# Linguaggi Formali e Compilatori
# (Formal Languages and Compilers)

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

## Prova scritta - 16 settembre 2010 - 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:

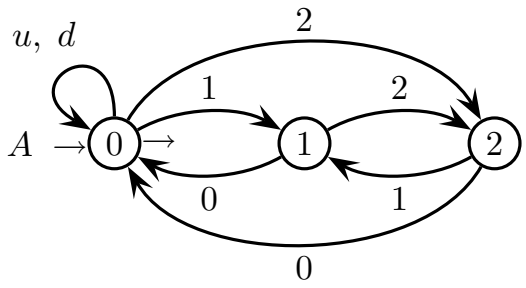
MATRICOLA: FIRMA:

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. Un ascensore serve tre piani numerati 0, 1 e 2. I tasti '0', '1' e '2' comandano il piano da raggiungere secondo il grafo dell'automa $A$ seguente:




Prima di entrare in cabina al piano 0, il passeggero deve inserire una carta che porta un codice con il suo diritto di accesso ai piani: *uno* dà accesso ai piani 0 e 1, e *due* permette l'accesso a tutti i piani. Dopo il ritorno al piano 0 un solo passeggero può entrare, e così via.

Preso l'alfabeto terminale $\Sigma = \{ d,\ u,\ 0,\ 1,\ 2 \}$, si consideri il linguaggio regolare $C$ (con $C \subseteq \Sigma^*$) delle sequenze che rispettano i diritti di accesso nel modo spiegato dagli esempi e contresempi seguenti.

Esempi:

$d\ d\ u\ 1\ 0 \qquad u\ 1\ 0\ d\ 2\ 0 \qquad u\ 1\ 0\ u\ d\ 2\ 0 \qquad u\ 1\ 0\ d\ 2\ 1\ 0$

$d\ 1\ 0\ d\ 2\ 0\ d\ 1\ 2\ 0\ 1\ 0$

Contresempi:

$u\ 2\ 0 \qquad u\ 1\ 2\ 0 \qquad u\ 1\ 0\ d\ u\ 2\ 0$

Si risponda alle domande seguenti:

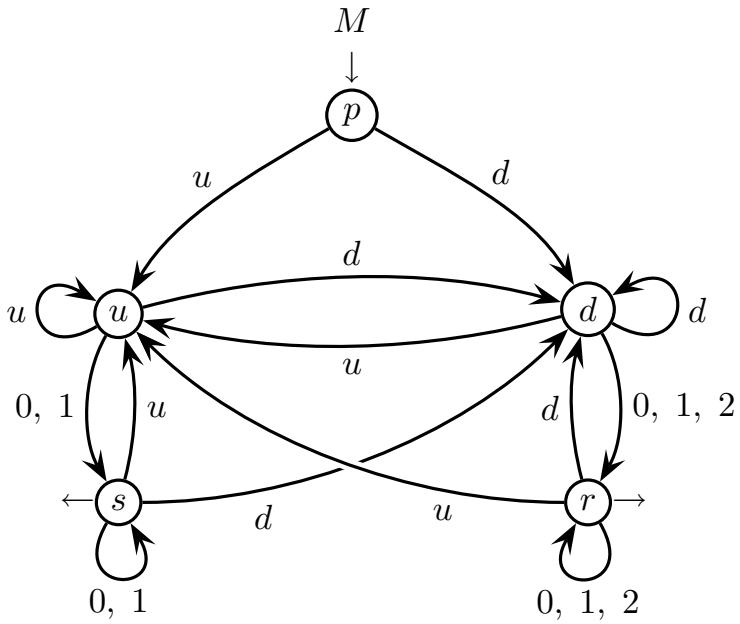
(a) Si progetti il riconoscitore del linguaggio $C$ sotto forma di *macchina prodotto cartesiano* di due automi finiti, ossia:
- l'automa $A$ disegnato sopra
- e un secondo automa $M$ (deterministico o no), da progettare

(b) (facoltativa) Si dica se il linguaggio $L(A)$ è locale, motivando la risposta.

---

## Soluzione

(a) Per trovare un automa $M$ tale che $L(A) \cap L(M) = C$, si considerino i vincoli da imporre, dati dalle stringhe di esempio e contresempio. Si vede che: la stringa deve iniziare con codice di accesso $u$ o $d$; la stringa non può finire né con $u$ né con $d$; e per ogni comparsa del codice di tasto '2' nella stringa, l'ultimo codice di accesso precedente, anche non immediatamente, dev'essere $d$. Nessuno di questi vincoli è già imposto dall'automa $A$ e dunque essi vanno imposti da $M$. Inoltre si vede che il codice di tasto '0' non può essere consecutivo né a $u$ né a $d$ e che la stringa non può terminare né con '1' né con '2', ma tali vincoli sono già imposti da $A$ e dunque non serve ripeterli in $M$. Pertanto l'insieme delle stringhe che soddisfano detti vincoli è riconosciuto dall'automa $M$ qui sotto:




È facile verficare che l'automa $M$ è minimo. La macchina prodotto $A \times M$ ha gli stati coppia $\{\ 0,\ 1,\ 2\ \} \times \{\ p,\ u,\ d,\ s,\ r\ \}$, con stato iniziale $(\ 0,\ p\ )$ e stati finali $(\ 0,\ s\ )$ e $(\ 0,\ r\ )$. Se ne potrebbero costruire le transizioni congiungendo (and logico) le transizioni delle due macchine $A$ e $M$.

(b) Il linguaggio $L(A)$ è locale, proprietà riscontrabile in due modi diversi.
Primo, gli insiemi seguenti

$$Inizi = \{\ u\ d\ 1\ 2\ \} \qquad Fini = \{\ 0\ \}$$

$$Digrammi = \{\ 01\ \ 0d\ 0u \quad 10\ 12 \quad 20\ 21 \quad d1\ d2\ dd\ du \quad u1\ u2\ ud\ uu\ \}$$

non permettono di produrre stringhe che non siano in $L(A)$.
Secondo, è facile trasformare l'automa $A$ nella forma locale, avente gli stati $\{\ q_0,\ 0,\ 1,\ 2,\ d,\ u\ \}$ caratterizzati dal fatto che, per ogni carattere terminale, nello stato corrispondente entrano soltanto archi etichettati da tale carattere.

2. È data l'espressione regolare $E$ seguente:

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

Si risponda alle domande seguenti:

(a) Si mostri una frase ambigua appartenente al linguaggio definito dall'espressione regolare $E$. Si discuta se il grado di ambiguità di $E$ è limitato o illimitato.

(b) Si costruisca un automa deterministico $A$ equivalente all'espressione regolare $E$, tramite un metodo sistematico.

---

## Soluzione

(a) L'espressione regolare $E_{\#}$ seguente, con generatori numerati progressivamente:

$$E_{\#} = \left( \; b_1^* \; ( \; b_2 \; \mid \; c_3 \; ) \; a_4 \; b_5^+ \; \right)^*$$

mette in evidenza due modi diversi di ottenere la stringa $b\,a\,b\,b\,b\,a\,b$:

$$b_2 \; a_4 \; b_5 \; b_5 \; b_2 \; a_4 \; b_5 \qquad b_2 \; a_4 \; b_5 \; b_1 \; b_2 \; a_4 \; b_5$$

Tanto basta per concludere che l'espressione regolare $E$ è ambigua.

Raffinando l'analisi, si vede che il grado di ambiguità dell'espressione regolare $E$ è illimitato. Si considerino per esempio le stringhe di tipo seguente:

$$b \; a \; b \; b^+ \; b \; a \; b$$

Succede che al crescere del numero di lettere $b$ frapposte alle lettere $a$, il numero di derivazioni della stringa aumenta. Infatti la sequenza di $b$ fra le due $a$, ossia $b^n$ $(n \geq 3)$, ha la struttura seguente:

$$b_5 \; b_5^h \; b_1^k \; b_2 \qquad\qquad h + k \geq 1$$

Per esponenti $h$ e $k$ differenti tali sequenze sono tutte generate in modo diverso, e al crescere di $n = h + k + 2$ il loro numero cresce. Per esempio il caso di ambiguità mostrato sopra viene da $h, k = 1, 0$ e $h, k = 0, 1$, con $n = 3$.

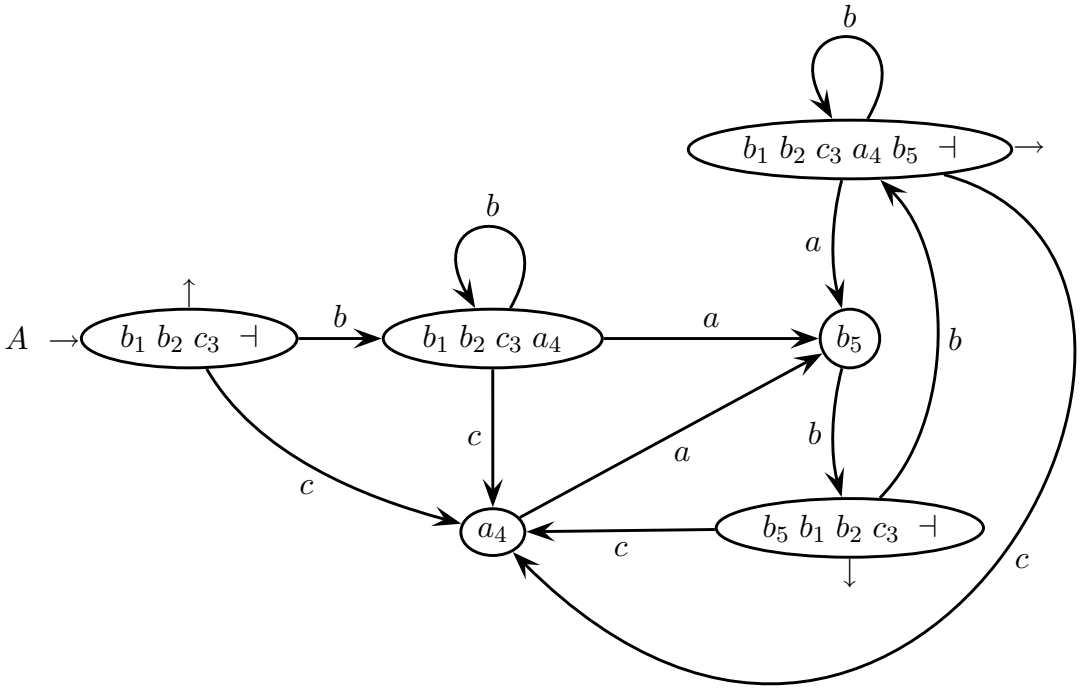
Però il grado di ambiguità dell'espressione regolare $E$ non è infinito, giacché il linguaggio $L(E)$ non contiene alcuna specifica stringa che sia generata da $E$ con un numero infinito di derivazioni differenti.

