Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 2 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. Dato l'alfabeto ternario $\Sigma = \{a, b, c\}$, si consideri l'espressione regolare R seguente:

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

   Si risponda alle domande seguenti:

   (a) Si ricavi l'automa deterministico A che riconosce il linguaggio $L(R)$ tramite l'algoritmo di Berri-Sethi.

   (b) Si dica, argomentando la risposta, se l'automa A trovato è minimo.

   (c) (facoltativa) Si dica, argomentando la risposta, se il linguaggio $L(R)$ è locale.

Soluzione

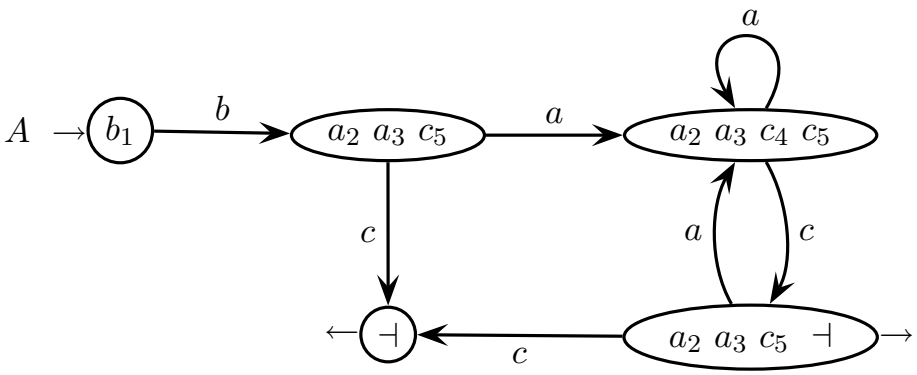
(a) Ecco l'espressione R numerata:

$$R = b_1 \ (\ a_2 \ | \ a_3 \ c_4 \ )^* \ c_5 \ \dashv$$

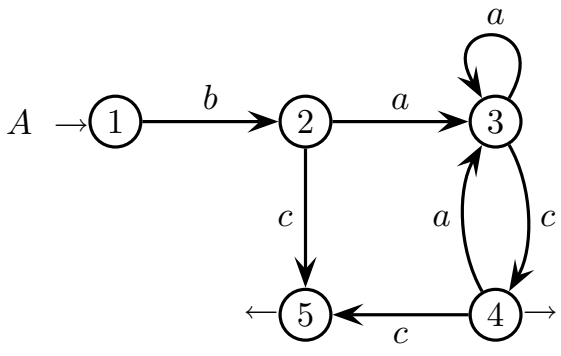
Il linguaggio $L(R)$ ha inizio b_1 e fine $\dashv$. I seguiti sono così:

b_1	$a_2 \ a_3 \ c_5$
a_2	$a_2 \ a_3 \ c_5$
a_3	c_4
c_4	$a_2 \ a_3 \ c_5$
c_5	$\dashv$

L'automa A deterministico di Berri-Sethi è dunque il seguente:



(b) Rinominando gli stati del grafo di A:



si ha la tabella degli stati seguente:

stato	a	b	c	finale
1		2		no
2	3		5	no
3	3		4	no
4	3		5	sì
5				sì

Ragionando: gli stati finali 4 e 5 sono distinguibili poiché hanno successori diversamente mancanti; lo stato non finale 1 è distinguibile dagli stati non finali 2 e 3 per lo stesso motivo; e gli stati non finali 2 e 3 sono distinguibili in quanto i c-successori 5 e 4 rispettivi lo sono (da prima); pertanto l'automa A è minimo.

(c) Tutte le stringhe di $L(R)$ iniziano con la lettera b e finiscono con la lettera c. Per la stringa $b\,c \in L(R)$ è permesso il digramma $b\,c$, e per la stringa $b\,a\,c\,c \in L(R)$ è permesso il digramma $c\,c$; pertanto anche la stringa $b\,c\,c$ dovrebbe appartenere a $L(R)$, ciò che è falso; in conclusione $L(R)$ non è locale (con digrammi - potrebbe esserlo con terne, o quaterne, ecc).

