Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 20 giugno 2011 - 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:

MATRICOLA: FIRMA:

ISTRUZIONI - LEGGERE CON ATTENZIONE:

- L'esame si compone di due parti:
  - I (80%) Teoria:
    1. espressioni regolari e automi finiti
    2. grammatiche libere e automi a pila
    3. analisi sintattica e parsificatori
    4. traduzione sintattica e analisi semantica
  - II (20%) Esercitazioni Flex e Bison
- Per superare l'esame l'allievo deve sostenere con successo entrambe le parti (I e II), in un solo appello oppure in appelli diversi, ma entro un anno.
- Per superare la parte I (teoria) occorre dimostrare di possedere conoscenza sufficiente di tutte le quattro sezioni (1-4), rispondendo alle domande obbligatorie.
- È permesso consultare libri e appunti personali.
- Per scrivere si utilizzi lo spazio libero e se occorre anche il tergo del foglio; è vietato allegare nuovi fogli o sostituirne di esistenti.
- Tempo: Parte I (teoria): 2h.30m - Parte II (esercitazioni): 45m

1 Espressioni regolari e automi finiti 20%

1. È data l'espressione regolare R seguente:

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

Si risponda alle domande seguenti:

(a) Si scrivano la stringa più breve ambigua e quella più breve non ambigua del linguaggio $L(R)$.

(b) Usando il metodo Berri-Sethi (BS), si ricavi un automa deterministico A che accetta il linguaggio $L(R)$.

(c) (facoltativa) Si minimizzi il numero degli stati dell'automa A.

Soluzione

(a) Ciascuna croce deve generare almeno una lettera. Dunque le stringhe più brevi del linguaggio sono quattro e hanno lunghezza due:

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

Tutte sono non ambigue, come si verifica numerando l'espressione così:

$$(a_1 \mid b_2)^+ \ (b_3 \mid c_4)^+ \ (a_5 \mid c_6)^*$$

e osservando che:

$a_1\ b_3 \qquad a_1\ c_4 \qquad b_2\ b_3 \qquad b_2\ c_4$

Dunque una stringa ambigua deve avere lunghezza almeno tre. Una stringa ambigua ovvia è $b\ b\ b$, dove la prima croce genera la prima e la seconda lettera b e la seconda croce genera la terza b, oppure la prima croce genera la prima b e la seconda croce genera la seconda e la terza b, ossia:

$b_2\ b_2\ b_3 \qquad b_2\ b_3\ b_3$

Ci sono anche altre stringhe ambigue di lunghezza tre, per esempio $a\ c\ c$, e più ancora crescendo in lunghezza.

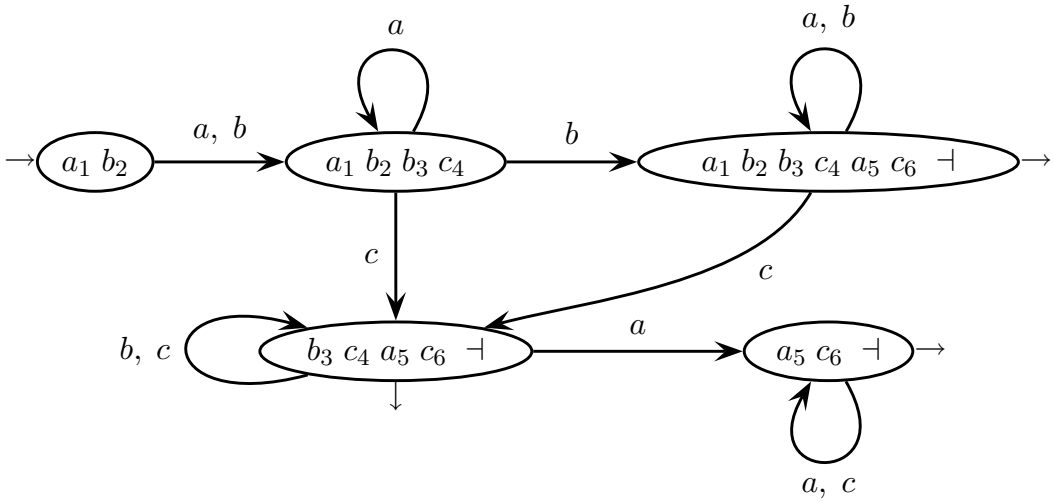
(b) Ecco l'espressione numerata $R_\#$:

$$R_\# = (a_1 \mid b_2)^+ \ (b_3 \mid c_4)^+ \ (a_5 \mid c_6)^*$$

Inizi e seguiti sono i seguenti:

inizi	$a_1\ b_2$
generatore	seguiti
a_1	$a_1\ b_2\ b_3\ c_4$
b_2	$a_1\ b_2\ b_3\ c_4$
b_3	$b_3\ c_4\ a_5\ c_6\ \dashv$
c_4	$b_3\ c_4\ a_5\ c_6\ \dashv$
a_5	$a_5\ c_6\ \dashv$
c_6	$a_5\ c_6\ \dashv$

Ecco l'automa deterministico di Berri-Sethi:



(c) L'automa deterministico è in forma ridotta (stati tutti utili) e minima. I due stati non finali sono distinguibili poiché quello iniziale manca di arco uscente c. I tre stati finali sono distinguibili poiché lo stato $a_5\ c_6\ \dashv$ manca di arco uscente b e i due rimanenti vanno con a in stati distinguibili.

2. Si consideri l'espressione regolare R seguente:

$$R = a^* \ (\ b \ c \ | \ c \ )^+$$

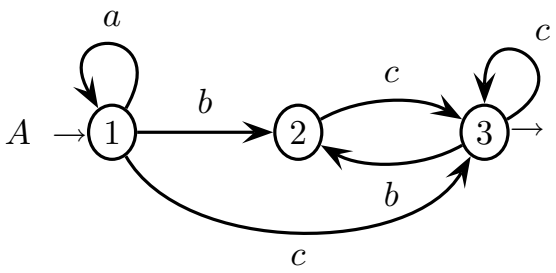
Si risponda alle domande seguenti:

(a) Si disegni l'automa deterministico A' che riconosce il linguaggio complemento di $L(R)$, eliminando eventuali stati inutili (ossia irraggiungibili o indefiniti).

(b) (facoltativa) Si argomenti se il linguaggio generato dall'espressione R è locale o no, e se è locale o no il linguaggio complemento di $L(R)$.

