Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 5 marzo 2008 - Parte I: Teoria

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

NOME:

COGNOME:

MATRICOLA: FIRMA:

ISTRUZIONI - LEGGERE CON ATTENZIONE:

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

1 Espressioni regolari e automi finiti 20%

1. Sono date le tre espressioni regolari E_1, E_2 e E seguenti, estese con operatore di complemento e intersezione, sull'alfabeto $\Sigma = \{ a, +, - \}$:

$$
\begin{aligned}
E_1 &= a\ ((\ +\ \mid\ -\ )\ a\ )^* \\
E_2 &= \neg\ (\ \Sigma^*\ -\ \Sigma^*\ +\ \Sigma^*\ ) \\
E &= E_1\ \cap\ E_2
\end{aligned}
$$

Il simbolo "¬" indica il complemento insiemistico. Si badi bene che qui il carattere "–" è un elemento dell'alfabeto Σ (non l'operatore di differenza insiemistica).

Si risponda alle domande seguenti:

(a) Si scrivano nella tabella le stringhe di lunghezza pari a esattamente cinque caratteri generate dalle espressioni E_1 e E (il numero di righe della tabella non è significativo).

(b) Seguendo un metodo a scelta si scriva un'espressione regolare equivalente all'espressione E ma non estesa (cioè solo con operatori di concatenamento unione e stella o croce). Si giustifichi a parole il metodo di costruzione seguìto.

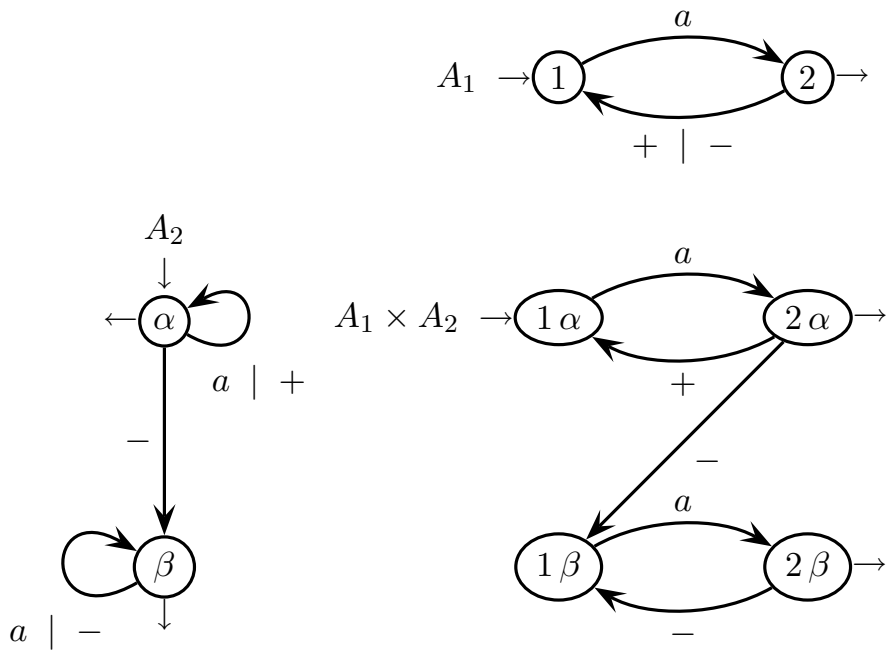
Soluzione

(a) Le stringhe di lunghezza pari a esattamente 5 caratteri generate dalle espressioni regolari E_1 e E sono le seguenti:

#	E_1	E
	$a\ +\ a\ +\ a$	$a\ +\ a\ +\ a$
	$a\ +\ a\ -\ a$	$a\ +\ a\ -\ a$
	$a\ -\ a\ +\ a$	$a\ -\ a\ -\ a$
	$a\ -\ a\ -\ a$	

Infatti E_1 genera espressioni (non vuote) di tipo somma algebrica con operatori $+$ e $-$ e con termini a, mentre E_2 genera stringhe qualsiasi (in generale infatti non sono espressioni ben formate) dove però il carattere $-$ non precede il carattere $+$; pertanto l'intersezione di E_1 e E_2 genera espressioni ben formate come quelle di E_1, ma dove l'operatore $-$ non precede l'operatore $+$.

(b) Per trovare l'espressione regolare E si costruiscono prima gli automi delle espressioni regolari E_1 e E_2 e poi l'automa prodotto, dal quale infine si ricava E.
Esaminando le espressioni regolari E_1 e E_2 si trova che E_1 definisce una lista di una o più lettere a separate da operatore $+$ o $-$, mentre E_2 definisce una stringa generica purché non contenga un operatore $+$ dopo un operatore $-$. Risulta allora semplice disegnare direttamente i due automi, disponendoli uno in orizzontale e l'altro in verticale in modo da rendere più ordinata la costruzione successiva dell'automa prodotto. Ecco l'intera costruzione:



È facile ricavare l'espressione regolare E del linguaggio notando l'esistenza di un circuito tra i nodi $(2\ \alpha)$ e $(1\ \alpha)$ che produce la lista $(+\ a)^*$, e di un circuito tra i nodi $(2\ \beta)$ e $(1\ \beta)$ che produce la lista $(-\ a)^*$. Eliminati tali nodi, applicando in modo sbrigativo il metodo BMC, si ottiene l'espressione non estesa E:

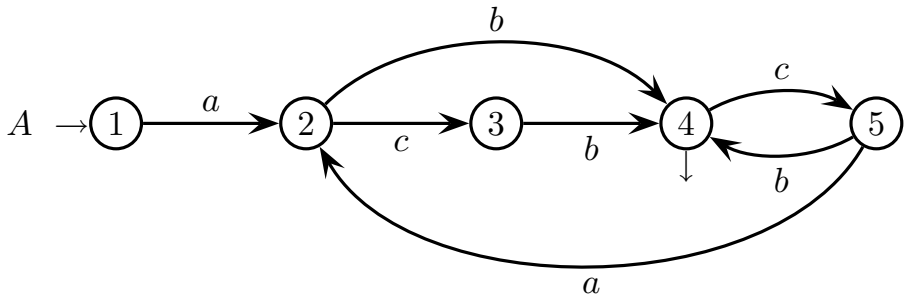
$$E = a\ (\ +\ a\ )^*\ (\ \varepsilon\ \mid\ -\ a\ (\ -\ a\ )^*\ )$$

Con un attimo di riflessione si arriva alla forma seguente di E:

$$E = a\ (\ +\ a\ )^*\ (\ -\ a\ )^*$$

di aspetto più compatto ed elegante.

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



sull'alfabeto di ingresso $\Sigma = \{a, b, c\}$.

Si applichi all'automa A la trasformazione alfabetica seguente (omomorfismo):

$$h(a) = a \qquad h(b) = b \qquad h(c) = \varepsilon$$

Si risponda alle domande seguenti:

(a) Si disegni il grafo stato-transizione dell'automa deterministico A' che riconosce il linguaggio $h(L(A))$ (immagine omomorfa di $L(A)$).

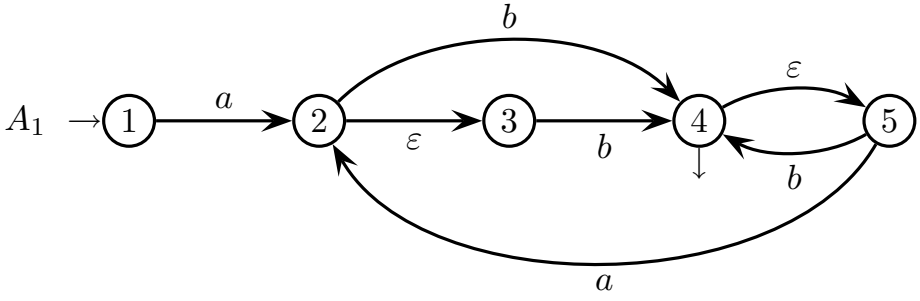
(b) Si dica, motivando la risposta, se i due linguaggi seguenti:

$$L(A) \qquad \text{e} \qquad h(L(A))$$

cioè quelli riconosciuti dagli automi A e A' rispettivamente, siano locali o no.

