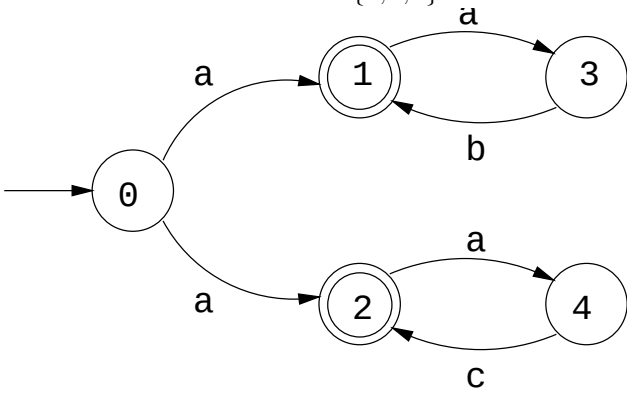
PROGRAMMAZIONE – I Appello del 3/12/2002

SOLUZIONE

ESERCIZIO 1 (6 punti)

Dato il seguente automa sull'alfabeto $\Lambda = \{a, b, c\}$

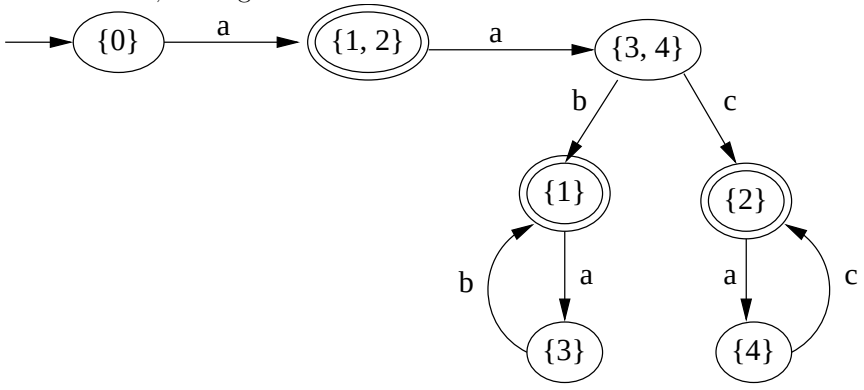


(a) Descrivere formalmente il linguaggio riconosciuto dall'automa.

(b) Trasformare l'automa dato in un automa equivalente deterministico usando l'algoritmo della costruzione dei sottoinsiemi. Spiegare almeno un passo dell'algoritmo.

Soluzione

Il linguaggio accettato è l'insieme $\{a(ab)^n \mid n \geq 0\} \cup \{a(ac)^n \mid n \geq 0\}$.
L'automa deterministico equivalente, costruito con l'algoritmo della costruzione dei sottoinsiemi, è il seguente:



Lo stato iniziale è, per definizione, $\{0\}$. Per costruire le transizioni uscenti da questo stato si calcola $T^a_{\{0\}} = \{1, 2\}$ cioè l'insieme degli stati che si raggiungono, nell'automa di partenza, a partire da uno stato qualsiasi dell'insieme $\{0\}$ e leggendo il simbolo a. Si costruisce allora una transizione dallo stato $\{0\}$ allo

stato $\{1, 2\}$ dell'automa risultato, etichettata con il simbolo a. Si calcola poi lo stesso insieme per gli altri simboli dell'alfabeto Λ. In questo caso $T^b_{\{0\}} = T^c_{\{0\}} = \{\}$. Quindi non ci sono transizioni uscenti dallo stato $\{0\}$ etichettate con b o con c, dato che l'automa di partenza, nello stato 0, non ha transizioni uscenti etichettate con questi simboli.

ESERCIZIO 2 (6 punti)

Siano date le seguenti categorie sintattiche con relative produzioni:

```
<Cifra> ::= 0|1|...|9
<Num> ::= <Cifra> | <Num> <Cifra>
<Alfa> ::= a|b|c|..|z|A|B|C|...|Z
<Ide> ::= <Alfa> | <Alfa> <Idesub>
<Idesub> ::= <Alfa> <Idesub> | <Cifra> <Idesub> | <Alfa> | <Cifra>
```

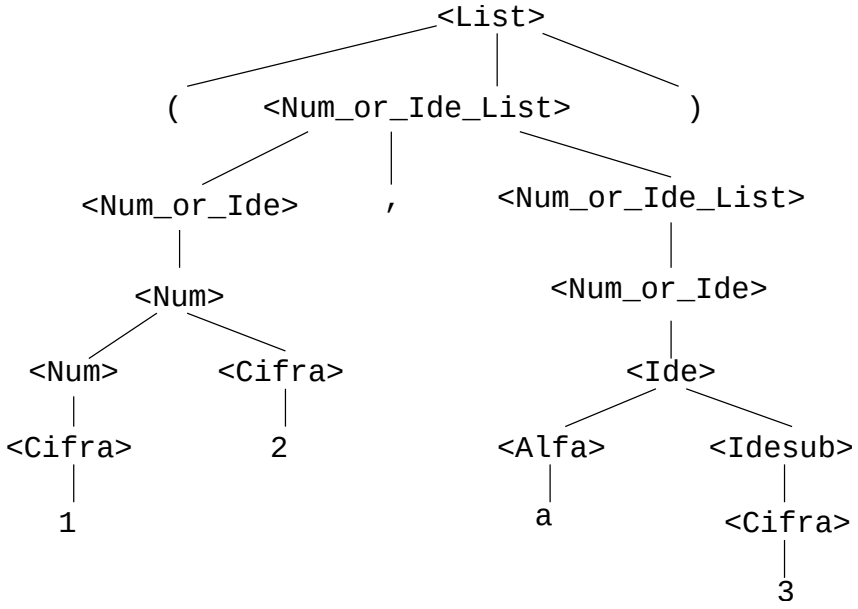
(a) Usando le categorie sintattiche date come sottocategorie, dare le produzioni di una categoria sintattica `<List>` (ed eventuali sottocategorie di comodo) che genera le liste di numeri (elementi della categoria sintattica `<Num>`) e/o di identificatori (elementi della categoria sintattica `<Ide>`). Ogni lista si apre con una parentesi tonda aperta, gli elementi sono separati da virgole, può contenere un numero qualsiasi di elementi (anche zero) e si chiude con una parentesi tonda chiusa. La lista vuota è la stringa "`()`". Altri esempi sono "`(1)`", "`(23, a)`" e "`(12, a34, pippo, 0)`".

(b) Disegnare l'albero di derivazione della lista "`(12, a3)`" usando la grammatica che si è data al punto (a).

Una possibile soluzione

```
<List> ::= () | ( <Num_or_Ide_List> )
<Num_or_Ide_List> ::= <Num_or_Ide> | <Num_or_Ide> , <Num_or_Ide_List>
<Num_or_Ide> ::= Num | Ide
```

L'albero di derivazione è il seguente.



ESERCIZIO 3 (6 punti)

Si supponga di estendere la categoria sintattica dei comandi con la seguente produzione

```
<Com> ::= caserepeat (<Exp>, <Com>, <Com>) until <Exp>;
```

Un comando `caserepeat (E1, C1, C2) until E` è un ciclo in cui il corpo viene eseguito in ogni caso alla prima iterazione. Se la guardia E ha valore **falso** nello stato risultante dall'esecuzione del corpo allora il ciclo viene ripetuto, altrimenti l'esecuzione del comando termina. L'esecuzione del corpo `(E1, C1, C2)` consiste nell'esecuzione del comando `C1` se l'espressione `E1`, nello stato corrente, ha valore vero o nell'esecuzione del comando `C2` se l'espressione `E1`, nello stato corrente, ha valore falso.
Definire le regole di semantica operazionale per il nuovo comando con riferimento allo stato composto solo da una pila di frame.

Soluzione

$$com_{caserep-tt-tt} \quad \frac{\langle \texttt{E1}, \sigma\rangle \rightarrow_{exp} \mathbf{tt} \quad \langle \texttt{C1}, \sigma\rangle \rightarrow_{com} \sigma' \quad \langle \texttt{E}, \sigma'\rangle \rightarrow_{exp} \mathbf{tt}}{\langle \texttt{caserepeat (E1, C1, C2) until E;}, \sigma\rangle \rightarrow_{com} \sigma'}$$

$$com_{caserep-ff-tt} \quad \frac{\langle \texttt{E1}, \sigma\rangle \rightarrow_{exp} \mathbf{ff} \quad \langle \texttt{C2}, \sigma\rangle \rightarrow_{com} \sigma' \quad \langle \texttt{E}, \sigma'\rangle \rightarrow_{exp} \mathbf{tt}}{\langle \texttt{caserepeat (E1, C1, C2) until E;}, \sigma\rangle \rightarrow_{com} \sigma'}$$