Soluzione

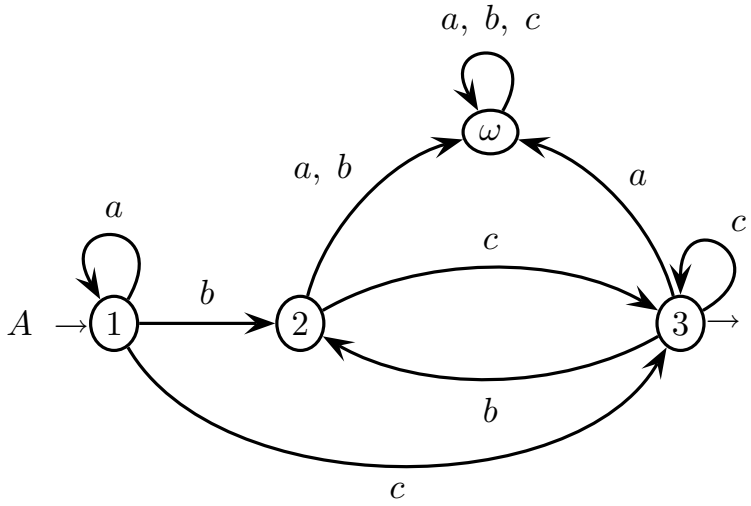
(a) Ecco l'automa A, già deterministico, disegnato intuitivamente:



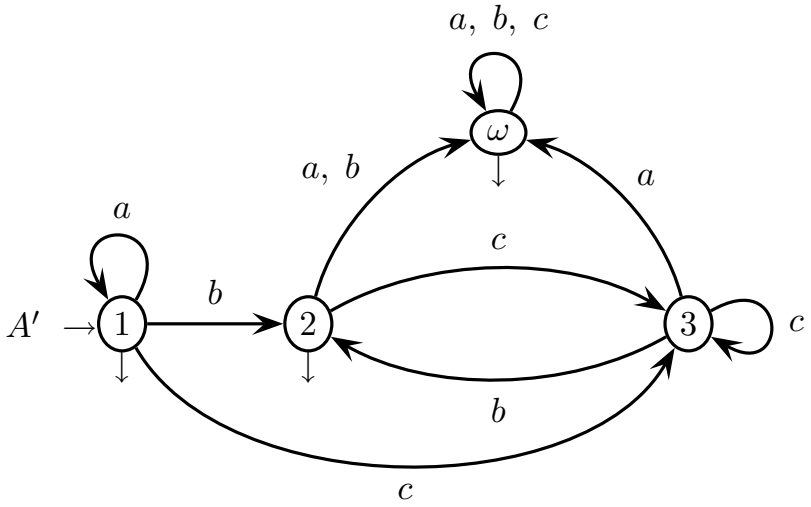
Si potrebbe ricavare A sistematicamente tramite il metodo di Berri-Sethi.

Benché non richiesto, si verifica che l'automa A è minimo. Basta ragionale così: intanto si vede che A non ha stati inutili (tutti sono raggiungibili e definiti); lo stato 3 è finale e distinguibile dagli stati 1 e 2, non finali; lo stato 1 ha l'arco uscente a e lo stato 2 non lo ha, dunque 1 e 2 sono distinguibili. I tre stati sono allora distinguibili e l'automa è minimo.

Per disegnare l'automa complemento A', bisogna prima completare l'automa deterministico A aggiungendogli lo stato d'errore ω, così:

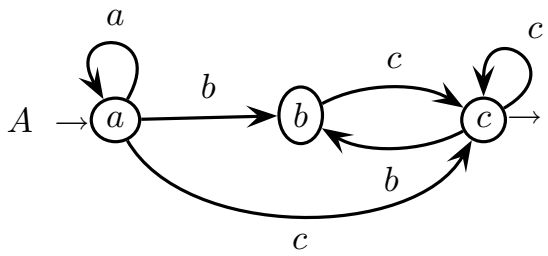


Ora si può complementare, così:



Gli stati di A' sono tutti utili: raggiungibili e definiti. Benché non richiesto, si verifica che A' è minimo: lo stato non finale 3 è distinguibile dagli stati finali 1, 2 e ω; 1 e 2 hanno arco c diretto verso lo stato non finale 3, mentre da ω con c si resta in ω, dunque 1 e 2 sono distinguibili da ω; da 1 con a si resta in 1, mentre da 2 con a si va in ω che è distinguibile da 1, dunque 1 e 2 sono distinguibili. I quattro stati sono allora distinguibili e l'automa A' è minimo.

(b) Il linguaggio $L(R)$ è locale. Basta osservare che nell'automa A in ogni stato entrano archi con la medesima etichetta, diversa per ogni stato, così:



Una risposta un po' intuitiva ma più breve è questa: il linguaggio è locale poiché è definito dall'espressione regolare lineare (ossia dove ogni lettera compare una sola volta) R_l seguente, equivalente a R:

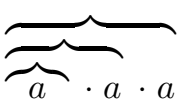
$$R_l = a^* \left(\left(b \mid \varepsilon \right) c \right)^*$$

Il linguaggio complemento non è locale. Guardando l'automa A', si vede che la stringa $b\,c$ è rifiutata; tuttavia ci sono stringhe accettate che contengono il digramma $b\,c$ e che iniziano e finiscono come la stringa rifiutata: per esempio la stringa $b\,b\,c$ è accettata. Pertanto il linguaggio non è formulabile in termini di digrammi e dunque non è locale.

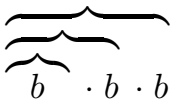
2 Grammatiche libere e automi a pila 20%

1. Si consideri il linguaggio generato dall'espressione regolare $a^+ \ (\ a \ | \ b \ )^*$. Si noti che si possono scrivere le frasi del linguaggio come $a^{i_1} \ b^{i_2} \ a^{i_3} \ \ldots$ ($k \geq 1$ e $i_k \geq 1$), ossia come serie di gruppi di lettere a e b alternati. La struttura sintattica da assegnare alle frasi del linguaggio è specificata nel modo seguente:

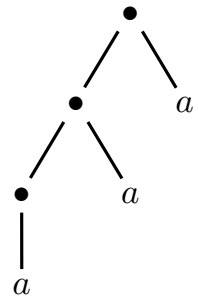
   - per ogni gruppo a^+ e per ogni gruppo b^+ l'albero sintattico è lineare a sinistra, ossia ha la forma seguente:



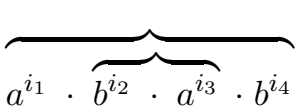
e

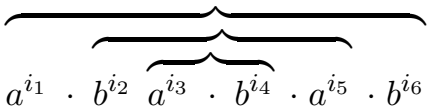


p. es.