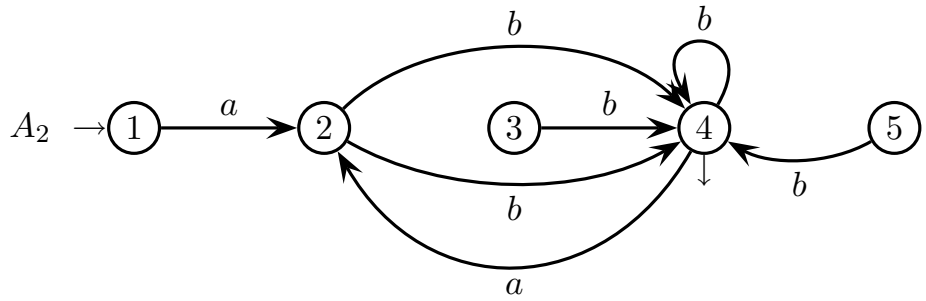
Soluzione

(a) In primo luogo ecco il grafo stato-transizione dell'automa indeterminstico A_1 che riconosce il linguaggio $h(L(A))$:

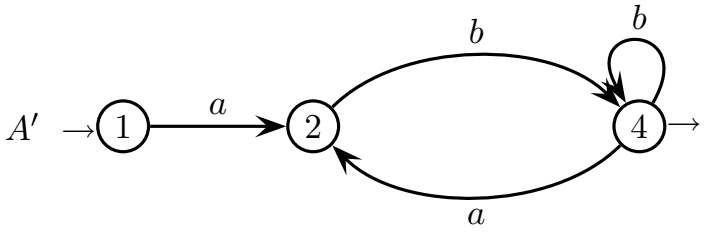


L'automa A_1 è ottenuto applicando direttamente l'omomorfismo h alle etichette degli archi dell'automa A. Poiché compaiono due transizioni spontanee ε-transizioni) l'automa A_1 è indeterministico.

È facile rendere deterministico l'automa A_1 tagliando le transizioni spontanee (ε-transizioni). Ecco l'automa A_2 così ottenuto:

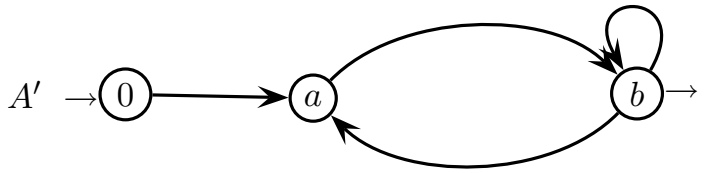


Si possono eliminare gli stati 3 e 5, in quanto irraggiungibili, e ridurre a uno solo i due archi dallo stato 2 allo stato 4 con etichetta b, giacché rappresentano la stessa transizione, ottenendo l'automa A' deterministico richiesto:



L'automa A' così ottenuto è minimo: lo stato finale 4 è distinguibile dagli stati 1 e 2 (non finali), e gli stati 1 e 2 sono distinguibili tra sé poiché non hanno archi uscenti con le stesse etichette.

(b) Il linguaggio $h\,(L\,(A))$ coincide con il linguaggio $L\,(A')$ e l'automa (minimo) A' è locale. Infatti gli archi entranti in ogni stato sono sempre etichettati con la stessa lettera, la quale si può dunque scrivere nello stato come segue:



L'automa A' così ridisegnato è evidentemente locale: gli stati hanno etichettatura univoca poiché essendo l'automa minimo è impossibile che due stati si fondano; pertanto il linguaggio $h(L(A)) = L(A')$ è locale.

Invece il linguaggio $L\,(A)$ non è locale: osservando l'automa A si vede che il linguaggio $L(A)$ ammette i digrammi $a\,b$, $a\,c$, $c\,a$, $c\,b$ e $b\,c$, e ha carattere iniziale a e finale b. Ma allora un ipotetico automa locale costruito secondo tali digrammi ammissibili inevitabilmente riconoscerebbe tutte le stringhe $(a\,c)^n\ b$ (con $n \geq 2$), le quali sono conformi ai digrammi e iniziano e terminano come richiesto, ma non sono riconosciute dall'automa A. Dunque non può esistere un automa locale che riconosca il linguaggio $L\,(A)$, il quale pertanto non è un linguaggio locale.

2 Grammatiche libere e automi a pila 20%

1. Si consideri il linguaggio di Dyck L_D con parentesi tonde e quadre, cioè di alfabeto $\Sigma = \{\,$'(', ')', '[', ']'$\,\}$. Si definisca il linguaggio L come sottoinsieme del linguaggio L_D con il vincolo seguente: le stringhe di L non contengono mai tre o più parentesi tonde aperte consecutive.

   Ecco alcune stringe non valide, cioè non appartenenti al linguaggio L:

   [((( ) [ ]))] (((( ) [ ])))

   Si risponda alle domande seguenti:

   (a) Si scriva una grammatica G, in forma non estesa (BNF), che generi il linguaggio L (non importa se G è ambigua).

   (b) Si formuli il linguaggio L come intersezione del linguaggio di Dyck e di un'espressione regolare E, ovvero:

   $$L = L_D \cap L(E)$$

   Si deve dare l'espressione regolare E (a scelta se in forma estesa o no).

Soluzione

(a) Ecco la grammatica G che modella il linguaggio L richiesto (assioma S), ricavata dalla regola di Dyck standard $S \to (\ S\ )\ S \mid \varepsilon$:

$$G \begin{cases} S & \to & \text{`('}\ S_1\ \text{`)}\ S \\ S & \to & \text{`['}\ S\ \text{`]}\ S \\ S & \to & \varepsilon \\ S_1 & \to & \text{`('}\ S_2\ \text{`)}\ S \\ S_1 & \to & \text{`['}\ S\ \text{`]}\ S \\ S_2 & \to & \text{`['}\ S\ \text{`]}\ S \end{cases}$$

È facile convincersi della non ambiguità della grammatica G. Può darsi esistano soluzioni ambigue più compatte di G.

(b) Posto l'alfabeto $\Sigma = \{\ \text{`('}, \text{`)'}, \text{`['}, \text{`]'}\ \}$, ecco una possibile forma estesa per l'espressione regolare E richiesta (contiene l'operatore di differenza insiemistica):

$$E = \Sigma^* \ -\ (\ \Sigma^*\ \text{`(}\ (\ (\text{'}\ \Sigma^*\ ) = \neg (\ \Sigma^*\ \text{`(}\ (\ (\text{'}\ \Sigma^*\ )$$

Se però si preferisce una formulazione non estesa per l'espressione regolare E, eccone una:

$$E' = \big(\ (\ \varepsilon \mid \text{`('} \mid \text{`(}\ (\text{'}\ )\ (\ \text{`)'} \mid \text{`['} \mid \text{`]'}\ )^+\ \big)^*\ (\ \varepsilon \mid \text{`('} \mid \text{`(}\ (\text{'}\ )$$

L'espressione regolare E' è equivalente a E, cioè si ha $L(E') = L(E)$. Peraltro volendo si può economizzare un po'. Ecco una seconda formualzione E'' non estesa, più breve di E':

$$E'' = (\ \text{`['} \mid \text{`]'}\ )^*\ \big(\ (\ \text{`('} \mid \text{`(}\ (\text{'}\ )\ (\ \text{`)'} \mid \text{`['} \mid \text{`]'}\ )^+\ \big)^*$$

Si noti che vale l'inclusione stretta $L(E'') \subset L(E)$, perché l'espressione E'' non genera stringhe inizianti con parentesi tonde chiuse o terminanti con parentesi tonde aperte (mentre l'espressione E fa entrambe le cose). D'altra parte ciò non è necessario (benché non sia un errore), giacché una stringa di Dyck corretta (e dunque appartenente a L_D) non potrebbe comunque né iniziare con parentesi tonde chiuse né terminare con parentesi tonde aperte.

2. Si consideri un linguaggio per modellare una sezione di dichiarazione di dati, ragionevolmente simile a quella di un generico linguaggio di programmazione (qui ci si ispira a Pascal), comprendente gli aspetti sintattici seguenti:

   - La sezione di dichiarazione è costituita da una lista di dichiarazioni di variabile separate da ";" (punto e virgola).
   - Sono possibili variabili di tipo scalare: intero e reale. Per esempio:

     ```
     age : integer ;
     weight, height : real ;
     ```

   - Sono possobili variabili di tipo record (o structure) contenenti uno o più campi di ogni specie, cioè scalare, vettore (array) (vedi sotto) e record. Per esempio:

     ```
     city : record
         position : record
             x, y : real ;
         end record ;
         population : integer ;
     end record ;
     ```

   - Sono possibili variabili di tipo array (vettore), a una o più dimensioni, i cui elementi possono essere di tipo scalare o record, ma non di tipo array. Per esempio:

     ```
     tabella : array [1 ... 10, 1 ... 100, -10 ... 10] of
     record
         re, im : real ;
     end record ;
     ```

   Si risponda alle domande seguenti:

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

   (b) (facoltativo) Si rappresentino le regole della grammatica G progettata sotto forma di diagrammi sintattici.

