Fondamenti di Programmazione - 29/11/2012
Esame totale sessione straordinaria laureandi e fuoricorso A.A. 11/12
I Parziale A.A. 12/13

Cognome:	Nome:	Matricola: ....

▫ TOTALE:	**Svolgere Es.1, Es.2, Es.4**	Tempo: 1 ora e 45 min.
▫ I PARZIALE:	**Svolgere Es.1, Es.2, Es.3**	Tempo: 1 ora e 45 min.

Anno di immatricolazione:	

Es. 1) Sia Λ={`Questo, È, Un, Esercizio, Già, Fatto, In, Classe, Ma, Attenti`} e siano $\mathcal{L}_1 = \{$`Questo`^{n} `È Un Esercizio`$^n \mid n \geqslant 2\}$, $\mathcal{L}_2 = \{$`Già`^{n} `Fatto`$^{n+m}$ `(In Classe)`$^m \mid n \geqslant 2,\ m \geqslant 1\}$ e $\mathcal{L}_3 = \{$`Ma`^{n} `Attenti`$^m \mid n \geqslant 0,\ m \geqslant 0\}$ linguaggi su Λ.

a) Definire una grammatica che genera il linguaggio:

$$\mathcal{L}_{123} = \{s_1 s_2 s_3 \mid s_1 \in \mathcal{L}_1,\ s_2 \in \mathcal{L}_2,\ s_3 \in \mathcal{L}_3\}.$$

Per esempio, la stringa
`QuestoQuestoÈUnEsercizioEsercizioGiàGiàGiàFattoFattoFattoFattoFattoInClasseInClasseMaMaAttenti`
appartiene a $\mathcal{L}_{123}$.

b) La stringa
`QuestoQuestoÈUnEsercizioEsercizioGiàFattoInClasseInClasseInClasseMa`
appartiene a $\mathcal{L}_{123}$**?** Se no, motivare la risposta. Se si, mostrare l'albero di derivazione.

c) Facoltativo - Se la grammatica definita è ambigua, dimostrarlo.

SOLUZIONE:
a)

S ::= $S_1 S_2 S_3$
S_1 ::= `Questo` S_1 `Esercizio` | `QuestoQuestoÈUnEsercizioEsercizio`
S_2 ::= $S_2' S_2''$
S_2' ::= `Già` S_2' `Fatto` | `GiàGiàFattoFatto`
S_2'' ::= `Fatto` S_2'' `InClasse` | `FattoInClasse`
S_3 ::= $S_3' S_3''$
S_3' ::= `Ma` S_3' | ε
S_3'' ::= `Attenti` S_3'' | ε

b) La stringa
`QuestoQuestoÈUnEsercizioEsercizioGiàFattoInClasseInClasseInClasse`
NON appartiene a $\mathcal{L}_{123}$ in quanto devono esserci almeno 2 "`Già`" e almeno 2 "`Fatto`". Inoltre, il numero di "`Fatto`" non è corretto rispetto al numero di "`InClasse`".

c) La grammatica definita nel punto **a)** non è ambigua. Infatti, ... (ragionare sulla struttura delle produzioni e quindi degli alberi di derivazione).

Es. 2) Dato A={`Facile,Difficile`}, sia A^+ l'insieme di tutte le stringhe su A, ad esclusione della stringa ε. Definire un sistema di transizione per $A_G = \{(\texttt{CompitoCompito}\cdot s)^n | n \geqslant 1, s \in A\}$ in modo che la semantica di una stringa $s \in A_G$ sia **(i)** la stringa `BENE` se il numero di `Facile` in s è il doppio del numero di `Difficile`; **(ii)** la stringa `MALE` se il numero di `Difficile` in s è il doppio del numero di `Facile`; la stringa `DIPENDE` in ogni altro caso.

