Linguaggi Formali e Compilatori
(Formal Languages and Compilers)

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

Prova scritta - 30 settembre 2011 - 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:

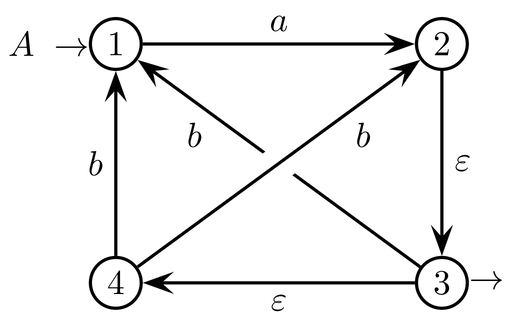
MATRICOLA: FIRMA:

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. È dato l'automa finito A seguente:



Si risponda alle domande seguenti:

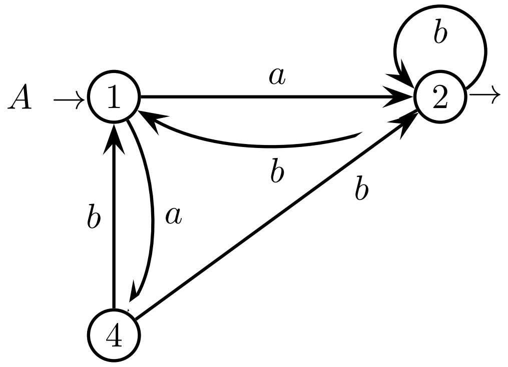
(a) Si disegni un automa finito A' equivalente all'automa A, non importa se deterministico o no, senza transizioni spontanee (ε-transizioni).

(b) Se necessario, si disegni un automa finito deterministico A'' equivalente all'automa A'.

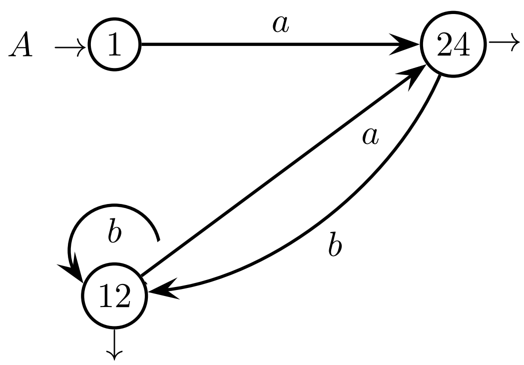
(c) (facoltativa) Si verifichi se l'automa finito deterministico A'' è minimo.

Soluzione

(a) Si disegni un automa finito A' equivalente all'automa A, non importa se deterministico o no, senza transizioni spontanee (ε-transizioni).



(b) Mediante l'usuale algoritmo di determinizzazione si ottiene il seguente.



(c) L'automa è minimo, in quanto lo stato 1 è distinguibile dagli altri due perchè non finale, e lo stato 24 è distinguibile dallo stato 12 perchè non accetta in ingresso il simbolo a.

2. Il linguaggio L di alfabeto $\{p = \text{ producer}, c = \text{ consumer}\}$ rappresenta le sequenze (non vuote) di azioni di produzione/consumo di un elemento in un buffer inizialmente vuoto che può contenere fino a 2 elementi. Es.: $p, ppc, pcpcpc, ppccp, \ldots$. Controesempi: $c, pcc, pppccc, \ldots$.

   Si risponda alle domande seguenti:

   (a) Si elenchino le prime cinque frasi di L in ordine lessicografico e si scriva l'espressione regolare del linguaggio L, usando soltanto gli operatori $\cup, *, +, \cdot$ e cercando di scrivere l'espressione in modo chiaro e semplice.

   (b) (facoltativa) Si verifichi che l'espressione regolare non sia ambigua.

Soluzione

(a) Si elenchino le prime cinque frasi di L in ordine lessicografico e si scriva l'espressione regolare del linguaggio L, usando soltanto gli operatori $\cup, *, +, \cdot$ e cercando di scrivere l'espressione in modo chiaro e semplice.