Soluzione

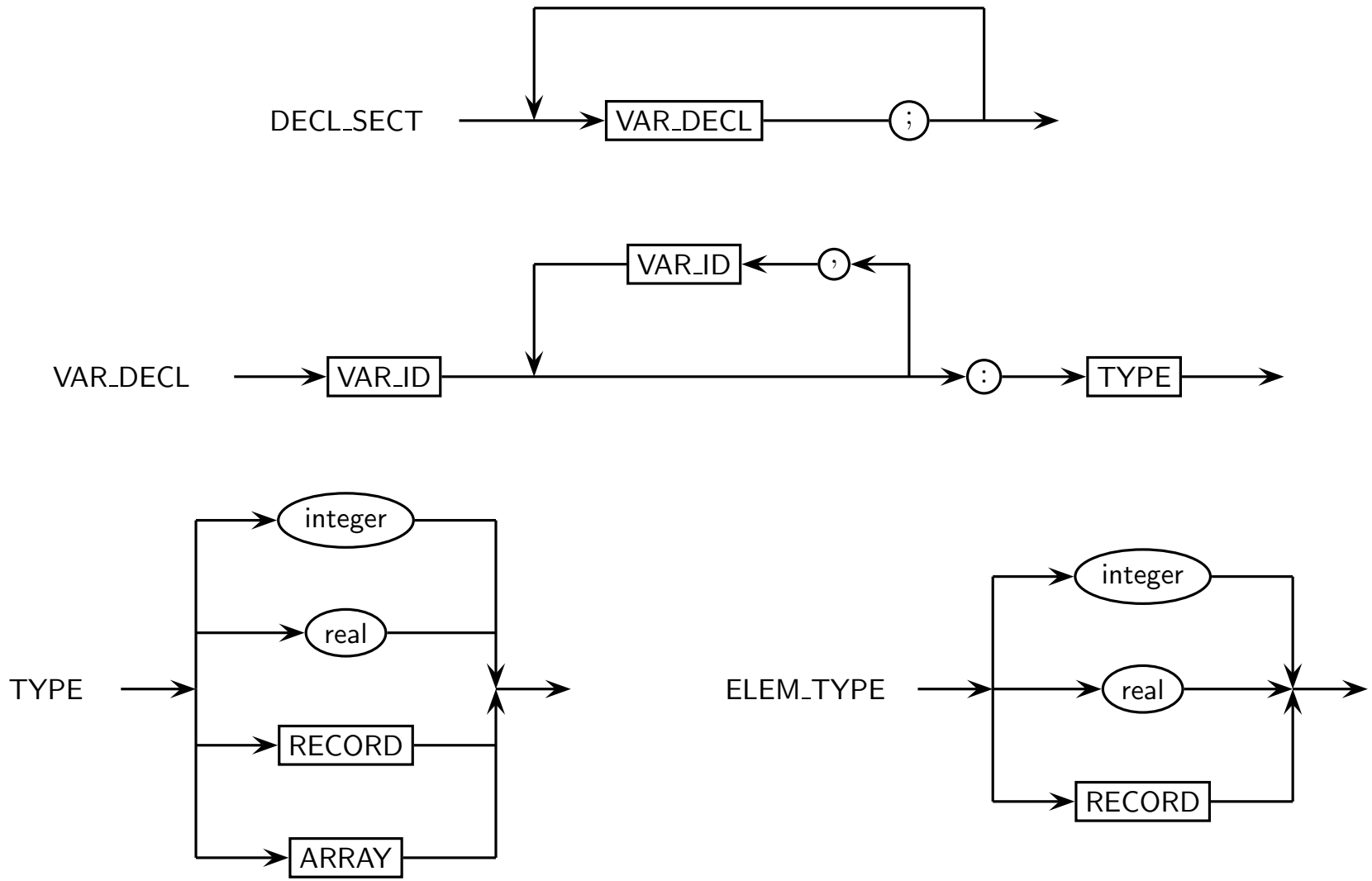
(a) Ecco la grammatica per modellare il linguaggio descritto (assioma ⟨DECL_SECT⟩:

$$
\begin{aligned}
\langle \text{DECL\_SECT} \rangle &\rightarrow (\ \langle \text{VAR\_DECL} \rangle\ \text{';'}\ )^{+} \\
\langle \text{VAR\_DECL} \rangle &\rightarrow \langle \text{VAR\_ID} \rangle\ (\ \text{','}\ \langle \text{VAR\_ID} \rangle\ )^{*}\ \text{': '}\ \langle \text{TYPE} \rangle \\
\langle \text{TYPE} \rangle &\rightarrow \text{'integer'} \mid \text{'real'} \mid \langle \text{RECORD} \rangle \mid \langle \text{ARRAY} \rangle \\
\langle \text{ELEM\_TYPE} \rangle &\rightarrow \text{'integer'} \mid \text{'real'} \mid \langle \text{RECORD} \rangle \\
\langle \text{RECORD} \rangle &\rightarrow \text{'record'}\ (\ \langle \text{FIELD\_DECL} \rangle\ \text{';'}\ )^{+}\ \ \text{'end record'} \\
\langle \text{ARRAY} \rangle &\rightarrow \text{'array'}\ \text{'['}\ \langle \text{RANGE\_LIST} \rangle\ \text{']'}\ \text{'of'}\ \langle \text{ELEM\_TYPE} \rangle \\
\langle \text{FIELD\_DECL} \rangle &\rightarrow \langle \text{FIELD\_ID} \rangle\ (\ \text{','}\ \langle \text{FIELD\_ID} \rangle\ )^{*}\ \text{': '}\ \langle \text{TYPE} \rangle \\
\langle \text{RANGE\_LIST} \rangle &\rightarrow \langle \text{RANGE} \rangle\ (\ \text{','}\ \langle \text{RANGE} \rangle\ )^{*} \\
\langle \text{VAR\_ID} \rangle &\rightarrow \langle \text{IDENTIFIER} \rangle \\
\langle \text{FIELD\_ID} \rangle &\rightarrow \langle \text{IDENTIFIER} \rangle \\
\langle \text{RANGE} \rangle &\rightarrow \langle \text{INTEGER} \rangle\ \text{'...'}\ \langle \text{INTEGER} \rangle \\
\langle \text{IDENTIFIER} \rangle &\rightarrow [\text{'A'} - \text{'Z'}]\ (\ [\text{'A'} - \text{'Z'}] \mid [\text{'0'} - \text{'9'}] \mid \text{'\_'})^{*} \\
\langle \text{INTEGER} \rangle &\rightarrow [\text{'+'} \mid \text{'}-\text{'}]\ [\text{'1'} - \text{'9'}]\ [\text{'0'} - \text{'9'}]^{*}
\end{aligned}
$$

Si noti come sarebbe possibile eliminare la classe sintattica ⟨FIELD_DECL⟩ sostituendola con ⟨VAR_DECL⟩ (si vede subito che sono equivalenti), così riducendo la taglia della grammatica. Tale scelta sarebbe qui del tutto accettabile. Tuttavia non è neppure inopportuno tenerle separate allo scopo di facilitare la distinzione semantica tra variabile e campo di record, giacché il secondo ha uno "scope" (visibilità) ristretto al solo record dove è dichiarato, mentre la prima potrebbe essere globale e pertanto visibile ovunque. Analogo ragionamento vale, su scala ridotta, per le classi sintattiche ⟨VAR_ID⟩ e ⟨FIELD_ID⟩ (le quali volendo sono unificabili). Inoltre la classe ⟨ELEM_TYPE⟩ è un sottinsieme della classe ⟨TYPE⟩ e dunque si potrebbe usare la prima come parte della seconda; tuttavia anche in questo caso tenerle distinte rende la grammatica più chiara evitando rimandi.

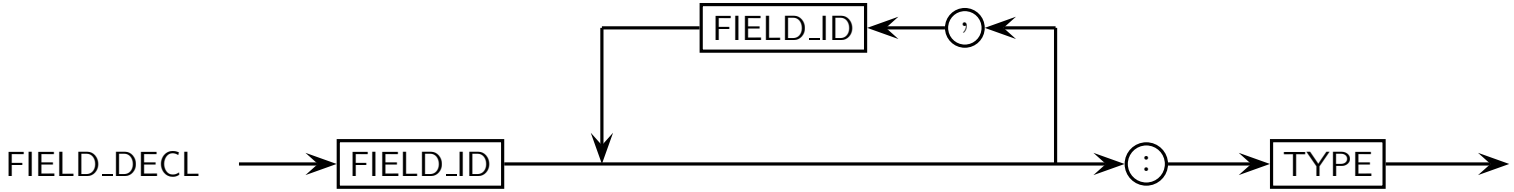
Benché non sia richiesto dall'esercizio ecco gli aspetti del linguaggio non modellabili sintatticamente: la dichiarazione di una variabile deve precederne l'uso; è vietato dichiarare due o più variabili omonime; l'estremo inferiore di una dimensione dell'array deve essere minore dell'estremo superiore; e forse altri ancora.

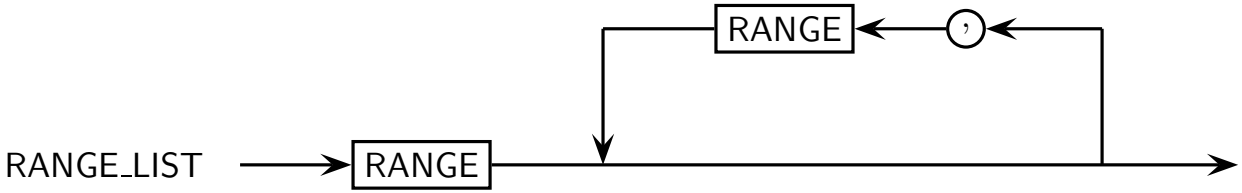
(b) Ecco i diagrammi sintattici della grammatica G progettata (non presentano difficoltà di sorta).



