Linguaggi Formali e Compilatori

Prof. L. Breveglieri e S. Crespi Reghizzi

Soluzione Prova scritta 01/07/2005 - Parte I: Teoria

ISTRUZIONI:

- L'esame si compone di due parti:
  - I (80%) Teoria:
    1. espressioni reg. e automi finiti
    2. grammatiche
    3. analisi sintattica
    4. traduzione e semantica
  - II (20%) Esercitazioni Flex e Bison
- Per superare l'esame l'allievo deve superare entrambe le parti (I e II) nello stesso appello oppure in appelli diversi della stessa sessione d'esame.
- Per superare la parte I (teoria) occorre dimostrare sufficiente conoscenza di tutte le quattro sottoparti (1-4).
- Tempo: Parte I (esercitazioni): 30 min - Parte II (teoria): 2.30 ore.
- È permesso consultare libri e appunti personali.

1 Espressioni regolari e automi finiti 20%

1. Dato il ling. di alfabeto $\Sigma = \{a, b, c\}$

$$L = \Sigma^* ab\Sigma^* \cap \neg\left(\Sigma^* bc\Sigma^*\right) \cap a\Sigma^*$$

   (a) Si elenchino la (o le) frasi di lunghezza ≤ 3

   (b) Si costruisca, commentando il procedimento seguito, l'automa che riconosce L.

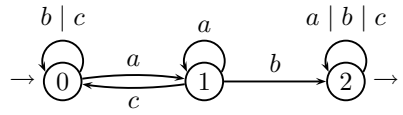
   (c) (facoltativo) Si calcoli l'espressione regolare di L con i soli operatori di concatenamento, unione, stella e croce.

Soluzione

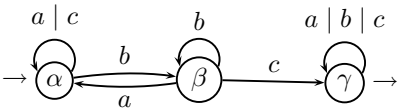
(a) ab, aab, aba, abb

(b) $L = \underbrace{\Sigma^* ab\Sigma^*}_{M_1} \cap \underbrace{\neg\left(\Sigma^* bc\Sigma^*\right)}_{M_2} \cap \underbrace{a\Sigma^*}_{M_3}$

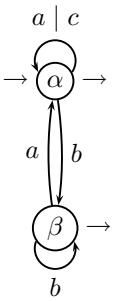
M_1 :



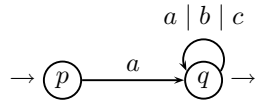
$\neg M_2$:



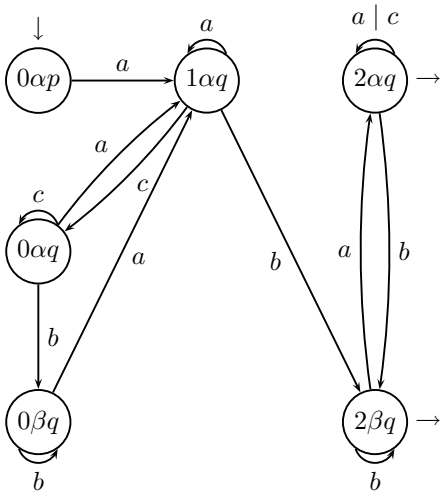
M_2 :



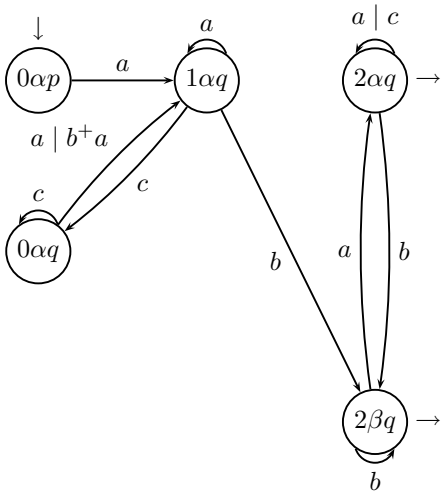
M_3 :



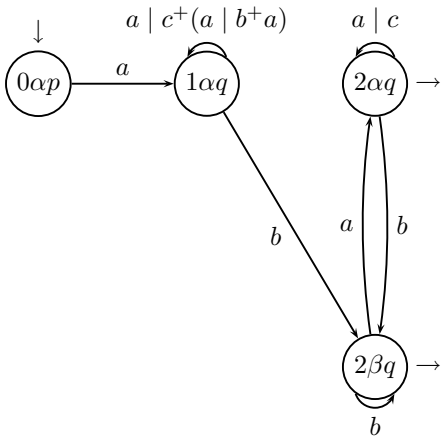
prodotto cartesiano $M_1 \times M_2 \times M_3$:



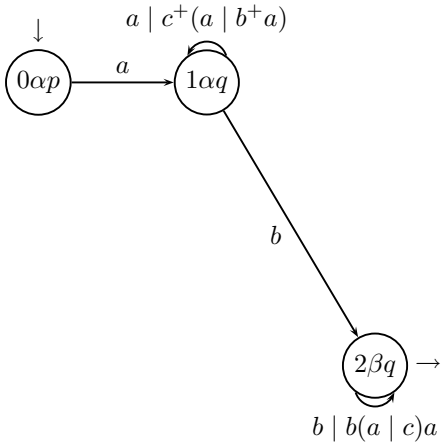
(c) Per calcolare l'espressione regolare si può usare il metodo di eliminazione. Prima si elimina lo stato $0\beta q$:



Eliminando lo stato $0\alpha q$:

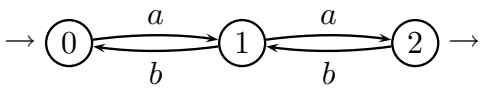


Eliminando lo stato e $2\alpha q$:



$$L = a\left(a \mid c^+\left(a \mid b^+a\right)\right)^* b\left(b \mid b\left(a \mid c\right)a\right)^*$$

2. È dato l'automa M:



(a) Si consideri l'insieme dei suffissi propri delle frasi di $L(M)$ ossia il ling.

$$L_S = \{y \mid zy \in L(M) \wedge y \neq \varepsilon \wedge z \neq \varepsilon\}$$

Si elenchino le frasi di lunghezza ≤ 2 del ling. L_S

(b) Si costruisca il riconoscitore deterministico minimo di L_S spiegando il procedimento seguito.

Soluzione

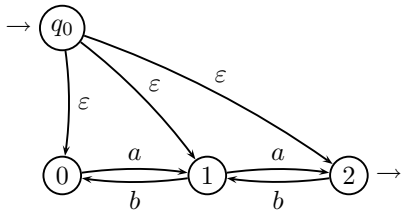
(a) Frasi di lunghezza ≤ 2: a, aa, ba

(b) Per costruire il riconoscitore deterministico si vede che i suffissi propri sono:

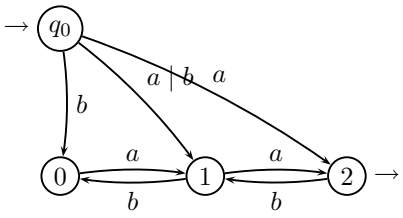
i. le stringhe non vuote che M riconosce partendo da uno stato non iniziale, cioè da 1 e da 2;

ii. ogni frase $x \in L(M)$ è suffisso di un'altra frase $abx \in L(M)$.

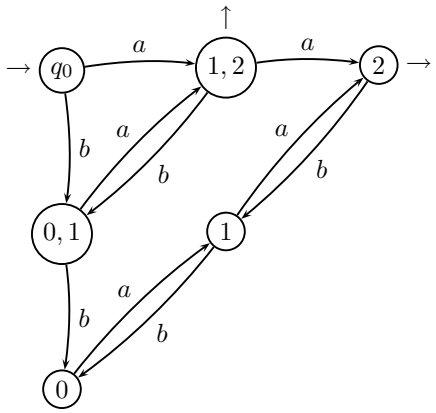
Si modifica allora l'automa in modo che tutti gli stati siano iniziali e, poiché ε non deve essere riconosciuta, nessuno stato iniziale sia finale:



Si eliminano le mosse spontanee (taglio ε-transizioni):



Si determinizza (costruzione dei sottinsiemi):



Non è detto che l'automa determinizzato sia in forma minima.

2 Grammatiche 20%

1. Si consideri il ling. di Dyck di alfabeto ‘(’, ‘)’, ‘[’, ‘]’.

   (a) Trovare la grammatica che genera tutte e sole le stringhe di Dyck dove: (1) se il numero totale di coppie di parentesi tonde è pari, il numero totale di coppie di parentesi quadre è dispari.
   Esempi: [ ], ([ ]), (([ ])), (([ ]([ ]))).
   Contresempio: (([ ][ ])).

Soluzione

$S \to S_{pd}$
$S_{pp} \to (S_{pp})S_{dp} \mid (S_{pd})S_{dd} \mid (S_{dp})S_{pp} \mid (S_{dd})S_{pd}$
$S_{pp} \to [S_{pp}]S_{pd} \mid [S_{dp}]S_{dd} \mid [S_{pd}]S_{pp} \mid [S_{dd}]S_{dp} \mid \varepsilon$
. . . (similmente per gli altri tre nonterminali)

Chiaramente, dovendosi contare indipendentemente sia le parentesi quadre sia quelle tonde, occorrono dei nonterminali che codifichino tali informazioni.

2. Costruire la grammatica EBNF non ambigua del linguaggio della teoria elementare delle classi, così definito. Sono ammesse classi di elementi e classi di classi, ecc, a qualsiasi livello di gerarchia e non necessariamente omogenee. Il linguaggio consta delle parti seguenti:

   (a) I simboli ';', '{', '}', '(', ')', '|', '$\cup$', '$\cap$', '$\neg$', '$\subset$', '$\in$', '$:=$' sono terminali.

   (b) Le lettere minuscole $a, \ldots, z$ sono simboli terminali rappresentanti elementi.

   (c) Le lettere maiuscole $A, \ldots, Z$ sono simboli terminali rappresentanti nomi di classi, di qualunque tipo.

   (d) Se C_1, C_2 sono classi ed e è un elemento, $C_1 \subset C_2$, $e \in C_1$ e $C_1 \in C_2$ sono predicati.

   (e) Se C_1, C_2 sono classi, $C_1 \cup C_2$, $C_1 \cap C_2$ e $\neg C_1$ sono classi.

   (f) Se A è un nome di classe ed e è un elemento, si possono formare classi secondo le regole seguenti:

      i. $\{\ \}$ (è la classe vuota)
      ii. $\{e\}$
      iii. $\{A\}$
      iv. $\{e|\quad \text{predicato}\}$
      v. $\{A|\quad \text{predicato}\}$

   (g) Se A è un nome di classe, la scrittura "$A:=$ classe;" è un assegnamento che definisce la nuova classe di nome A, con contenuto "classe" (una qualsiasi stringa di terminali e nonterminali che definisca una classe secondo le regole da (a-f)).

   Il linguaggio consiste in una successione (non vuota) di assegnamenti a nomi di classi. La grammatica che lo genera deve essere non ambigua, e le precedenze sono $\neg$ precede $\cap$ precede $\cup$.

   Esempi:

   $A:=\{e\}$; $\quad A:=\{e\}\cup\{\{d\}\}$; $\quad A:=\{e|\ e\in A\}$; $\quad A:=B\cup(C\cap D)$;

   $A:=\{B|\ B\in\{e\}\cup D\}$; $\quad A:=\{B|\ B\in\{C|\quad C\subset D\}\}$;

   Quali aspetti semantici non sono modellabili mediante la grammatica proposta?

Soluzione

$$
\begin{array}{rcl}
S & \rightarrow & \left(\langle ass \rangle \text{ `;}' \right)^{+} \\
\langle nomeclasse \rangle & \rightarrow & [\text{`}A' - \text{`}Z'] \\
\langle elem \rangle & \rightarrow & [\text{`}a' - \text{`}z'] \\
\langle ass \rangle & \rightarrow & \langle nomeclasse \rangle \text{ `:=}' \ \langle espr \rangle \\
\langle espr \rangle & \rightarrow & \langle term \rangle \left(\text{ `}\cup' \ \langle term \rangle \right)^{*} \\
\langle term \rangle & \rightarrow & \langle fatt \rangle \left(\text{ `}\cap' \ \langle fatt \rangle \right)^{*} \\
\langle fatt \rangle & \rightarrow & \text{`}\neg' \ \langle var \rangle \mid \langle var \rangle \\
\langle var \rangle & \rightarrow & \text{`(}' \ \langle espr \rangle \text{ `)}' \mid \\
 & & \mid \text{`\{}' \text{ `\}}' \mid \\
 & & \mid \text{`\{}' \ \langle elem \rangle \left( \ \varepsilon \mid \text{`}|' \ \langle pred \rangle \ \right) \ \text{`\}}' \mid \\
 & & \mid \text{`\{}' \ \langle nomeclasse \rangle \left( \ \varepsilon \mid \text{`}|' \ \langle pred \rangle \ \right) \ \text{`\}}' \\
\langle pred \rangle & \rightarrow & \langle espr \rangle \text{ `}\subset' \ \langle espr \rangle \mid \\
 & & \mid \langle elem \rangle \text{ `}\in' \ \langle espr \rangle \mid \\
 & & \mid \langle espr \rangle \text{ `}\in' \ \langle espr \rangle
\end{array}
$$

Non sono modellabili sintatticamente: che un nome di classe usato in un'espressione sia già stato assegnato; che un nome di classe venga assegnato due o più volte; che non si tenti di porre l'appartenenza di una classe a un elemento; ecc.

3 Grammatiche e analisi sintattica 20%

1. Calcolare gli insiemi guida delle grammatica data per $k = 1$, indicare dove essa non risulta $LL(1)$, e trovare il valore di $k > 1$ per cui la grammatica risulta $LL(k)$, motivando in breve la risposta.

Regola	Insieme Guida
$S \to XaBSc$	
$X \to aY$	
$X \to \varepsilon$	
$Y \to bX$	
$Y \to \varepsilon$	
$B \to abD$	
$D \to bD$	
$D \to b$	

Soluzione

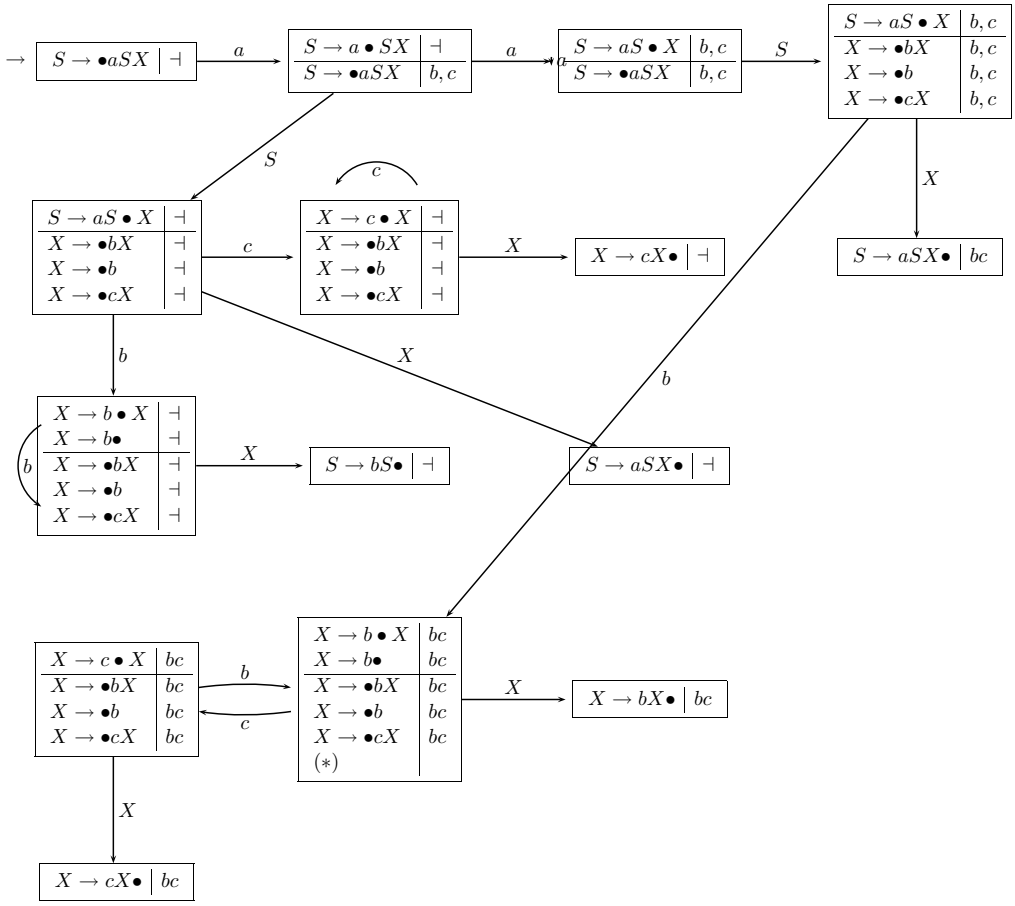
Regola	Guida $k=1$	$k>1$
$S \to XaBSc$	a	
$X \to aY$	$a:$ ✗	$aaa, aba:$ ✓ $k=3$
$X \to \varepsilon$	$a:$ ✗	$aab:$ ✓ $k=3$
$Y \to bX$	$b:$ ✓	
$Y \to \varepsilon$	$a:$ ✓	
$B \to abD$	a	
$D \to bD$	$b:$ ✗	$bb:$ ✓ $k=2$
$D \to b$	$b:$ ✗	$ba:$ ✓ $k=2$

La gramamtica risulta dunque $LL\,(3)$.

2. Costruire il riconoscitore deterministico dei prefissi ascendenti e indicare in quale(i) stato(i) sono violate le condizioni LR(1).

$$S \to aSX \qquad X \to bX \qquad X \to b \qquad X \to cX$$

Soluzione



Conflitto LR(1) nello stato (*). Nota bene: la grammatica di per sé è nonterminante, ma ciò non impatta sull'esame della grammatica.

3. Domanda facoltativa. Si consideri il linguaggio delle espressioni aritmetiche con tre operatori binari $+$, $-$ e $\times$, e un nome di variabile v. I due operatori $+$ e $\times$ sono di tipo prefisso, mentre l'operatore $-$ è di tipo postfisso (si tratta dunque di una forma mista prefissa-postfissa).

   Al fine di interpretare la forma mista in modo univoco, si rifletta se sia necessario o no avere parentesi, e si dica eventualmente per quale(i) operatore(i). Qualunque sia la conclusione si scriva la grammatica che genera il linguaggio, in forma ambigua o no, a scelta, ma specificando quale delle due.

   Suggerimento: si rifletta sulla stringa $v\ +\ \times\ v\ v\ \times\ v\ v\ -\ v\ -$.

Soluzione

Si esaminino le due parentetizzazioni seguenti, entrambe valide:

$((v\ (+\ (\times\ v\ v)\ (\times\ v\ v))\ -)\ v\ -)$

$(v\ (+\ (\times\ v\ v)\ (\times\ (v\ v\ -)\ v))\ -)$

Ne viene che l'interpretazione è ambigua, e che dunque bisogna dotare di parentesi almeno alcuni operatori.

Conviene mettere le parentesi all'operatore $-$, piuttosto che a $+$ e $\times$, per economia di parentesi.

Quanto alla grammatica, è facile: generazione in forma prefissa per $+$ e $\times$, senza parentesi, e in forma postfissa per $-$, ma con parentesi. Si lascia l'esercizio al lettore.

4 Traduzione e semantica 20%

1. Il ling. sorgente L_1 è una variante del ling. di Dyck, di alfabeto $\Sigma = \{'(','), b\}$, con b che designa uno spazio bianco. Le lettere b possono comparire in ogni posizione, anche ripetutamente.

   (a) Si progetti una gramm. di traduzione ossia uno schema di traduzione (semplice, senza attributi semantici) per convertire una frase di L_1 nella corrispondente stringa in cui due parentesi sono sempre separate da una, e una sola, b e non ci sono altre b oltre a queste.

   Es. :

frase sorgente	frase pozzo
$bb(())$	$(b(b)b)$
(b)	(b)
bbb	ε
$()(bb(bbb)bb)b$	$(b)b(b(b)b)$

   (b) Si verifichi se sia possibile costruire un traduttore deterministico partendo dalla vostra grammatica di traduzione.

Soluzione

Si presentano due soluzioni.

(a) Prima soluzione

i. Gramm. di traduzione: l'idea per la gramm. sorgente è di modificare una grammatica di Dyck inserendo il nonterminale B che genera 0 o più b tra ogni coppia di parentesi. Nella gramm. pozzo il nonterminale B genererà invece una sola b. Perfezionando l'idea, si vede però che

- le stringhe b^* poste a prefisso e a suffisso non vanno tradotte in una lettera b ma cancellate. Occorre quindi differenziare tale caso dal precedente, introducendo un altro nonterminale A.
- è meglio che la frase b^* sia derivata direttamente dall'assioma.

gramm. sorgente	gramm. pozzo
$S \to AXA$	$S \to AXA$
$S \to A$	$S \to A$
$A \to bA$	$A \to A$
$A \to \varepsilon$	$A \to \varepsilon$
$X \to (BXB)BX$	$X \to (BXB)BX$
$X \to (B)BX$	$X \to (B)BX$
$X \to (BXB)$	$X \to (BXB)$
$B \to bB$	$B \to B$
$B \to \varepsilon$	$B \to b$

ii. Per vedere se sia possibile costruire un traduttore deterministico, si può verificare se la gramm. sorgente sia LL(k) o possa essere trasformata in una gramm equivalente LL(k). I nonterminali S e X non sono LL(1) ma fattorizzando a sin. le alternative conflittuali si ottiene la gramm. equivalente

gramm. sorgente	insieme guida
$S \to AW$	
$W \to XA$	(
$W \to \varepsilon$	$\dashv$
$A \to bA$	b
$A \to \varepsilon$	$\dashv$, (
$X \to (BZY$	
$Z \to XB$	(
$Z \to)$	)
$Y \to BX$	b, (
$Y \to \varepsilon$	b, (
$B \to bB$	b
$B \to \varepsilon$	(

nella quale resta però ancora un conflitto per Y. Questa tecnica non permette dunque di ottenere un traduttore a pila deterministico.

(b) Seconda soluzione, più compatta ma la gramm. sorgente è ambigua:

gramm. sorgente	insieme guida
$S_0 \to B(BSB)B$	$S \to B(bBSB)bBSB$
$S \to B(BSB)BSB$	$S \to B$
$S \to B$	$\dashv$
$B \to b^*$	$B \to \varepsilon$

Ovviamente, essendo la grammatica sorgente ambigua, le determinizzazione risulta ben più ardua se non impossibile tranne modificando radicalmente la gramamtica stessa.

2. È data la sintassi di certi programmi contenenti istruzioni di assegnamento:
   $S \to AS \mid A$
   $A \to i := E$
   $E \to i + E \mid i \mid c$
   dove i è un identificatore di variabile, e c una costante.

   (a) Progettare una gramm. ad attributi per controllare che ogni variabile usata in un'espressione abbia un valore precedentemente calcolato da un assegnamento. Si veda l'albero dell'esempio.
   Si usino i seguenti attributi, estendendoli o aggiungendone altri se necessario:

   predicato semantico: "α of S", sintetizzato: dice se il programma supera il controllo.

   variabili usate: "use of E", sintetizzato: l'insieme degli identificatori che compaiono nella espressione E.

   variabili definite: " def of A", ereditato: l'insieme degli identificatori inizializzati da un precedente assegnamento

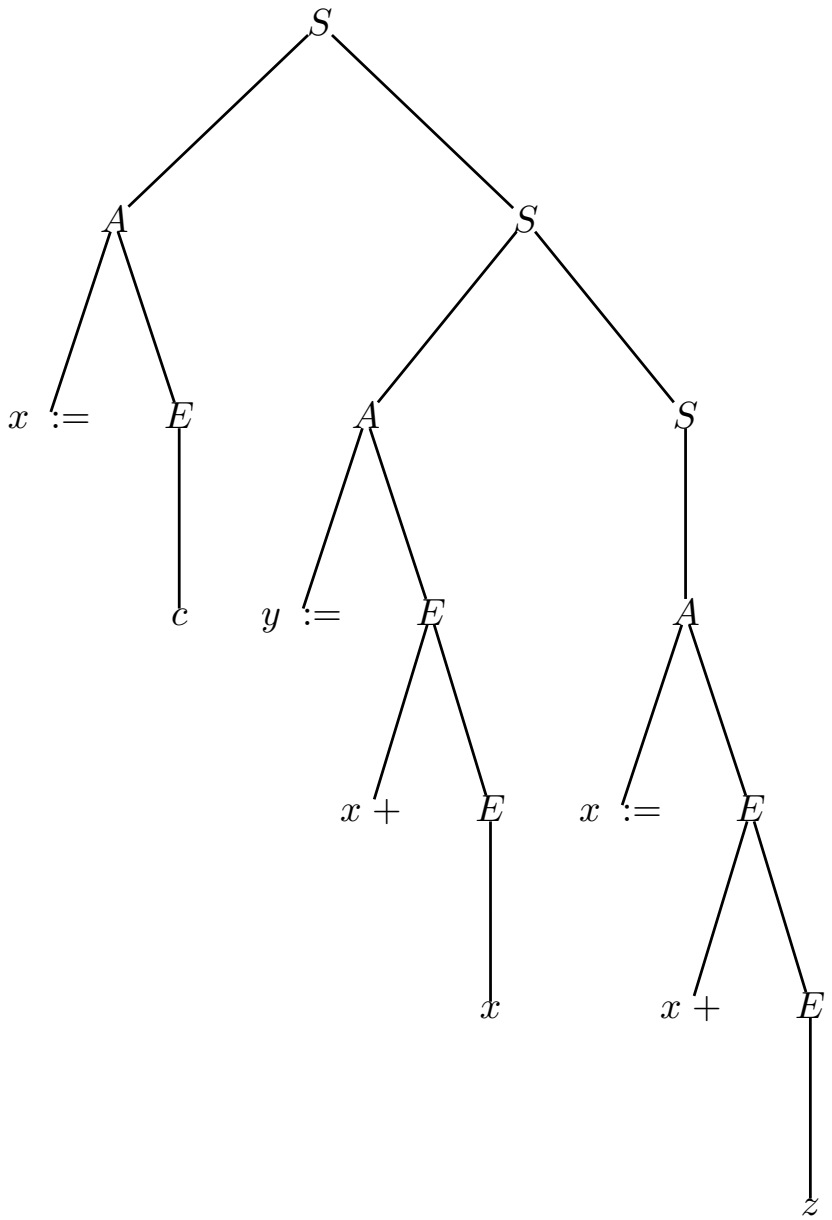
   Si scrivano le funzioni semantiche.

   (b) Disegnare sull'albero i valori degli attributi e le frecce delle dipendenze funzionali tra di essi.

   (c) Scrivere lo pseudocodice del valutatore semantico, supponendo che l'albero sintattico sia stato già costruito dal parsificatore.

   (d) Facoltativo: estendere i controlli semantici in modo da verificare se un assegnamento, diverso dall'ultimo, sia inutile, ossia se il valore della variabile da esso calcolata non sia utilizzato in un'espressione successiva.

Sintassi	Funzioni semantiche
$S \to AS$	
$S \to A$	
$A \to i := E$	
$E \to i + E$	
$E \to i$	
$E \to c$	



Il primo assegnamento è corretto, il secondo è corretto (ma è inutile), il terzo è scorretto poiché la variabile z non ha un valore.

Soluzione

Altro attributo sintetizzato: $lp\ of\ A$, l'identificatore della parte sin. dell'assegnamento (left part of A). Inoltre si estendono l'attributo sintetizzato α anche al nonterminale A: $\alpha\ of\ A$; e l'attributo ereditato def anche al nonterminale S: $def\ of\ S$. Per comodità sotto si numerano tutti i nonterminali.

Sintassi	Funzioni semantiche
$Assioma \rightarrow S_1$	$def\ of\ S_1 := \emptyset$
$S_0 \rightarrow A_1 S_2$	$\alpha\ of\ S_0 := (\alpha\ of\ A_1\ \wedge\ \alpha\ of\ S_2)$ $def\ of\ S_2 := (def\ of\ S_0\ \cup\ lp\ of\ A_1)$ $def\ of\ A_1 := def\ of\ S_0$
$S_0 \rightarrow A_1$	$\alpha\ of\ S_0 := \alpha\ of\ A_1$ $def\ of\ A_1 := def\ of\ S_0$
$A_0 \rightarrow i := E_1$	$\alpha\ of\ A_0 := (use\ of\ E_1 \subseteq def\ of\ A_0)$ $lp\ of\ A_0 :=$ if $(\alpha\ of\ A_0 =$ true$)$ then $\{i\}$ else $\emptyset$
$E_0 \rightarrow i + E_1$	$use\ of\ E_0 := use\ of\ E_1 \cup \{i\}$
$E_0 \rightarrow i$	$use\ of\ E_0 := \{i\}$
$E_0 \rightarrow c$	$use\ of\ E_0 := \emptyset$

Per quanto riguarda l'albero, applicare gli attributi in questione e tirare le frecce: si lascia l'esercizio al lettore.

Per quanto riguarda lo pseudocodice, è un esercizio abbastanza ovvio.

Per quanto riguarda l'estensione dei controlli semantici, basta raccogliere tutti gli usi di variabili (occorre un nuovo attributo, sintetizzato) e introdurre la verifica che alla fine della lista il $def\ of\ S$ sia contenuto nell'insieme di usi raccolto (poi la verifica deve risalire alla radice dell'albero nel predicato di correttezza). È possibile vi siano altre soluzioni.