Per esempio, la semantica della stringa
"`CompitoCompitoDifficileCompitoCompitoFacileCompitoCompitoDifficileCompitoCompitoDifficile`" è `DIPENDE`, della stringa
"`CompitoCompitoFacileCompitoCompitoDifficileCompitoCompitoDifficileCompitoCompitoFacile`
`CompitoCompitoFacileCompitoCompitoFacile`" è `BENE`, della stringa
"`CompitoCompitoDifficileCompitoCompitoDifficile`" è `DIPENDE`, mentre della stringa
"`CompitoCompitoFacileCompitoCompitoDifficileCompitoCompitoDifficile`" è `MALE`.

Le configurazioni del sistema contengono, tra le altre, $\{s | s \in A_G\}$.
Svolgere l'esercizio specificando anche le restanti configurazioni.

SOLUZIONE:

$$\Gamma = \{s | s \in A_G\} \bigcup$$
$$\{\langle s, n, m\rangle | s \in A_G \cup \{\varepsilon\},\ n \in \mathbb{N},\ m \in \mathbb{N}\} \bigcup$$
$$\{s | s \in \{\texttt{BENE}, \texttt{MALE}, \texttt{DIPENDE}\}\}$$

$$T = \{s | s \in \{\texttt{BENE}, \texttt{MALE}, \texttt{DIPENDE}\}\}$$

Assumendo di saper fare la somma di numeri naturali, la moltiplicazione di numeri naturali e il loro confronto (e usando "==" per *uguale* e "!=" per *diverso*), la relazione di transizione è così definita:

$$\frac{}{s \longrightarrow \langle s, 0, 0\rangle} \quad (Iniziale)$$

$$\frac{s \in A_G \cup \{\varepsilon\} \quad k = n + 1}{\langle \texttt{CompitoCompitoFacile}s, n, m\rangle \longrightarrow \langle s, k, m\rangle} \quad (Intermedia1)$$

$$\frac{s \in A_G \cup \{\varepsilon\} \quad k = m + 1}{\langle \texttt{CompitoCompitoDifficile}s, n, m\rangle \longrightarrow \langle s, n, k\rangle} \quad (Intermedia2)$$

$$\frac{n{==}2 * m}{\langle \varepsilon, n, m\rangle \longrightarrow \texttt{BENE}} \quad (Finale1)$$

$$\frac{m{==}2 * n}{\langle \varepsilon, n, m\rangle \longrightarrow \texttt{MALE}} \quad (Finale2)$$

$$\frac{m!{=}2 * n \quad n!{=}2 * m}{\langle \varepsilon, n, m\rangle \longrightarrow \texttt{DIPENDE}} \quad (Finale3)$$

Es. 3)

Dire se i seguenti comandi `COM1` e `COM2` sono equivalenti.

`COM1:` `if E then` C_1`;`C_3 `else` C_2`;`C_3 `fi`

`COM2:` `if E then` C_1 `else` C_2 `fi;` C_3

SOLUZIONE:

N.B.: NON si fanno assunzioni sullo stato iniziale. Si consideri quindi un generico stato σ. la dimostrazione avviene poi ragionando per casi.

CASO I: Consideriamo il caso in cui $\varepsilon[\![\mathtt{E}]\!]\sigma = tt$.

Per il comando `COM1` abbiamo:

$$\frac{\varepsilon[\![\mathtt{E}]\!]\sigma = tt \qquad \dfrac{\langle \mathtt{C}_1,\sigma\rangle \longrightarrow_{com} \sigma' \quad \langle \mathtt{C}_3,\sigma'\rangle \longrightarrow_{com} \sigma''}{\langle \mathtt{C}_1\mathtt{;C}_3,\sigma\rangle \longrightarrow_{com} \sigma''}\;(com_;)}{\langle \mathtt{if\ E\ then\ C}_1\mathtt{;C}_3\ \mathtt{else\ C}_2\mathtt{;C}_3\ \mathtt{fi},\sigma\rangle \longrightarrow_{com} \sigma''}\;(com_{if-tt})$$