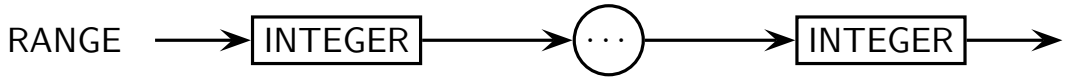

RECORD
record
FIELD_DECL
;
end record

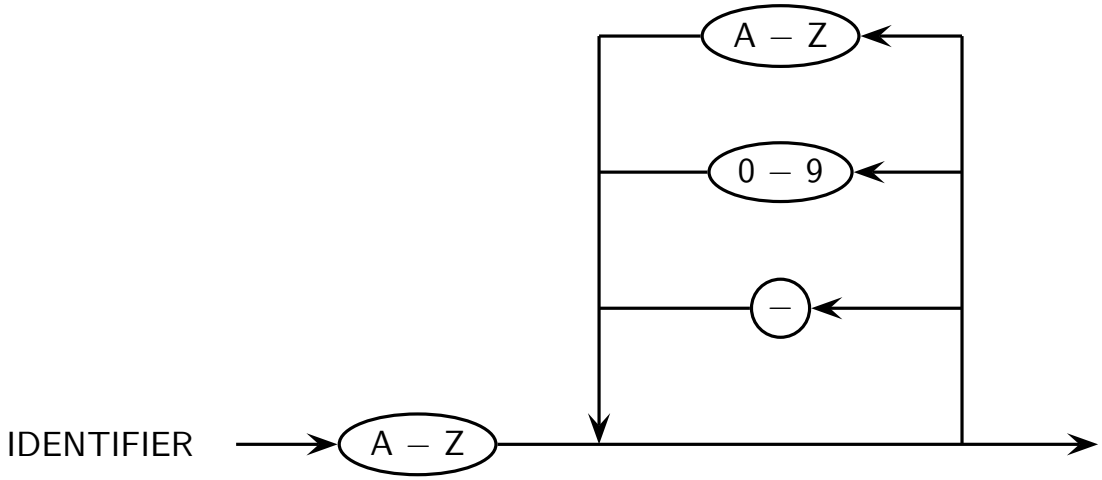

ARRAY
array
[
RANGE_LIST
]
of
ELEM_TYPE

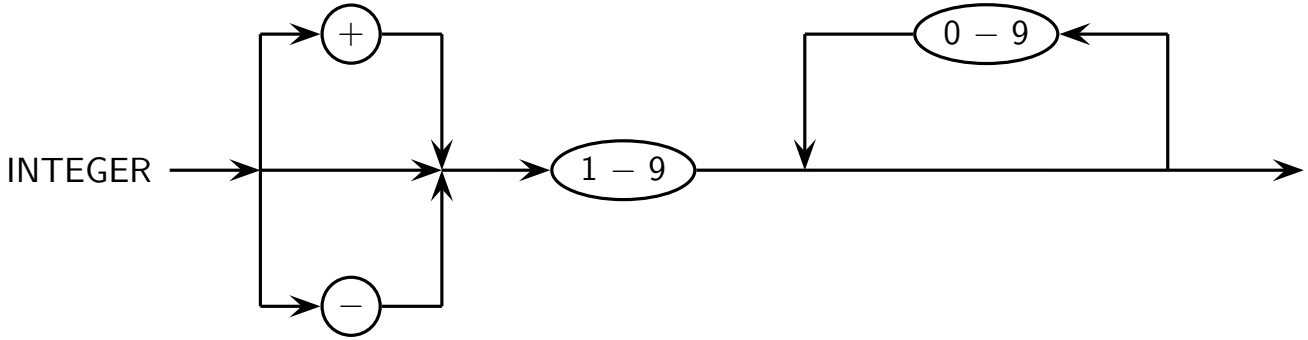

FIELD_ID
,
FIELD_DECL
FIELD_ID
:
TYPE


RANGE
,
RANGE_LIST
RANGE


VAR_ID
IDENTIFIER
FIELD_ID
IDENTIFIER


RANGE
INTEGER
· · ·
INTEGER


A − Z
0 − 9
−
IDENTIFIER
A − Z


+
0 − 9
INTEGER
1 − 9
−

3 Analisi sintattica e parsificatori 20%

1. È data la grammatica G seguente (assioma S) in forma estesa (EBNF):

$$G \begin{cases} S & \rightarrow & A\ b \ \mid \ a\ [\ c\ S\ ] \\ A & \rightarrow & a\ (\ b\ A\ )^*\ S \end{cases}$$

definita sull'alfabeto terminale $\{a, b, c\}$.

Si risponda alle domande seguenti:

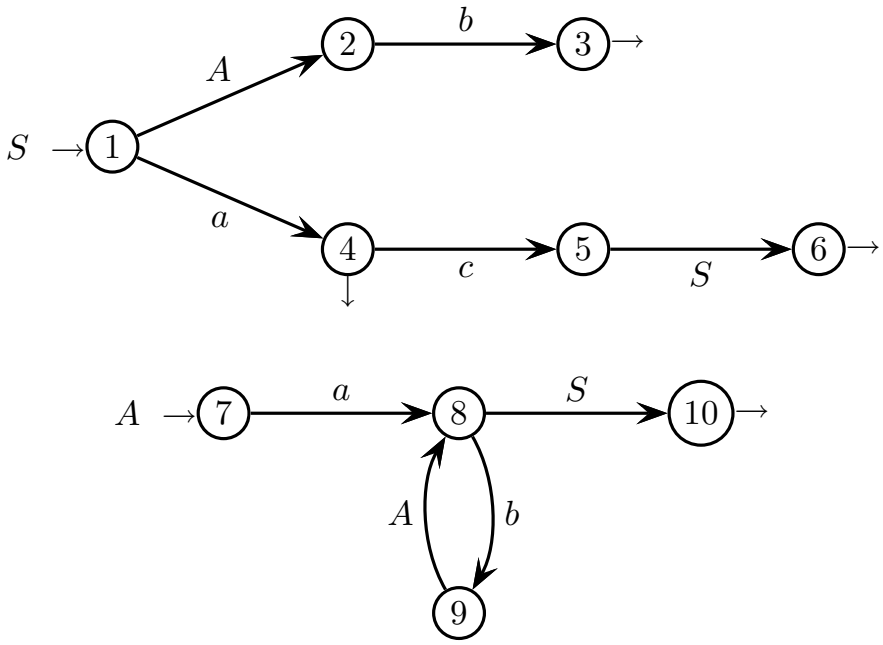
(a) Si rappresenti la grammatica G in forma di rete di automi ricorsivi a stati finiti sull'alfabeto totale (unione di terminali e nonterminali).

(b) Si calcolino tutti gli insiemi guida della grammatica G (in forma di rete di automi) con $k = 1$ e si dica se G sia di tipo $LL(1)$.

(c) Si dica se la grammatica G sia di tipo $LL(2)$ calcolando gli insiemi guida con $k = 2$ eventualmente necessari.

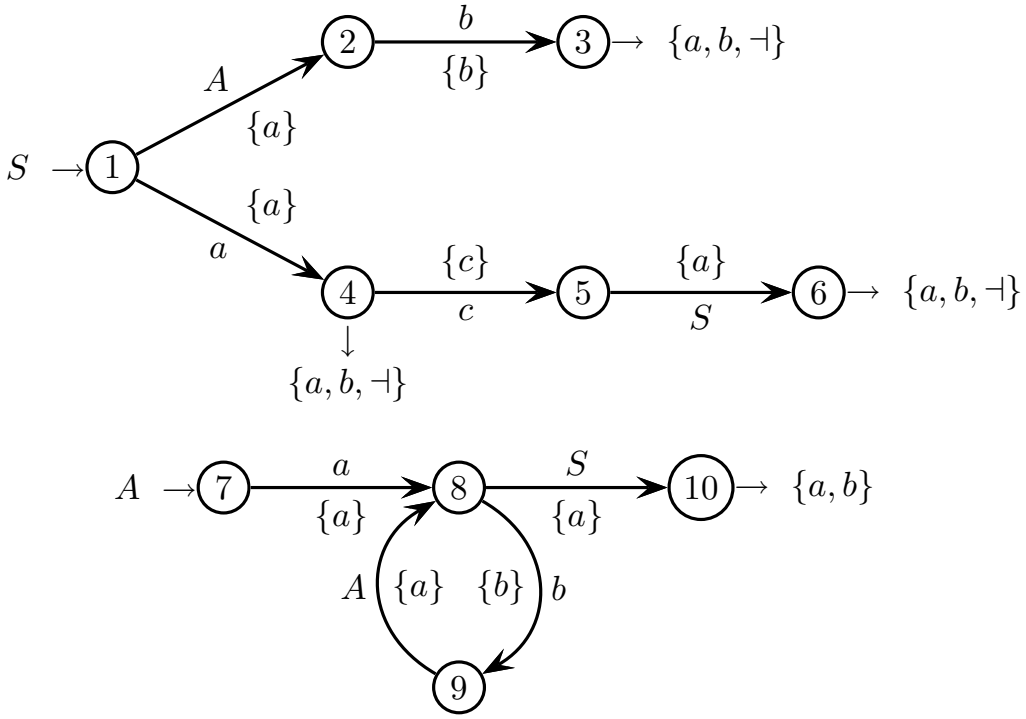
Soluzione

(a) Si ricordi che le parentesi quadre sono metacaratteri (non elementi dell'alfabeto terminale) che rappresentano opzionalità: l'argomento può esserci o no. Ecco la rete di automi ricorsivi che modella la grammatica G:



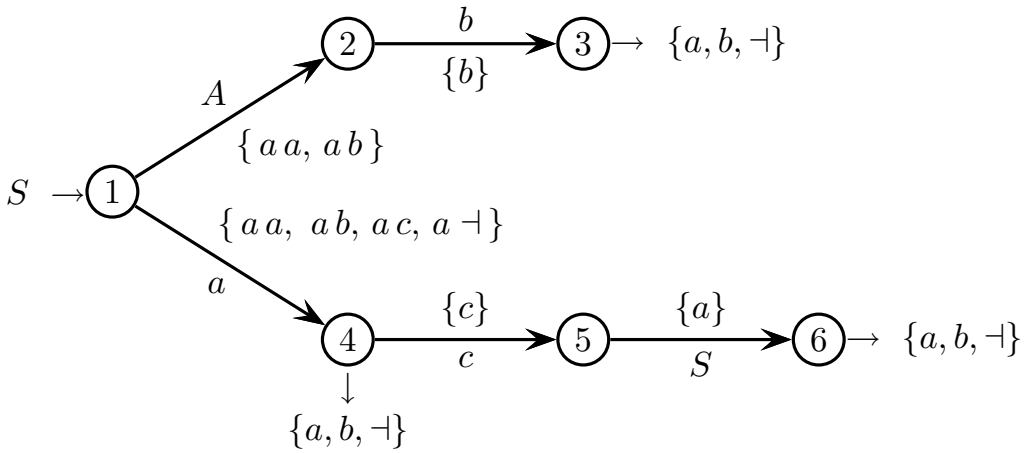
Entrambi gli automi di S e A sono deterministici sull'alfabeto totale.

(b) Ecco gli insiemi guida della grammatica G, con $k = 1$, calcolati sulla rappresentazione come rete ricorsiva di automi:



La biforcazione allo stato 1 nell'automa S ha insiemi guida non disgiunti, pertanto la grammatica G non è di tipo $LL(1)$. In G non ci sono altre violazioni alla condizione $LL(1)$.

(c) Ecco gli insiemi guida con $k = 2$ calcolati sulla biforcazione allo stato 1 (è l'unico punto dove sia necessario conoscerli):



Si vede che la biforcazione allo stato 1 ha ancora insiemi guida non disgiunti (le stringhe $a\,a$ e $a\,b$ sono comuni), pertanto la grammatica G non è neppure di tipo $LL(2)$.

2. Si consideri la grammatica G seguente (assioma S), non in forma estesa (BNF), definita sull'alfabeto terminale $\{ s, e \}$ (assioma S):

$$G \begin{cases} S & \rightarrow & S\ s\ E \ \mid \ E \\ E & \rightarrow & e\ E \ \mid \ s\ E \ \mid \ \varepsilon \end{cases}$$

Si risponda alle domande seguenti:

(a) Tramite il metodo di Earley si mostri la simulazione dell'analisi della stringa $s\,s\,e\,e \in L\,(G)$. Allo scopo si usi la tabella preparata di seguito.

(b) Si disegni un albero sintattico della stringa $s\,s\,e\,e$, evidenziando la corrispondenza tra i sottoalberi e le candidate di riduzione prodotte dall'algoritmo di analisi.

(c) (facoltativo) Si dica se la grammatica G sia ambigua o no, e in particolare se sia ambigua la stringa $s\,s\,e\,e$ e quanto valga il suo grado di ambiguità; in caso di ambiguità si disegnino tutti gli alberi sintattici di tale stringa.

Soluzione

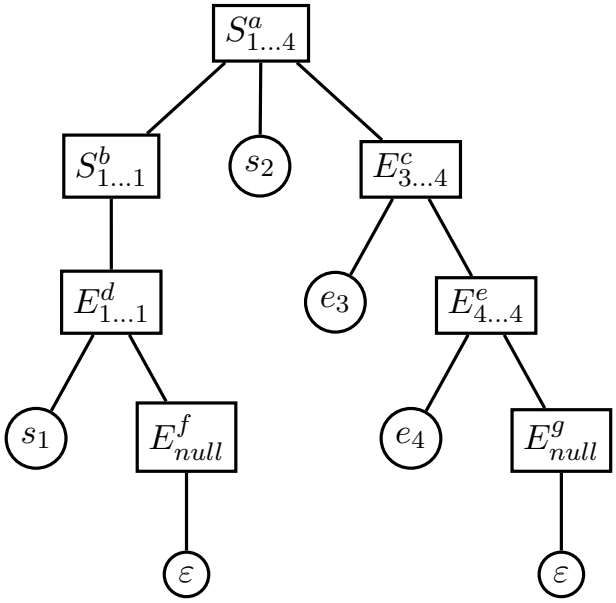
(a) Ecco la tabella di simulazione di Earley per la stringa $s\,s\,e\,e$:

Schema di simulazione dell'algoritmo di Earley

state 0	pos. s	state 1	pos. s	state 2	pos. e	state 3	pos. e	state 4	pos. $\dashv$
$S \rightarrow \bullet\ S\ s\ E$	0	$E \rightarrow s \bullet E$	0	$E \rightarrow s \bullet E$	1	$E \rightarrow e \bullet E$	2	$E \rightarrow e \bullet E$	3
$S \rightarrow \bullet\ E$	0	$S \rightarrow S\ s \bullet E$	0	$S \rightarrow S\ s \bullet E$	1	$E \rightarrow \bullet\ e\ E$	3	$E \rightarrow \bullet\ e\ E$	4
$E \rightarrow \bullet\ e\ E$	0	$E \rightarrow \bullet\ e\ E$	1	$E \rightarrow \bullet\ e\ E$	2	$E \rightarrow \bullet\ s\ E$	3	$E \rightarrow \bullet\ s\ E$	4
$E \rightarrow \bullet\ s\ E$	0	$E \rightarrow \bullet\ s\ E$	1	$E \rightarrow \bullet\ s\ E$	2	$E \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	3	$E^g \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	4
$E \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	0	$E^f \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	1	$E \rightarrow \bullet\ \varepsilon = \varepsilon \bullet$	2	$E \rightarrow e\ E \bullet$	2	$E^e \rightarrow e\ E \bullet$	3
$S \rightarrow E \bullet$	0	$E^d \rightarrow s\ E \bullet$	0	$E \rightarrow s\ E \bullet$	1	$E \rightarrow s\ E \bullet$	1	$E^c \rightarrow e\ E \bullet$	2
$S \rightarrow S \bullet s\ E$	0	$S \rightarrow S\ s\ E \bullet$	0	$S \rightarrow S\ s\ E \bullet$	1	$S \rightarrow S\ s\ E \bullet$	1	$E \rightarrow s\ E \bullet$	1
		$S^b \rightarrow E \bullet$	0	$E \rightarrow s\ E \bullet$	0	$E \rightarrow s\ E \bullet$	0	$S \rightarrow S\ s\ E \bullet$	1
		$S \rightarrow S \bullet s\ E$	0	$S \rightarrow S\ s\ E \bullet$	0	$S \rightarrow S\ s\ E \bullet$	0	$E \rightarrow s\ E \bullet$	0
				$S \rightarrow E \bullet$	0	$S \rightarrow E \bullet$	0	$S^a \rightarrow S\ s\ E \bullet$	0
				$S \rightarrow S \bullet s\ E$	0	$S \rightarrow S \bullet s\ E$	0	$S \rightarrow E \bullet$	0
								$S \rightarrow S \bullet s\ E$	0

Lo stato 4 contiene due candidate assiomatiche completate ($S \rightarrow S\ s\ E$ e $S \rightarrow E$) riferite allo stato 0, pertanto la stringa è riconosciuta.
Poiché le candidate riconoscenti sono due, la stringa riconosciuta è certamente ambigua di grado almeno 2
(il grado di ambiguità però potrebbe essere maggiore di 2).

