PROGRAMMAZIONE – III Appello del 7/03/2003

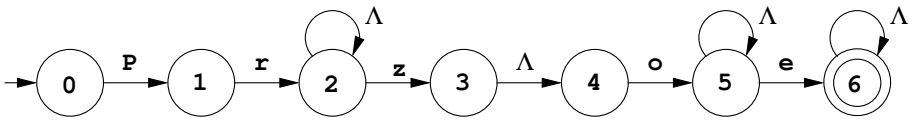
Scrivere **in stampatello** COGNOME, NOME e NUMERO DI MATRICOLA (se conosciuto) su ogni foglio consegnato e sul testo, che va consegnato insieme al compito.

ESERCIZIO 1 (6 punti)

Si consideri l'alfabeto Λ come l'insieme dei simboli ASCII (lettere, cifre, segni di punteggiatura etc.). Le stringhe con caratteri jolly sono rappresentazioni sintetiche di stringhe di Λ^*. I caratteri jolly sono `*` e `?`. Il carattere jolly `*` rappresenta zero o più caratteri di Λ, mentre il carattere jolly `?` rappresenta esattamente *un* carattere di Λ. Una stringa che contiene dei caratteri jolly rappresenta un insieme di stringhe di Λ^*. Ad esempio la stringa con caratteri jolly `a*` rappresenta tutte le stringhe che cominciano con un carattere `a` (a posto di `*` posso sostituire una qualsiasi stringa, anche la stringa vuota ϵ). La stringa con caratteri jolly `1??` rappresenta tutte le stringhe lunghe 3 caratteri e che cominciano con il carattere `1` (a posto di ogni `?` devo mettere un carattere). `*` e `?` possono essere anche combinati. Ad esempio la stringa con caratteri jolly `esa*?*tex` rappresenta tutte le stringhe che cominciano con `esa`, che finiscono con `tex` e che contengono almeno un altro carattere (devo metterne uno a posto del `?` e a posto degli `*` posso mettere le stringhe che voglio, anche la stringa vuota ϵ in tutti e due i casi. Alcune stringhe dell'insieme rappresentato da `esa*?*tex` sono `esa1tex`, `esame.tex` o `esa1234567890tex`.

- Disegnare un automa non deterministico che accetta tutte e sole le stringhe rappresentate dalla stringa con caratteri jolly `Pr*z?o*e*`
- Mostrare un cammino di accettazione e uno di non accettazione per la stringa `Programmazione`

SOLUZIONE



Un cammino di accettazione è il seguente:

$$0 \xrightarrow{\texttt{P}} 1 \xrightarrow{\texttt{r}} 2 \xrightarrow{\texttt{o}} 2 \xrightarrow{\texttt{g}} 2 \xrightarrow{\texttt{r}} 2 \xrightarrow{\texttt{a}} 2 \xrightarrow{\texttt{m}} 2 \xrightarrow{\texttt{m}} \xrightarrow{\texttt{a}} 2 \xrightarrow{\texttt{z}} 3 \xrightarrow{\texttt{i}} 4 \xrightarrow{\texttt{o}} 5 \xrightarrow{\texttt{n}} 5 \xrightarrow{\texttt{e}} 6$$

Un cammino di non accettazione è il seguente:

$$0 \xrightarrow{\texttt{P}} 1 \xrightarrow{\texttt{r}} 2 \xrightarrow{\texttt{o}} 2 \xrightarrow{\texttt{g}} 2 \xrightarrow{\texttt{r}} 2 \xrightarrow{\texttt{a}} 2 \xrightarrow{\texttt{m}} 2 \xrightarrow{\texttt{m}} \xrightarrow{\texttt{a}} 2 \xrightarrow{\texttt{z}} 2 \xrightarrow{\texttt{i}} 2 \xrightarrow{\texttt{o}} 2 \xrightarrow{\texttt{n}} 2 \xrightarrow{\texttt{e}} 2$$

ESERCIZIO 2 (6 punti)