Per il comando `COM2` abbiamo:

$$\frac{\dfrac{\varepsilon[\![\mathtt{E}]\!]\sigma = tt \quad \langle \mathtt{C}_1,\sigma\rangle \longrightarrow_{com} \sigma'}{\mathtt{if\ E\ then\ C}_1\ \mathtt{else\ C}_2\ \mathtt{fi},\sigma\rangle \longrightarrow_{com} \sigma'}\;(com_{if-tt}) \quad \langle \mathtt{C}_3,\sigma'\rangle \longrightarrow_{com} \sigma''}{\langle \mathtt{if\ E\ then\ C}_1\ \mathtt{else\ C}_2\ \mathtt{fi;\ C}_3,\sigma\rangle \longrightarrow_{com} \sigma''}\;(com_;)$$

CASO II: Consideriamo ora il caso in cui $\varepsilon[\![\mathtt{E}]\!]\sigma = ff$

Per il comando `COM1` abbiamo:

$$\frac{\varepsilon[\![\mathtt{E}]\!]\sigma = ff \qquad \dfrac{\langle \mathtt{C}_2,\sigma\rangle \longrightarrow_{com} \sigma_1' \quad \langle \mathtt{C}_3,\sigma_1'\rangle \longrightarrow_{com} \sigma_1''}{\langle \mathtt{C}_2\mathtt{;C}_3,\sigma\rangle \longrightarrow_{com} \sigma''}\;(com_;)}{\langle \mathtt{if\ E\ then\ C}_1\mathtt{;C}_3\ \mathtt{else\ C}_2\mathtt{;C}_3\ \mathtt{fi},\sigma\rangle \longrightarrow_{com} \sigma_1''}\;(com_{if-ff})$$

Per il comando `COM2` abbiamo:

$$\frac{\dfrac{\varepsilon[\![\mathtt{E}]\!]\sigma = ff \quad \langle \mathtt{C}_2,\sigma\rangle \longrightarrow_{com} \sigma_1'}{\mathtt{if\ E\ then\ C}_1\ \mathtt{else\ C}_2\ \mathtt{fi},\sigma\rangle \longrightarrow_{com} \sigma_1'}\;(com_{if-ff}) \quad \langle \mathtt{C}_3,\sigma_1'\rangle \longrightarrow_{com} \sigma_1''}{\langle \mathtt{if\ E\ then\ C}_1\ \mathtt{else\ C}_2\ \mathtt{fi;\ C}_3,\sigma\rangle \longrightarrow_{com} \sigma_1''}\;(com_;)$$

Non si fatta alcuna ipotesi sullo stato iniziale σ e si sono ottenuti i seguenti finali: nel **CASO I**, σ'' sia per `COM1` sia per `COM2`; nel **CASO II**, σ_1'' sia per `COM1` sia per `COM2` Quindi, partendo da un generico stato, i due comandi portano in tuti i casi possibili nello stesso stato e quindi nella stessa configurazione terminale. I comandi sono pertanto equivalenti.

Es. 4) I seguenti programmi `P1` e `P2` in +/- Java sono identici fino alla linea 29. Considerando che il simbolo "?" indica un generico valore intero che inizializza la variabile `j`:

a) si calcoli direttamente lo stato σ_{29} alla linea 29 in tutte le sue componenti ρ_i, μ_j e δ_1. <u>Non</u> è necessario mostrare i passaggi e le regole utilizzate.

b) eseguendo le rispettive linee 30-35 dallo stato σ_{29} (si ricorda che $\rho_i(\mathtt{j})$ =? valore generico) dimostrare se i due programmi `P1` e `P2` sono equivalenti o meno. Per la dimostrazione è sufficiente indicare le regole da utilizzare nelle linee `30-35` di `P1` e `P2`, mostrare stacks e heaps risultanti, e trarre le conclusioni per i vari casi.

P1

```
01: class Producer {
02:   private int resource;
03:
04:   public int getResource() {
05:     return this.resource;
06:   }
07:}
08:
09:class Consumer {
10:   private Producer p;
11:
12:   public Consumer(int eur) {
13:     if(eur >= 8) this.p = new Producer();
14:     else ;
15:   }
16:
17:   public Producer producerFactory(int x) {
18:     if(x < 8) this.p = new Producer();
19:     return this.p;
20:   }
21:}
22:
23:public class Program {
24:
25:   public static void main(String[] cLine) {
26:     int j = ?;
27:     Consumer c = new Consumer(j);
28:     Producer p = c.producerFactory(j);
29:
30:     if (j >= 8)
31:      j = p.getResource();
32:     else {
33:      p = c.producerFactory(j);
34:      j = p.getResource();
35:     }
36:
37:   }
38:}
```

P2

```
01: class Producer {
02:   private int resource;
03:
04:   public int getResource() {
05:     return this.resource;
06:   }
07:}
08:
09:class Consumer {
10:   private Producer p;
11:
12:   public Consumer(int eur) {
13:     if(eur >= 8) this.p = new Producer();
14:     else ;
15:   }
16:
17:   public Producer producerFactory(int x) {
18:     if(x < 8) this.p = new Producer();
19:     return this.p;
20:   }
21:}
22:
23:public class Program {
24:
25:   public static void main(String[] cLine) {
26:     int j = ?;
27:     Consumer c = new Consumer(j);
28:     Producer p = c.producerFactory(j);
29:
30:     while(j >= 8) {
31:         j = p.getResource();
32:     }
33:
34:     p = c.producerFactory(j);
35:     j = p.getResource();
36:
37:   }
38:}
```

SOLUZIONE:

a) Lo stato alla linea `29` è il seguente:

$$\sigma_{29} = \langle \rho_i, \mu_j, \delta_1 \rangle$$

ρ_i :

cLine	*null*
j	?
c	l_1
p	?

μ_j :

l_1	$\langle\ \varphi_{v(l_1)} :$ (p | ?) $,\ \varphi_{m(Consumer)} \rangle$
?	?

δ_1 :

Producer	$\langle \varphi_{v(Producer)}, \varphi_{m(Producer)} \rangle$
Consumer	$\langle \varphi_{v(Consumer)}, \varphi_{m(Consumer)} \rangle$

$\varphi_{v(Producer)}$:

resource	0

$\varphi_{m(Producer)}$:

getResource	$\langle$ int, $\perp$, {return this.resource;} $\rangle$

$\varphi_{v(Consumer)}$:

p	*null*

$\varphi_{m(Consumer)}$:

producerFactory	$\langle$ Producer, x, {if(x < 8) this.p = new Producer(); return this.p;} $\rangle$
<init>	$\langle$ void, eur, {if(eur >= 8) this.p = new Producer(); else ;} $\rangle$

b) Si ragiona per casi:

CASO I: Supponendo che j $\geqslant 8$ si ha che $\sigma_{29} = \langle \rho_i, \mu_j, \delta_1 \rangle$ è lo stato in cui ρ_i e μ_j sono:

ρ_i :

cLine	*null*
j	x (con $x \geqslant 8$)
c	l_1
p	l_2

μ_j :

l_1	$\langle\ \varphi_{v(l_1)}$: [p | l_2] , $\varphi_{m(Consumer)} \rangle$
l_2	$\langle\ \varphi_{v(l_2)}$: [resource | 0] , $\varphi_{m(Producer)} \rangle$

Si osservi che, fino alla linea 29, per j $\geqslant 8$ l'istanza della classe Producer riferita da l_2, e quindi da p, è stata creata dal costruttore della classe Consumer.

- **Programma P1:** Applicazione della regola com_{if_true} alle linee 30-35, applicazione della regola com_{assign} per j = p.getResource(); e quindi della regola $exp_{mcallnodecl}$ alla linea 31 per la chiamata p.getResource() che ritorna 0. Quindi, il sistema di transizione porta in una configurazione terminale $\sigma'_{29} = \langle \rho'_i, \mu_j, \delta_1 \rangle$ dove

ρ'_i :

cLine	$null$
j	0
c	l_1
p	l_2

- **Programma P2:** Applicazione della regola com_{while_true} alla linea 30-32 (il comando while effettua un solo ciclo), applicazione della regola com_{concat} alle linee 34-35, applicazione della regola com_{assign} per p = c.producerFactory(j); e quindi della regola $exp_{mcallparnodecl}$ alla linea 34 per la chiamata c.producerFactory(j) che ritorna una nuova istanza della classe Producer riferita da l_3, applicazione della regola com_{assign} per j = p.getResource(); e quindi della regola $exp_{mcallnodecl}$ alla linea 35 per la chiamata p.getResource() che ritorna 0.

  Quindi, il sistema di transizione porta in una configurazione terminale $\sigma'_{29} = \langle \rho'_i, \mu'_j, \delta_1 \rangle$ dove

ρ'_i :

cLine	$null$
j	0
c	l_1
p	l_3

μ'_j :

l_1	$\langle \; \varphi_{v(l_1)} : \boxed{\texttt{p} \mid l_2} \; , \varphi_{m(Consumer)} \rangle$
l_2	$\langle \; \varphi_{v(l_2)} : \boxed{\texttt{resource} \mid 0} \; , \varphi_{m(Producer)} \rangle$
l_3	$\langle \; \varphi_{v(l_3)} : \boxed{\texttt{resource} \mid 0} \; , \varphi_{m(Producer)} \rangle$

Si osservi che ora si ha una ulteriore istanza della classe Producer, e quindi μ_j del programma **P1** è diverso da μ'_j del programma **P2**.

CASO II: Supponendo che j < 8 si ha che $\sigma_{29} = \langle \rho_i, \mu_j, \delta_1 \rangle$ è lo stato in cui ρ_i e μ_j sono:

ρ_i :

`cLine`	*null*
`j`	x (con $x < 8$)
`c`	l_1
`p`	l_2

μ_j :

l_1	$\langle\ \varphi_{v(l_1)} : \begin{array}{\|c\|c\|}\hline \texttt{p} & l_2 \\ \hline\end{array}\ , \varphi_{m(Consumer)}\ \rangle$
l_2	$\langle\ \varphi_{v(l_2)} : \begin{array}{\|c\|c\|}\hline \texttt{resource} & 0 \\ \hline\end{array}\ , \varphi_{m(Producer)}\ \rangle$

Si osservi che, fino alla linea 29, l'istanza della classe Producer riferita da l_2 è stata ora creata dalla chiamata alla funzione c.producerFactory(j).

- **Programma P1:** Applicazione della regola com_{if_false} alle linee 30-35, applicazione della regola com_{assign} per p = c.producerFactory(j); e quindi della regola $exp_{mcallparnodecl}$ alla linea 33 per la chiamata c.producerFactory(j) che ritorna una nuova istanza della classe Producer riferita da l_3, applicazione della regola com_{assign} per j = p.getResource(); e quindi della regola $exp_{mcallnodecl}$ alla linea 34 per la chiamata p.getResource() che ritorna 0.

- **Programma P2:** Applicazione della regola com_{while_false} alle linee 30-32, applicazione della regola com_{concat} alle linee 34-35, applicazione della regola com_{assign} per p = c.producerFactory(j); e quindi della regola $exp_{mcallparnodecl}$ alla linea 34 per la chiamata c.producerFactory(j) che ritorna una nuova istanza della classe Producer riferita da l_3, applicazione della regola com_{assign} per j = p.getResource(); e quindi della regola $exp_{mcallnodecl}$ alla linea 35 per la chiamata p.getResource() che ritorna 0.

  Le tabelle rappresentanti gli stati finali per i due programmi nal **CASO II** non vengono mostrate. Si può comunque facilmente dedurre che i due programmi portano allo stesso stato finale.

- **Conclusione:** I **CASI I** e **II** esauriscono tutti i casi possibili. Sebbene per il **CASO II** i due programmi `P1` e `P2` portano allo stesso stato finale, per il **CASO I** i due programmi portano a <u>differenti</u> stati finali. Quindi, `P1` e `P2` NON sono equivalenti.