Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

prof. S. Crespi Reghizzi, prof.ssa L. Sbattella
(prof. Luca Breveglieri)

Prova scritta - 7 settembre 2006 - Parte I: Teoria

CON SOLUZIONI PARZIALI

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 e automi a pila
    3. analisi sintattica e parsificatori
    4. traduzione 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 sufficiente conoscenza 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; in fondo a ogni sezione (1-4) c'è un foglio bianco addizionale.
- Tempo: Parte I (teoria): 2h.30m - Parte II (esercitazioni): 45m

1 Espressioni regolari e automi finiti 20%

1. È data l'espressione regolare seguente:

$$(a \mid b)^+ \, c \, a^+ \mid b^+ \, c \, (a \mid b)^+$$

(a) Si costruisca l'automa riconoscitore deterministico del linguaggio, descrivendo il ragionamento fatto per ottenerlo.

(b) Si minimizzi tale automa, se necessario.

(c) (facoltativo) Si verifichi se il linguaggio sia del tipo *locale*.

Soluzione

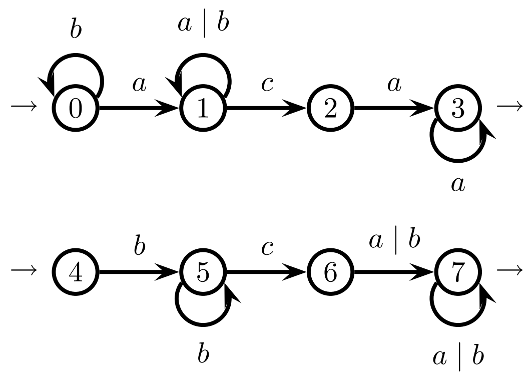
(a) Si preferisce trasformare l'espressione regolare, prima di costruire l'automa. Essa denota l'unione di due linguaggi non disgiunti:

$$\underbrace{(a \mid b)^+ \, c \, a^+}_{L_1} \mid \underbrace{b^+ \, c \, (a \mid b)^+}_{L_2}$$

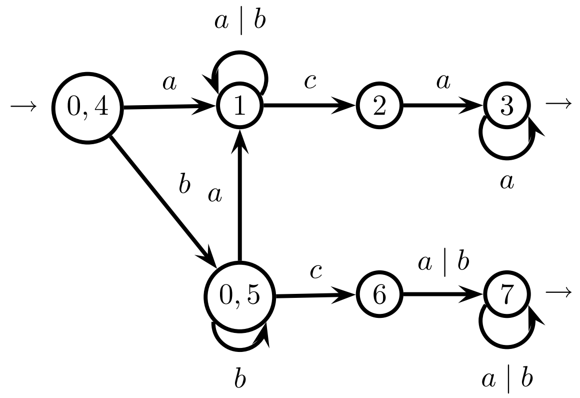
Infatti le stringhe $b^+ \, c \, a^+$ stanno in entrambi i linguaggi. Togliendo a L_1 tali stringhe comuni, il linguaggio non cambia, ed è definito dall'espressione seguente

$$\underbrace{b^* \, a \, (a \mid b)^* \, c \, a^+}_{L_3} \mid \underbrace{b^+ \, c \, (a \mid b)^+}_{L_2}$$

dalla quale si costruisce l'automa, indeterministico possedendo due stati iniziali:



Si può determinizzare facilmente l'automa, ottenendo quanto segue:



(b) L'automa costruito è minimo: ogni coppia di stati è infatti distinguibile. P. es. gli stati 2 e 6 sono distinguibili perché da 6, ma non da 2, è definita la mossa che legge b.

(c) Il linguaggio L non è locale. Per dimostrarlo basta mostrare due stringhe, una appartenente a L l'altra non appartenente, tali che:

- le due stringhe contengano gli stessi digrammi, e
- le due stringhe inizino e terminino con gli stessi caratteri

La frase $a\,c\,a$ e la stringa non valida $a\,c\,a\,c\,a$ soddisfano entrambe le condizioni.

2. Il linguaggio L_1 di alfabeto

$$\Sigma = \{ \ v, \ +, \ - \ \}$$

contiene tutte le espressioni aritmetiche con la variabile v e gli operatori somma e differenza, che soddisfano la condizione seguente:

$$\textsf{l'espressione contiene almeno una sottrazione} \tag{1}$$

Esempi:

$$v \ - \ v \qquad\qquad v \ + \ v \ - \ v \qquad\qquad v \ - \ v \ - \ v \ - \ v \ - \ v$$

Si svolgano i punti seguenti:

(a) Si scriva un'espressione regolare non ambigua del linguaggio L_1.

(b) Si costruisca in modo algoritmico l'automa (det. o non det. a scelta) riconoscitore del linguaggio L_1, spiegando il procedimento seguito per costruirlo.

(c) Si aggiunga al linguaggio L_1 il vincolo

$$\textsf{l'ultimo operatore dell'espressione deve essere una somma} \tag{2}$$

e si costruisca, spiegando il procedimento seguito, l'automa (det. o non det. a scelta) che riconosce il linguaggio

$$L_2 = \{ x \ \mid \ x \in L_1 \land x \text{ soddisfa il vincolo (2) } \}$$

Soluzione

(a) In prima battuta conviene vedere il linguaggio L_1 come il concatenamento di tre parti:

$$L_1 = L' \ \text{`}-\text{'} \ L'$$

dove L' è un'espressione aritmetica qualsiasi (non vuota), definita dalla e.r.:

$$L' = v \, ((\text{`}+\text{'} \ \mid \ \text{`}-\text{'}) \, v)^*$$

Tuttavia tale formula è notoriamente ambigua, quando la stringa contiene due o più sottrazioni. Per costruire una e.r. inambigua si differenziano i linguaggi che precedono e seguono il carattere '−', in modo che uno dei due soltanto possa contenere i caratteri '−'. Si scrive allora come segue:

$$L_1 = L'' \ \text{`}-\text{'} \ L'$$

dove si ha:

$$L'' = v \ \ (\text{`}+\text{'} \ v)^*$$

La e.r. così costruita

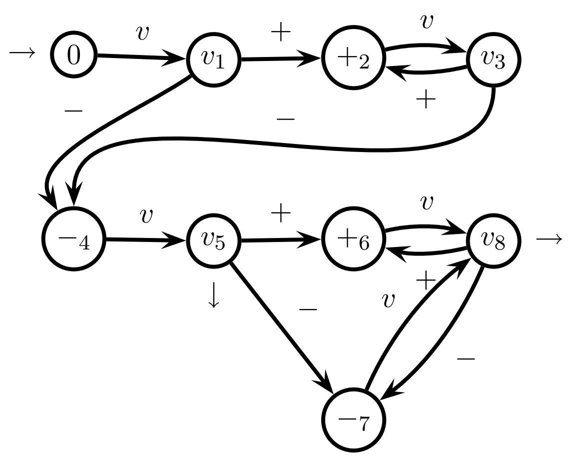
$$\underbrace{v \ (\text{`}+\text{'} \ v)^*}_{L''} \quad \underbrace{\text{`}-\text{'}}_{\text{prima sottrazione}} \quad \underbrace{v \ ((\text{`}+\text{'} \ \mid \ \text{`}-\text{'}) \ v)^*}_{L'}$$

è non ambigua. Infatti L'' arriva a coprire tutto il prefisso della stringa fino al primo segno di sottrazione escluso. Il termine L' copre tutto il rimenente suffisso della stringa. Inoltre entrambe le e.r. L'' e L' sono evidentemente inambigue (definiscono delle liste in modo associativo a destra).

(b) Per costruire il riconoscitore A_1 di L_1 conviene applicare il metodo più semplice. Scritta la e.r. numerata seguente (omettendo gli apici):

$$N_1 = v_1\ (+_2\ v_3)^*\ -_4\ \ v_5\ ((+_6\ \mid\ -_7)\ v_8)^*$$

si disegna il riconoscitore del ling. locale, come segue:



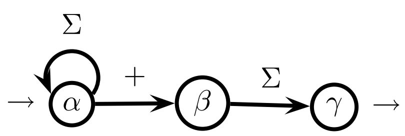
(c) Il linguaggio L_2, essendo definito dall'aggiunta della condizione 2, si può scrivere come intersezione di due linguaggi:

$$L_2 = L_1 \cap \{x \mid x \text{ è una espr. aritmetica che soddisfa la (2)}\}$$

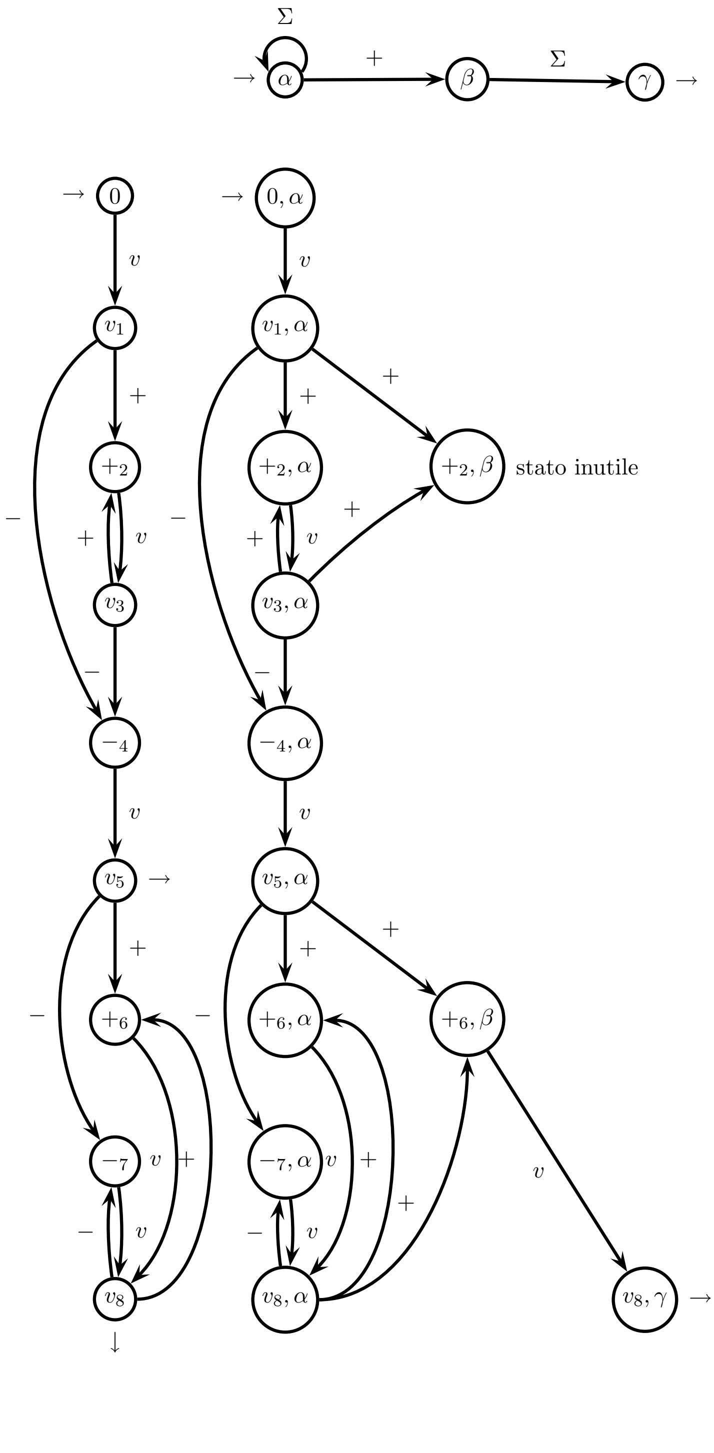
Senza perdita di precisione si può semplificare il secondo termine, scrivendo:

$$L_2 = L_1 \cap \underbrace{\{x\ \mid\ x \in \Sigma^* \wedge (\text{il penultimo carattere è `+'})\}}_{L_3}$$

Il riconoscitore di L_2 si costruirà ora come prodotto cartesiano di due macchine, M_1 e il riconoscitore M_3 di L_3, sotto mostrato:



Macchina prodotto cartesiano:



2 Grammatiche libere e automi a pila 20%

1. Si consideri il linguaggio di Dyck di alfabeto $\Sigma = \{a, \overline{a}, b, \overline{b}\}$, dove a, b sono parentesi aperte e $\overline{a}$, $\overline{b}$ sono le corrispondenti parentesi chiuse. Eccone la grammatica G:

$$G = \begin{cases} S & \to & a\ S\ \overline{a}\ S \\ S & \to & b\ S\ \overline{b}\ S \\ S & \to & \varepsilon \end{cases}$$

Poi è dato il linguaggio regolare R seguente (definito mediante complemento $\neg$):

$$R = \neg\left(\Sigma^* \{aa, \overline{a}b\}\, \Sigma^*\right)$$

Infine si ponga:

$$L = \text{Dyck} \cap R$$

Dunque L è definito come intersezione del linguaggio di Dyck di cui sopra e di R.

Si svolgano i punti seguenti:

(a) Si elenchino tutte le stringhe di L di lunghezza ≤ 4.

(b) Si scriva la grammatica G' di L in forma BNF (non estesa).

Soluzione

(a) $\varepsilon \quad a\overline{a} \quad b\overline{b} \quad ab\overline{b}\overline{a} \quad bb\overline{bb} \quad a\overline{a}a\overline{a} \quad b\overline{b}b\overline{b} \quad b\overline{b}a\overline{a} \quad b\overline{b}a\overline{a}$

(b)

$$G' = \begin{cases} S & \to & A \mid B \\ A & \to & a\ B\ \overline{a}\ A \mid \varepsilon \\ B & \to & b\ S\ \overline{b}\ S \mid \varepsilon \\ S & \to & \varepsilon \end{cases}$$

2. Si progetti la grammatica G in forma EBNF (estesa) non ambigua che modella il ciclo `while` in linguaggio C:

   - il corpo del ciclo non è vuoto e può contenere istruzioni elementari C (separate da ";") e altri cicli `while` annidati
   - l'istruzione elementare C può essere:
     - assegnamento di espressione a variabile
     - chiamata a procedura con parametri attuali (anche nesuno)
   - le espressioni contengono identificatori di variabile (come in sintassi C), numeri interi positivi o nulli, operatori "+" (addizione) e "$*$" (moltiplicazione); gli operatori sono in forma infissa e le espressioni possono contenere eventuali parentesi tonde "(" e ")"; la moltiplicazione precede l'addizione
   - la condizione del ciclo è del tipo: "espr. op. rel. espr.", dove l'operatore relazionale può essere "==", "! =", "<" o ">"

   Esempio:

```
while (a < 10) {
   a = b + c * 17;
   proc1 (a, 10, 5 + c);
   while (b == 20) {
      c = (30 + a) * 7;
   }
   proc2 ( );
}
```

   Si scriva la grammatica G in questione (in forma EBNF non ambigua). Quali aspetti semantici del ciclo `while` effettivo del linguaggio di programmazione C non sono esprimibili sintatticamente ?

Soluzione

Abbastanza ovvia ... da fare.

3 Analisi sintattica e parsificatori 20%

1. È data la grammatica estesa (EBNF) G seguente:

$$S \rightarrow D\ (d\ D)^*$$

$$D \rightarrow a\ D\ b\ D \mid \varepsilon$$

Si svolgano i punti seguenti:

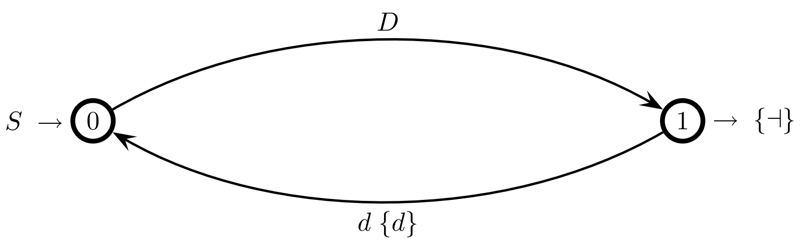
(a) Si disegni la rete delle macchine ricorsive equivalente a G.

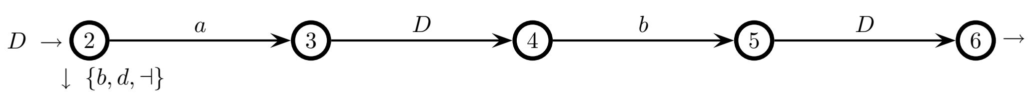
(b) Si calcolino gli insiemi guida e si verifichi se la rete delle macchine sia $LL(k)$.

(c) Se necessario, si trasformi la grammatica G per ottenere una grammatica equivalente $LL(k)$.

Soluzione

(a) Si disegna già nella forma deterministica la rete di automi, composta dalle macchine S e D:





(b) Gli insiemi guida rilevanti (dove ci sono biforcazioni) sono già sul grafo. Nessuno stato viola la condizione $LL(1)$.

(c) Non c'è nulla da fare, si ha già $k = 1$.

2. È data la grammatica G seguente:

$$S \rightarrow a\,S\,b \mid a\,S\,b\,b \mid c$$

Nota: non è permesso modificare la grammatica.

Si svolgano i punti seguenti:

(a) Per G si applichi l'algoritmo di Earley alla stringa seguente:

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

calcolando gli stati attraversati dall'algoritmo e riportandoli nella tabella predisposta alla pagina successiva.

(b) Si mostrino l'albero o gli alberi sintattici trovati dall'algoritmo.

(c) Si discuta se il linguaggio $L(G)$ sia deterministico.

Soluzione

(a) Da fare ...

(b) Da fare ... la stringa analizzata è ambigua di grado 2, ammette 2 alberi sinatttici distinti.

Bozza di schema per la simulazione dell'algoritmo di Earley								
stato 0	pos. a	stato 1	pos. a	stato 2	pos. c	stato 3	pos. b	stato 4

Bozza di schema per la simulazione dell'algoritmo di Earley

stato 4	pos. b	stato 5	pos. b	stato 6	pos. $\dashv$	stato 7	pos. non usare !	non usare !

(c) Vedere per esempio se si possa fare con un automa a pila det. Ragionevolmente sì, perché si usa la pila per assicurare che le b non siano più del doppio delle a ... da fare

4 Traduzione e analisi semantica 20%

1. Dato l'alfabeto sorgente $\Sigma = \{ \text{`1'}, \text{`('}, \text{`)'} \}$, si consideri il linguaggio sorgente generato dalla grammatica seguente:

$$S \to \text{`('}\ S\ \text{`)'}\ S \ \mid \ \text{`1'}\ S \ \mid \ \varepsilon$$

Le stringhe sorgente hanno struttura parentetica alla Dyck, ma con inseriti terminali "1" a piacere in ogni modo e numero possibile (vedi sotto).

Dato l'alfabeto pozzo $\Delta = \{ \text{`}p\text{'}, \text{`}d\text{'}, \text{`('}, \text{`)'} \}$, si consideri la traduzione sintattica $\tau \colon \Sigma \to \Delta$ esemplificata come segue:

sorg.	ε	1	$1\ 1$	$(\ )$	$(\ 1\ )$	$(\ 1\ 1\ )$	$(\ 1\ (\ 1\ )\ 1\ 1\ 1\ )\ 1$	$1\ (\ 1\ (\ 1\ 1\ )\ 1\ 1\ 1\ )\ 1\ 1$
pozzo	p	d	p	(p)	$(\ d\ )$	$(\ \ p\ )$	$(\ \ (\ d\ )\ \ \ p\ )\ d$	$(\ \ (\ \ p\ )\ \ \ p\ )\ \ d$

Chiaramente la traduzione τ si comporta come segue:

- conserva inalterata la struttura parentetica della sorgente
- cancella tutti terminali "1" presenti nella sorgente
- per ogni coppia di parentesi conta i terminali "1" immediatamente contenuti nella coppia (allo stesso livello), e nella stringa pozzo emette p o d in corrispondenza dell'ultimo "1" della coppia secondo il conteggio abbia dato valore *pari* o *dispari* (0 è da considerare pari)

Si svolga *a scelta* uno dei due punti seguenti (non entrambi):

(a) Si progetti lo schema sintattico di traduzione (gramm. sorgente e pozzo) che realizza la traduzione τ. Suggerimento (non obbligatorio): si prenda spunto dalla grammatica sorgente data sopra modificandola come e dove opportuno.

(b) Si progetti l'automa trasduttore a pila det. che realizza la traduzione τ.

Soluzione

(a) Soluzione accettabile:

$$G_\tau = \left\{ \begin{array}{lcll} & & \text{sorgente} & \text{pozzo} \\ S & \to & P & P \\ P & \to & \text{`1'}\ D \mid \text{`('}\ P\ \text{`)'}\ P \mid \varepsilon & D \mid \text{`('}\ P\ \text{`)'}\ P \mid p \\ D & \to & \text{`1'}\ P \mid \text{`('}\ P\ \text{`)'}\ D \mid \varepsilon & P \mid \text{`('}\ P\ \text{`)'}\ D \mid d \end{array} \right.$$

(b) Da fare ...

2. È dato un supporto sintattico G che genera espressioni algebriche (senza sottoespr. tra parentesi) contenenti addizione e moltiplicazione di numeri complessi (la moltiplicazione precede l'addizione), i num. compl. sono rappresentati nella forma "parte reale + parte imm. i"; per non fare confusione ogni num. compl. è racchiuso tra parentesi; "$\langle \text{int} \rangle$" è un numero intero che non occorre espandere; "i" è un terminale.

$$G \begin{cases} S \rightarrow A \text{ `}+\text{' } S \\ S \rightarrow A \\ A \rightarrow C \text{ `}\times\text{' } A \\ A \rightarrow C \\ C \rightarrow (\langle \text{int} \rangle \text{ `}+\text{' } \langle \text{int} \rangle \text{ `}i\text{'}) \\ \langle \text{int} \rangle \rightarrow \dots \end{cases}$$

esempio di espressione:
$(1+1i) \ + \ (1+2i) \ \times \ (2+1i)$

Si deve progettare una grammatica con attributi G', estendendo la sintassi G con regole semantiche, che effettui il calcolo dell'espressione. Attributi già dati:

- l'attributo α (valevole per "$\langle \text{int} \rangle$") che restituisce il valore di "$\langle \text{int} \rangle$"
- l'attributo β (valevole per S) che dà il valore dell'espressione

Si possono estendere gli attributi ad altri nonterminali e introdurre nuovi attributi, come sembra necessario od opportuno.

Ecco i punti da svolgere:

(a) Progettare lo schema che calcola l'espressione. Sottopunti:
   i. Elencare gli attributi, con il rispettivo tipo e significato.
   ii. Scrivere le funzioni semantiche che calcolano gli attributi (alle pagine successive sono già pronti gli schemi da compilare).
   iii. Disegnare i grafi delle dipendenze funzionali tra attributi, per ciascuna produzione separatamente.
   iv. Stabilire se la grammatica sia di tipo a una sola scansione.
   v. Stabilire se la grammatica sia di tipo L.

(b) Il calcolo di addizione e moltiplicazione di num. compl. si riconduce a un certo numero di operazioni su reali. Si osservi che il calcolo di un termine contenente un fattore complesso nullo (parte re. $=$ parte im. $= 0$) si può semplificare riducendo il numero di operazioni su reali. Estendere la grammatica ad attributi in modo da introdurre tale ottimizzazione che risparmia operazioni su reali (già dato un ulteriore attributo γ, vedi pagina seguente).

---

tipo	nome	nonterminali	dominio	significato
		già dati		(per punti (a) e (b)
sx	α	$\langle$int$\rangle$	numero int.	valore parte re. o im.
sx	β	S	numeri compl.	valore espressione
		da aggiungere o estendere		per punto (a)
		da aggiungere o estendere		per punto (b)
sx	γ	$\langle$int$\rangle$	V / F	nullità di "$\langle$int$\rangle$"

sintassi	*funzioni semantiche* (per punto (a))
$S \to A$ ‘ $+$ ’ S	
$S \to A$	
$A \to C$ ‘ $\times$ ’ A	

sintassi	*funzioni semantiche* (per punto (a))
$A \to C$	
$C \to (\langle \text{int} \rangle \text{ `}+\text{' } \langle \text{int} \rangle \text{ `}i\text{'})$	
$\langle \text{int} \rangle \to \dots$	

sintassi	*funzioni semantiche* (per punto (b))
$S \to A$ ‘ $+$ ’ S	
$S \to A$	
$A \to C$ ‘ $\times$ ’ A	

sintassi	*funzioni semantiche* (per punto (b))
$A \to C$	
$C \to (\langle\text{int}\rangle\ `+\text{'}\ \langle\text{int}\rangle\ `i\text{'})$	
$\langle\text{int}\rangle \to \ldots$	

Soluzione

(a) Da fare
(b) Da fare ...
(c) Da fare ...
(d) Da fare ...
(e) Da fare ...