Le prime frasi sono: p, pc, pp, pcp, ppc. Sia B l'insieme delle sequenze che partendo da buffer vuoto lo lasciano vuoto; siano B_1 (risp. B_2) l'insieme delle sequenze che, partendo da buffer vuoto e mai svuotandolo completamente, lasciano un elemento (risp. due elementi) nel buffer. Le espressioni dei tre linguaggi sono semplici:

$$B = \left(p\,(pc)^*\,c\right)^+ \tag{1}$$

$$B_1 = p(pc)^* \tag{2}$$

$$B_2 = p(pc)^*p \tag{3}$$

L'espressione reg. richiesta è:

$$e = B \cdot (\varepsilon \cup B_1 \cup B_2)$$

(b) (facoltativa) Si verifichi che l'espressione regolare non sia ambigua.

First, we observe that regular expressions, B, B_1, B_2 are unambiguous. Then we observe that the languages $\{\varepsilon\}$, B_1, and B_2 are disjoint. Moreover there exists no prefix of B_1 or B_2 that, if appended to the end of a string of B, produces a string of B. Therefore any sentence $x \in L$ can be segmented in a unique way as

$$x = x_0 \quad \text{or} \tag{4}$$

$$x = x_0 \cdot x_1 \quad \text{or} \tag{5}$$

$$x = x_0 \cdot x_2 \quad \text{or} \tag{6}$$

$$\tag{7}$$

where $x_0 \in B, x_1 \in B_1, x_2 \in B_2$. Hence expression e is unambiguous.

2 Grammatiche libere e automi a pila 20%

1. Si consideri il linguaggio delle stringhe, di lunghezza arbitraria, composte da parentesi tonde aperte e chiuse, tali che, per ogni prefisso, il numero delle parentesi aperte è maggiore o uguale al numero delle parentesi chiuse. Esempi di stringhe appartenenti al linguaggio:

   $$\varepsilon, \qquad (()((, \qquad (()()(, \qquad (())()$$

   Esempi di stringhe non appartenenti al linguaggio

   $$()), \qquad)), \qquad ()()))(($$

   Si risponda alle domande seguenti:

   (a) Scrivere una semplice grammatica ambigua per il linguaggio, fornendo anche come esempio una stringa e due suoi alberi sintattici.

   (b) Scrivere una grammatica non ambigua del linguaggio, dando una spiegazione di tale non ambiguità, e mostrando l'unico albero sintattico della stringa fornita al punto precedente.

Soluzione

(a) Scrivere una semplice grammatica ambigua per il linguaggio, fornendo anche come esempio una stringa e due suoi alberi sintattici.

1. Il linguaggio può essere visto come una variante di quello di Dyck, con parentesi aperte aggiunte in qualsiasi posizione; o, anche, come l'insieme delle stringhe che sono prefissi delle frasi del linguaggio di Dyck E' quindi facile ottenere la grammatica ambigua aggiungendo una singola alternativa alla grammatica del linguaggio di Dyck

$$S \to (S)S \mid \varepsilon \mid (S$$

Come esempio di stringa consideriamo (() e i due alberi

$$\overbrace{(\quad (\quad \overbrace{\varepsilon}^{S} \quad) \quad \overbrace{\varepsilon}^{S}}^{S}$$

$$\overbrace{(\quad \overbrace{(\quad \varepsilon}^{S} \quad) \quad \overbrace{\varepsilon}^{S}}^{S}$$

(b) 1. Una grammatica non ambigua si ottiene osservando che il linguaggio è il prefisso di quello di Dyck.

Si farà in modo che ogni parentesi sinistra sia bilanciata, quando possibile, dalla più vicina parentesi destra successiva atta a corrisponderle. In questa ipotesi si osserva poi che non ci può essere una parentesi sinistra '(' sbilanciata all'interno di una coppia bilanciata di parentesi corrispondenti $'('\ldots'('\ldots')'$, altrimenti la parentesi sinistra '(' interna potrebbe fare coppia con la parentesi destra ')' rendendo sbilanciata la prima delle due parentesi sinistre. Quindi all'interno di una coppia (...) bilanciata non ci possono essere stringhe sbilanciate, aventi cioè parentesi sinistre in eccesso.