Si consideri l'alfabeto $\Lambda = \{a, b, c, @\}$. Definire una grammatica libera dal contesto che generi tutte e sole le stringhe del seguente insieme:

$$L = \{a^m \ @ \ b^n \ @ \ c^n \ b^m \mid m \geq 1, n \geq 0\}$$

SOLUZIONE

```
<S> ::= a <S> b | a @ <T> b
<T> ::= b <T> c | @
```

ESERCIZIO 3 (6 punti)

Date due `Slist`, `S1` e `S2`, esse si dicono equivalenti se e solo se, dato uno stato σ qualsiasi, si verifica una ed una sola delle seguenti condizioni:

- $\langle \texttt{S1}, \sigma\rangle \not\rightarrow_{slist}$ **e** $\langle \texttt{S2}, \sigma\rangle \not\rightarrow_{slist}$
- $\langle \texttt{S1}, \sigma\rangle \rightarrow_{slist} \sigma'$, $\langle \texttt{S2}, \sigma\rangle \rightarrow_{slist} \sigma''$ **e** $\sigma' = \sigma''$.

Dimostrare, usando le regole di semantica operazionale e le derivazioni, che le seguenti `Slist` sono equivalenti, a partire da uno stato $\sigma \neq \Omega$:

```
S1: int x = 1;                                  S2: int x = 0;
    while (x > 0) x = x - 1;
```

SOLUZIONE

L'ipotesi $\sigma \neq \Omega$ è equivalente a considerare uno stato di partenza della forma $\phi.\sigma$, dove ϕ è un frame qualunque e σ una pila di frame qualunque (anche vuota).
Dato che le due `Slist` utilizzano solo la variabile `x`, che dichiarano, sicuramente le regole funzioneranno senza bloccarsi (la definizione di modifica di frame funziona anche se si ridefinisce una variabile `x` che era già definita nello stesso frame). Per quanto riguarda il `while` di `S1`, possiamo affermare (si veda la derivazione) che termina sempre, a partire da ogni stato della forma $\phi.\sigma$. Quindi la prima condizione dell'equivalenza non si verificherà mai. Pertanto dobbiamo dimostrare che in tutti i casi le due `Slist` portano allo stesso stato finale, partendo dallo stato generico $\phi.\sigma$. Vediamo cosa succede per `S1`:

$\langle \texttt{int x} = 1; \ \texttt{while} \ (\texttt{x} > 0) \ \texttt{x} = \texttt{x} - 1;, \phi.\sigma\rangle$
$\quad \rightarrow_{slist} \quad \{ \quad (slist_{list}) :$
$\quad\quad (d1) : \ \langle \texttt{int x} = 1; \ , \phi.\sigma\rangle \rightarrow_{slist} \phi[\texttt{1}/\texttt{x}].\sigma$
$\quad\quad (d2) : \ \langle \texttt{while} \ (\texttt{x} > 0) \ \texttt{x} = \texttt{x} - 1;, \phi[\texttt{1}/\texttt{x}].\sigma\rangle \rightarrow_{slist} \phi[\texttt{1}/\texttt{x}][\texttt{0}/\texttt{x}].\sigma \quad \}$
$\phi[\texttt{1}/\texttt{x}][\texttt{0}/\texttt{x}].\sigma$

$(d1)$:
$\langle \texttt{int x} = 1; , \phi.\sigma \rangle$
$\quad \rightarrow_{slist} \quad \{ \quad (slist_{dec}) :$
$\texttt{int x} = 1; \in \texttt{Decl},$
$(dec_{var-2}) :$
$\mathcal{E}[\![1]\!]_{\phi.\sigma} = \mathbf{1},$
$\langle \texttt{int x} = 1; , \phi.\sigma \rangle \rightarrow_{dec} \phi[^{\mathbf{1}}/\texttt{x}].\sigma \ \}$
$\phi[^{\mathbf{1}}/\texttt{x}].\sigma$