2. Dato l'alfabeto ternario $\Sigma = \{a, b, c\}$, si consideri il linguaggio regolare L con le caratteristiche seguenti:

   (a) *tutte* le stringhe di L *iniziano* con la lettera a e *non finiscono* con la lettera a

   (b) in *nessuna* stringa di L ci sono lettere a *adiacenti*

   (c) in *ogni* stringa di L ci sono *tante* lettere a *quante* lettere c

   (d) la lettera b può mancare oppure fare una o più comparse nella stringa di L, e ciascuna comparsa di b è *immediatamente preceduta* da almeno una lettera c

   Si risponda alle domandi seguenti:

   (a) Si scriva un'espressione regolare R standard (ossia contenente solo concatenamento, unione, stella e croce) che genera il linguaggio L.

   Suggerimento: si esaminino i punti (a, b, c, d) in progressione, supponendo all'inizio che il punto (d) sia soddisfatto banalmente, non ammettendo nella stringa alcuna b; poi si generalizzi.

   (b) (facoltativa) Si dica, argomentando la risposta, se l'espressione regolare R trovata prima è ambigua o no.

Soluzione

(a) Considerando il punto (a) e supponendo non ci siano lettere b (così il punto (d) è soddisfatto senz'altro), si avrebbe:

$$a\ (\ a\ |\ c\ )^*\ c$$

Aggiungendo il punto (b), si avrebbe:

$$(\ a\ c^+\ )^+$$

E aggiungendo il punto (c), si avrebbe:

$$(\ a\ c\ )^+$$

Ora si osservi che la lettera b può comparire solo a seguito della lettera c e non ha dipendenze con altre lettere, sicché il digramma $c\ b$ funziona come la singola lettera c; pertanto si può modificare l'espressione inserendo facoltativamente una lettera b immediatamente dopo ciascuna lettera c, così ottenendo R:

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

Oppure si può ragionare immaginando di inserire prima una, poi due, ecc, lettere b dopo altrettante lettere c scelte a caso. Mettendo dunque esattamente una lettera b così da soddisfare il punto (d), si avrebbe:

$$(\ a\ c\ )^*\ (\ a\ c\ b\ )\ (\ a\ c\ )^*$$

e mettendone esattamente due:

$$(\ a\ c\ )^*\ (\ a\ c\ b\ )\ (\ a\ c\ )^*\ (\ a\ c\ b\ )\ (\ a\ c\ )^*$$

e così via; ossia per una, due, ecc, lettere b, si avrebbe:

$$(\ (\ a\ c\ )^*\ (\ a\ c\ b\ )\ )^+\ (\ a\ c\ )^*$$

Pertanto R è così, unendo i due casi precedenti (b non c'è e c'è):

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

ed è evidentemente equivalente alla formulazione precedente.
A posteriori, entrambe le formulazioni sono equivalenti alla seguente:

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

meno facile da ottenere partendo dalla descrizione, per come essa è data.

(b) L'espressione regolare R non è ambigua. Guardando la prima formulazione, ogni lettera compare in R una sola volta, l'espressione è lineare (e il linguaggio L è locale) e dunque non ambigua.
Guardando la seconda formulazione, si osserva che il primo membro dell'unione è disgiunto dal secondo; espandendo la croce a secondo membro si ha:

$$(\ a\ c\ )^*\ (\ a\ c\ b\ )\ (\ a\ c\ )^*\ |\ (\ a\ c\ )^*\ (\ a\ c\ b\ )\ (\ a\ c\ )^*\ (\ a\ c\ b\ )\ (\ a\ c\ )^*\ |\ \ldots$$

membri pure tutti disgiunti; e infine ciascun membro non è ambiguo (non ci sono le condizioni per ambiguità di concatenamento e neppure di stella); dunque tutta l'espressione R non è ambigua. Similmente per la terza formulazione.

2 Grammatiche libere e automi a pila 20%

1. Si consideri il linguaggio L_{TQ} delle espressioni parentetiche correttamente bilanciate, costruite con parentesi tonde e quadre.

   Si risponda alle domande seguenti:

   (a) Si definisca la grammatica in forma non estesa di un linguaggio L, sottoinsieme di L_{TQ}, con in aggiunta il vincolo che le sue stringhe non devono presentare due coppie di parentesi tonde consecutive allo stesso livello di annidamento.

   Esempi di stringhe permesse:

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

   Esempi di stringhe vietate:

   $$(\,)\,(\,) \qquad [\,]\,\Big[\,(\,)\,\big(\,[\,]\,\big)\,\Big] \qquad \Big(\,[\,]\,\big(\,[\,]\,\big)\,(\,)\,\Big)$$

   (b) Si definisca il linguaggio L come intersezione del linguaggio L_{TQ} e del linguaggio di un'espressione regolare E opportuna, scritta in forma preferibilmente non estesa, ossia così:

   $$L = L_{TQ} \cap L(E)$$

Soluzione

(a) La grammatica è una variante di quella per il linguaggio di Dyck.

$$\begin{cases} S & \rightarrow & \text{`('}\ S\ \text{`)'}\ S_{[} \ \mid\ \text{`['}\ S\ \text{`]'}\ S \ \mid\ \varepsilon \\ S_{[} & \rightarrow & \text{`['}\ S\ \text{`]'}\ S \ \mid\ \varepsilon \end{cases}$$

(b) Definito l'alfabeto $\Sigma = \{\ (,\ ),\ [,\ ]\ \}$, l'espressione E in forma estesa è la seguente:

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

mentre in forma non estesa è

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

equivalente alla prima e non ambigua.

2. Si progetti la grammatica estesa (EBNF) per la definizione della parte d'interfaccia dei moduli di un linguaggio di programmazione. Ogni interfaccia contiene zero o più clausole d'importazione e una o più dichiarazioni di sottoprogramma; tutte sono terminate da punto e virgola ';'.

   In ogni clausola d'importazione si cita il nome di un altro modulo indicando la lista (eventualmente vuota) delle entità da esso importate (cfr. righe 2-4 dell'esempio).

   I sottoprogrammi dichiarati possono essere funzioni o procedure. Nelle funzioni la parola chiave **subprogram** è preceduta dall'identificatore di tipo del valore restituito (cioè uno tra i tipi predefiniti **integer, real, char, boolean**, o un tipo definito dall'utente), e tutti i parametri sono passati per valore (cfr. righe 5 e 7). Nelle procedure la parola chiave **subprogram** non è preceduta da nulla, e almeno uno dei parametri è passato per indirizzo (cfr. riga 6).

   La dichiarazione di parametro passato per indirizzo inizia con la parola chiave **var**, mentre la dichiarazione di parametro passato per valore inizia direttamente con la lista dei nomi dei parametri (cfr. righe 5-7).

   Ogni dichiarazione di sottoprogramma contiene una o più dichiarazioni di parametro, separate da punto e virgola ';'. Ogni dichiarazione di parametro elenca uno o più parametri, separati da virgola ',', tutti dello stesso tipo e passati allo stesso modo (per valore o per indirizzo) (cfr. righe 5-7).

   Il tipo di parametro può essere indicato con l'identificatore di un tipo definito altrove oppure può essere un array, a una o più dimensioni (si intende che array a più dimensioni siano array i cui elementi sono array, per esempio una matrice a due dimensioni è un array i cui elementi sono array a una dimensione) (cfr. righe 6-7).

   Per ogni dimensione di parametro array si può specificare il numero dei componenti, con un numero intero, o lasciare tale numero non specificato (cfr. righe 6-7). Un array con indice non specificato non può contenere, né direttamente né indirettamente, un array con indice specificato; tutte le altre combinazioni sono permesse. Per esermpio, "**array of** [15] **array of real**" è corretto, mentre "**array of array of** [15] **real**" è errato.

   Per semplicità si supponga che tutti gli identificatori siano rappresentati dal terminale **id** e tutti i numeri dal terminale **num**.

   Ecco un esempio di dichiarazione d'interfaccia di modulo (parole chiave e identificatori riservati sono sottolineati).

```
1)  interface module modInt;
2)      from otherMod1 import type1, type2;
3)      from otherMod2 import;
4)      from otherMod3 import type3;
5)      type1  subprogram funcId1 (parId1, parId2: type1;  parId3: type2);
6)      subprogram procId1 (parId4: char; var parid5, parId6: type3;
                            arId1: array of [10] array of [5] integer);
7)      real subprogram funcId2 (arId2, arId3: array of array of real);
8)  end interface module modInt;
```

Soluzione

Ecco la grammatica (assioma INTERFACE):

⟨INTERFACE⟩	→	interface module id ⟨IMPORT_CLAUSE_LIST⟩ ⟨SUBPRO_HEADER_LIST⟩ end interface module id ';'
⟨IMPORT_CLAUSE_LIST⟩	→	(⟨IMPORT_CLAUSE⟩ ';')*
⟨SUBPRO_HEADER_LIST⟩	→	(⟨SUBPRO_HEADER⟩ ';')*
⟨IMPORT_CLAUSE⟩	→	from id import (ε | ⟨ID_LIST⟩)
⟨SUBPRO_HEADER⟩	→	⟨PROCEDURE⟩ | ⟨FUNCTION⟩
⟨PROCEDURE⟩	→	subprogram id '(' ⟨PROC_PAR_LIST⟩ ')'
⟨FUNCTION⟩	→	⟨TYPE⟩ subprogram id '(' ⟨FUNC_PAR_LIST⟩ ')'
⟨PROC_PAR_LIST⟩	→	(⟨PARAMS⟩ ';')* ⟨VAR_PARAMS⟩ ⟨MORE_PARMS⟩
⟨FUNC_PAR_LIST⟩	→	⟨PARAMS⟩ (';' ⟨PARAMS⟩)*
⟨PARAMS⟩	→	⟨ID_LIST⟩ ': ' (⟨TYPE⟩ | ⟨ARRAY⟩)
⟨VAR_PARAMS⟩	→	var ⟨PARAMS⟩
⟨MORE_PARAMS⟩	→	(';' (⟨PARAMS⟩ | ⟨VAR_PARAMS⟩))*
⟨ARRAY⟩	→	array of '[' num ']' (⟨TYPE⟩ | ⟨ARRAY⟩) | ⟨ARRAY_UNSP⟩
⟨ARRAY_UNSP⟩	→	array of (⟨TYPE⟩ | ⟨ARRAY_UNSP⟩)
⟨ID_LIST⟩	→	id (',' id)*
⟨TYPE⟩	→	integer | real | char | boolean | id

La struttura modulare stratificata della grammatica ne assicura la correttezza (o almeno la giustifica ragionevolmente).

3 Analisi sintattica e parsificatori 20%

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

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

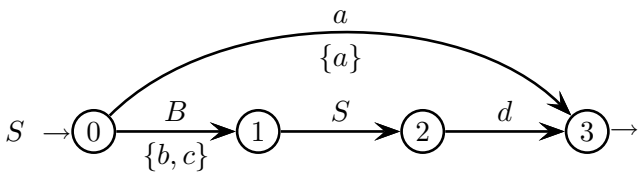
   Si risponda alle domande seguenti:

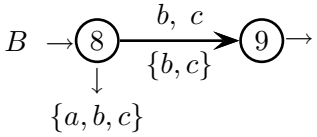
   (a) Si disegnino i grafi delle macchine S e B e si calcolino gli insiemi guida, verificando se G soddisfi la condizione $LL(k)$.

   (b) Se necessario, si progetti una grammatica equivalente a G che goda della proprietà $LL(1)$.

Soluzione

(a) Ecco i due grafi con gli insiemi guida per $k = 1$ (dove serve):





Per la macchina di B, la grammatica non è $LL(1)$. L'analisi con $k = 2$ non muta la situazione (verifica lasciata al lettore).

Poiché comunque la grammatica G ammette la derivazione ricorsiva a sinistra seguente

$$S \;\Rightarrow\; B\ S\ d \;\Rightarrow\; S\ d$$

essa certamente viola la condizione $LL(k)$, per qualunque $k \geq 1$.

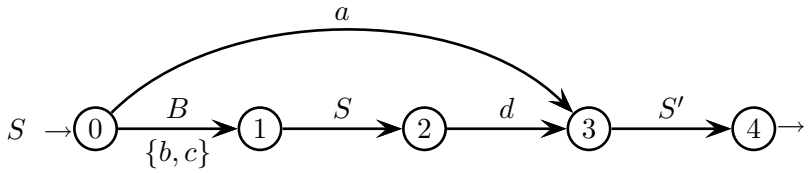
(b) Viene naturale di trasformare la ricorsione sinistra in ricorsione destra. Eliminando la regola vuota $B \to \varepsilon$ si ottiene la grammatica seguente