(b) Si sceglie di costruire il riconoscitore deterministico del linguaggio $L(E)$ tramite il metodo di Berry-Sethi (ma si potrebbe procedere anche in altro modo).
Ecco l'analisi degli inizi e dei seguiti dell'espressione regolare $E$, con generatori numerati progressivamente:

$$E_{\#} = \left( \; b_1^* \;\; ( \; b_2 \;\; | \;\; c_3 \; ) \;\; a_4 \; b_5^+ \; \right)^*$$

| inizi | $b_1 \; b_2 \; c_3 \; \dashv$ |
|---|---|
| generatore | seguiti |
| $b_1$ | $b_1 \; b_2 \; c_3$ |
| $b_2$ | $a_4$ |
| $c_3$ | $a_4$ |
| $a_4$ | $b_5$ |
| $b_5$ | $b_5 \; b_1 \; b_2 \; c_3 \; \dashv$ |

Ed ecco l'automa deterministico $A$ equivalente all'espressione regolare $E$, secondo il metodo di Berry-Sethi:




L'automa deterministico $A$ ha tre stati finali, uno dei quali è quello iniziale. Infatti la stringa vuota $\varepsilon$ appartiene al linguaggio generato dall'espressione regolare $E$ e pertanto l'automa $A$ la deve riconoscere.
Non è difficile concludere che l'automa $A$ così trovato è minimo: basta esaminare le coppie di stati e constatare che tutti sono facilmente distinguibili.

## 2 Grammatiche libere e automi a pila 20%

1. Si considerino le ben note espressioni aritmetiche con le operazioni di addizione '+', moltiplicazione '×', le parentesi tonde aperta '(' e chiusa ')', e variabili simbolizzate tramite il carattere $a$. Come d'uso la moltiplicazione ha precedenza sull'addizione.

   Si risponda alle domande seguenti:

   (a) Si scriva la grammatica $G_1$, non ambigua e in forma non estesa (BNF), delle espressioni aritmetiche con il vincolo aggiuntivo di *contenere solo* un numero pari di addizioni (zero compreso).

   (b) (facoltativa) Si scriva la grammatica $G_2$, non ambigua e in forma non estesa (BNF), delle espressioni aritmetiche con il vincolo aggiuntivo di *contenere solo* un numero dispari di (sotto)espressioni parentetiche.

---

## Soluzione

(a) Ecco la nota grammatica $G$ (non ambigua e non estesa) delle espressioni aritmetiche da considerare, senza vincoli di parità sulle addizioni (assioma $E$):

$$G \begin{cases} E & \to & T \ + \ E \ \mid \ T \\ T & \to & F \ \times \ T \ \mid \ F \\ F & \to & a \ \mid \ ( \ E \ ) \end{cases}$$

Per ottenere la grammatica $G_1$, si possono differenziare i nonterminali $E$, $T$ e $F$ secondo essi generino sottostringhe contenenti un numero pari (even) o dispari (odd) di addizioni. Ecco pertanto la grammatica $G_1$ (assioma $E$):

$$G_1 \begin{cases} E & \to & E_e \\ E_e & \to & T_e \ + \ E_o \ \mid \ T_o \ + \ E_e \ \mid \ T_e \\ E_o & \to & T_e \ + \ E_e \ \mid \ T_o \ + \ E_o \ \mid \ T_o \\ T_e & \to & F_e \ \times \ T_e \ \mid \ F_o \ \times \ T_o \ \mid \ F_e \\ T_o & \to & F_e \ \times \ T_o \ \mid \ F_o \ \times \ T_e \ \mid \ F_o \\ F_e & \to & a \ \mid \ ( \ E_e \ ) \\ F_o & \to & ( \ E_o \ ) \end{cases}$$

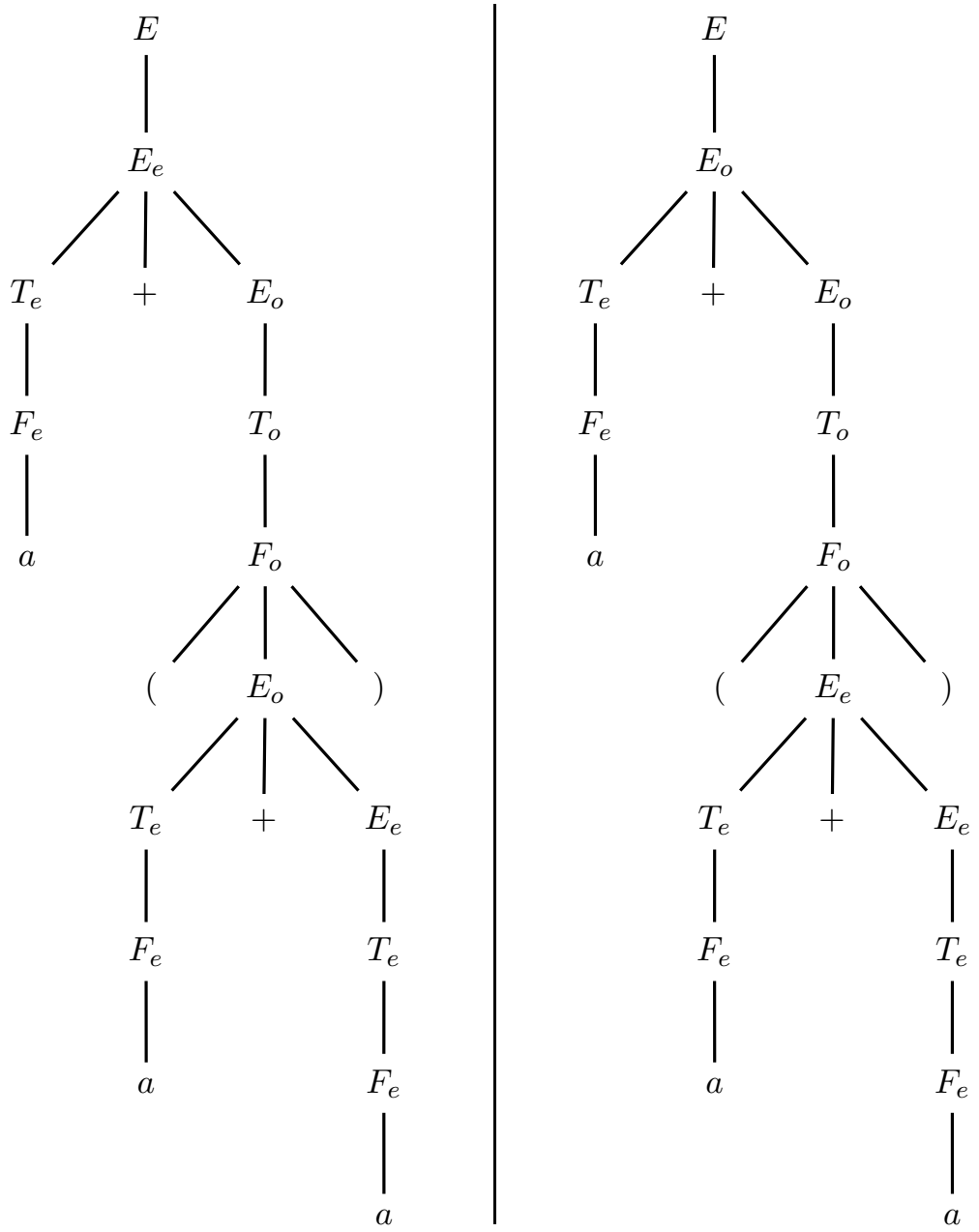
(b) Si può ripartire dalla grammatica $G$, e aggiungere il vincolo differenziando i nonterminali e valutando la parità delle sottoespressioni parenteriche. Ecco dunque la grammatica $G_2$ (assioma $E$):

$$G_2 \begin{cases} E & \to & E_o \\ E_e & \to & T_e \ + \ E_e \ \mid \ T_o \ + \ E_o \ \mid \ T_e \\ E_o & \to & T_e \ + \ E_o \ \mid \ T_o \ + \ E_e \ \mid \ T_o \\ T_e & \to & F_e \ \times \ T_e \ \mid \ F_o \ \times \ T_o \ \mid \ F_e \\ T_o & \to & F_e \ \times \ T_o \ \mid \ F_o \ \times \ T_e \ \mid \ F_o \\ F_e & \to & a \ \mid \ ( \ E_o \ ) \\ F_o & \to & ( \ E_e \ ) \end{cases}$$

Le regole di $G_2$ sono simili a quelle della grammatica $G_1$, ma aggiustate per contare modulo 2 le parentesi e non le addizioni.