$(d2)$:
$\langle \texttt{while (x} > 0) \texttt{ x} = \texttt{x} - 1;, \phi[^{\mathbf{1}}/\texttt{x}].\sigma \rangle$
$\quad \rightarrow_{slist} \quad \{ \quad (slist_{com}) :$
$\texttt{while (x} > 0) \texttt{ x} = \texttt{x} - 1; \in \texttt{Com},$
$(d3) : \ \langle \texttt{while (x} > 0) \texttt{ x} = \texttt{x} - 1;, \phi[^{\mathbf{1}}/\texttt{x}].\sigma \rangle \rightarrow_{com} \phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma \ \}$
$\phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma$

$(d3)$:
$\langle \texttt{while (x} > 0) \texttt{ x} = \texttt{x} - 1;, \phi[^{\mathbf{1}}/\texttt{x}].\sigma \rangle$
$\quad \rightarrow_{com} \quad \{ \quad (com_{while-tt}) :$
$\mathcal{E}[\![\texttt{x} > 0]\!]_{\phi[^{\mathbf{1}}/\texttt{x}].\sigma} = \mathbf{tt},$
$(com_{=}) :$
$\mathcal{E}[\![\texttt{x} - 1]\!]_{\phi[^{\mathbf{1}}/\texttt{x}].\sigma} = \mathbf{0},$
$\langle \texttt{x} = \texttt{x} - 1;, \phi[^{\mathbf{1}}/\texttt{x}].\sigma \rangle \rightarrow_{com} \phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma,$
$(com_{while-ff}) :$
$\mathcal{E}[\![\texttt{x} > 0]\!]_{\phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma} = \mathbf{ff},$
$\langle \texttt{while (x} > 0) \texttt{ x} = \texttt{x} - 1;, \phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma \rangle \rightarrow_{com} \phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma \ \}$
$\phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma$

Per `S2` abbiamo la seguente derivazione:

$\langle \texttt{int x} = 0;, \phi.\sigma \rangle$
$\quad \rightarrow_{slist} \quad \{ \quad (slist_{dec}) :$
$\texttt{int x} = 0; \in \texttt{Decl},$
$(dec_{var-2}) :$
$\mathcal{E}[\![0]\!]_{\phi.\sigma} = \mathbf{0},$
$\langle \texttt{int x} = 0; , \phi.\sigma \rangle \rightarrow_{dec} \phi[^{\mathbf{0}}/\texttt{x}].\sigma$
$\phi[^{\mathbf{0}}/\texttt{x}].\sigma$

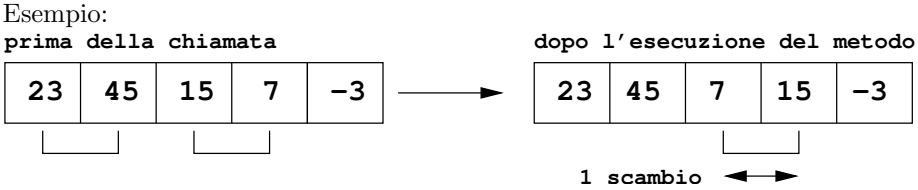
Per la definizione della modifica dei frame, si ha che le espressioni $\phi[^{\mathbf{1}}/\texttt{x}][^{\mathbf{0}}/\texttt{x}].\sigma$ e $\phi[^{\mathbf{0}}/\texttt{x}].\sigma$ rappresentano lo stesso stato. Quindi le due `Slist` sono equivalenti.

ESERCIZIO 4 (6 punti)

Completare la definizione del seguente metodo.

```
public int metodo(int[] a)
/** Scandisce l'array due elementi alla volta (il primo e il secondo,
 * il terzo ed il quarto etc.). Se l'array ha un numero dispari di
 * elementi l'ultimo elemento non viene considerato. Per ogni coppia
 * scandita, se il primo elemento e' maggiore del secondo, i due elementi
 * vengono scambiati di posto. Il metodo restituisce il numero degli
 * scambi effettuati.
 */
```

Esempio:



SOLUZIONE

```
public int metodo(int[] a) {
  int scambi = 0;
  int appoggio;
  int stop;
  if (a.length % 2 == 0)
    stop = a.length;
  else
    stop = a.length - 1;
  for (int i = 0; i < stop; i = i + 2)
    if (a[i] > a[i+1]) {
      scambi = scambi + 1;
      appoggio = a[i];
      a[i] = a[i+1];
      a[i+1] = appoggio;
    }
  return scambi;
}
```

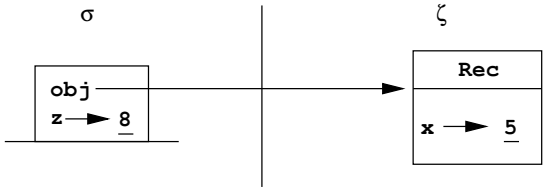
ESERCIZIO 5 (6 punti)

Si consideri la seguente definizione di classe.

```
class Rec {
  public int x;
  public void recm (int y) {
    if (y > 1) {
```

```
      this.x = this.x * this.x;
      this.recm(y-1);
    }
  }
}
```

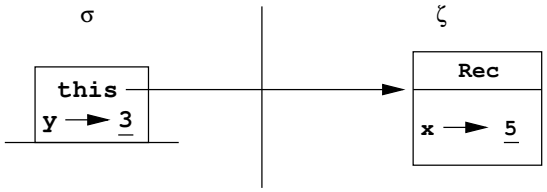
Si esegua il comando `obj.recm(3);` a partire dallo stato rappresentato in figura.



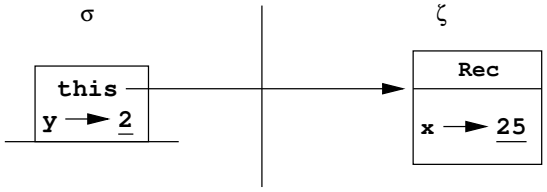
Si disegnino tutti gli stati (σ e ζ) intermedi nell'ordine dato dall'esecuzione ricorsiva: lo stato all'inizio dell'esecuzione della prima chiamata, poi lo stato all'inizio dell'esecuzione della seconda chiamata, poi lo stato all'inizio dell'esecuzione della terza chiamata. A questo punto si disegni lo stato immediatamente precedente all'uscita della terza chiamata ricorsiva. Poi lo stato immediatamente precedente all'uscita della seconda chiamata e poi lo stato immediatamente precedente all'uscita della prima chiamata. Infine si disegni lo stato risultante dall'esecuzione del comando dato.

SOLUZIONE

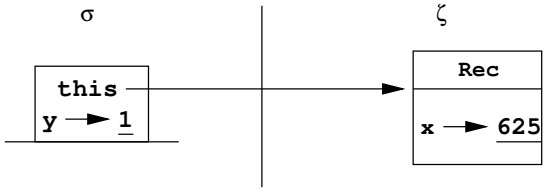
Inizio prima chiamata:



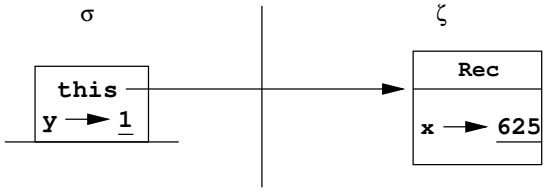
Inizio seconda chiamata:



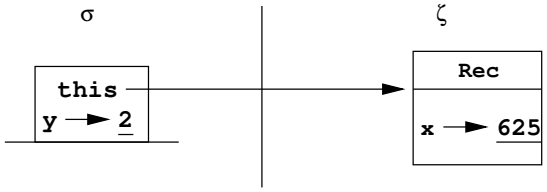
Inizio terza chiamata:



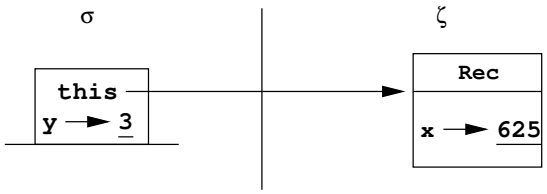
Fine terza chiamata:



Fine seconda chiamata:



Fine prima chiamata:



Stato dopo l'esecuzione di `obj.recm(3);`: