vede che il trasduttore T ha dovuto "indovinare" che la scelta corretta è di smettere di emettere lettere c (e di passare a emettere lettere b) in corrispondenza dell'ultima lettera b in ingresso. Ma tale scelta indeterministica è motivabile solo sapendo già che il numero di lettere c in ingresso è proprio pari al numero di lettere b meno una. Tuttavia la conferma di tale scelta arriva solo in seguito, quando le lettere c in ingresso vengono lette e la pila si svuota. Similmente se le lettere c eccedono le lettere b o se le uguagliano in numero.

Questo è un comportamento fortemente indeterministico e rappresenta il prezzo da pagare per avere un trasduttore con un solo stato. Il trasduttore T ha anche altre lievi forme di indeterminismo, poco rilevanti.

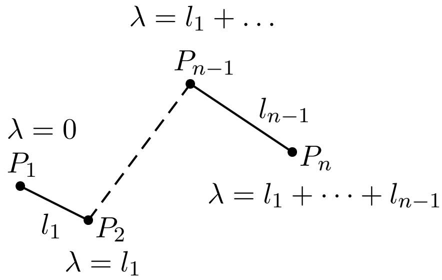
2. Una linea spezzata connessa è descritta dai suoi n punti (con $n \geq 2$):

$$P_1 = (\ x_1,\ y_1) \quad P_2 = (\ x_2,\ y_2\ ) \quad \ldots \quad P_n = (\ x_n,\ y_n\ )$$

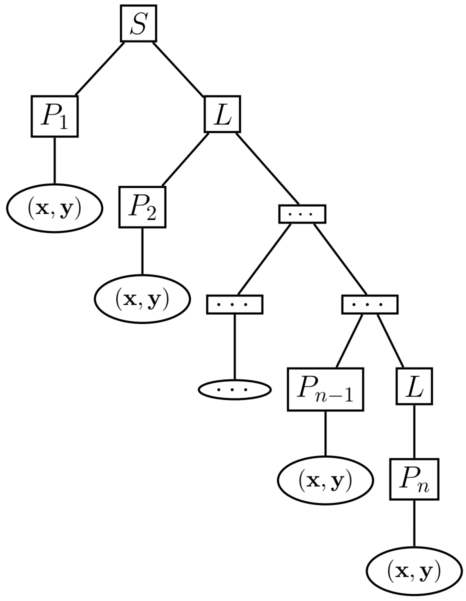
L'elenco dei punti è generato dalla sintassi G seguente (assioma S):

$$G \begin{cases} 1: & S \rightarrow P\ L \\ 2: & L \rightarrow P\ L \\ 3: & L \rightarrow P \\ 4: & P \rightarrow (\ \mathbf{x},\ \mathbf{y}\ ) \end{cases}$$

supporto sintattico



linea spezzata



albero sintattico

Si supponga che i punti generati siano numerati così: $S \stackrel{*}{\Rightarrow} P_1\ P_2\ \ldots\ P_{n-1}\ P_n$; come mostra anche l'albero sintattico a lato.

Si supponga di avere due attributi lessicali di tipo intero α e ω (ascissa e ordinata), sinistri, associati al nonterminale P, precalcolati (ossia immediatamente disponibili) contenenti il valore dei terminali $\mathbf{x}$ e $\mathbf{y}$, rispettivamente.

Si risponda alle domande seguenti:

(a) Si scrivano nell'apposita tabella (data a pagina seguente) le regole semantiche di una grammatica con attributi per avere nel generico nonterminale P associato allo i-esimo punto P_i ($1 \leq i \leq n$), la lunghezza λ (numero di tipo reale) della spezzata $P_1,\ P_2,\ \ldots,\ P_i$ (si veda la figura sopra); per definizione in P_1 si ha $\lambda = 0$. Se necessario si introducano altri attributi, specificandone il tipo.

(b) (facoltativa) Si verifichi se la grammatica con attributi progettata al punto precedente è di tipo a una scansione (one sweep).

sintassi	*funzioni semantiche*
1: $S_0 \to P_1\ L_2$	
2: $L_0 \to P_1\ L_2$	
3: $L_0 \to P_1$	
4: $P_0 \to (\ \mathbf{x},\ \mathbf{y}\ )$	

Soluzione

(a) Il calcolo della lunghezza della spezzata procede lungo l'albero sintattico dalla radice alle foglie, come implica la numerazione dei nodi P. Ne viene che l'attributo λ indicato nel testo dev'essere di tipo destro (ereditato). Il calcolo effettivo di λ è svolto in associazione con il nonterminale L, poi λ viene passato al nonterminale P. In aggiunta ai due attributi lessicali α e ω, per passare le coordinate dei punti un livello più sotto nell'albero occorrono due attributi ausiliari μ e ν, con lo stesso significato di α e ω, rispettivamente.

Pertanto si supponga l'attributo λ di tipo destro, associato ai nonterminali L e P. Per calcolare la lunghezza del segmento i-esimo $P_i - P_{i+1}$ $(1 \leq i \leq n-1)$, la coppia di attributi lessicali sinistri α e ω del punto P_i viene memorizzata in una coppia di attributi destri di L denominati μ e ν. Quivi una nota formula calcola la lunghezza del segmento di coordinate $(\ \alpha,\ \omega\ )$, $(\ \mu,\ \nu\ )$, dove α e β si riferiscono al punto $(i+1)$-esimo, e μ e ν al punto i-esimo; nel punto n-esimo il calcolo è effettuato direttamente in P. La lunghezza del segmento i-esimo viene aggiunta alla lunghezza λ della spezzata $P_1,\ P_2,\ \ldots,\ P_i$ per ottenere il valore di λ nel punto P_{i+1}, ossia la lunghezza della spezzata $P_1,\ P_2,\ \ldots,\ P_i,\ P_{i+1}$.

Riassumendo gli attributi:

- λ è destro, reale, associato a L e P
- μ, ν sono destri, interi, associati a L
- α, ω sono sinistri, interi, associati a P, funzione dei terminali **x** e **y**

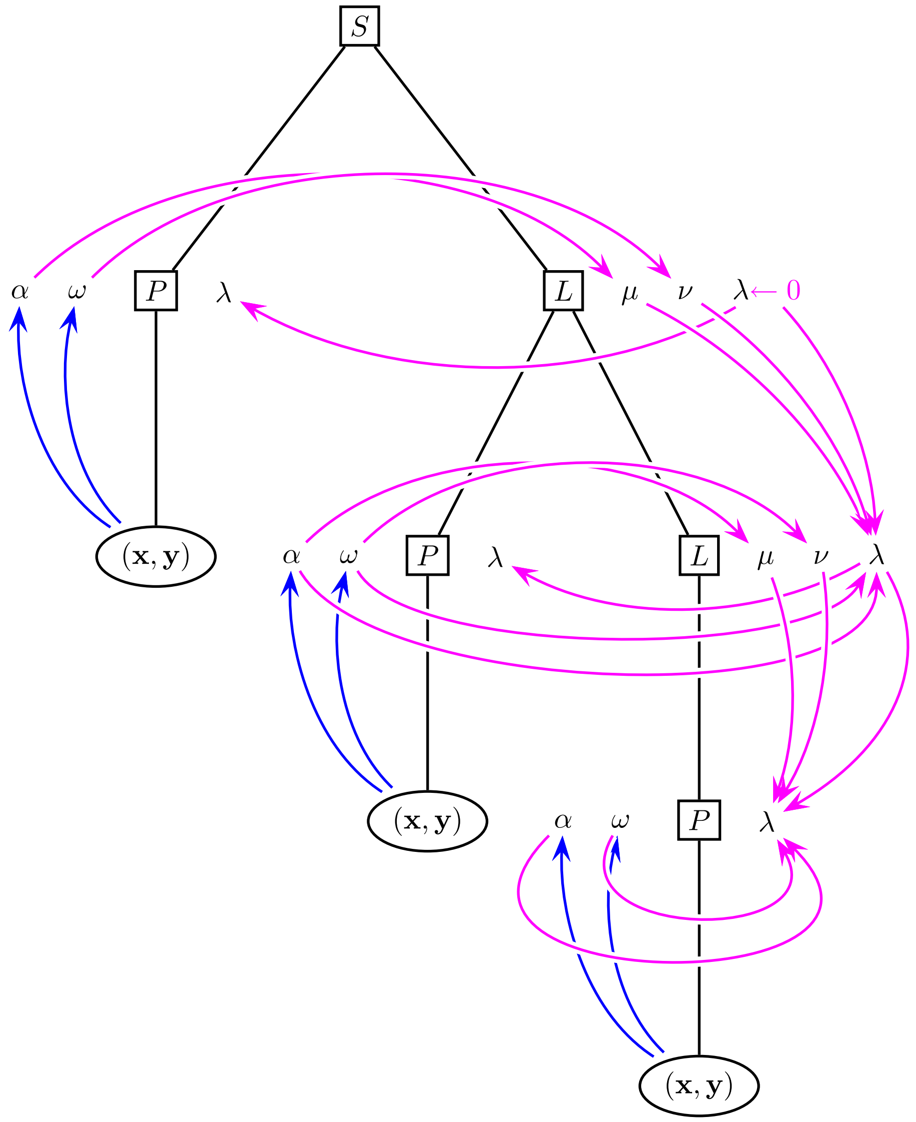
Ecco la grammatica con attributi:

	sintassi	*funzioni semantiche*
1:	$S_0 \to P_1\ L_2$	$\lambda_2 = 0$ $\lambda_1 = \lambda_2$ $\mu_2 = \alpha_1$ $\nu_2 = \omega_1$
2:	$L_0 \to P_1\ L_2$	$\lambda_2 = \lambda_0 + \sqrt{(\ \alpha_1 - \mu_0\ )^2 + (\ \omega_1 - \nu_0\ )^2}$ $\lambda_1 = \lambda_2$ $\mu_2 = \alpha_1$ $\nu_2 = \omega_1$
3:	$L_0 \to P_1$	$\lambda_1 = \lambda_0 + \sqrt{(\ \alpha_1 - \mu_0\ )^2 + (\ \omega_1 - \nu_0\ )^2}$
4:	$P_0 \to (\ \mathbf{x},\ \mathbf{y}\ )$	$\alpha_0 =$ valore di $\mathbf{x}$ $\omega_0 =$ valore di $\mathbf{y}$

Beninteso ci sono varianti. Per esempio il calcolo della distanza tra due punti potrebbe essere fatto in P e poi passato a L; il resto non cambia. E forse ci sono anche altre soluzioni, più diversificate.

È anche possibile effettuare il calcolo in modo sintetizzato, visitando l'albero bottom-up (e allora λ è di tipo sinistro). Tuttavia così facendo la lunghezza della spezzata completa viene associata al punto P_1, non al punto P_n, cambiando (sia pure formalmente non sostanzialmente) l'enunciato del problema proposto.

(b) Ecco le dipendenze funzionali della grammatica con attributi, qui esemplificate decorando un albero sintattico a quattro livelli che definisce una spezzata di tre punti, cioè con due segmenti:



albero sintattico decorato con dipendenze funzionali

Intanto si vede che la grammatica con attributi è aciclica, dunque corretta. Pertanto il calcolo è fattibile visitando i nodi in ordine topologico.

Dato che gli attributi lessicali α e ω sono disponibili (ossia precalcolati), la grammatica con attributi è a una scansione (one sweep). La valutazione procede top-down dalla radice S al punto P_n, e l'ordine di valutazione degli attributi è il seguente: μ, ν e λ del figlio L, poi λ del figlio P. Chiaramente nella regola $2\colon L \rightarrow P\ L$ l'ordine di valutazione dei nodi è il seguente: prima il figlio L (μ, ν e λ), poi il figlio P (λ); ed è diverso da quello sintattico left-to-right.