Per illustrare il funzionamento delle due grammatiche $G_1$ e $G_2$, ecco due alberi sintattici per la medesima stringa generata sia da $G_1$ sia da $G_2$:

$$a \ + \ ( \ a \ + \ a \ )$$




Strutturalmente, ossia a prescindere dai nomi dei nodi interni, i due alberi di $G_1$ e $G_2$ sono identici, e del resto coincidono con l'albero della grammatica $G$ delle espressioni aritmetiche senza vincoli aggiuntivi di parità; ma tra gli alberi di $G_1$ e $G_2$ cambia il modo di codificare la parità mediante i nomi dei nonterminali.

2. Si consideri un frammento del linguaggio *Attempto Controlled English* (o *ACE*) per modellare una parte semplificata della grammatica inglese, illustrato così:

   (a) Frase semplice o Simple Sentence $SS$:

   $$\overbrace{\overbrace{a\ customer}^{NP}\ \overbrace{inserts}^{VP}\ \overbrace{the\ card}^{NP}}^{SS}\ .$$

   dove $NP$ è una NounPhrase e $VP$ una VerbPhrase. I sostantivi come *customer* e *card* sono indicati dal simbolo generico $N$. Possono essere preceduti dall'articolo indeterminativo *a* o determinativo *the*, indifferentemente.

   (b) Una $NP$ può anche contenere uno o più aggettivi, denotati dal simbolo $A$:

   $$a\ \overbrace{rich}^{A}\ customer\ inserts\ a\ \overbrace{small}^{A}\ and\ \overbrace{green}^{A}\ card\ .$$

   Se gli aggettivi sono due vanno separati con la congiunzione *and*, se sono tre o più vanno separati con virgola tranne l'ultimo che va preceduto dalla congiunzione *and*, come in *a small, thin and green card* .

   (c) Ci sono due tipi di verbo: transitivo $TV$ e intransitivo $IV$

   $$a\ customer\ \overbrace{inserts}^{TV}\ the\ card\ . \qquad a\ customer\ \overbrace{waits}^{IV}\ .$$

   Il verbo ha sempre il soggetto espresso. Tuttavia l'oggetto del verbo transitivo può mancare: *the customer calls* .

   (d) Due o più frasi semplici $SS$ sono coordinabili tramite le congiunzioni *or* e *and*, e anche tramite le congiunzioni *, or* e *, and* (si noti la virgola !). La congiunzione *and* prende precedenza sulla congiunzione *or*, ma le congiunzioni con virgola cedono precedenza a quelle senza virgola, nel modo illustrato sotto:

   $$\underbrace{the\ bell\ rings}\ or\ \overbrace{\underbrace{the\ girl\ waits}\ and\ \underbrace{the\ boy\ opens\ the\ door}}\ .$$

   $$\overbrace{\underbrace{the\ bell\ rings}\ or\ \underbrace{the\ girl\ waits}}\ ,and\ \underbrace{the\ boy\ opens\ the\ door}\ .$$

   Però nella frase ci può essere una sola congiunzione con virgola.

L'alfabeto terminale del linguaggio è dunque il seguente:

$$\Sigma = \{\ a,\ the,\ N,\ A,\ IV,\ TV,\ and,\ or,\ ',',\ '.'\ \}$$

Si scriva una grammatica, non ambigua e in forma estesa (EBNF), per il frammento di linguaggio *ACE* illustrato.

---

## Soluzione

Ecco la grammatica richiesta per modellare il linguaggio $ACE$ (assioma ACE):

$$
\begin{cases}
\langle \mathsf{ACE} \rangle \;\to\; \langle \mathsf{ACE} \rangle_1 \;\big[\; \text{`,'} \;(\; \mathsf{or} \;\mid\; \mathsf{and} \;)\; \langle \mathsf{ACE} \rangle_1 \;\big]\; \text{`.'} \\
\hline
\langle \mathsf{ACE} \rangle_1 \;\to\; \langle \mathsf{ACE} \rangle_2 \;\big(\; \mathsf{or}\; \langle \mathsf{ACE} \rangle_2 \;\big)^* \\
\hline
\langle \mathsf{ACE} \rangle_2 \;\to\; \langle \mathsf{SS} \rangle \;\big(\; \mathsf{and}\; \langle \mathsf{SS} \rangle \;\big)^* \\
\hline
\langle \mathsf{SS} \rangle \;\to\; \langle \mathsf{NP} \rangle\; \langle \mathsf{VP} \rangle\; \big[\; \langle \mathsf{NP} \rangle \;\big] \\
\hline
\langle \mathsf{NP} \rangle \;\to\; \big[\; \mathsf{a} \;\mid\; \mathsf{the} \;\big]\; \big[\; \langle \mathsf{A\_LIST} \rangle \;\big]\; N \\
\langle \mathsf{VP} \rangle \;\to\; TV \;\mid\; IV \\
\hline
\langle \mathsf{A\_LIST} \rangle \;\to\; A \;\big[\; (\; \text{`,'}\; A \;)^*\; \mathsf{and}\; A \;\big]
\end{cases}
$$

Le parentesi quadre indicano opzionalità. Si noti come la precedenza tra congiunzioni senza virgola sia imposta stratificando le regole, come nelle espressioni aritmetiche. Tuttavia le congiunzioni con virgola sono modellate a pari livello (infatti entrambe compaiono nella medesima regola), giacché in ogni caso nella frase ne può figurare una sola e dunque imporre precedenza anche tra loro due sarebbe del tutto superfluo. Però entrambe compaiono nella regola assiomatica, a livello superiore delle regole per le congiunzioni senza virgola, giacché cedono precedenza a queste ultime.

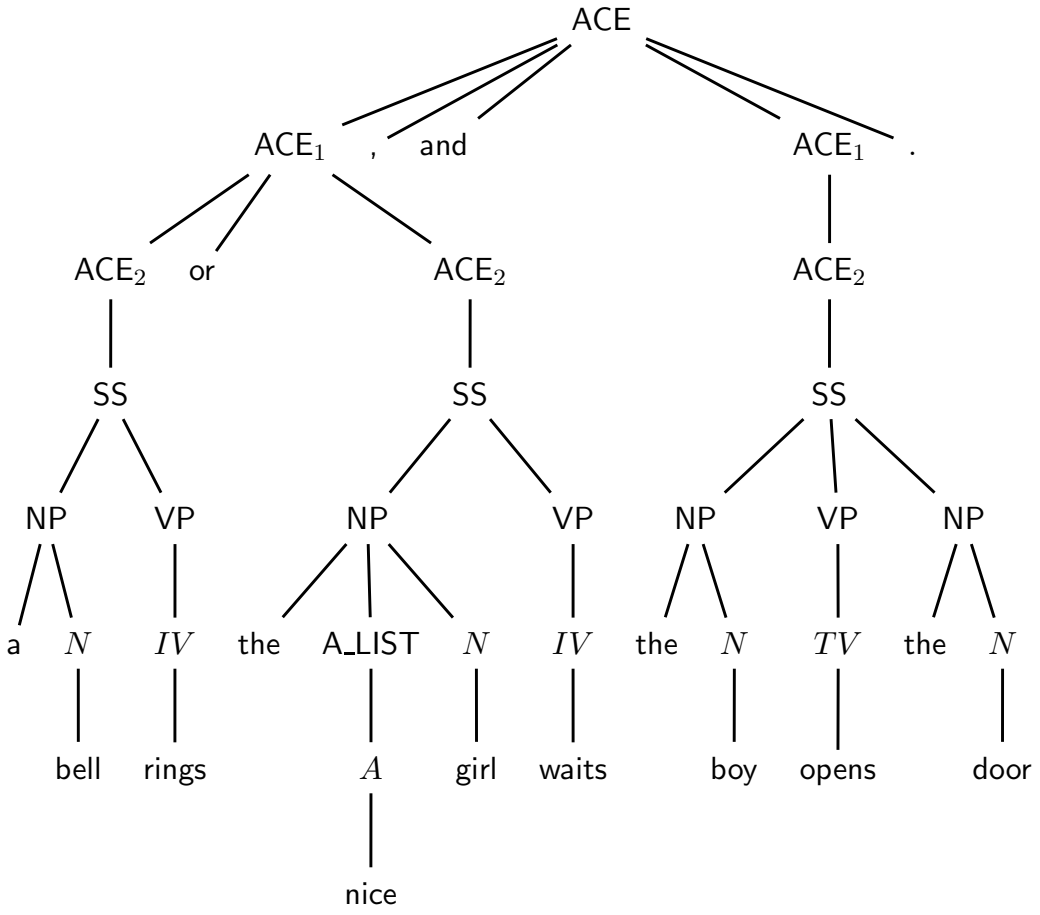
Per come è data questa grammatica, l'oggetto potrebbe ben figurare a seguito di un verbo intransitivo e ciò non avrebbe significato linguistico; ma qui tale aspetto di correttezza è rimandato a livello semantico, non sintattico. Naturalmente si potrebbe anche scegliere di trattarlo sintatticamente, per esempio così:

$$
\begin{array}{rcl}
\langle \mathsf{SS} \rangle & \to & \langle \mathsf{NP} \rangle\; \langle \mathsf{VP} \rangle \\
\hline
\langle \mathsf{NP} \rangle & \to & \ldots \\
\langle \mathsf{VP} \rangle & \to & TV \;\big[\; \langle \mathsf{NP} \rangle \;\big] \;\mid\; IV
\end{array}
$$

Il resto della grammatica è molto semplice e si commenta da sé. La struttura modulare a livelli della grammatica ne giustifica a sufficienza la correttezza sintattica. La grammatica è non ambigua, in quanto composizione di regole notoriamente non ambigue (in effetti essa è perfino $LL(1)$ e a maggior ragione non ambigua - si veda più avanti). Beninteso non è l'unica soluzione possibile.