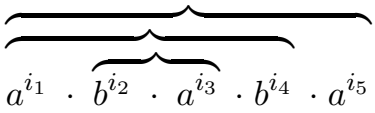


   - la struttura delle frasi che contengono un numero pari di gruppi a^+ e b^+ alternati è bilanciata, come sotto mostrato:





   - se l'ultimo gruppo è del tipo a^+ e dunque i gruppi sono in numero dispari, esso va aggiunto all'albero bilanciato, così:



   Si risponda alle domande seguenti:

   (a) Si scriva una grammatica BNF (non estesa) che assegna alle frasi la struttura specificata; si disegni l'albero sintattico della frase campione seguente:

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

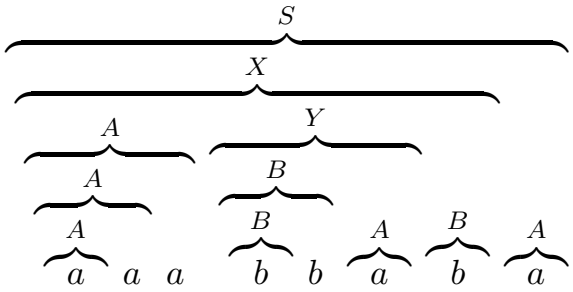
   (b) (facoltativa) Si verifichi che la grammatica non sia ambigua.

Soluzione

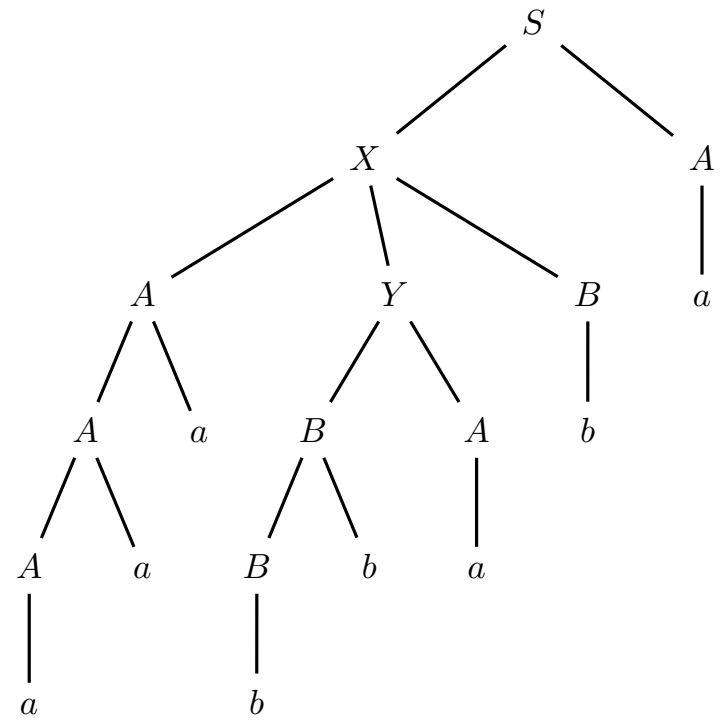
(a) Ecco la grammatica richiesta, conforme alle specifiche (assioma S):

$$\begin{cases} S & \rightarrow & X\ A \ \mid\ X \ \mid\ A \\ X & \rightarrow & A\ Y\ B \ \mid\ A\ B \\ Y & \rightarrow & B\ X\ A \ \mid\ B\ A \\ A & \rightarrow & A\ a \ \mid\ a \\ B & \rightarrow & B\ b \ \mid\ b \end{cases}$$

Ed ecco l'albero sintattico, in forma parentetica, della frase campione:



Eccone invece la forma grafica, per la stessa frase campione:



Gli esempi illustrano sufficientemente il meccanismo sintattico che genera le frasi, conforme alle specifiche date.

(b) La grammatica non è ambigua. Per dimostrarlo rigorosamente, un modo è tentare la costruzione di un analizzatore sintattico deterministico. Essendo ricorsiva a sinistra, la grammatica non è LL. Si potrebbe costruire il grafo pilota LR e verificare se sia deterministico, ma la grammatica ha numerose regole, la costruzione è allora lunga e l'esito imprevedibile.

Alternativamente si può dimostrare che ogni stringa generata dalla grammatica ammette una sola derivazione ossia un solo albero sintattico. Ecco una traccia di dimostrazione sufficientemente analitica e completa.

Le regole $A \to A\ a \mid a$ e analogamente per il nonterminale B, sono ricorsive a sinistra e non danno luogo ad ambiguità poiché l'albero di una stringa come a^n ($n \geq 1$) è lineare e dunque unico per la stringa; similmente per b^n.

Esaminando le regole che espandono il nonterminale X, si vede subito che le stringhe generate da X iniziano e terminano con le lettere b e a, rispettivamente. Ne segue che le tre regole assiomatiche $S \to X\ A$, $S \to X$ e $S \to A$ non danno luogo ad ambiguità d'unione, poiché non generano frasi comuni dato che queste iniziano o terminano con lettere diverse per ciascuna regola.

Se la regola binaria $S \to X\ A$ presa come iniziale desse luogo ad ambiguità, questa potrebbe essere dovuta al concatenamento tra X e A, al solo A o al solo X. Ecco i tre casi discussi separatamente:

- Caso $X\ A$: Non si ha ambiguità di concatenamento tra X e A. Infatti il nonterminale X genera stringhe che terminano con la lettera b e il nonterminale A che lo segue genera solo lettere a, una o più.
- Caso A: Il nonterminale A preso da solo non genera stringhe ambiguamente, come detto prima. Vale pure per la regola di copia $S \to A$ presa come iniziale.
- Caso X: Il nonterminale X preso da solo non genera stringhe ambiguamente. Vale pure per la regola di copia $S \to X$ presa come iniziale. Le regole messe in gioco e i linguaggi generati dai nonterminali coinvolti sono così (ricordare che $L(A) = a^+$ e $L(B) = b^+$ e che questi linguaggi presi isolatamente sono generati in modo non ambiguo):

$$\begin{aligned} 1\colon &\quad X \to a^+\ Y\ b^+ \\ 2\colon &\quad X \to a^+\ b^+ \\ 3\colon &\quad Y \to b^+\ X\ a^+ \\ 4\colon &\quad Y \to b^+\ a^+ \end{aligned}$$

  Il linguaggio $L(X)$ (come anche $L(Y)$) è generato da regole autoinclusive che si alternano nella derivazione: 1, 3, 1, 3, ..., 2 o 4. Ciascuna regola aggiunge un fattore a^+ e un fattore b^+ alternandoli. Poiché gli alfabeti di tali fattori sono disgiunti, non c'è ambiguità di concatenamento in $a^+\ Y\ b^+$ o $b^+\ X\ a^+$, e naturalmente neppure in $a^+\ b^+$ o $b^+\ a^+$.

Non ci sono altri casi rimasti da esaminare. Pertanto in generale la grammatica non è ambigua. Con ciò la dimostrazione è conclusa.

2. Si vuole formalizzare la sintassi di un ipotetico linguaggio di programmazione ispirato al notissimo MATLAB, limitatamente alla definizione di funzione: testata di interfaccia e corpo esecutivo.

   La testata della funzione ha numero ≥ 0 di parametri in ingresso e numero ≥ 0 di parametri in uscita. I parametri in ingresso sono racchiusi in una coppia di parentesi tonde, quelli in uscita in una di parentesi quadre. Quando i parametri in ingresso o uscita mancano, ci devono comunque essere le parentesi tonde o quadre.

   Il corpo della funzione consta di una o più istruzioni. L'istruzione termina facoltativamente con punto-e-virgola ' ; ' ed è sempre seguita (dopo l'eventuale punto-e-virgola) da almeno un carattere separatore, per esempio spazio o ritorno a capo: terminale blank o newline.

   Le istruzioni ammesse sono: assegnamento, condizionale e chiamata a funzione.

   L'istruzione di assegnamento dà a una variabile il valore di un'espressione o il risultato di una chiamata a funzione. In quest'ultimo caso, se la funzione ha più di un parametro in uscita allora a sinistra dell'operatore di assegnamento vanno elencate tra parentesi quadre le variabili che fungono da parametri attuali in uscita.

   Questa notazione che mette tra parentesi quadre i parametri attuali in uscita, va però usata solo quando il numero di parametri è > 1. Una funzione è anche invocabile come procedura ossia senza assegnarne il risultato ad alcuna variabile, e allora si mette solo il nome della funzione con i parametri attuali in ingresso.

   Il condizionale if ha gli elementi seguenti: una clausola di controllo, ossia un'espressione logica o relazionale; un ramo then, ma la parola chiave then va omessa; un numero ≥ 0 di alternative elseif; e una sola alternativa finale else facoltativa.

   Non occorre espandere né identificatori né espressioni algebriche, logiche o relazionali: basta modellare i primi con il terminale id e le seconde con il terminale exp.

   Ecco un esempio di definizione di funzione con testata e corpo:

```
function [o1,  o2]  func1 (i1,  i2,  i3)
   o1 =  i1 + i3 ;
   o2 = func2 (i2, i3)
   if  o1 > o2
      [o1,  o2] = func3 (o1) ;
      func4 ( ) ;
   elseif a1 = a2
      o1 = o2 - i2 ;
      if i2 < i3
         o2 = o1 ;
      end
   else o2 = i3
   end
end
```

   Si chiede di scrivere la grammatica estesa ($EBNF$) per il linguaggio così tratteggiato. Tale grammatica deve generare una lista non vuota di definizioni di funzione.

Soluzione

Ecco la grammatica $EBNF$ richiesta (assioma PROG):

$$
\left\{
\begin{array}{rcl}
\langle \text{PROG} \rangle & \rightarrow & \big(\ \langle \text{F\_DEF} \rangle \ \big)^{+} \\
\hline
\langle \text{F\_DEF} \rangle & \rightarrow & \text{function} \ \langle \text{HEAD} \rangle \ \langle \text{BODY} \rangle \ \text{end} \\
\hline
\langle \text{HEAD} \rangle & \rightarrow & \text{`[' } \langle \text{ID\_LIST} \rangle \mid \varepsilon \text{ `]' id `(' } \langle \text{ID\_LIST} \rangle \mid \varepsilon \text{ `)'} \\
\langle \text{BODY} \rangle & \rightarrow & \big(\ \langle \text{STAT} \rangle \ [\text{ `;' }] \ (\ \text{blank} \mid \text{newline} \ ) \ \big)^{+} \\
\hline
\langle \text{ID\_LIST} \rangle & \rightarrow & \text{id} \ (\text{ `,' id })^{*} \\
\langle \text{STAT} \rangle & \rightarrow & \langle \text{ASGN} \rangle \mid \langle \text{IF} \rangle \mid \langle \text{CALL} \rangle \\
\hline
\langle \text{ASGN} \rangle & \rightarrow & \big(\ \text{id} \mid \text{`[' id `,' } \langle \text{ID\_LIST} \rangle \text{ `]' } \big) \text{ `=' } \langle \text{CALL} \rangle \mid \text{id `=' exp} \\
\langle \text{IF} \rangle & \rightarrow & \text{if} \ \langle \text{IF\_BODY} \rangle \\
\langle \text{CALL} \rangle & \rightarrow & \text{id `(' } \langle \text{ID\_LIST} \rangle \mid \varepsilon \text{ `)'} \\
\hline
\langle \text{IF\_BODY} \rangle & \rightarrow & \text{exp} \ \langle \text{BODY} \rangle \ \big(\ \text{elseif} \ \langle \text{IF\_BODY} \rangle \ \big)^{*} \ \big[\ \text{else} \ \langle \text{BODY} \rangle \ \big] \ \text{end}
\end{array}
\right.
$$

La grammatica è stratificata e facilmente interpretabile.

3 Analisi sintattica e parsificatori 20%

1. È data la grammatica G seguente, in forma $EBNF$ ossia estesa, con alfabeto terminale $\{ a, b \}$ (assioma S):

   $$S \rightarrow (a \mid b) \; S \; b \mid \varepsilon$$

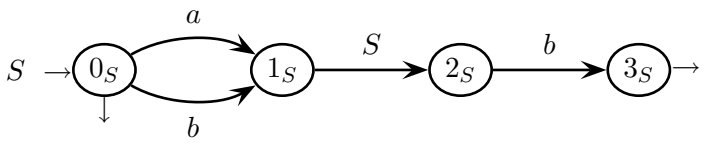
   Si risponda alle domande seguenti:

   (a) Si dica se la grammatica G è di tipo $LL(k)$ ($k \geq 1$) e si giustifichi adeguatamente la risposta data.