Di conseguenza ogni stringa del linguaggio è formata da una lista di sequenze di parentesi sinistre non bilanciate (ognuna di tali sequenze può eventualmente essere di lunghezza nulla), separate da nidi di parentesi bilanciate. Ciò porta a definire la grammatica seguente (dove il nonterminale A rappresenta una sequenza di parentesi sinistre non bilanciate e B una stringa di parentesi bilanciate, come nel linguaggio di Dyck):

$$\begin{array}{lcl} S & \to & A(B)S \mid A \\ A & \to & (A \mid \varepsilon \\ B & \to & (B)B \mid \varepsilon \end{array}$$

2. Con riferimento al linguaggio del pre-processore C, si considerino i seguenti costrutti.

   Direttive di definizione di variabili, ad es.:

   ```
   #define MAX   100  /* il valore assegnato  è un numero */
   #define STRING_ERR "Rilevato errore!"
   ```

   Direttive di definizione di macro, ad es.:

   ```
   #define CERCHIO1(r)   r*r*3.14  /* il corpo della macro è codice C */
   ```

   Direttive condizionali, ad es.:

   ```
   #ifdef DEBUG
      ..........
   #else ......
   #endif
   ```

   La parte `#else` è facoltativa. Al posto dei puntini può stare una sequenza di pezzi di codice C, o di direttive di definizione di variabile o di macro, o di direttive condizionali.

   Nello scrivere la grammatica si diano per disponibili i nonterminali:

   - $< C\ \mathit{number} >$ che rappresenta un numero;
   - $< C\ \mathit{string} >$ che rappresenta una stringa;
   - $< C\ \mathit{code} >$ che rappresenta un pezzo di codice;
   - $< \mathit{ID} >$ che rappresenta un identificatore.

   Si risponda alle domande seguenti:

   (a) Si definisca mediante una grammatica EBNF non ambigua una lista con i costrutti descritti.

Soluzione

Grammatica EBNF:

$$
\begin{gathered}
S \to (V \mid M \mid C)^{+} \\
V \to \#\mathrm{def} \quad <ID> \ \ (<C\ number> \mid <C\ string>) \\
M \to \#\mathrm{def} \quad <ID> \ \ \text{'('}\varepsilon \mid <ID> \ (\ , \ <ID>)^{*} \text{')'} <C\ code> \\
C \to \#\mathrm{ifdef} \quad <ID> \ \ (<C\ code> \mid V \mid M \mid C)^{+} \\
\big[\#\mathrm{else} \ \ (<C\ code> \mid V \mid M \mid C)^{+}\big] \ \ \#\mathrm{endif}
\end{gathered}
$$

3 Analisi sintattica e parsificatori 20%

1. Si considerino i linguaggi

$$
\begin{aligned}
A &= \{a^n b^n \mid n \geq 1\} \\
B &= (ab)^{*} \\
C &= A \cdot B \\
D &= B \cdot A
\end{aligned}
$$

Si risponda alle domande seguenti:

(a) Si scrivano in pseudocodice le procedure dell'analizzatore a discesa ricorsiva $LL(1)$ per i linguaggi A, B e C.

(b) (facoltativa) Si scriva la procedura a discesa ricorsiva per il linguaggio D.

Soluzione

(a) Si scrivano in pseudocodice le procedure dell'analizzatore a discesa ricorsiva $LL(1)$ per i linguaggi A, B e C.

Data la semplicità dei linguaggi, invece di formalizzarli mediante reti di macchine, passiamo alla costruzione diretta delle procedure.

La procedura per A legge a quindi, se il prossimo carattere è a si autoinvoca, e infine legge b.

La procedura per B è un ciclo con la condizione finale che il carattere corrente CT sia diverso da a.

La procedura per C può essere scritta combinando le due procedure precedenti.

(b) (facoltativa) Si scriva la procedura a discesa ricorsiva per il linguaggio D.

Questo caso differisce dal caso C nel richiedere una lunghezza di prospezione maggiore. Si osservi che la composizione $D \to B \cdot A$ porta ad una grammatica che non è $LL(1)$, come appare dai seguenti casi:

$$\overbrace{\overbrace{a\ b}^{B}\ \overbrace{a\ b}^{A}}^{D} \dashv \qquad \overbrace{\overbrace{a\ b\ a\ b}^{B}\ \overbrace{a\ b}^{A}}^{D} \dashv \qquad \overbrace{\overbrace{\varepsilon}^{B}\ \ \overbrace{a\ b}^{A}}^{D} \dashv$$

$$\overbrace{\overbrace{a\ b}^{B}\ \overbrace{a\ a \ldots}^{A}}^{D} \dashv \qquad \overbrace{\overbrace{a\ b\ a\ b}^{B}\overbrace{a\ a \ldots}^{A}}^{D} \dashv \qquad \overbrace{\overbrace{\varepsilon}^{B}\ \ \overbrace{a\ a \ldots}^{A}}^{D} \dashv$$

Al fine di scegliere se invocare la procedura di A piuttosto che quella di B, si devono esaminare più caratteri: $ab \dashv$, aa. Non è difficile scrivere la procedura che effettua il test sul secondo e terzo carattere oltre quello corrente.

2. La seguente grammatica non è LR(1).

$$
\begin{array}{lcl}
S & \to & CcdS \\
S & \to & DceS \\
S & \to & \varepsilon \\
C & \to & abC \\
C & \to & ab \\
D & \to & abD \\
D & \to & ab
\end{array}
$$

Si risponda alle domande seguenti:

(a) Mostrare, disegnando il minor numero possibile di stati dell'automa pilota, quale tipo di conflitto è presente. (NB: ogni stato deve tuttavia essere completato in ogni sua parte)

(b) Dire se la grammatica è $LR(k)$ per qualche $k > 1$.

(c) (facoltativa) Modificare, nella misura minima possibile, la grammatica in modo tale da renderla $LR(1)$ e mostrare come per la nuova grammatica, negli stati corrispondenti a quelli individuati al punto 1, non è presente alcun conflitto.

Soluzione

(a) Mostrare, disegnando il minor numero possibile di stati dell'automa pilota, quale tipo di conflitto è presente. (NB: ogni stato deve tuttavia essere completato in ogni sua parte)

Dallo stato iniziale, leggendo a e poi b, si va in uno stato che presenta un conflitto riduzione-riduzione. Ciò è causato dal fatto che i due n.t. C e D, che generano lo stesso linguaggio $(ab)^+$, sono seguiti, nelle due prime produzioni, dallo stesso carattere c.

(b) Dire se la grammatica è $LR(k)$ per qualche $k > 1$.

Il secondo carattere successivo è diverso per i due n.t. C e D: è d per C e e per D. Quindi la grammatica è $LR(2)$.

(c) (facoltativa) Modificare, nella misura minima possibile, la grammatica in modo tale da renderla $LR(1)$ e mostrare come per la nuova grammatica, negli stati corrispondenti a quelli individuati al punto 1, non è presente alcun conflitto.

Si può rendere la grammatica $LR(1)$ modificando i due nonterminali C e D in modo che incorporino anche il successivo carattere c:

$$\begin{array}{lcl} S & \rightarrow & CdS \\ S & \rightarrow & DeS \\ S & \rightarrow & \varepsilon \\ C & \rightarrow & abC \\ C & \rightarrow & abc \\ D & \rightarrow & abD \\ D & \rightarrow & abc \end{array}$$

Soluzione alternativa. Poiché C e D generano lo stesso linguaggio, uno dei due puo' essere eliminato ottenendo la grammatica $LR(1)$ seguente:

$$\begin{array}{lcl} S & \rightarrow & CcdS \mid CceS \\ S & \rightarrow & \varepsilon \\ C & \rightarrow & abC \\ C & \rightarrow & ab \end{array}$$

4 Traduzione e analisi semantica 20%

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

$$G \begin{cases} S \rightarrow \text{'}f\text{' '(' } A \text{ ',' } A \text{ ')'} \\ S \rightarrow \text{'}g\text{' '(' } A \text{ ',' } A \text{ ')'} \\ A \rightarrow \text{'}a\text{'} \mid S \end{cases}$$

La grammatica G genera un linguaggio di funzioni composte f e g, entrambe a due argomenti. L'argomento è una funzione oppure la lettera a che rappresenta una variabile. Esempio:

$$f\left(a,\ g\left(f\ (a,\ a\ ),\ a\right)\right)$$

Si risponda alle domande seguenti:

(a) Modificando se opportuno la grammatica G, si scriva una grammatica di traduzione G' (o uno schema sintattico di traduzione) che traduce le stringhe del linguaggio di G, mettendole in forma infissa per la funzione f e postfissa per la funzione g; le virgole vengono eliminate, mentre le parentesi restano solo per f. Esempio:

$$f_1\left(a_2,\ g_3\left(f_4\ (a_5,\ a_6\ ),\ a_7\right)\right) \Rightarrow \left(a_2\ f_1\ \left(a_5\ f_4\ a_6\right)\ a_7\ g_3\right)$$

N.B.: i pedici sono indicati per chiarezza, ma non fanno parte del linguaggio.

(b) (facoltativa) Si supponga ora che la funzione f possa avere un numero arbitrario ≥ 1 di argomenti, e si modifichi la grammatica di traduzione G' mettendo le regole di f in forma EBNF, traducendo f in forma postfissa (e non più infissa) e conservando per f le parentesi. Esempio:

$$f_1\left(a_2,\ g_3\left(f_4\ (a_5,\ a_6\ ),\ a_7,\ a_8\right)\right) \Rightarrow \left(a_2\ \left(a_5\ a_6\ f_4\right)\ a_7\ g_3\ a_8\ f_1\right)$$

Soluzione

(a)
$$S \to \frac{f}{\varepsilon}\frac{\text{`('}}{\text{`('}}A\frac{,}{f}A\frac{\text{`)'}}{\text{`)'}}$$

$$S \to \frac{g}{\varepsilon}\frac{\text{`('}}{\varepsilon}A\frac{,}{\varepsilon}A\frac{\varepsilon}{g}\frac{\text{`)'}}{\varepsilon}$$

(b)
$$S \to \frac{f}{\varepsilon}\frac{\text{`('}}{\text{`('}}A\left(\frac{,}{\varepsilon}A\right)^{*}\frac{\varepsilon}{f}\frac{\text{`)'}}{\text{`)'}}$$

2. Si richiede l'analisi statica del seguente programma.

```
1:  c=a*b
2:  c=c+2
3:  d=d+3
4:  read(a)
5:  if (a*d > 10)
6:  then  b=2*b
7:  else  a=2*a
8:  if a==0 then goto 3
9   print(c)
```

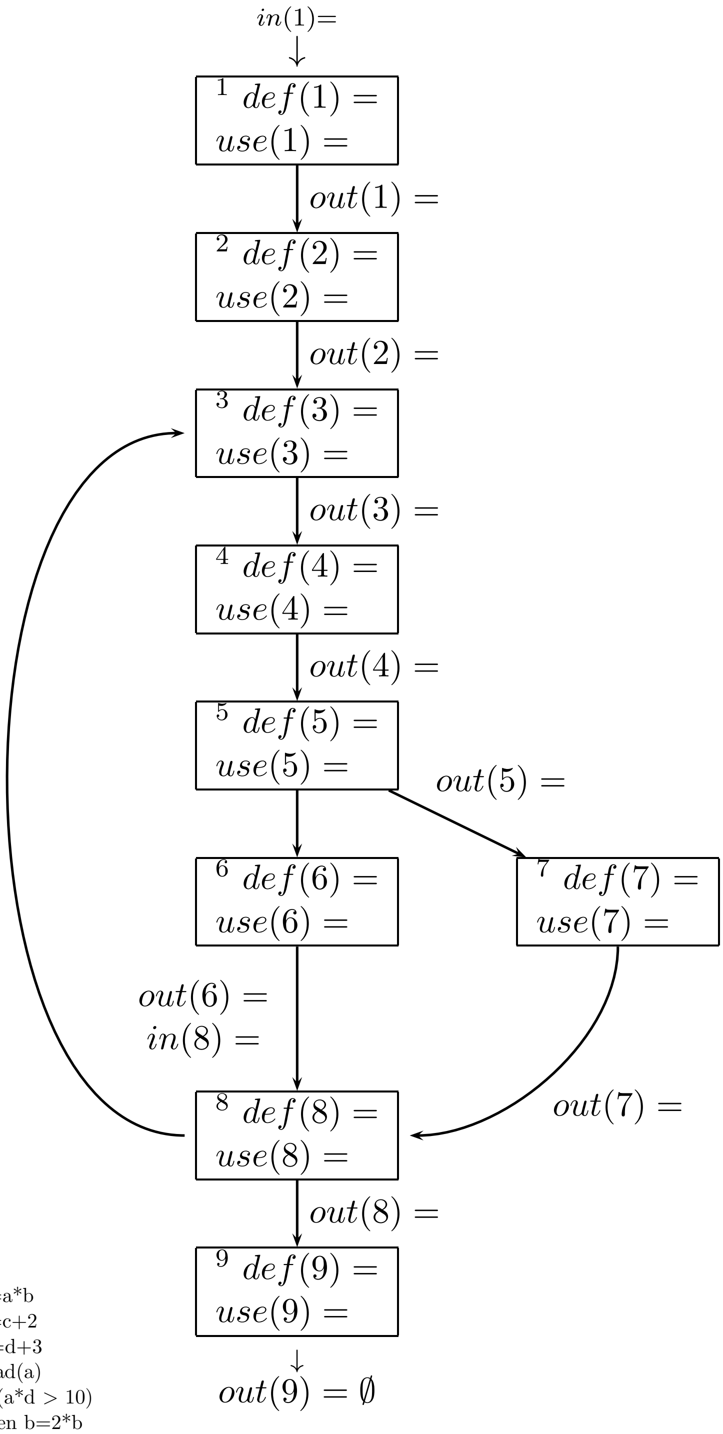
Si supponga che le variabili prive di valore iniziale siano passate come parametri d'ingresso.

Si risponda alle domande seguenti:

(a) Si completi il grafo di flusso predisposto alla pagina seguente inserendo gli elementi degl'insiemi def e use.

- Si scrivano poi le equazioni di flusso relative al nodo 8 per calcolare gli insiemi $live_{in}$ e $live_{out}$;
- Si indichi in quali punti del programma è viva la variabile c.

(b) (facoltativa) Si calcolino i valori degli insiemi $live_{in}$ e $live_{out}$ nei nodi del grafo. Si indichi quante celle di memoria sono necessarie per le variabili a, b, c, d. Si vuole minimizzare il numero delle celle di memoria usate dal programma. A tale fine si trasformi il programma riordinando le istruzioni ma senza modificare la semantica dello stesso.

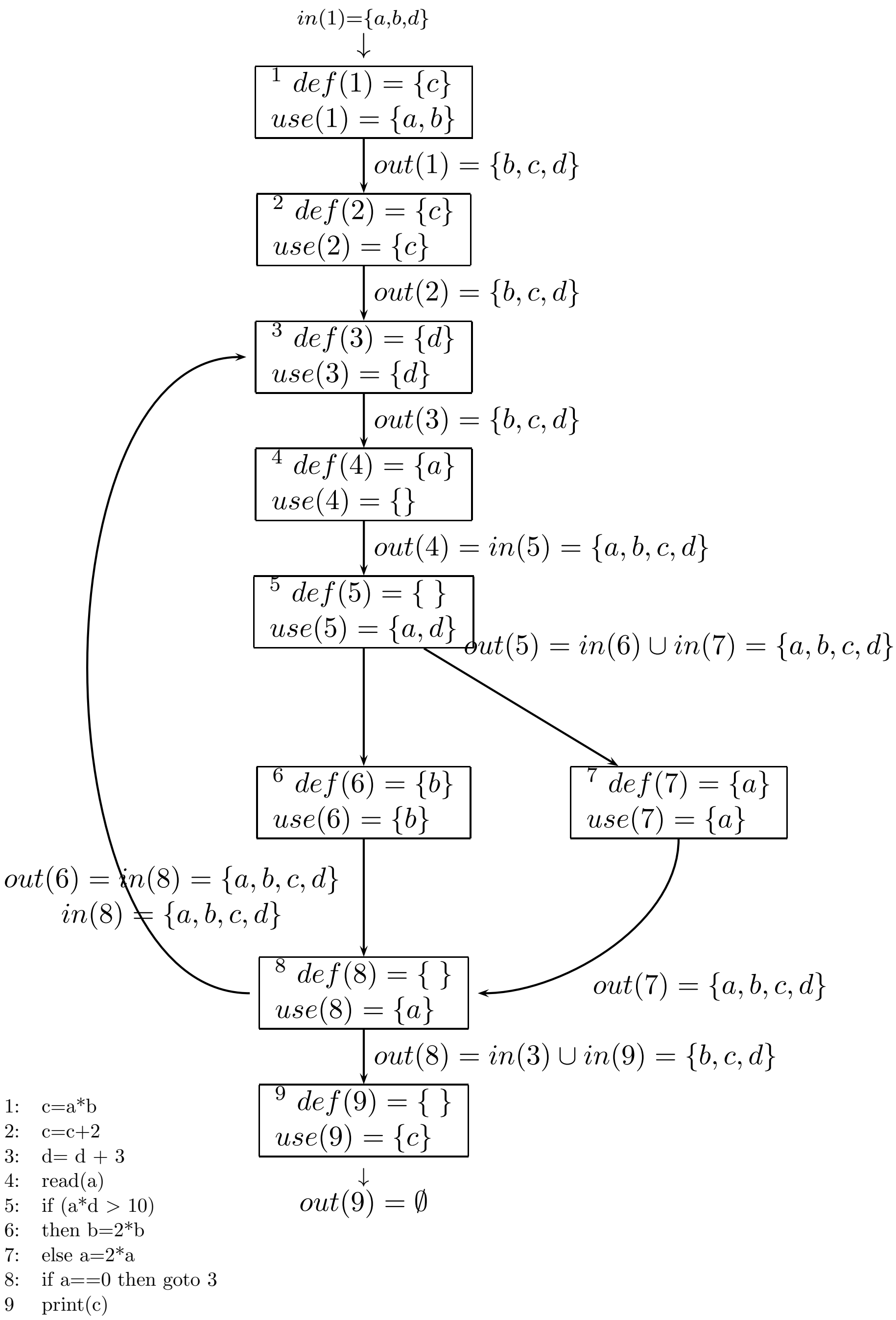



```
1:  c=a*b
2:  c=c+2
3:  d=d+3
4:  read(a)
5:  if (a*d > 10)
6:  then b=2*b
7:  else a=2*a
8:  if a==0 then goto 3
9   print(c)
```

Soluzione

(a) Equazioni per il nodo 8:

$$in(8) = use(8) \cup (out(8) \setminus def(8)) = \{a\} \cup out(8)$$
$$out(8) = in(3) \cup in(9)$$



(b) Poiché le variabili a, b, c, d sono contemporaneamente vive (ad es. all'uscita dell'istruzione 5), sono necessarie quattro celle di memoria per contenere i loro valori.

Si osserva però che l'intervallo di vita di c, da $out(1)$ fino a $in(9)$, è inutilmente esteso: infatti l'unico uso di c, dopo la sua definizione in 2, è la stampa in 9, operazione che può essere anticipata a subito dopo 2 senza alterare i risultati prodotti dal programma. Così facendo c scompare dagli insiemi *in* e *out* dei nodi $3, 4, \ldots, 9$, il che permette di usare tre celle invece di quattro: a e c possono infatti stare nella stessa cella perché i loro valori non sono vivi contemporaneamente (si dice che non interferiscono).