L'albero sintattico della frase seguente aiuta a comprenderne il funzionamento:

*a bell rings or the nice girl waits, and the boy opens the door.*




Dato che la congiunzione and con virgola si trova più vicina alla radice, cede precedenza alla congiunzione or senza vigola, più lontana, sebbene l'ordine usuale sia che and abbia precedenza su or. Inoltre si vede che, dopo avere generata una congiunzione con virgola, la grammatica non ne può generare un'altra.

Si procede similmente per gli altri tipi di frase del linguaggio $ACE$. Si noti come una tale grammatica modelli la struttura logica della frase inglese, come peraltro si potrebbe ben fare anche per altre lingue.

La grammatica è $LL(1)$ e pertanto $LR(1)$. L'eventuale verifica è facile e pertanto è lasciata al lettore. L'analisi sintattica della frase, per esempio proprio con il metodo $LL$ o $LR$, corrisponde dunque a effettuarne la cosiddetta "analisi logica" di scolastica memoria. Si consulti Wikipedia per un'esposizione sintetica delle caratteristiche sintattiche del linguaggio $ACE$ completo, assai chiaro ed elegante.

## 3 Analisi sintattica e parsificatori 20%

1. È data la grammatica $G$ seguente, di alfabeto $\{a, b, d, e\}$ (assioma $S$):

$$G \begin{cases} S & \rightarrow & a\ B \mid D \mid a\ b\ e \\ B & \rightarrow & b\ e\ D \\ D & \rightarrow & a\ b\ d\ S \end{cases}$$

Si risponda alle domande seguenti:

(a) Si disegni almeno la macchina per la regola assiomatica e si mostri che la grammatica $G$ non è né $LL(1)$ né $LL(2)$.

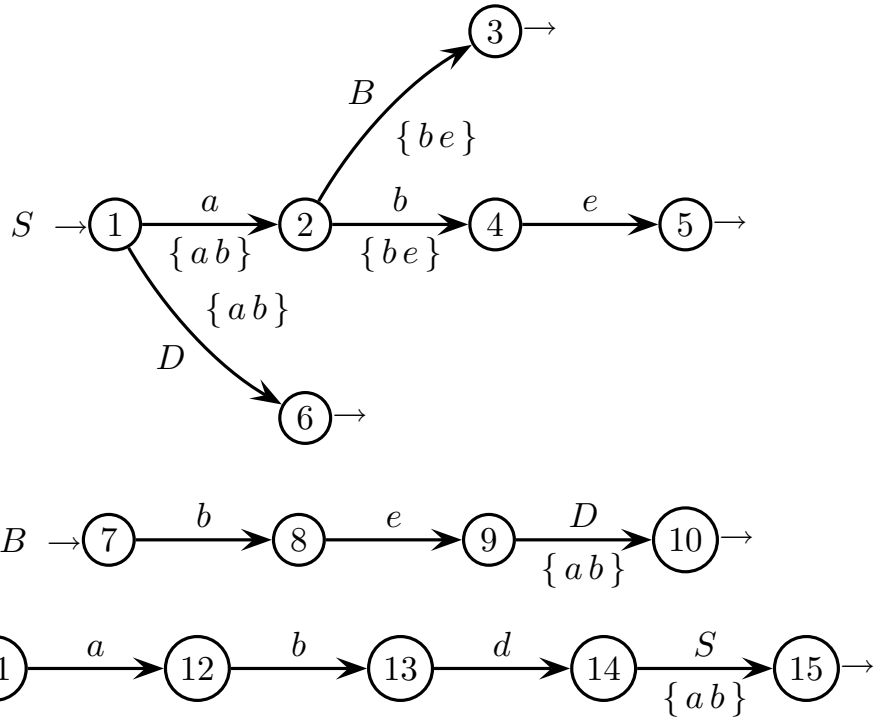
(b) Si trovi il minimo $k$ per cui la grammatica $G$ è $LL(k)$.

(c) (facoltativa) Considerato il linguaggio generato dalla grammatica $G$, si dica se esso ammette una grammatica $LL(1)$, giustificando la risposta.

---

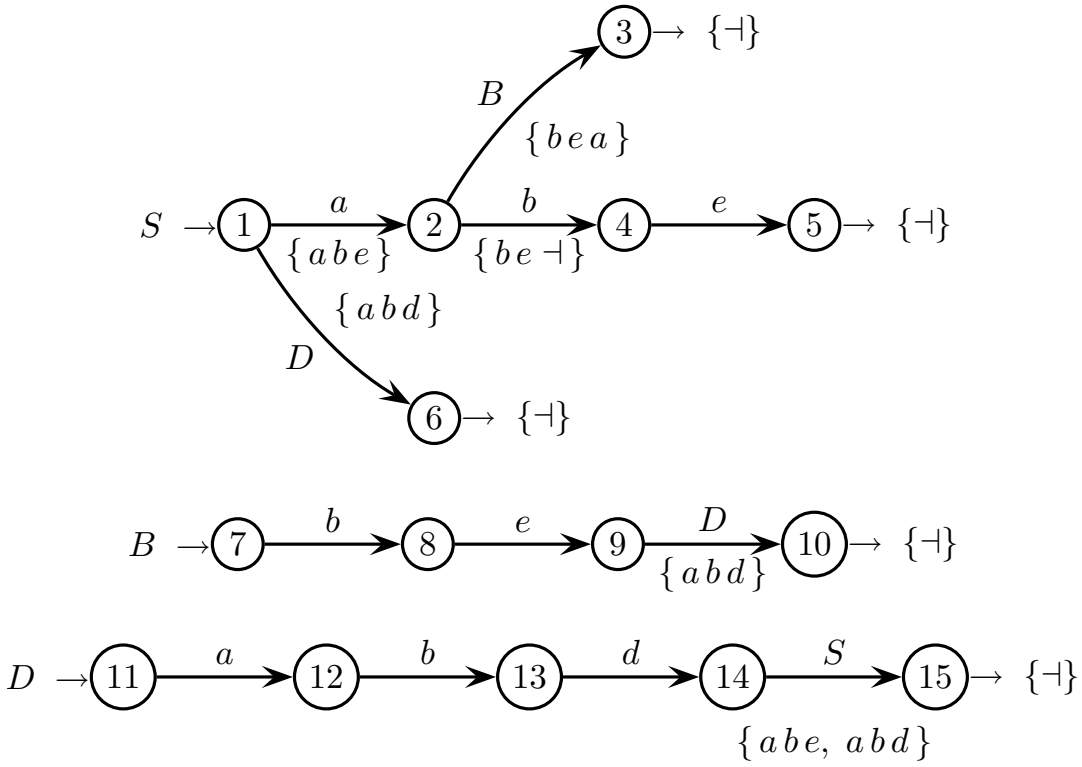
### Soluzione

(a) Ecco l'analisi $LL$ della grammatica $G$ per $k \leq 2$, svolta sulle macchine deterministiche (qui non minime) che rappresentano le regole:



È immediato notare che la grammatica $G$ non è $LL(1)$, per via della biforcazione sullo stato 2 dove il terminale $b$ figura come prospezione di profondità $k = 1$ su ambo gli archi uscenti; analogamente per la biforcazione sullo stato 1. E si vede subito che essa non è nemmeno $LL(2)$, perché per esempio nella macchina di $S$ c'è la biforcazione $2 \xrightarrow{B} 3$ e $2 \xrightarrow{be} 5$, e nella macchina di $B$ si vede che il nonterminale $B$ inizia con la sequenza $be$; similmente per la biforcazione sullo stato 1. Pertanto gli insiemi guida con $k = 2$ alla biforcazione sugli stati 2 e 1 non sono disgiunti.

(b) Tuttavia la grammatica $G$ è $LL(3)$. Eccone l'analisi per $k = 3$:




dove si vede che gli insiemi guida con $k = 3$ alle biforcazioni sono disgiunti.

(c) Si nota che la grammatica $G$ è lineare a destra, seppure con parti terminali di lunghezza differente. Dunque il linguaggio $L(G)$ generato da $G$ è regolare. Con un attimo di riflessione, eccone un'espressione regolare $R$:

$$R = ( \; a \; b \; d \;\; | \;\; a \; b \; e \; a \; b \; d \; )^* \;\; a \; b \; e$$

Pertanto il linguaggio $L(G)$ ammette senz'altro una grammatica $LL(1)$. Si lascia al lettore di trovarla, eventualmente, o intuitivamente o ricorrendo a una trasformazione sistematica, per esempio prima trovando l'automa deterministico equivalente e poi riscrivendolo come grammatica lineare a destra.

2. Si consideri la grammatica $G_1$ seguente (assioma $S$):

$$G_1 \begin{cases} S & \rightarrow & B\ a\ S \mid e \\ B & \rightarrow & a\ B\ c \mid a\ B\ d \mid b \end{cases}$$

Si risponda alle domande seguenti:

(a) Considerando le regole della grammatica $G_1$ e senza costruirne l'automa pilota bensì giustificando la risposta, si discuta se $G_1$ è $LR(1)$.

(b) Modificando in $G_1$ la regola per $B$, si consideri la grammatica $G_2$ seguente:

$$G_2 \begin{cases} S & \rightarrow & B\ a\ S \mid e \\ B & \rightarrow & a\ B\ c \mid a\ B\ a \mid a \end{cases}$$

Si discuta se $G_2$ è $LR(1)$, giustificando la risposta.