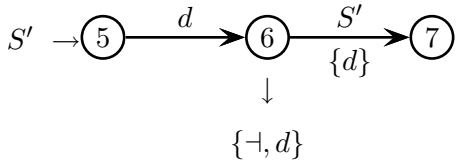
$$\begin{aligned} S &\;\to\; B\ S\ d \;\mid\; S\ d \;\mid\; a \\ B &\;\to\; b \;\mid\; c \end{aligned}$$

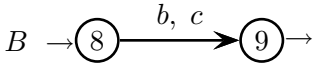
Trasformando la ricorsione sinistra in destra, si ottiene la grammatica equivalente seguente

$$\begin{aligned} S &\;\to\; (\ B\ S\ d \;\mid\; a\ )\ S' \\ S' &\;\to\; d\ S' \;\mid\; d \\ B &\;\to\; b \;\mid\; c \end{aligned}$$

la quale però ancora non risulta $LL(1)$, come si verifica dal calcolo degli insiemi guida:







A meglio vedere, si trova che la grammatica è ambigua: $b\,a\,d\,d\,d\,d$ ha due alberi

$\overbrace{\overbrace{b\,a\,d\,d}^{b\,S\,d}\,d}^{S'}$ oppure $\overbrace{\overbrace{b\,a\,d\,d}^{b\,S\,d}}^{S'}\overbrace{d\,d}$

In altro modo, osservando il linguaggio $L(G)$ si vede che le frasi hanno la forma

$B^m\,a\,d^n$ dove $B \in \{\ b,\ c\ \}$ e $0 \le m \le n$

generabile dalla grammatica $LL(1)$ seguente:

$$\begin{aligned} S &\rightarrow X\ d^* \\ X &\rightarrow (\ b\ \mid\ c\ )\ X\ d\ \mid\ a \end{aligned}$$

2. Data la grammatica G seguente (assioma S):

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

Si risponda alle domande seguenti:

(a) Si costruisca l'automa pilota di G e si dica se la grammatica è $LR(1)$.

(b) (facoltativa) Si costruisca una grammatica equivalente a G e avente la proprietà $LALR(1)$, e si discuta se sia anche possibile ottenere una grammatica con la proprietà $LR(0)$.

Soluzione

(a) Si lascia a lattore di tracciare il grafo $LR(1)$, non tanto grande, e di trovare i conflitti. Basta svoluppare il grafo sul percorso $b\,b\,S\,d$ per trovare, nel macrostato raggiunto da S, un conflitto spostamento-riduzione (spostare su d oppure fare la riduzione $D \to \varepsilon$).

Comunque, certamente la grammatica non è $LR(1)$ perché è ambigua.

(b) Il linguaggio $L(G) = a \mid b^+\, a\, d^*$ è regolare e quindi deterministico, ma contiene prefissi appartenenti al linguaggio stesso. Di conseguenza esso non è $LR(0)$.

La grammatica lineare a destra seguente

$$\begin{aligned} S &\to a \mid b\,X \\ X &\to b\,X \mid a\,Y \mid a \\ Y &\to d\,Y \mid d \end{aligned}$$

soddisfa la condizione $LALR(1)$ (facile verifica lasciata al lettore).

4 Traduzione e analisi semantica 20%

1. Certe espressioni aritmetiche sono definite dalla grammatica G seguente (assioma E):

$$G \begin{cases} E & \rightarrow & E + T \mid T \\ T & \rightarrow & i \times T \mid r \times T \mid i \mid r \end{cases}$$

dove i e r stanno per una costante intera e reale, rispettivamente (NB: si presume che il risultato di un'operazione in cui almeno uno dei due operandi è reale sia reale).

Si deve progettare uno schema puramente sintattico di traduzione per trasformare tali espressioni nel modo seguente:

- le operazioni tra due operandi interi si trasformano nella forma polacca prefissa; gli operatori si chiamano add e mult
- le operazioni dove almeno uno degli operandi non è intero si trasformano nella notazione funzionale con parentesi; gli operatori si chiamano fadd e fmult

Esempi:

$$\begin{aligned} i + i \times i &\Rightarrow \underline{add}\ i\ \underline{mult}\ i\ i \\ r + i \times i &\Rightarrow \underline{fadd}\ (\ r\ \underline{mult}\ i\ i\ ) \\ r + r \times r &\Rightarrow \underline{fadd}\ (\ r\ \underline{fmult}\ (\ r\ r\ )\ ) \end{aligned}$$

Si risponda alle domande seguenti:

(a) Si scriva lo schema di traduzione (modificando se necessario la grammatica sorgente G) e si disegnino gli alberi sintattici sorgente e immagine della frase $i + r \times i + i$.