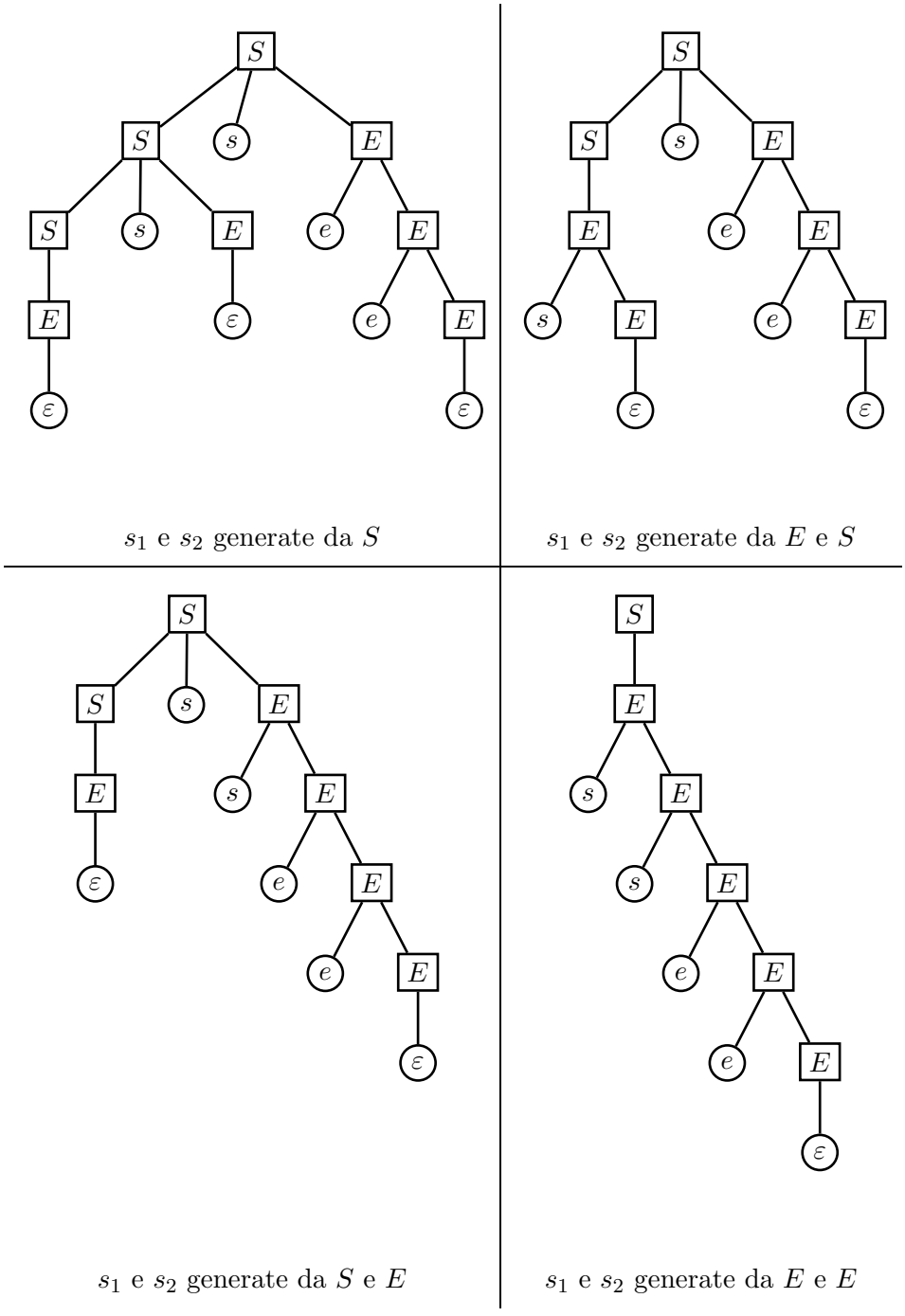
(b) Ecco un albero sintattico della stringa riconosciuta $s\,s\,e\,e$:



Le lettere riportate come apice permettono di rintracciare nella tabella le candidate di riduzione che danno luogo questo albero sintattico.

(c) Come già osservato prima, la stringa riconosciuta è certamente ambigua perché alla fine dell'analisi ci sono due candidate assiomatiche. Tanto basta per sapere che il suo grado di ambiguità vale almeno 2, ma potrebbe essere maggiore. Per determinarne il grado effettivo di ambiguità occorre pertanto un'analisi più approfondita.

Esaminando la grammatica G si vede che il terminale s è generato da entrambi i nonterminali S e E, mentre il terminale e è generato solo dal nonterminale E. Poiché la stringa $s\,s\,e\,e$ contiene due terminali s, l'intuito suggerisce che il suo grado effettivo di ambiguità valga esattamente $2 \times 2 = 4$. Per dimostrarlo rigorosamente bisogna esibire tutti gli alberi sintattici della stringa. Ecco i quattro alberi sintattici corrispondenti:



s_1 e s_2 generate da S

s_1 e s_2 generate da E e S

s_1 e s_2 generate da S e E

s_1 e s_2 generate da E e E

Basta poco per convincersi che questi sono i soli quattro alberi sintattici per la striga proposta, pertanto il grado effettivo di ambiguità vale esattamente 4. Il lettore può divertirsi a rintracciare le candidate di riduzione che danno origine a ciascuno di essi. L'albero ricostruito dato prima è quello in alto a destra.

4 Traduzione e analisi semantica 20%

1. Si consideri un linguaggio sorgente L per modellare un programma assembler. Le istruzioni macchina sono le seguenti (a lato è data l'interpretazione come commento):

```
Istruzione:              Interpretazione (commento):
add s, d                 d <- s + d
add 1, d                 d <- d + 1
add -1, d                d <- d - 1
nop                      nessuna operazione
```

   Gli argomenti "s" e "d" indicano due registri generici del processore, sorgente e destinazione rispettivamente; gli argomenti "1" e "-1" sono costanti.

   Il programma è semplicemente una lista di istruzioni del tipo sopra indicato, separate dal carattere speciale "⟨NL⟩" (newline o ritorno a capo).

   Il programma può contenere errori del tipo seguente: l'istruzione "add" ha come argomento destinazione una costante (per esempio "add s, 1").

   Si vuole progettare un traduttore sintattico T che traduca il programma sostituendo le istruzioni macchina correnti con nuove istruzioni ottimizzate (se possibile) nel modo seguente (a lato è data una spiegazione come commento):

```
Istruzione:          Sostituzione:       Spiegazione
add s, d             add s, d            resta inalterata
add 1, d             inc d               viene ottimizzata
add -1, d            dec d               viene ottimizzata
nop                                      viene eliminata
add s, 1             errore              segnala errore
add s, -1            errore              segnala errore
```

   Si risponda alle domande seguenti:

   (a) Si dica, motivando brevemente la risposta, quale potrebbe essere il modello di traduzione più adatto (automa a stati finiti, a pila, schema o grammatica di traduzione, ecc) per realizzare il traduttore T descritto sopra.

   (b) Scelto il modello, si realizzi il traduttore T descritto sopra.

   (c) (facoltativo) Si dica se il traduttore T realizzato sia deterministico o no, e se fosse indeterministico si provi a individuarne una possibile realizzazione deterministica (magari cambiando modello di trasduttore, se servisse).

Soluzione

(a) Per realizzare il trasduttore si possono usare modelli diversi: un automa trasduttore (ossia IO-automa) finito oppure uno schema sintattico di traduzione. Una scelta meno buona sarebbe quella di usare un automa trasduttore a pila, perché questa traduzione non ha bisogno di pila né per controllare la correttezza

delle stringhe sorgente né per calcolarne l'immagine; pertanto una macchina a pila sarebbe inutilmente complicata per tale scopo.

Tra i due modelli di cui sopra si sceglie di usare lo schema piuttosto che l'automa a stati finiti, perché dà luogo a una soluzione più espressiva, ordinata e breve. Invece un automa finito imporrebbe di descrivere in dettaglio gli stati interni del trasduttore, con qualche evitabile fastidio.

(b) La traduzione T è definita dallo schema di traduzione seguente:

$G_{sorgente}$:	$G_{destinazione}$:
$S \rightarrow I\ \langle\mathsf{NL}\rangle\ S$	$S \rightarrow I\ \langle\mathsf{NL}\rangle\ S$
$S \rightarrow I$	$S \rightarrow I$
$I \rightarrow \mathsf{add},\ s,\ d$	$I \rightarrow \mathsf{add},\ s,\ d$
$I \rightarrow \mathsf{add},\ s,\ 1$	$I \rightarrow \mathsf{error}$
$I \rightarrow \mathsf{add},\ 1,\ s$	$I \rightarrow \mathsf{inc},\ s$
$I \rightarrow \mathsf{add},\ -1,\ s$	$I \rightarrow \mathsf{dec},\ s$
$I \rightarrow \mathsf{nop}$	$I \rightarrow \varepsilon$

(c) Dallo schema si può costruire il programma del trasduttore realizzando un analizzatore sintattico del linguaggio sorgente e aggiungendo le azioni di stampa della stringa immagine.

Scegliendo la tecnica discendente $LL(k)$ si vede che la grammatica $G_{sorgente}$ non è deterministica per $k = 1$. Per esempio le prime due regole hanno il prefisso comune I che impedisce una scelta deterministica. Conviene riscrivere lo schema di traduzione mettendo in evidenza i prefissi più lunghi comuni alle alternative:

$G_{sorgente}$:	$G_{destinazione}$:
$S \rightarrow I\ A$	$S \rightarrow I\ A$
$A \rightarrow \langle\mathsf{NL}\rangle\ S$	$A \rightarrow \langle\mathsf{NL}\rangle\ S$
$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
$\ldots$	$\ldots$

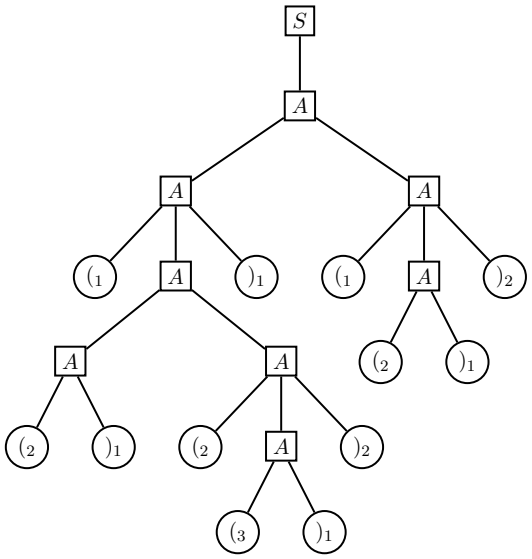
Analogamente si dovrebbe fare, con qualche fatica, per rendere deterministiche le regole alternative che espandono il nonterminale I (qui tale sviluppo è omesso).

Risulta più semplice realizzare il trasduttore ascendente tramite la tecnica $LR(1)$. Le condizioni affinché ciò sia possibile vanno tuttavia verificate. Eccole:

- Lo schema di traduzione deve essere in forma postfissa. Soltanto la regola $S \rightarrow I\ \langle\mathsf{NL}\rangle\ S$ vìola tale condizione, perché il terminale $\langle\mathsf{NL}\rangle$ è seguito dal nonterminale S. Però suddividendo la regola nel modo seguente:
$$S \rightarrow I\ N\ S \qquad S \rightarrow I \qquad N \rightarrow \langle\mathsf{NL}\rangle$$
si elimina il problema.
- La grammatica sorgente, dopo la modifica precedente, deve essere $LR(1)$. Questa condizione va verificata costruendo l'automa pilota (qui omesso).

2. Si consideri il linguaggio L di Dyck con parentesi tonde aperte e chiuse, cioè di alfabeto $\Sigma = \{\text{`('}, \text{`)'}\}$, esclusa la stringa vuota ε. Ecco il supporto sintattico G che genera tale linguaggio L (assioma S):

$$G \begin{cases} S & \rightarrow & A \\ A & \rightarrow & A\ A \\ A & \rightarrow & \text{`('}\ A\ \text{`)'} \\ A & \rightarrow & \text{`('}\ \text{`)'} \end{cases}$$



Si vogliono associare alle parentesi aperte "(" e chiuse ")" un indice di profondità di annidamento e un indice di posizione nella lista dei fratelli, rispettivamente (entrambi gli indici sono numeri interi ≥ 1). Ecco un esempio:

$$(_1\ (_2\ )_1\ (_2\ (_3\ )_1\ )_2\ )_1\ (_1\ (_2\ )_1\ )_2$$

L'albero sintattico con indici della stringa di esempio è mostrato sopra a destra del supporto sintattico G. Inoltre si vuole calcolare anche l'indice di altezza totale della stringa, coincidente con l'indice di profondità del nido più interno di parentesi.

Sono assegnati gli attributi seguenti:

- α associato a S: altezza totale della stringa
- π associato a A: profondità del nido corrente di parentesi
- λ associato a A: posizione del nido corrente di parentesi

Se occorre o lo si ritiene opportuno si possono estendere (ad altri nonterminali) gli attributi già assegnati o aggiungerne di nuovi.

Si risponda alle domande seguenti:

(a) Si completi se necessario l'elenco degli attributi, e si dica quali attributi siano sinistri e quali destri (sintetizzati ed ereditati). Si usi la tabella predisposta di séguito.

(b) Si scrivano le funzioni semantiche associate alle regole della grammatica G. Si usi la tabella predisposta di séguito.

(c) Si disegnino i grafi delle dipendenze degli attributi per ciascuna regola della grammatica G.

(d) Si traccino i grafi dei fratelli delle regole e si verifichi se la grammatica con attributi sia di tipo a una scansione (one-sweep) ed eventualmente di tipo L.

Soluzione

(a) Ecco l'elenco degli attributi di G:

attributi da usare per la grammatica

tipo	nome	(non)terminali	dominio	significato
già dati nel testo dell'esercizio				
sx	α	S	num. int.	altezza totale di stringa
dx	π	A	num. int.	profondità di nido
dx	λ	A	num. int.	posizione di nido
eventualmente da estendere e / o aggiungerne di nuovi				
sx	α	A	num. int.	altezza totale di stringa
sx	μ	A	num. int.	ultima posizione del nido di parentesi nella lista dei fratelli

L'attributo α è esteso al nonterminale A. L'attributo μ è nuovo e serve per contare i nidi di parentesi nella lista dei fratelli. Si noti che il solo attributo (destro) λ non basta per contare correttamente i nidi fratelli; occorre anche l'attributo (sinistro) μ, il cui valore risale nel sottoalbero di regole $A \to A\ A$ innestate e poi ridiscende sotto forma di attributo (destro) λ (si vedano le funzioni semantiche).

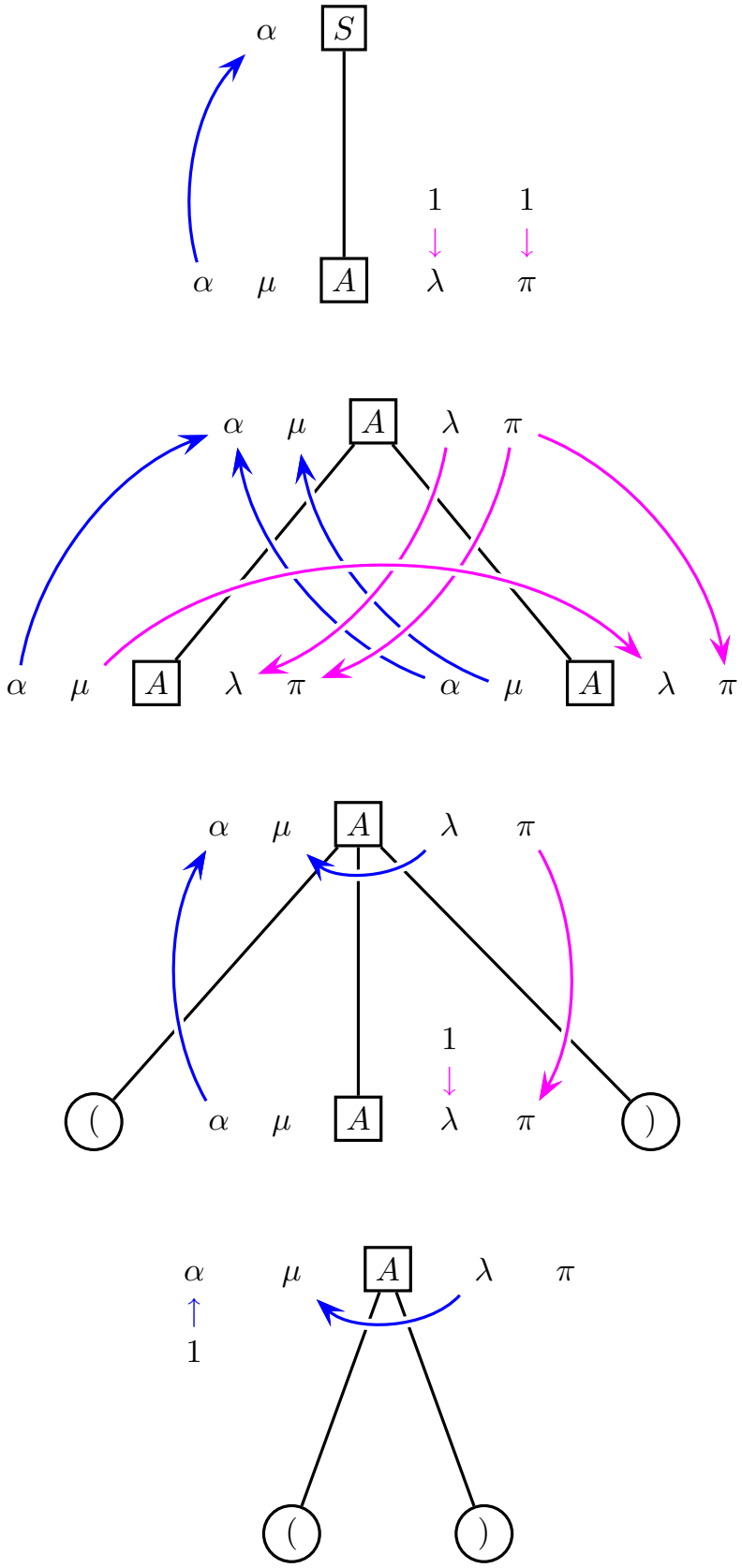
(b) Ecco le funzioni semantiche di G:

sintassi	*funzioni semantiche*
$S_0 \rightarrow A_1$	$\alpha_0 = \alpha_1$ $\lambda_1 = 1$ $\pi_1 = 1$
$A_0 \rightarrow A_1\ A_2$	$\alpha_0 = \max(\ \alpha_1,\ \alpha_2\ )$ $\mu_0 = \mu_2$ $\lambda_1 = \lambda_0$ $\lambda_2 = \mu_1 + 1$ $\pi_1 = \pi_0$ $\pi_2 = \pi_0$
$A_0 \rightarrow$ ‘(’ A_1 ‘)’	$\alpha_0 = \alpha_1 + 1$ $\mu_0 = \lambda_0$ $\lambda_1 = 1$ $\pi_1 = \pi_0 + 1$
$A_0 \rightarrow$ ‘(’ ‘)’	$\alpha_0 = 1$ $\mu_0 = \lambda_0$

La sola aggiunta rilevante è l'attributo sinistro μ (con le relative funzioni semantiche), che risale lungo il sottoalbero generato dalla regola $A_0 \rightarrow A_1\ A_2$ per tenere traccia della posizione dell'ultimo nido di parentesi generato da A_1 nella lista dei fratelli che prosegue con A_2. Infatti l'attributo λ è destro e da solo non potrebbe svolgere tale funzione.

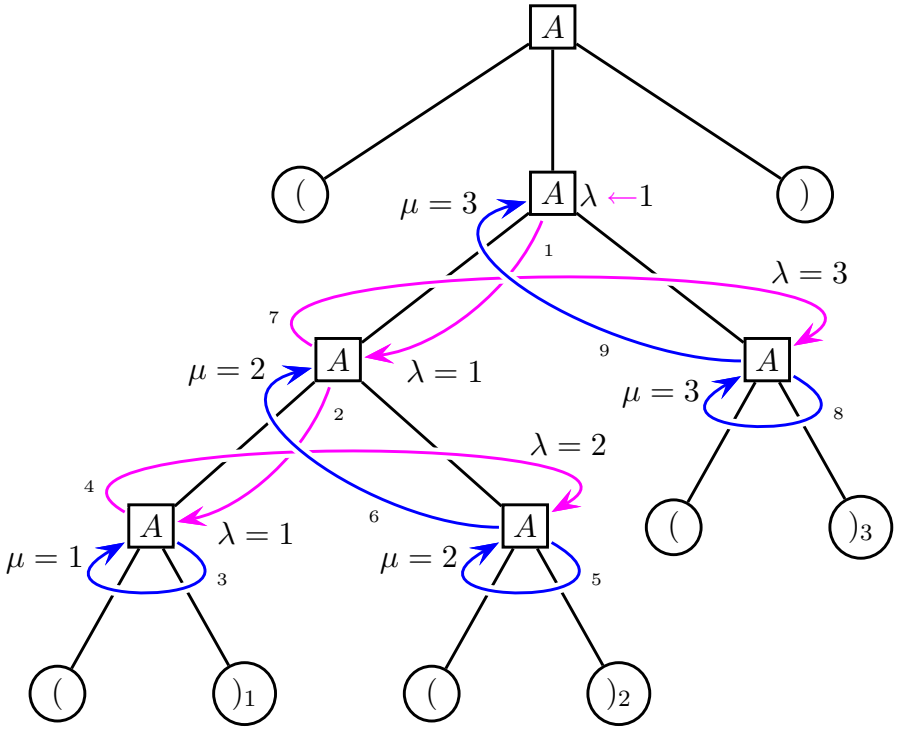
Per risolvere l'esercizio è sufficiente calcolare correttamente gli attributi. Comunque volendo si possono aggiungere le funzioni per stamparli oppure definire un attributo τ di tipo stringa dove costruire progressivamente per concatenamento la stringa parentetica con indici, nel modo usuale, fino alla radice S.

(c) Ecco i grafi di dipendenza degli attributi per ciascuna regola di G:



Alcuni attributi vengono soltanto inizializzati. I grafi di dipendenza sono tutti aciclici e molto semplici. È facile convincersi che anche gli alberi sintattici decorati con dipendenze sono aciclici. Pertanto la grammatica G è corretta.

Benché non sia affatto necessario per risolvere l'esercizio, è però istruttivo seguire su un frammento di albero sintattico il flusso degli attributi μ e λ, giacché μ è l'unico attributo nuovo ed è strettamente legato a λ. Ecco l'esempietto:

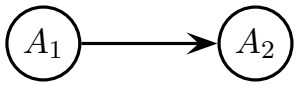


calcolo dell'indice di posizione nella stringa (( )$_1$ ( )$_2$ ( )$_3$) con tre nidi fratelli
(le frecce che aggiornano λ e μ sono colorate in rosa e blu rispettivamente)
(i numeretti da 1 a 9 sugli archi indicano progressivamente il percorso)

Si noti come avvenga il processo di calcolo e incremento dell'indice della lista dei fratelli da associare alla parentesi chiuse: nella regola $A \to A\ A$ l'attributo destro λ fa scendere tale indice da padre a primo figlio e poi lo passa da primo a secondo fratello (ma coinvolgendo anche l'attribito μ), e nel passaggio tra fratelli l'indice viene incrementato; invece l'attributo sinistro μ fa salire l'indice da secondo fratello a padre, e da qui l'indice verrà passato ad altri fratelli se ce ne sono. Né λ né μ da soli possono svolgere tale compito per intero, giacché esso prevede sia di scendere e passare di lato nell'albero (sono due azioni caratteristiche da attributo destro), sia di salire nell'albero (è un'azione caratteristica da attributo sinistro); pertanto questi due attributi sono entrambi necessari e devono cooperare.

Al riguardo va rilevato come l'esempio di stringa e albero dato nel testo dell'esercizio sia un po' addomesticato (volutamente), in quanto l'unica lista di nidi fratelli ha solo due elementi e dove pertanto non si verifica il caso di almeno due regole $A \to A\ A$ innestate (là infatti ne figura una sola), ciò che invece succede nell'esempietto dato sopra; del resto non bisogna dimenticare di chiedersi in quale misura un esempio dato sia rappresentativo di tutta quanta la realtà.

(d) L'unico grafo dei fratelli rilevante è quello associato alla regola $A_0 \to A_1\ A_2$, giacché le altre regole hanno un solo nodo nonterminale figlio o addirittura nessuno e pertanto i loro grafi di dipendenza sono tutti banali. Eccolo:



Qui l'ordinamento topologico dei nodi fratelli è quasi banale: A_1 precede A_2. Esso coincide semplicemente con l'ordine sintattico dei nodi nella regola.

È facile constatare che la grammatica G soddisfa alla condizione a una scansione. Ecco la verifica dei punti in cui si articola la condizione:

- i grafi delle dipendenze delle regole di G (e in generale degli alberi sintattici) sono tutti aciclici
- in ogni regola di G gli attributi sinistri del nodo padre dipendono solo da attributi (qualsiasi) dei nodi figli o da attributi destri nel nodo padre stesso
- in ogni regola di G nessun attributo destro di un nodo figlio dipende da attributi sinistri del nodo padre
- e i grafi dei fratelli delle regole di G sono tutti aciclici (in realtà come osservato sopra qui è significativo solo quello della regola $A \to A\ A$)

Pertanto la grammatica G è senz'altro di tipo a una scansione (one-sweep).

Per quanto riguarda la condizione L, come si capisce dal grafo dei fratelli la procedura semantica della regola $A_0 \to A_1\ A_2$ (l'unica significativa) può esaminare i nodi fratelli A_1 e A_2 in ordine sintattico; pertanto le funzioni semantiche della grammatica G sono compatibili con l'analisi sintattica discendente (top-down). In conclusione la grammatica G è di tipo L.

Tuttavia non è possibile integrare gli analizzatori semantico e sintattico a discesa ricorsiva (LL), poiché a causa della regola con ricorsione bilatera $A \to A\ A$ il supporto sintattico G è ambiguo e dunque non è di tipo $LL(k)$ per nessun $k \geq 1$.