(c) (facoltativa) Si consideri ora la grammatica $G_3$ seguente (ottenuta da $G_2$ modificando la regola per l'assioma $S$) e si discuta, giustificando la risposta, che cosa cambia rispetto a $G_2$ per quanto riguarda l'analisi $LR(1)$ del linguaggio generato.

$$G_3 \begin{cases} S & \rightarrow & B\ f\ S \mid e \\ B & \rightarrow & a\ B\ c \mid a\ B\ a \mid a \end{cases}$$

---

## Soluzione

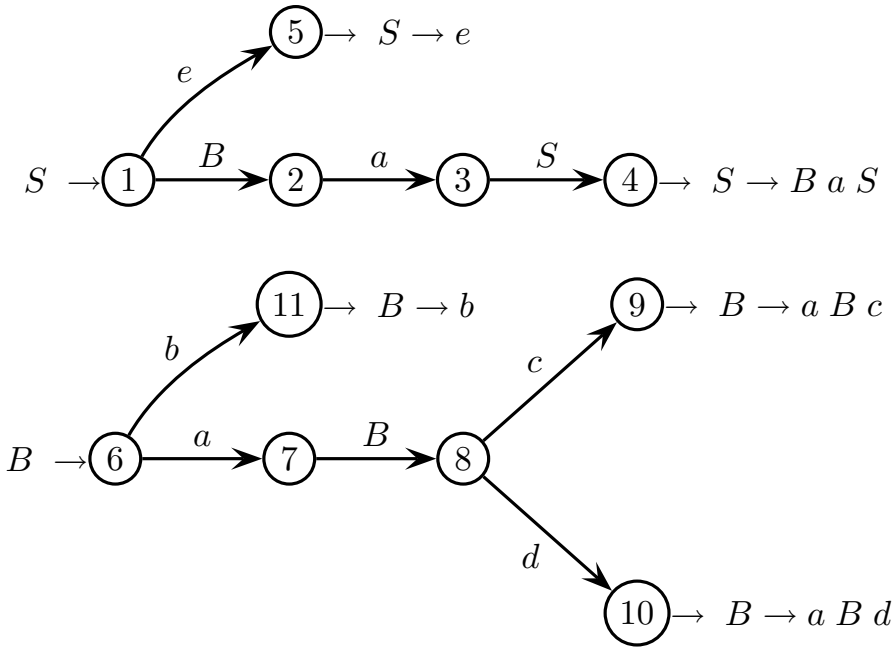
(a) La grammatica $G_1$ genera stringhe del tipo seguente (con $n \geq 0$):

$$(\ a^n\ b\ (\ c \mid d\ )^n\ a\ )^*\ e$$

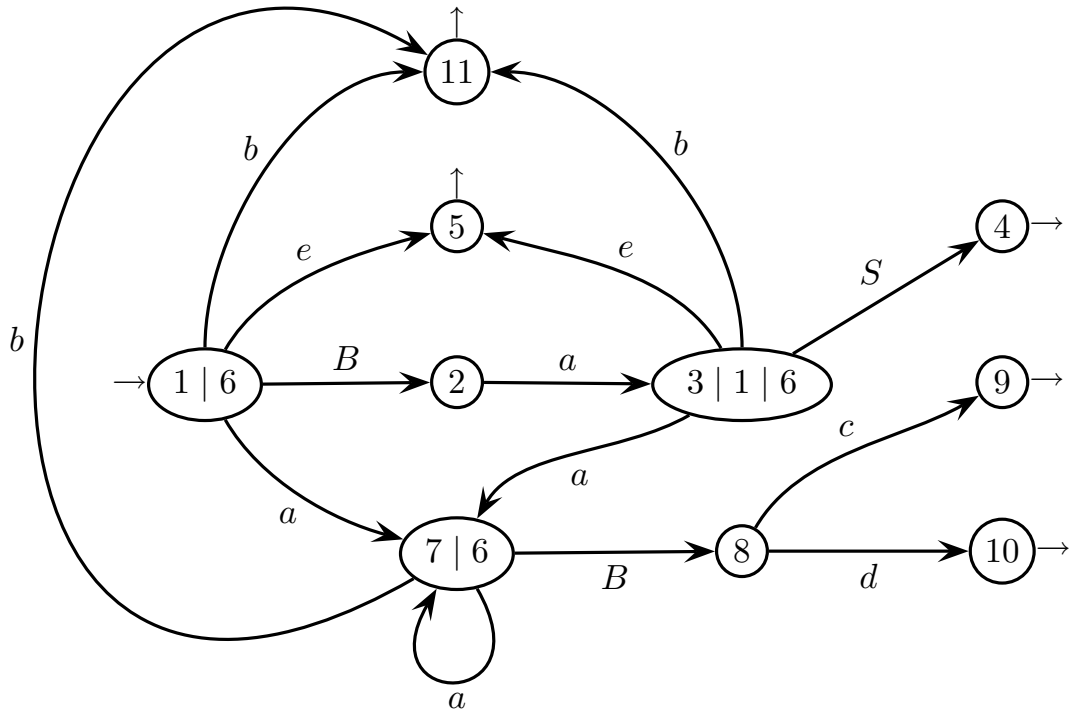
cioè liste di strutture parentetiche bilanciate, del tipo $a^n\ b\ (\ c \mid d\ )^n$ $(n \geq 0)$, ognuna con centro riconoscibile, qui la lettera $b$, e separate da un carattere distinto, qui la lettera $a$. Pertanto si può congetturare che $G_1$ sia di tipo $LR$, in quanto le strutture parentetiche con centro lo sono e la ripetizione di tali strutture è una caratteristica puramente regolare.

Per altra via, si trova che la grammatica $G_1$ è di tipo $LL(1)$ e dunque a maggior ragione di tipo $LR(1)$; si lascia al lettore la facilissima verifica.

Si verifica poi che la grammatica $G_1$ è addirittura $LR(0)$, così. Ecco le macchine deterministiche che rappresentano le regole di $G_1$, con uno stato finale distinto per ciascuna per regola e pertanto non minime:




Ed ecco il grafo pilota di $G_1$, di tipo $LR(0)$, tracciato usando gli stati delle macchine che rappresentano le regole della grammatica (invece delle regole marcate):

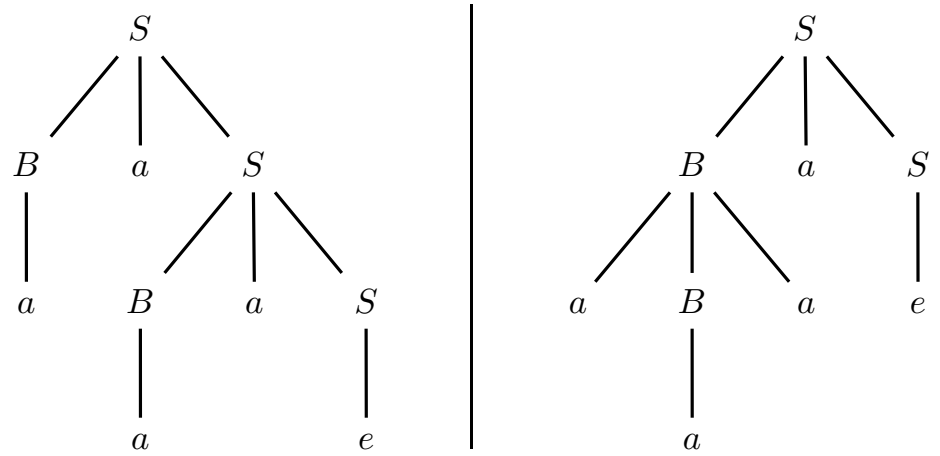



La barra verticale separa fasi di completamento successive del macrostato. Il grafo è senza conflitti: tutti gli stati finali, che corrispondono alla riduzione di una regola specifica, compaiono isolati in macrostati differenti, pertanto la riduzione non è mai in conflitto con lo spostamento o con un'altra riduzione. La grammatica $G_1$ è $LR(0)$ e, a maggior ragione, è $LR(1)$.

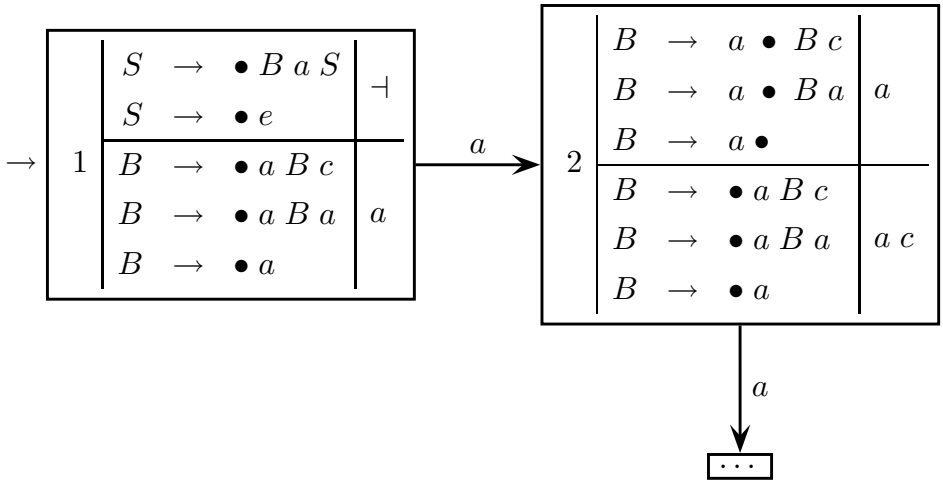
(b) La grammatica $G_2$ è ambigua. Per esempio, la stringa seguente:

$$a\ a\ a\ a\ e = a^4\ e$$

è ambigua e ha due alberi sintattici strutturalmente differenti. Eccoli:

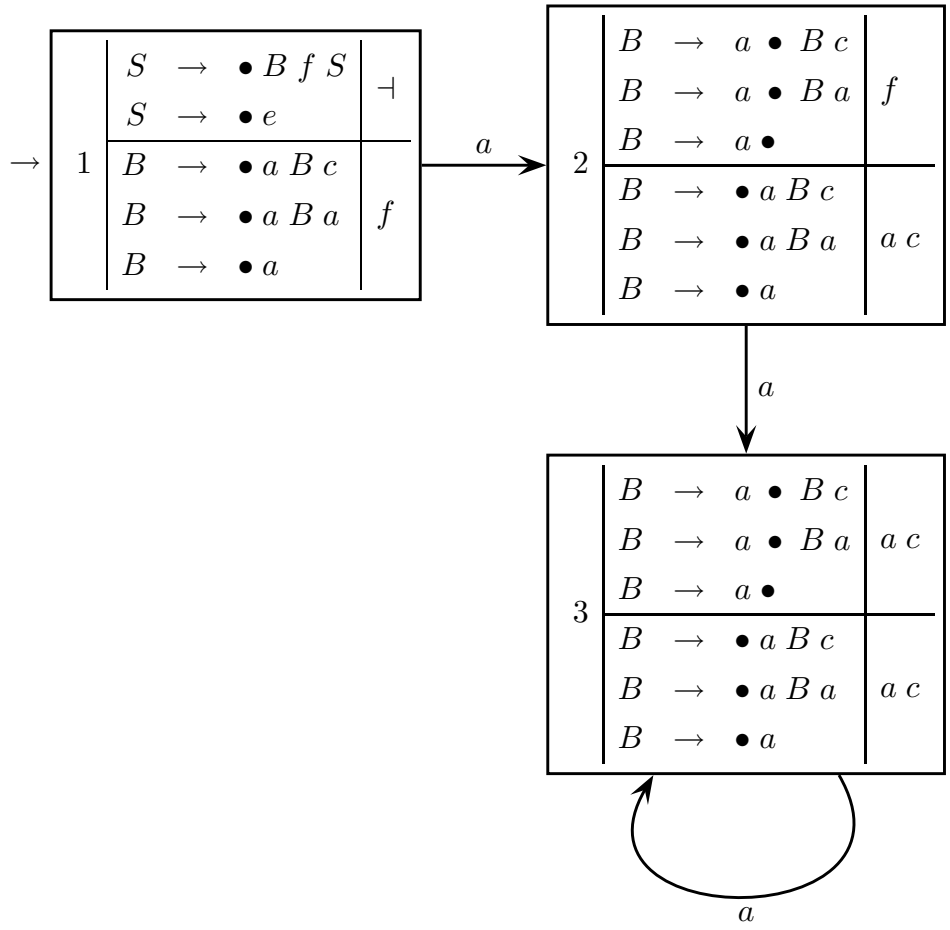



Pertanto la grammatica $G_2$ non è deterministica e in particolare non è $LR(1)$. Ci si potrebbe chiedere se il linguaggio di $G_2$ sia inerentemente ambiguo.
Per altra via, basta costruire un frammento di due soli macrostati dell'automa pilota $LR(1)$ di $G_2$ per scoprire un conflitto e arrivare alla stessa conclusione:




Il macrostato 2 ha un conflitto di tipo riduzione-spostamento: la riduzione $B \to a$ con prospezione $a$ è in conflitto con lo spostamento dato dall'arco $a$ uscente.
Del resto ciò è intuibile: la regola assiomatica $S \to B\ a\ S$ genera ripetizioni del nonterminale $B$ separate dal terminale $a$, ma anche $B$ può generare $a$. Il carattere di prospezione $a$ è dunque interpretabile sia come separatore dei nonerminali $B$, dunque demarcante la riduzione di $B$, sia come elemento generato espandendo un nonterminale $B$, dunque demarcante uno spostamento abilitato da $B$; donde il conflitto, che si manifesta subito (dopo avere letto il carattere $a$ iniziale) giacché il primo $B$ si può ridurre a un solo terminale.

(c) La grammatica $G_3$ non è ambigua e dunque nemmeno il linguaggio generato da $G_3$ lo è, tuttavia neppure $G_3$ è $LR(1)$; occorre disegnare almeno parte dell'automa pilota per scoprire un conflitto, ma bastano tre macrostati. Ecco il frammento di grafo (con macrostati etichettati da regole marcate):




Il macrostato 3 contiene un conflitto tra la riduzione $B \to a$ con prospezione $a$ e l'arco uscente (qui autoanello) di spostamento su $a$. Si noti che il macrostato 2, pur contenendo la stessa riduzione e avendo un arco uscente etichettato con $a$, non ha conflitto giacché la prospezione della riduzione è diversa, ossia $f$.

Del resto ciò è intuibile: la stringa consta di strutture parentetiche $a^n\ a\ (\ a\ |\ c\ )^n$ $(n \geq 0)$ generate dal nonterminale $B$, concatenate e separate dalla lettera $f$ generata dall'assioma. Non c'è ambiguità, ma la struttura parentetica non ha centro (qui $a$) distinguibile dai caratteri a sinistra (tutti $a$) e da quelli che potrebbero stare a destra (lettere $a$ e $c$ disposte liberamente). Pertanto l'analizzatore sintattico bottom-up derivato dalla grammatica $G_3$ non può decidere deterministicamente se spilare invece d'impilare.

## 4 Traduzione e analisi semantica 20%

1. Si consideri un linguaggio di liste a un solo livello, delimitate da parentesi e non vuote, con elementi rappresentati dal carattere '$e$' e separati dal carattere virgola ','. La traduzione di tali liste è esemplificata così:

| sorgente | traduzione |
|---|---|
| ( $e$ ) | single $e$ |
| ( $e$, $e$ ) | former $e$ latter $e$ |
| ( $e$, $e$, $e$ ) | first $e$ next $e$ last $e$ |
| ( $e$, $e$, $e$, $e$ ) | first $e$ next $e$ next $e$ last $e$ |
| ... | ... |

dove single, former, latter, first, next e last sono i nuovi delimitatori e separatori.

Ecco la grammatica $G$ che genera il linguaggio sorgente (assioma $S$):

$$G \begin{cases} S \rightarrow \text{`(' } L \text{ `)'} \\ L \rightarrow e \text{ `,' } L \mid e \end{cases}$$

Si risponda alle domande seguenti:

(a) Si scriva la grammatica (o lo schema) per la traduzione descritta, modificando opportunamente la grammatica sorgente $G$.

(b) (facoltativa) Si immagini di ampliare il linguaggio sorgente ammettendo, oltre alle precedenti, anche liste annidate a qualunque livello di profondità, e si estenda a queste ultime la traduzione, così:

| sorgente | traduzione |
|---|---|
| ( ( $e$ ) ) | SINGLE $e$ |
| ( ( $e$ ) , $e$ ) | former single $e$ latter $e$ |
| ( ( $e$, $e$, ) , ( $e$ ) ) | former former $e$ latter $e$ latter single $e$ |
| ( $e$, $e$, ( $e$, $e$ ) ) | first $e$ next $e$ last former $e$ latter $e$ |
| ... | ... |

dove SINGLE è una forma stenografica per abbreviare la ripetizione single single, sia pure perdendo un po' d'informazione.

Similmente per tradurre liste a tre o più livelli, dove si abbrevia single single single ... in SINGLE (invece la ripetizione di former o first non va stenografata).

Si scriva la grammatica (o lo schema) per la traduzione estesa, modificando opportunamente la grammatica sorgente $G$.

## Soluzione

(a) Ecco lo schema di traduzione per liste a un solo livello (assioma $S$):

| | | | | | |
|---|---|---|---|---|---|
| $S$ | $\rightarrow$ | ‘(’ $e$ ‘)’ | $S$ | $\rightarrow$ | single $e$ |
| $S$ | $\rightarrow$ | ‘(’ $e$ ‘,’ $e$ ‘)’ | $S$ | $\rightarrow$ | former $e$ latter $e$ |
| $S$ | $\rightarrow$ | ‘(’ $e$ ‘,’ $e$ ‘,’ $L$ ‘)’ | $S$ | $\rightarrow$ | first $e$ next $e$ $L$ |
| $L$ | $\rightarrow$ | $e$ ‘,’ $L$ $\mid$ $e$ | $L$ | $\rightarrow$ | next $e$ $L$ $\mid$ last $e$ |

L’idea è di differenziare i casi di lista di lunghezza uno, due, e tre o maggiore.

(b) Prima conviene realizzare l’annidamento senza stenografia (assioma $S$):

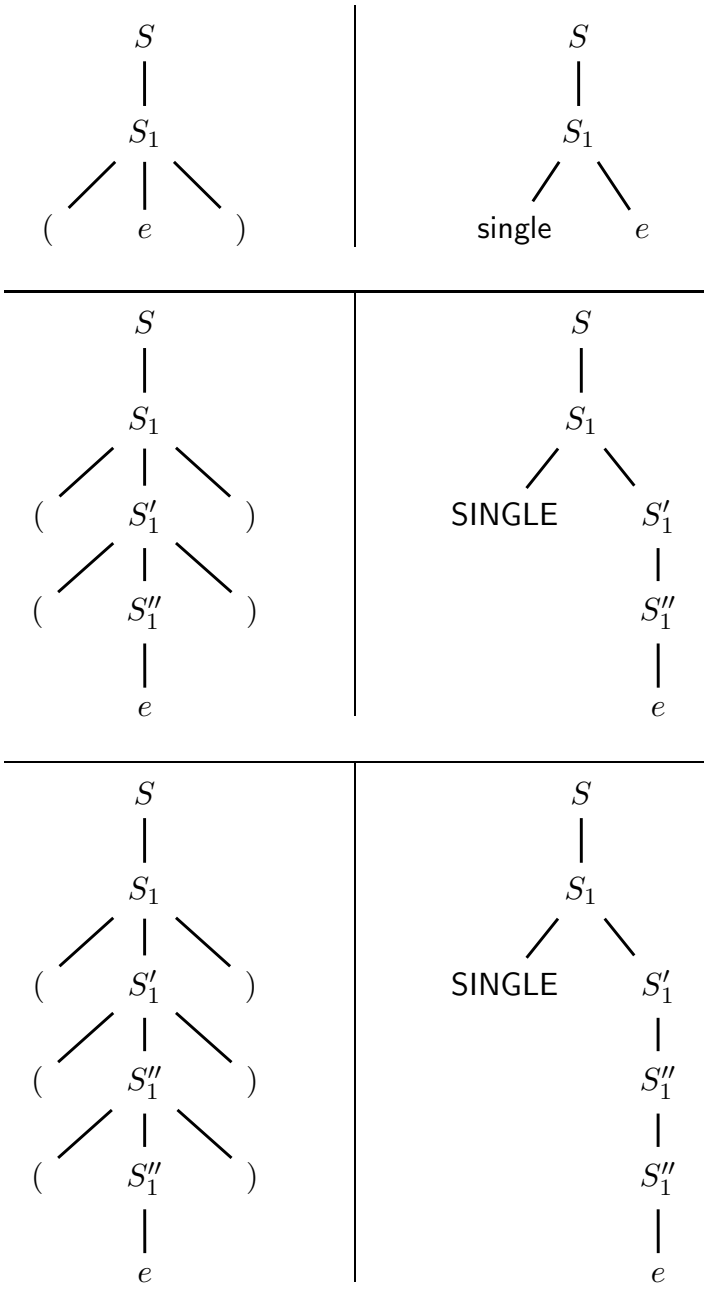
| | | | | | |
|---|---|---|---|---|---|
| $S$ | $\rightarrow$ | ‘(’ $E$ ‘)’ | $S$ | $\rightarrow$ | single $E$ |
| $S$ | $\rightarrow$ | ‘(’ $E$ ‘,’ $E$ ‘)’ | $S$ | $\rightarrow$ | former $E$ latter $E$ |
| $S$ | $\rightarrow$ | ‘(’ $E$ ‘,’ $E$ ‘,’ $L$ ‘)’ | $S$ | $\rightarrow$ | first $E$ next $E$ $L$ |
| $L$ | $\rightarrow$ | $E$ ‘,’ $L$ $\mid$ $E$ | $L$ | $\rightarrow$ | next $E$ $L$ $\mid$ last $E$ |
| $E$ | $\rightarrow$ | $S$ $\mid$ $e$ | $E$ | $\rightarrow$ | $S$ $\mid$ $e$ |

dove l’elemento è modellato dal nonterminale $E$, che si espande nell’elemento terminale $e$ (regola $E \rightarrow e$) o in una sottolista (regola di copia $E \rightarrow S$).
La forma stenografica compare quando si usa ricorsivamente la prima regola assiomatica $S \rightarrow$ ‘(’ $E$ ‘)’ (passando per la regola di copia $E \rightarrow S$). Tale regola va dunque differenziata, per esempio così (assioma $S$):

| | | | | | |
|---|---|---|---|---|---|
| $S$ | $\rightarrow$ | $S_1$ $\mid$ $S_{\geq 2}$ | $S$ | $\rightarrow$ | $S_1$ $\mid$ $S_{\geq 2}$ |
| $S_1$ | $\rightarrow$ | ‘(’ $S_{\geq 2}$ ‘)’ $\mid$ ‘(’ $e$ ‘)’ | $S_1$ | $\rightarrow$ | single $S_{\geq 2}$ $\mid$ single $e$ |
| $S_1$ | $\rightarrow$ | ‘(’ $S'_1$ ‘)’ | $S_1$ | $\rightarrow$ | SINGLE $S'_1$ |
| $S'_1$ | $\rightarrow$ | ‘(’ $S''_1$ ‘)’ | $S'_1$ | $\rightarrow$ | $S''_1$ |
| $S''_1$ | $\rightarrow$ | ‘(’ $S''_1$ ‘)’ $\mid$ $S_{\geq 2}$ $\mid$ $e$ | $S''_1$ | $\rightarrow$ | $S''_1$ $\mid$ $S_{\geq 2}$ $\mid$ $e$ |
| $S_{\geq 2}$ | $\rightarrow$ | ‘(’ $E$ ‘,’ $E$ ‘)’ | $S_{\geq 2}$ | $\rightarrow$ | former $E$ latter $E$ |
| $S_{\geq 2}$ | $\rightarrow$ | ‘(’ $E$ ‘,’ $E$ ‘,’ $L$ ‘)’ | $S_{\geq 2}$ | $\rightarrow$ | first $E$ next $E$ $L$ |
| $L$ | $\rightarrow$ | $E$ ‘,’ $L$ $\mid$ $E$ | $L$ | $\rightarrow$ | next $E$ $L$ $\mid$ last $E$ |
| $E$ | $\rightarrow$ | $S$ $\mid$ $e$ | $E$ | $\rightarrow$ | $S$ $\mid$ $e$ |

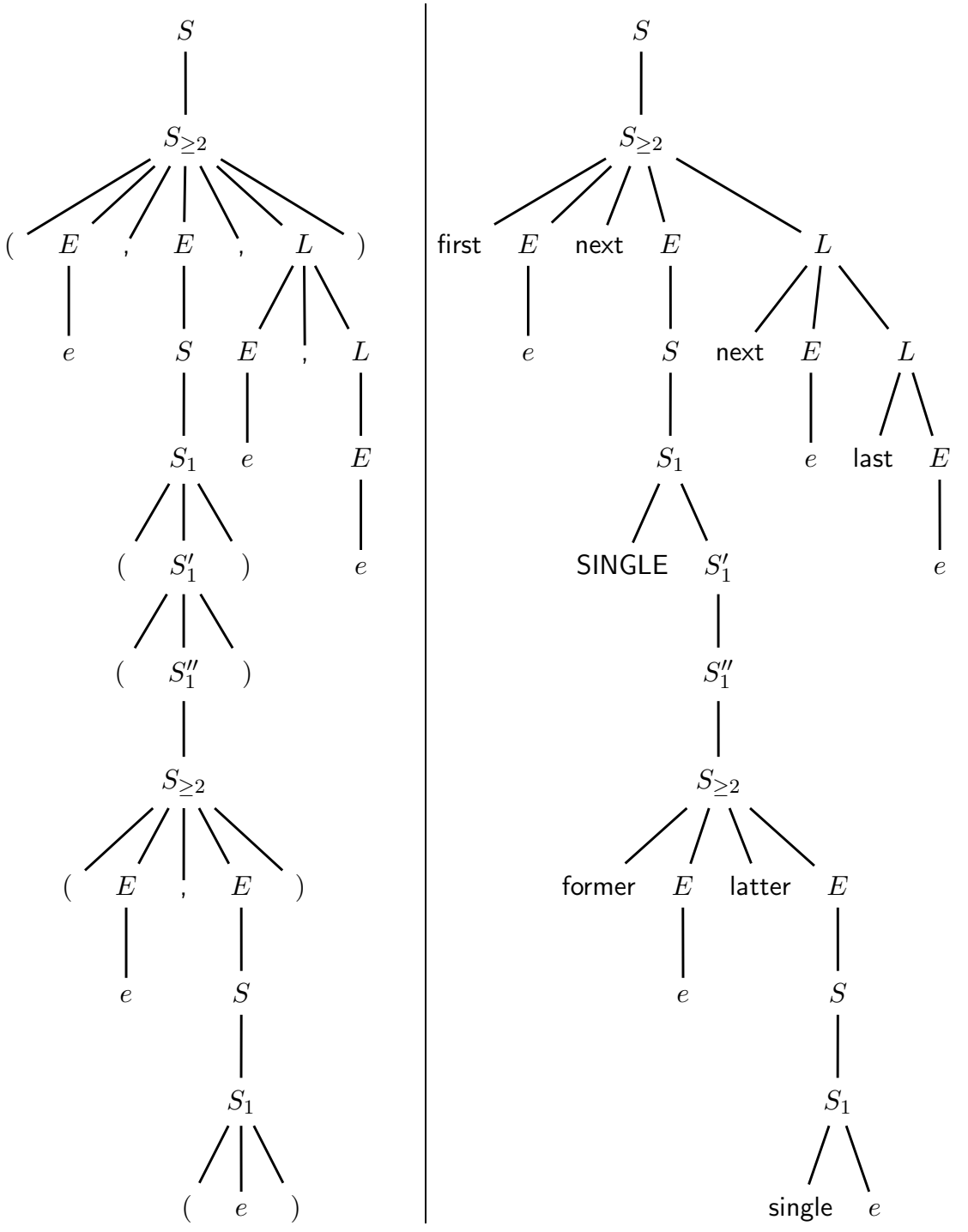
dove si distingue tra liste di un solo elemento oppure di due o più, e si distinguono le liste di un solo elemento annidate in prima posizione. Tale soluzione sarebbe facilmente applicabile anche per stenografare la ripetizione di former e first.

Ecco gli alberi sintattici sorgente e traduzione di tre casi significativi per capire il funzionamento della grammatica:




I tre casi mostrati si riferiscono alla parte di grammatica che realizza la forma stenografica SINGLE. Essi illustrano la traduzione di liste di lunghezza unitaria, che infine contengono un solo elemento. Naturalmente si può espandere l'elemento e farlo diventare una lista di lunghezza maggiore di uno (regola $S''_1 \to S_{\geq 2}$). Similmente, la lista unitaria può figurare come elemento di una lista di lunghezza maggiore di uno, posta a livello superiore (catena di regole di copia $E \to S \to S_1$). Ciò basta per giustificare la correttezza della grammatica.

Ed ecco un esempio finale sostanzialmente completo, con stringhe e alberi:




( $e$, ( ( ( $e$, ( $e$ ) ) ) ), $e$, $e$ )

first $e$ next SINGLE former $e$ latter single $e$ next $e$ last $e$

Sono contemplati tutti i costrutti (o quasi tutti - tranne le varianti ovvie), diversamente innestati, e le traduzioni rispettive.

2. Si consideri la porzione seguente della grammatica astratta di un linguaggio di programmazione (assioma statL):

$$
\begin{aligned}
\langle \mathsf{statL} \rangle &\rightarrow \langle \mathsf{stat} \rangle \ \langle \mathsf{statL} \rangle \\
\langle \mathsf{statL} \rangle &\rightarrow \langle \mathsf{stat} \rangle \\
\langle \mathsf{stat} \rangle &\rightarrow if \ \langle \mathsf{cond} \rangle \ then \ \langle \mathsf{statL} \rangle \ end \\
\langle \mathsf{stat} \rangle &\rightarrow if \ \langle \mathsf{cond} \rangle \ then \ \langle \mathsf{statL} \rangle \ else \ \langle \mathsf{statL} \rangle \ end \\
\langle \mathsf{stat} \rangle &\rightarrow asg \\
\langle \mathsf{cond} \rangle &\rightarrow c
\end{aligned}
$$

Si risponda alle domande seguenti:

(a) Si scriva una grammatica con attributi per applicare un'etichetta 'else' a ciascuna istruzione di assegnamento raggiungibile attraverso almeno una diramazione else. Allo scopo si utilizzino l'attributo booleano $in\_else$ e la funzione $tag\_by\_else\,(istr)$ per etichettare l'istruzione datale come argomento. Ecco un esempio:

```
asg                 -> non etichettare
if c then
    asg             -> non etichettare
    asg             -> non etichettare
else
    if c then
        asg         -> tag_by_else (asg)
    else
        asg         -> tag_by_else (asg)
    end
end
```

Si dica se la grammatica è a una passata (one sweep), spiegandone il motivo.

(b) Dato che statisticamente le istruzioni di assegnamento raggiungibili via diramazione else hanno bassa probabilità d'essere eseguite, si vuole stimare nella radice dell'albero sintattico del programma il rapporto tra numero di assegnamenti raggiungibili via diramazione else e numero complessivo di assegnamenti. Ciò può servire al compilatore per ottimizzare il codice macchina generato. Nell'esempio precedente il rapporto vale 2/5.

Si scriva una grammatica con attributi per realizzare il calcolo così descritto. Allo scopo si utilizzino gli attributi interi $n\_else$ e $n\_tot$ per contare i due tipi di assegnamento, e l'attributo reale $r$ per il rapporto. Si dica se la grammatica è di tipo a una passata (one-sweep) e in particolare di tipo L, spiegandone il motivo.

---

| sintassi | calcolo attributi (punto (a)) |
|---|---|
| 1: $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1 \ \langle \mathsf{statL} \rangle_2$ | |
| 2: $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1$ | |
| 3: $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2 \ end$ | |
| 4: $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2$ $else \ \langle \mathsf{statL} \rangle_3 \ end$ | |
| 5: $\langle \mathsf{stat} \rangle_0 \rightarrow asg$ | |
| 6: $\langle \mathsf{cond} \rangle_0 \rightarrow c$ | |

| sintassi | calcolo attributi (punto (b)) |
|---|---|
| 1: $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1 \ \langle \mathsf{statL} \rangle_2$ | |
| 2: $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1$ | |
| 3: $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2 \ end$ | |
| 4: $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2$ $else \ \langle \mathsf{statL} \rangle_3 \ end$ | |
| 5: $\langle \mathsf{stat} \rangle_0 \rightarrow asg$ | |
| 6: $\langle \mathsf{cond} \rangle_0 \rightarrow c$ | |

## Soluzione

(a) L'attributo booleano *in_else* è destro, associato ai nonterminali stat e statL, e lo schema di calcolo è puramente ereditato. Eccolo:

| | sintassi | calcolo attributi (punto (a)) |
|---|---|---|
| 1: | $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1 \ \langle \mathsf{statL} \rangle_2$ | $in\_else_1 = in\_else_0$<br>$in\_else_2 = in\_else_0$ |
| 2: | $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1$ | $in\_else_1 = in\_else_0$ |
| 3: | $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2 \ end$ | $in\_else_2 = in\_else_0$ |
| 4: | $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2$ $else \ \langle \mathsf{statL} \rangle_3 \ end$ | $in\_else_1 = in\_else_0$<br>$in\_else_2 = true$ |
| 5: | $\langle \mathsf{stat} \rangle_0 \rightarrow asg$ | **if** $in\_else_0 == true$ **then**<br>$tag\_by\_else\,(asg)$<br>**end** |
| 6: | $\langle \mathsf{cond} \rangle_0 \rightarrow c$ | |

L'idea è di propagare verso il basso l'attributo *in_else* e di renderlo vero non appena esso attraversa una diramazione else. Raggiunto il nodo di assegnamento, l'attributo *in_else* governa la condizione per attivare la funzione che etichetta l'istruzione di assegnamento. Naturalmente l'attributo *in_else* va inizializzato a $false$ nella radice dell'albero.

Lo schema è evidentemente di tipo a una passata, poiché c'è un solo attributo destro che dipende esclusivamente dal valore associato al padre.

(b) Bastano i due attributi sinistri interi $n\_else$ e $n\_tot$, associati ai nonterminali stat e statL, e l'attributo reale $r$ nella radice che è pure banalmente sinistro, associato solo a statL. Gli attributi sintetizzati dipendono solo da sé stessi e lo schema di calcolo è puramente sintetizzato. Eccolo:

| | sintassi | calc. attr. (punto (b)) - $1^a$ sol. |
|---|---|---|
| 1: | $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1 \ \langle \mathsf{statL} \rangle_2$ | $n\_else_0 = n\_else_1 + n\_else_2$<br>$n\_tot_0 = n\_tot_1 + n\_tot_2$<br>$r_0 = n\_else_0 / n\_tot_0$ |
| 2: | $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1$ | $n\_else_0 = n\_else_1$<br>$n\_tot_0 = n\_tot_1$<br>$r_0 = n\_else_0 / n\_tot_0$ |
| 3: | $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2 \ end$ | $n\_else_0 = n\_else_2$<br>$n\_tot_0 = n\_tot_2$ |
| 4: | $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2$<br>$else \ \langle \mathsf{statL} \rangle_3 \ end$ | $n\_else_0 = n\_else_2 + n\_tot_3$<br>$n\_tot_0 = n\_tot_2 + n\_tot_3$ |
| 5: | $\langle \mathsf{stat} \rangle_0 \rightarrow asg$ | $n\_else_0 = 0$<br>$n\_tot_0 = 1$ |
| 6: | $\langle \mathsf{cond} \rangle_0 \rightarrow c$ | |

L'idea è di propagare verso l'alto gli attributi $n\_else$ e $n\_tot$, cumulandoli così da contare gli assegnamenti sottoposti a diramazione else e gli assegnamenti totali; si veda la regola 4.

Lo schema è di tipo a una passata, poiché è puramente sintetizzato. Esso è anche di tipo L e la visita dei nodi figli può procedere da sx verso dx.

È anche possibile sfruttare l'attributo destro $in\_else$, così (le funzioni semantiche per calcolare $in\_else$ sono quelle della domanda (a) e qui non sono ripetute):

| | sintassi | calc. attr. (punto (b)) - $2^a$ sol. |
|---|---|---|
| 1: | $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1 \ \langle \mathsf{statL} \rangle_2$ | $n\_else_0 = n\_else_1 + n\_else_2$<br>$n\_tot_0 = n\_tot_1 + n\_tot_2$<br>$r_0 = n\_else_0 / n\_tot_0$ |
| 2: | $\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1$ | $n\_else_0 = n\_else_1$<br>$n\_tot_0 = n\_tot_1$<br>$r_0 = n\_else_0 / n\_tot_0$ |
| 3: | $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2 \ end$ | $n\_else_0 = n\_else_2$<br>$n\_tot_0 = n\_tot_2$ |
| 4: | $\langle \mathsf{stat} \rangle_0 \rightarrow if \ \langle \mathsf{cond} \rangle_1 \ then \ \langle \mathsf{statL} \rangle_2$<br>$else \ \langle \mathsf{statL} \rangle_3 \ end$ | $n\_else_0 = n\_else_2 + n\_else_3$<br>$n\_tot_0 = n\_tot_2 + n\_tot_3$ |
| 5: | $\langle \mathsf{stat} \rangle_0 \rightarrow asg$ | **if** $in\_else == true$ **then**<br>$n\_else_0 = 1$<br>**else**<br>$n\_else_0 = 0$<br>**end**<br>$n\_tot_0 = 1$ |
| 6: | $\langle \mathsf{cond} \rangle_0 \rightarrow c$ | |

L'idea è di ricorrere all'attributo $in\_else$ (già calcolato) per sapere se l'assegnamento $asg$ è sottoposto a diramazione else o no, e di conseguenza per annoverarlo subito nel conteggio svolto tramite l'attributo $n\_else$ oppure per escluderlo; il resto dello schema è quasi identico alla soluzione precedente, tranne ovviamente il modo di contare gli assegnamenti sottoposti a diramazione else (regola 4).

Questo schema è di tipo misto, più complesso del precedente giacché il calcolo prima è top-down (attributo $in\_else$) e poi bottom-up (attributi $n\_else$, $n\_tot$ e $r$), ma è a una passata nonché di tipo L. Si lascia al lettore la verifica, facilissima.

La domanda dà un esempio di come si possa ottenere lo stesso risultato tramite schemi di calcolo grammaticale significativamente differenti.