(b) (facoltativa) Si dica se la traduzione può essere calcolata deterministicamente da un traduttore a pila.

Soluzione

(a) Grammatica di traduzione (assioma E):

$$
\begin{array}{lcl}
E & \rightarrow & E_i \mid E_r \\
\hline
E_i & \rightarrow & \frac{\varepsilon}{\underline{add}}\ E_i\ +\ T_i \\
E_i & \rightarrow & T_i \\
T_i & \rightarrow & \frac{i\ \times}{\underline{mult}\ i}\ T_i \\
T_i & \rightarrow & \frac{i}{i} \\
 & & \\
E_r & \rightarrow & \frac{\varepsilon}{\underline{fadd}\ (}\ E_i\ +\ T_r\ \frac{\varepsilon}{)} \\
E_r & \rightarrow & \frac{\varepsilon}{\underline{fadd}\ (}\ E_r\ +\ T_i\ \frac{\varepsilon}{)} \\
E_r & \rightarrow & \frac{\varepsilon}{\underline{fadd}\ (}\ E_r\ +\ T_r\ \frac{\varepsilon}{)} \\
E_r & \rightarrow & T_r \\
 & & \\
T_r & \rightarrow & \frac{i\ \times}{\underline{fmult}\ (\ i}\ T_r\ \frac{\varepsilon}{)} \\
T_r & \rightarrow & \frac{r\ \times}{\underline{fmult}\ (\ r}\ T_i\ \frac{\varepsilon}{)} \\
T_r & \rightarrow & \frac{r\ \times}{\underline{fmult}\ (\ r}\ T_r\ \frac{\varepsilon}{)} \\
T_r & \rightarrow & \frac{r}{r}
\end{array}
$$

(b) Discutiamo le due possibilità, $LR(k)$ e $LL(k)$.

- Pur essendo la grammatica sorgente del tipo $LR(k)$, poiché lo schema di traduzione non è del tipo *postfisso*, cade l'ipotesi che garantirebbe la possibilità di calcolare deterministicamente la traduzione con un parsificatore $LR(k)$.
- D'altra parte la grammatica sorgente è ricorsiva a sinistra, quindi inadatta all'analisi $LL(k)$.
  Se anche si girassero le ricorsioni a destra, e si aggiustasse la grammatica di traduzione in accordo, resterebbe la scelta non deterministica dovuta all'alternativa $E \rightarrow E_i \mid E_r$.

2. Si consideri la seguente porzione della grammatica astratta di un linguaggio di programmazione (assioma $\langle \mathsf{statL} \rangle$).

$$
\begin{array}{lcl}
\langle \mathsf{statL} \rangle & \rightarrow & \langle \mathsf{stat} \rangle \ \langle \mathsf{statL} \rangle \\
\langle \mathsf{statL} \rangle & \rightarrow & \langle \mathsf{stat} \rangle \\
\langle \mathsf{stat} \rangle & \rightarrow & \mathit{if} \ \langle \mathsf{cond} \rangle \ \mathit{then} \ \langle \mathsf{statL} \rangle \ \mathit{end} \\
\langle \mathsf{stat} \rangle & \rightarrow & \mathit{while} \ \langle \mathsf{cond} \rangle \ \mathit{do} \ \langle \mathsf{statL} \rangle \ \mathit{end} \\
\langle \mathsf{stat} \rangle & \rightarrow & \mathit{asg} \\
\langle \mathsf{cond} \rangle & \rightarrow & c
\end{array}
$$

Si risponda alle domande seguenti:

(a) Si definisca una grammatica con attributi che calcoli il numero di istruzioni di assegnamento presenti in un frammento di codice derivato dal nonterminale $\langle \mathsf{statL} \rangle$. Usare allo scopo un attributo n di tipo intero, oltre eventualmente ad altri necessari. Definire possibilmente le regole in modo tale che la grammatica ad attributi sia di tipo L.