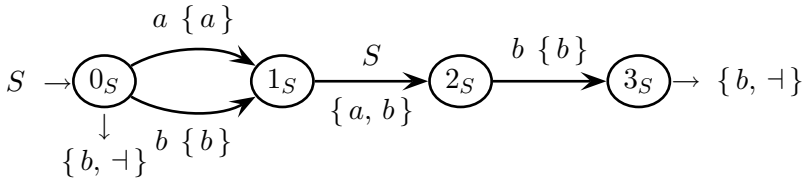
   (b) (facoltativa) Se necessario, si argomenti se il linguaggio $L(G)$ è deterministico.

Soluzione

(a) Ecco la rete di automi, deterministici e minimi:



Gli insiemi di look-ahead con $k = 1$ sono i seguenti:



La grammatica G non è $LL(1)$ nello stato 0_S.

La grammatica non è $LL(2)$. Per esempio, i look-ahead con $k = 2$ allo stato 0_S diventano $\{\ a\ a,\ a\ b,\ a\ \dashv\}$, $\{\ b\ a,\ b\ b,\ b\ \dashv\}$ e $\{\ b\ b,\ b\ \dashv\}$, dunque non disgiunti.

La situazione non cambia per $k \geq 2$ poiché nello stato 0_S resta sempre il look-ahead b^k su due archi distinti. Dunque la grammatica non è LL in generale.

(b) Il linguaggio $L(G)$ contiene, tra le altre, stringhe di tipo $(\,b\,b\,)^n = b^{2n}$ e $a^n\,b^n$ ($n \geq 0$). Le prime sono regolari e le seconde libere. Tuttavia entrambe hanno la metà finale di lunghezza arbitraria e costituita di sole lettere b. Però le stringhe di primo tipo vanno riconosciute senza usare la pila (infatti l'automa non può individuare deterministicamente il centro della stringa), mentre quelle di secondo tipo necessitano della pila per contare (e qui il centro della stringa è individuabile deterministicamente). È difficile immaginare un meccanismo deterministico unificato per riconosceree entrambe le tipologie. Insomma l'automa a pila, trovando una lettera b, deve decidere se iniziare a contare in pila oppure no, ma non ha elementi per decidere deterministicamente poiché non può localizzare il centro della stringa. Dunque ragionevolmente il linguaggio non è deterministico.

2. Il linguaggio libero $L = \{ a^m b^n \mid \ m \geq n \geq 1 \}$ è definito dalla grammatica G_1 seguente (assioma S):

$$G_1 \begin{cases} S & \rightarrow & a\ S\ b \ \mid \ a\ A\ b \\ A & \rightarrow & a\ A \ \mid \ \varepsilon \end{cases}$$

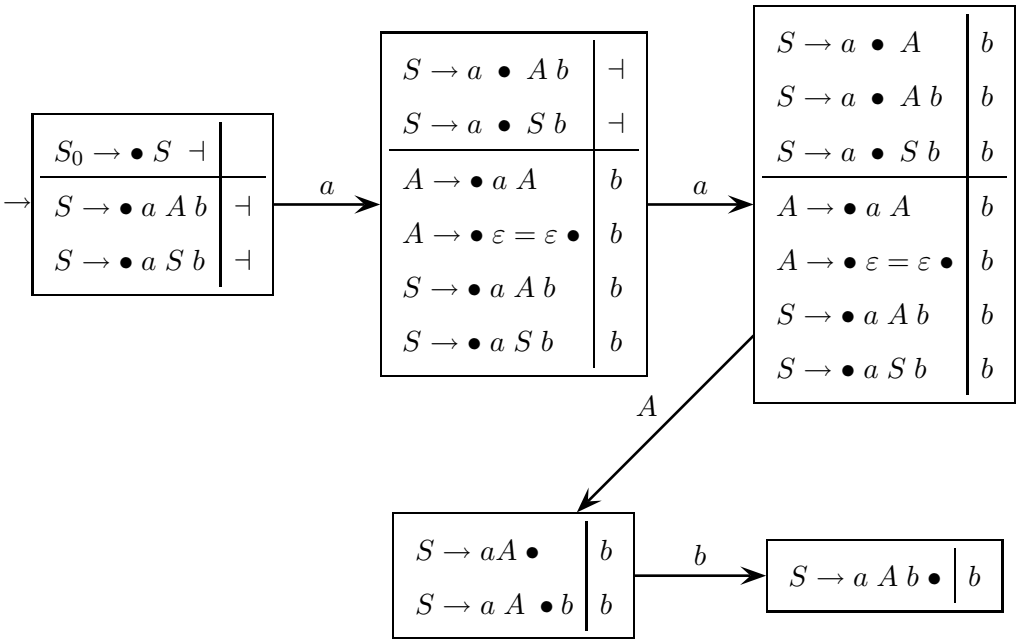
Si risponda alle domande seguenti:

(a) Si costruisca l'automa pilota della grammatica G_1 e si verifichi se esso soddisfa le condizioni $LR(1)$ e $LALR(1)$.

(b) In caso negativo, si scriva una grammatica G_2 equivalente a G_1 tale da soddisfare le condizioni violate da G_1.

Soluzione

(a) Ecco parte del grafo pilota $LR(1)$ della grammatica G_1 data:

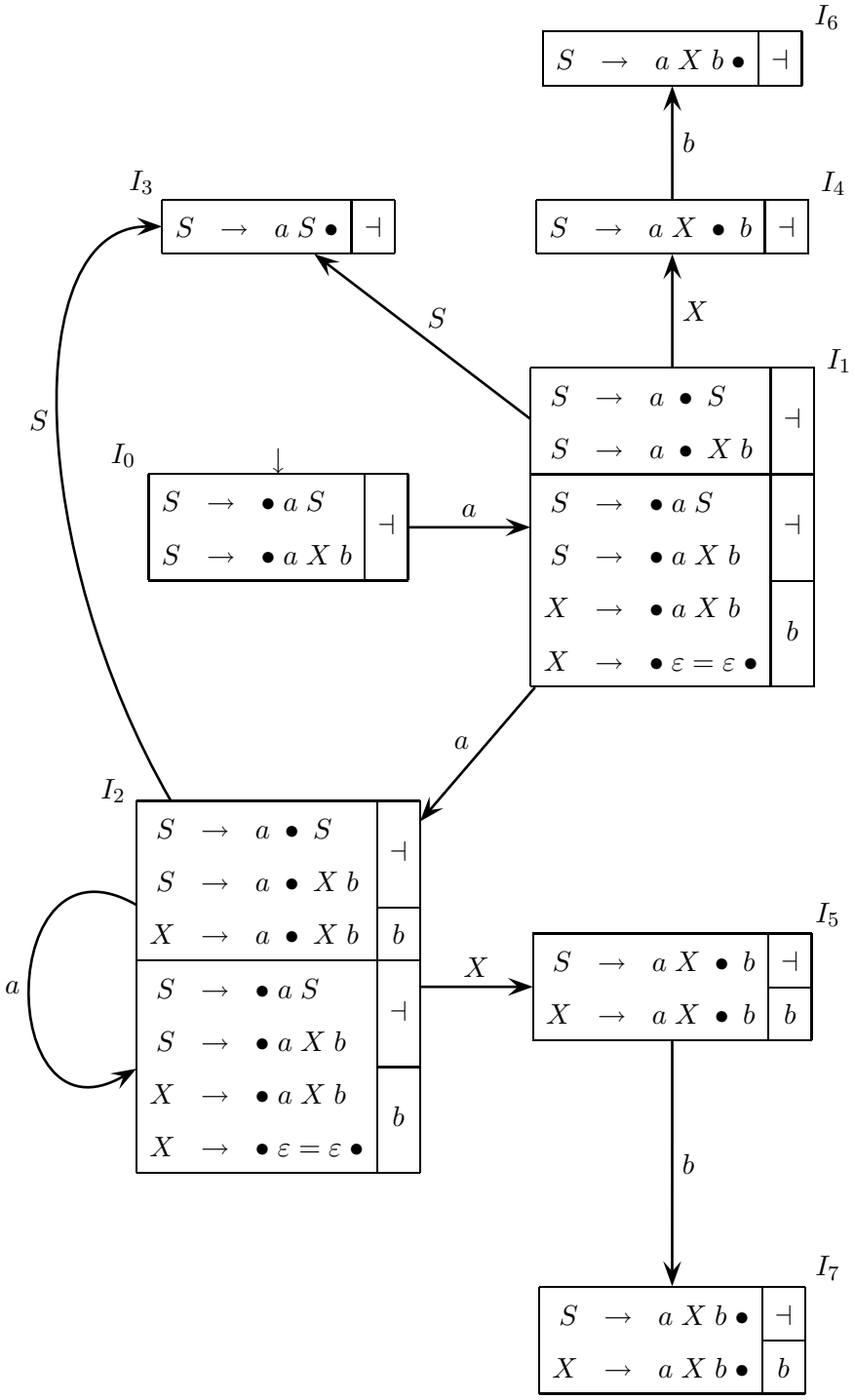


viola la cond. $LR(1)$
quindi anche $LALR(1)$

(b) Per rendere deterministica l'analisi basta scambiare l'ordine delle due ricorsioni, mettendo quella inclusiva all'interno dell'altra.

$$G_2 \begin{cases} S & \to & a\ S \mid a\ X\ b \\ X & \to & a\ X\ b \mid \varepsilon \end{cases}$$

Si può osservare che il linguaggio L è definibile anche come $L = a^*\ a^n\ b^n$ $(n \geq 1)$, dove chiaramente si può porre $a^*\ a^n = a^m$ e si ha $m \geq n \geq 1$ come nella definizione originale. La forma $a^*\ a^n\ b^n$ è ben nota per non essere un linguaggio LL, benché sia deterministico e dunque ammetta senz'altro una grammatica $LR(1)$. A riguardo si veda il libro di testo del corso. La domanda comunque chiede di esibire appunto una tale grammatica $LR(1)$. Basta disegnare il grafo pilota $LR(1)$ di G_2. Eccolo:



Il grafo pilota soddisfa la condizione $LR(1)$. Inoltre non essendoci in esso macro-stati identici a meno degli insiemi di prospezione, anche la condizione $LALR(1)$ è soddisfatta. Peraltro, il pilota $LALR$ ha lo stesso numero di macro-stati di quello LR e dunque non è effettivamente più semplice.

4 Traduzione e analisi semantica 20%

1. Si consideri la traduzione τ seguente, con alfabeti d'ingresso $\{\ a,\ b\ \}$ e d'uscita $\{\ c,\ d\ \}$:

   $$\tau\colon\ (\ a\ b\ )^*\ \ (\ a\ \mid\ b\ ) \rightarrow c^*\ \mid\ (\ d\ d\ )^*$$

   La traduzione τ agisce così:

   $$\begin{aligned} \tau\ \big(\ (\ a\ b\ )^n\ a\ \big) &= c^n & n \geq 0 \\ \tau\ \big(\ (\ a\ b\ )^n\ b\ \big) &= d^{2\,n} & n \geq 0 \end{aligned}$$

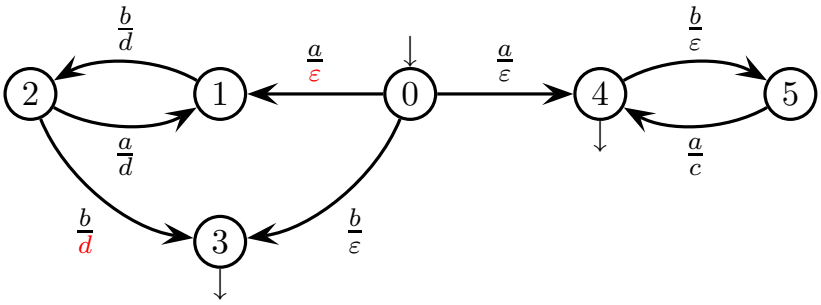
   Si risponda alle domande seguenti:

   (a) Si modelli la traduzione τ come IO-automa T a stati finiti nondeterministico.

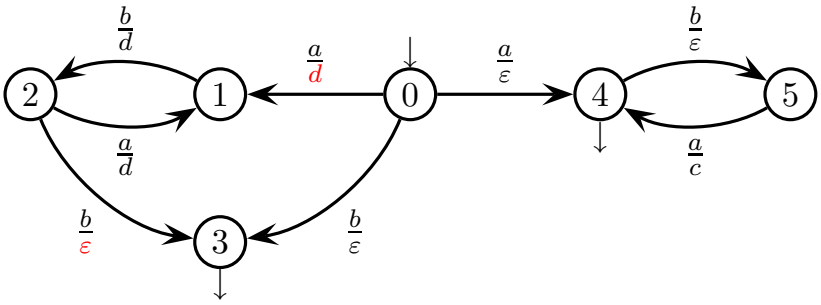
   (b) (facoltativa) Si modelli la traduzione τ come schema sintattico di traduzione G deterministico.

Soluzione

(a) Ecco il grafo stato-transizione di un IO-automa nondeterministico a stati finiti T che realizza la traduzione τ:



Non è l'unica soluzione con sei stati, eccone un'altra pure valida ma un po' diversa (le parti in rosso mettono in evidenza le differenze):



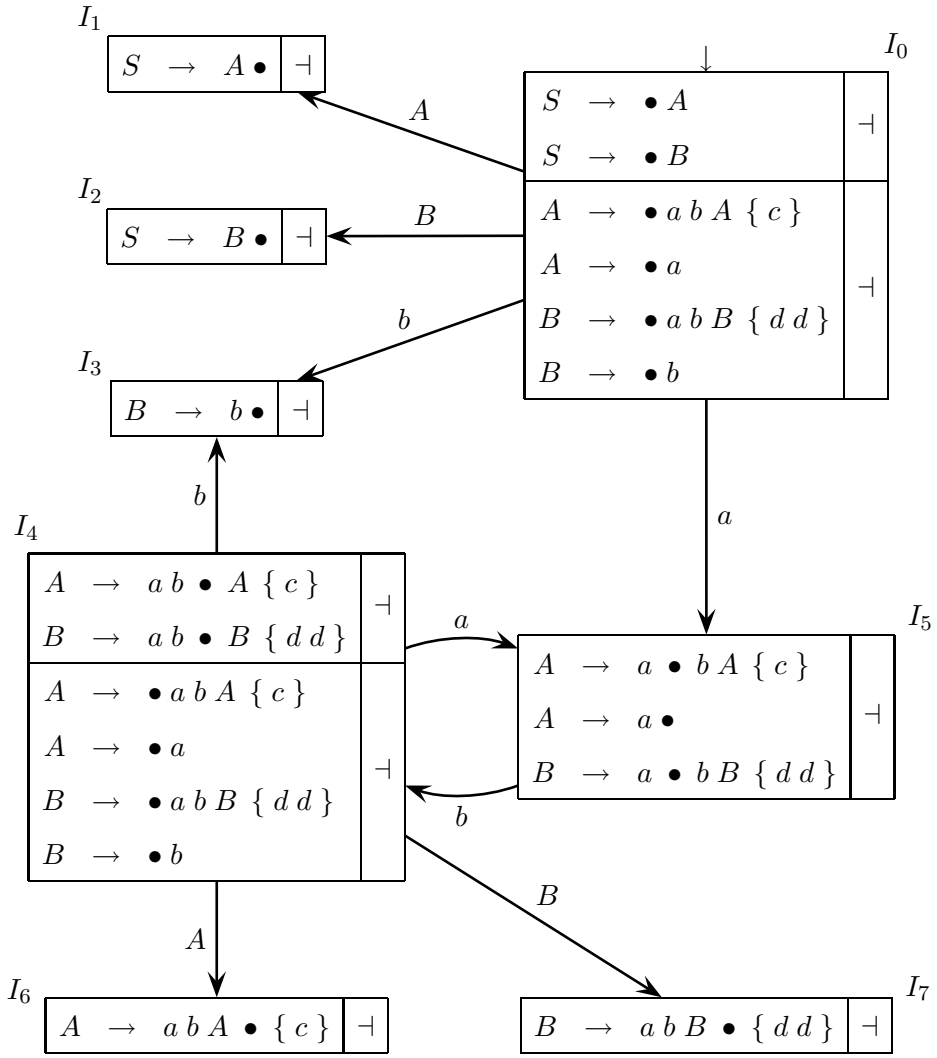
Entrambi gli IO-automi sono nondeterministici nello stato iniziale 0.

(b) Ecco una sintassi G che realizza la traduzione τ (assioma S), qui espressa come grammatica di traduzione (è immediato riscriverla come schema sintattico):

$$G \begin{cases} S \rightarrow A \mid B \\ A \rightarrow a\ b\ A\ \{\ c\ \} \mid a \\ B \rightarrow a\ b\ B\ \{\ d\ d\ \} \mid b \end{cases}$$

Sostanzialmente è una grammatica lineare a destra poiché la traduzione τ è razionale ossia a stati finiti. La grammatica sorgente di G non è LL poiché i nonterminali A e B hanno lo stesso look-ahead. Si vede facilmente però che essa è LR: intuitivamente legge ripetutamente la coppie $a\ b$ in ingresso spostando ogni volta; dunque le conta impilandole fino a quando prospeziona la lettera a o b finale e inizia a ridurre spilando ed emettendo altrettante lettere c o coppie di lettere d. Pertanto la sintassi di traduzione scritta è deterministica.

Per una dimostrazione sistematica basta disegnare il pilota $LR(1)$. Eccolo:



La condizione $LR(1)$ è facilmente soddisfatta poiché nelle riduzioni la prospezione è sempre il terminatore $\dashv$ e non confligge con gli spostamenti, e inoltre non ci sono macro-stati contenenti due o più riduzioni. Oltre a ciò si noti che la grammatica di traduzione G ha forma postfissa come richiesto per realizzare l'analizzatore sintattico LR: ogniqualvolta si sposta nei macro-stati I_6 o I_7, l'analizzatore riduce ed emette c o $d\ d$, rispettivamente. Resta così dimostrato rigorosamente che lo schema di traduzione è deterministico.

La condizione $LALR(1)$ è soddisfatta per lo stesso motivo di quella $LR(1)$. Tuttavia non ci sono macro-stati con base identica e dunque il traduttore $LALR$ non ha meno macro-stati di quello LR.

La condizione $LR(0)$ non è soddisfatta poiché il linguaggio sorgente stesso ha prefissi (p. es. a è prefisso di $a\ b\ a$), e dunque nessuna grammatica che lo genera può essere $LR(0)$. Si noti che il pilota $LR(1)$ si limita a prospezionare il terminatore $\dashv$: ciò serve e basta appunto per sapere se la stringa letta è l'intero contenuto del nastro di ingresso o solo un suo prefisso. Se si cancellasse la prospezione, in I_5 la riduzione confliggerebbe con gli spostamenti.

2. È data la sintassi seguente (assioma S), che genera espressioni con parantesi e fattori moltiplicativi stilizzati dal terminale ‘ a ’ (il segno di moltiplicazione è sottinteso):

$$\begin{cases} 0: & S & \rightarrow & E \\ 1: & E & \rightarrow & E\ E \\ 2: & E & \rightarrow & \text{‘(’}\ E\ \text{‘)’} \\ 3: & E & \rightarrow & a \end{cases}$$

Esempi:

$$a \quad a\,a \quad (\ a\ ) \quad a\ (\ a\ ) \quad (\ a\,a\ ) \quad (\ (\ a\ )\ ) \quad (\ a\ (\ a\,a\ )\ a\,a\,a\ )$$

Si vuole realizzare una grammatica con attributi che traduca il linguaggio in questione eliminando le parentesi e riscrivendo il monomio risultante $\underbrace{a\,a\,\ldots\,a}_{n \text{ volte}}$ in forma esponenziale come a^n ($n \geq 1$), però ignorando le lettere esterne alle parentesi e le lettere contenute in un numero pari di parentesi.

Per esempio:

$$(\ a\ (\ a\,a\ )\ a\,a\,a\ )\ a \Rightarrow (\ a\hat{}1\ (\ a\hat{}2\ )\ a\hat{}3\ )\ a\hat{}1 \Rightarrow a\hat{}1\ a\hat{}3 = a\hat{}4$$

Per scrivere l’esponente è a disposizione una funzione *string* (n) con argomento numerico $n \geq 1$, la quale emette la stringa numerica decimale che rappresenta il valore n. Il carattere ‘ ^ ’ rappresenta l’esponenziazione.

Si vuole realizzare la grammatica con attributi partendo dalla sintassi data sopra. Allo scopo vanno usate le tabelle predisposte nelle pagine seguenti.

Si risponda alle domande seguenti:

(a) Si progetti una grammatica con attributi per tradurre l’espressione, prima modificando la sintassi data in modo da rappresentare la parità del livello di annidamento delle parentesi, e quindi dando alla nuova sintassi gli attributi e le regole semantiche opportune.

(b) (facoltativa) Si progetti una grammatica con attributi per tradurre l’espressione, ma senza modificare la sintassi data.

---

attributi da usare per la grammatica - domanda (a)

tipo	nome	(non)terminali	dominio	significato

attributi da usare per la grammatica - domanda (b)

tipo	nome	(non)terminali	dominio	significato

sintassi modificata	calcolo attributi - domanda (a)

sintassi originale	calcolo attributi - domanda (b)
0: $S_0 \rightarrow E_1$	
1: $E_0 \rightarrow E_1\ E_2$	
2: $E_0 \rightarrow$ ‘(’ E_1 ‘)’	
3: $E_0 \rightarrow a$	

Soluzione

(a) Si può distinguere la parità di parentesi in modo sintattico, così (assioma S):

$$\begin{cases} 0: & S & \to & P \\ 1: & P & \to & P\ P \\ 2: & D & \to & D\ D \\ 3: & P & \to & \text{`(' } D \text{ `)'} \\ 4: & D & \to & \text{`(' } P \text{ `)'} \\ 5: & P & \to & a \\ 6: & D & \to & a \end{cases}$$

attributi da usare per la grammatica - domanda (a)

tipo	nome	(non)terminali	dominio	significato
sx	*cont*	P, D	intero	conta lettere a
sx	τ	S	stringa	traduzione

Ecco le funzioni semantiche, dove se contare o no è deciso sintatticamente:

sintassi modificata	calcolo attributi - domanda (a)
$0:\quad S_0 \to P_1$	$\tau_0 = cat\,(\,a,\,\hat{},\,string\,(\,cont_1\,)\,)$
$1:\quad P_0 \to P_1\ P_2$	$cont_0 = cont_1 + cont_2$
$2:\quad D_0 \to D_1\ D_2$	$cont_0 = cont_1 + cont_2$
$3:\quad P_0 \to$ `(' $D_1$ `)'`	$cont_0 = cont_1$
$4:\quad D_0 \to$ `(' $P_1$ `)'`	$cont_0 = cont_1$
$5:\quad P_0 \to a$	$cont_0 = 0$ – se pari non contare
$6:\quad D_0 \to a$	$cont_0 = 1$ – se dispari conta

La grammatica con attributi è puramente sintetizzata, dunque è a una passata.

(b) Senza modifcare la sintassi data, occorre prevedere un attributo booleano ereditato per distinguere la parità delle parentesi. Il conteggio delle lettere rimane sintetizzato con gli stessi attributi della domanda (a).

attributi da usare per la grammatica - domanda (b)

tipo	nome	(non)terminali	dominio	significato
sx	*cont*	E	intero	conta lettere a
sx	τ	S	stringa	traduzione
dx	π	E	booleano	parentesi dispari

Ecco le funzioni semantiche, dove se contare o no è deciso semanticamente:

sintassi originale	calcolo attributi - domanda (b)
0: $S_0 \rightarrow E_1$	$\pi_1 = false$ – all'esterno non sono dispari $\tau_0 = cat\,(\,a, \hat{}, string\,(\,cont_1\,)\,)$
1: $E_0 \rightarrow E_1\ E_2$	$\pi_1 = \pi_0$ – conserva parità $\pi_2 = \pi_0$ – conserva parità $cont_0 = cont_1 + cont_2$
2: $E_0 \rightarrow$ ‘(’ E_1 ‘)’	$\pi_1 =$ **not** π_0 – commuta parità $cont_0 = cont_1$
3: $E_0 \rightarrow a$	**if** $\pi_0 == true$ **then** – se dispari conta $cont_0 = 1$ **else** – se pari non contare $cont_0 = 0$ **endif**

La grammatica è comunque a una passata: in discesa si calcola l'attributo π, in salita l'attributo *cont* e infine nella radice la traduzione τ.