$$com_{caserep-tt-ff} \quad \frac{\langle \texttt{E1}, \sigma\rangle \rightarrow_{exp} \mathbf{tt} \quad \langle \texttt{C1}, \sigma\rangle \rightarrow_{com} \sigma' \quad \langle \texttt{E}, \sigma'\rangle \rightarrow_{exp} \mathbf{ff} \quad \langle \texttt{caserepeat (E1, C1, C2) until E;}, \sigma'\rangle \rightarrow_{com} \sigma''}{\langle \texttt{caserepeat (E1, C1, C2) until E;}, \sigma\rangle \rightarrow_{com} \sigma''}$$

$$com_{caserep-ff-ff} \quad \frac{\langle \texttt{E1}, \sigma\rangle \rightarrow_{exp} \mathbf{ff} \quad \langle \texttt{C2}, \sigma\rangle \rightarrow_{com} \sigma' \quad \langle \texttt{E}, \sigma'\rangle \rightarrow_{exp} \mathbf{ff} \quad \langle \texttt{caserepeat (E1, C1, C2) until E;}, \sigma'\rangle \rightarrow_{com} \sigma''}{\langle \texttt{caserepeat (E1, C1, C2) until E;}, \sigma\rangle \rightarrow_{com} \sigma''}$$

ESERCIZIO 4 (6 punti)

Completare la definizione del seguente metodo.

```
public int metodo(int[] a)
/** Dimezza tutti gli elementi pari e raddoppia tutti gli elementi
 *  dispari dell'array puntato da a. Restituisce il numero degli
 *  elementi dimezzati.
 */
```

Una possibile soluzione

```
public int metodo(int[] a) {
int dim = 0;
for (int i = 0; i < a.length; i++)
  if (a[i] % 2 == 0) {
     a[i] = a[i] / 2;
     dim = dim + 1;
  }
  else
    a[i] = a[i] * 2;
return dim;
}
```

ESERCIZIO 5 (6 punti)

Si consideri il seguente programma.

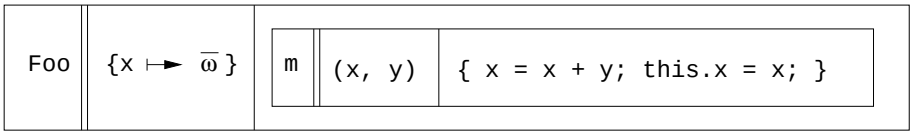
```
prog { class Foo {
         public int x;
         public void m(int x, int y) {
           x = x + y;
           this.x = x;
         }
       } (1)
       {
       Foo o1 = new Foo;
       o1.x = 10;
       Foo o2 = new Foo;
       o2.x = 20; (2)
       o2.m(o1.x, o2.x); (3)
       }
     }
```

(a) Disegnare l'ambiente delle classi al punto `(1)` del programma.

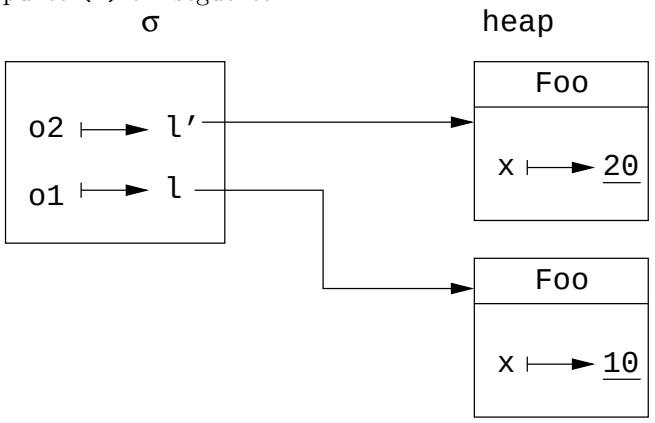
(b) Disegnare la pila di frame e l'heap ai punti `(2)` e `(3)` del programma.

Soluzione

L'ambiente delle classi ρ_c al punto `(1)` è il seguente.



Lo stato al punto (2) è il seguente.



Lo stato al punto (3) è il seguente.