(b) Sempre in riferimento a un frammento di codice derivato dalle regole proposte, si consideri, per ogni istruzione di assegnamento, il suo livello di annidamento in istruzioni iterative, cioè il numero di istruzioni *while* in cui essa risulta annidata. Per esempio, nel seguente frammento:

```
asg
while c do
    asg
    while c do
        asg
        asg
    end
end
```

per la prima istruzione di assegnamento il livello di annidamento è nullo (non è annidata in alcun *while*), per la seconda è uno, e per la terza e la quarta è due.

Si definiscano le regole per calcolare, in un attributo di nome p, il livello di annidamento di ogni istruzione di assegnamento, come sopra definito.

Si definiscano le regole per calcolare, in un attributo di nome np nella radice dell'albero, un valore intero dato dalla sommatoria del livello di annidamento di tutte le istruzioni di assegnamento. Per esempio nel frammento di codice sopra riportato il valore dell'attributo np nella radice è $5 = 0 + 1 + 2 + 2$.

Si dica se la grammatica definita è di tipo L, motivando la risposta.

---

sintassi	calcolo attributi (domanda (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: $\langle \mathsf{stat} \rangle_0 \rightarrow while \ \langle \mathsf{cond} \rangle_1 \ do \ \langle \mathsf{statL} \rangle_2 \ end$	
5: $\langle \mathsf{stat} \rangle_0 \rightarrow asg$	
6: $\langle \mathsf{cond} \rangle_0 \rightarrow c$	

sintassi	calcolo attributi (domanda (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: $\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{if} \ \langle \mathsf{cond} \rangle_1 \ \mathit{then} \ \langle \mathsf{statL} \rangle_2 \ \mathit{end}$	
4: $\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{while} \ \langle \mathsf{cond} \rangle_1 \ \mathit{do} \ \langle \mathsf{statL} \rangle_2 \ \mathit{end}$	
5: $\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{asg}$	
6: $\langle \mathsf{cond} \rangle_0 \rightarrow c$	

Soluzione

(a) L'attributo numerico n è sinistro:

	sintassi	calcolo attributi (domanda (a))
1:	$\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1 \ \langle \mathsf{statL} \rangle_2$	$n_0 = n_1 + n_2$
2:	$\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1$	$n_0 = n_1$
3:	$\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{if} \ \langle \mathsf{cond} \rangle_1 \ \mathit{then} \ \langle \mathsf{statL} \rangle_2 \ \mathit{end}$	$n_0 = n_2$
4:	$\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{while} \ \langle \mathsf{cond} \rangle_1 \ \mathit{do} \ \langle \mathsf{statL} \rangle_2 \ \mathit{end}$	$n_0 = n_2$
5:	$\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{asg}$	$n_0 = 1$
6:	$\langle \mathsf{cond} \rangle_0 \rightarrow c$	

Essendo puramente sintetizzata, la grammatica è di tipo L.

(b) Occorre un po' di eredità. L'attributo p è destro, mentre l'attributo np è sinistro.

	sintassi	calcolo attributi (domanda (b))
1:	$\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1 \ \langle \mathsf{statL} \rangle_2$	$p_1 = p_0$ $p_2 = p_0$ $np_0 = np_1 + np_2$
2:	$\langle \mathsf{statL} \rangle_0 \rightarrow \langle \mathsf{stat} \rangle_1$	$p_1 = p_0$ $np_0 = np_1$
3:	$\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{if} \ \langle \mathsf{cond} \rangle_1 \ \mathit{then} \ \langle \mathsf{statL} \rangle_2 \ \mathit{end}$	$p_2 = p_0$ $np_0 = np_2$
4:	$\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{while} \ \langle \mathsf{cond} \rangle_1 \ \mathit{do} \ \langle \mathsf{statL} \rangle_2 \ \mathit{end}$	$p_2 = p_0 + 1$ $np_0 = np_2$
5:	$\langle \mathsf{stat} \rangle_0 \rightarrow \mathit{asg}$	$np_0 = p_0$
6:	$\langle \mathsf{cond} \rangle_0 \rightarrow c$	

All'inizio l'attributo p vale 0. La grammatica è a una passata (one sweep), poiché l'attributo p dei figli dipende solo da attributi destri del padre e l'attributo np del padre dipende solo da attributi (non importa se destri o sinistri) dei figli. Inoltre per la valutazione semantica va bene l'ordine di visita dei figli da sinistra verso destra, dunque la grammatica è di tipo L.