Linguaggi per il Web – appello del 24/7/2013

COGNOME: _____

NOME: _____

MATRICOLA: _____

Autorizzo la pubblicazione del mio voto di questo esame sul sito web `http://www.dis.uniroma1.it/~rosati/lw`, secondo quanto prevede il decreto legislativo 196/2003 (codice in materia di protezione dei dati personali) che dichiaro di conoscere. In fede,

_____

Esercizio 1 (a) Data la seguente grammatica G:

$$S \to AB$$
$$A \to cA \mid bB$$
$$B \to Ccba$$
$$C \to baS \mid cSCB \mid ab$$

è possibile stabilire se G è una grammatica LL(1) senza costruire esplicitamente gli insiemi FIRST e FOLLOW e la tabella di parsing di G? motivare la risposta;

(b) Data la seguente grammatica G:

$$S \to SA \mid SB \mid SC \mid AD$$
$$A \to bcBA \mid bcBB$$
$$B \to CcbaD \mid Cc \mid d$$
$$C \to baS \mid cSCB \mid ab$$
$$D \to DDba \mid DDc \mid dc \mid S$$

scrivere una grammatica G' tale che G' non presenti né ricorsione sinistra diretta né prefissi comuni e tale che $\mathcal{L}(G') = \mathcal{L}(G)$.

Esercizio 2 Si consideri il frammento del linguaggio Java costituito dalle stringhe che corrispondono alla definizione di un metodo Java. Un metodo ha una intestazione e un corpo. L'intestazione è costituita da un tipo che può essere `void`, `int`, `float` o `String`, dal nome del metodo e da una sequenza di zero o più argomenti tra parentesi tonde e separati da virgole. Ogni argomento è di tipo `int` o `float` o `String`. Il corpo è racchiuso tra parentesi graffe ed è costituito da una sequenza di zero o più istruzioni. Ogni istruzione può essere: (i) una invocazione di metodo, in cui ogni argomento (parametro attuale) può essere o una costante intera, una costante di tipo `float`, una costante di tipo stringa, un identificatore di variabile, oppure una somma di sottoespressioni di questo tipo; (ii) una istruzione di tipo `if`, con ramo `else` opzionale, il cui ramo `then` e l'eventuale ramo `else` contengono una istruzione, e la cui condizione è un confronto di uguaglianza o disuguaglianza tra due espressioni, ognuna delle quali può essere una costante intera, una costante di tipo `float`, una costante di tipo stringa, un identificatore di variabile, o una somma di sottoespressioni di questo tipo; (iii) una assegnazione, il cui lato destro può essere una costante intera, una costante di tipo `float`, una costante di tipo stringa, un identificatore di variabile, una invocazione di metodo, o una somma di sottoespressioni di questo tipo; (iv) una sequenza (anche vuota) di istruzioni delimitata da parentesi graffe; (v) se il metodo non è di tipo `void`, allora deve essere presente una istruzione `return` come istruzione finale del corpo del metodo: l'argomento di tale istruzione può essere una costante intera, una costante di tipo `float`, una costante di tipo stringa, un identificatore di variabile, o una somma di sottoespressioni di questo tipo; se il metodo è di tipo `void`, allora non è presente nessuna istruzione `return`.

Esempi di stringhe appartenenti a questo linguaggio sono i seguenti:

```
void metodo1() {
  z=-1;
  x=metodo3(123+y);
  metodo4();
  { y = p(0, "ciao"); }
}
```

```
float metodo2(String x, int y) {
  paperino="ciao";
  { z=metodo5();
    {  y=met3(x+z,-1,3.0+12.4);
       z=metodo6("ciao",z,1); }
    if (p!="abc xyz" + q) a=1; }
  return -0.23E-12 + x;
}
```

```
String metodo3(int x, String y, float z) {
  a123=z;
  {  paperino=123;
     { x=met1(x,abc12,0,a,y); y="ciao"; }
  }
  if (z==3.14) y="ok" else pluto(25,x+"abc");
  return "abc xyz";
}
```

Scrivere una grammatica non contestuale per tale linguaggio, dividendo la specifica del lessico del linguaggio (che va definita mediante espressioni regolari) dalla specifica della sintassi vera e propria. Scrivere preferibilmente tale specifica come specifica JavaCC.

Esercizio 3 Data la seguente DTD:

```
<!DOCTYPE z [
  <!ELEMENT z ((a|b|c),(d|e|f),z?)>
  <!ELEMENT a ((a|b)*,(c,d)+)>
  <!ELEMENT b (#PCDATA|f)*>
  <!ELEMENT c ((c,d)*,(a|b)+)>
  <!ELEMENT d (#PCDATA|g)*>
  <!ELEMENT e (g?,f*,e?)>
  <!ELEMENT f EMPTY>
  <!ELEMENT g (#PCDATA|g)*>
  <!ATTLIST a attra CDATA #IMPLIED>
  <!ATTLIST b attrb CDATA #REQUIRED>
  <!ATTLIST x attrx CDATA #REQUIRED>
]>
```

1) dire se la DTD è corretta ed in caso negativo evidenziare gli errori presenti e correggerli; 2) scrivere un documento XML che sia valido rispetto alla DTD (eventualmente corretta).

Esercizio 4 Data la seguente DTD:

```
<!DOCTYPE r [
  <!ELEMENT r (a,c)*>
  <!ELEMENT a ((d|b)+,(g,f)*)>
  <!ELEMENT c (d?,b,g*)>
  <!ELEMENT d (#PCDATA)>
  <!ELEMENT b (#PCDATA)>
  <!ELEMENT g (#PCDATA)>
  <!ELEMENT f EMPTY>
  <!ATTLIST a attra CDATA #REQUIRED>
  <!ATTLIST c attrc CDATA #IMPLIED>
]>
```

scrivere un XML Schema corrispondente a tale DTD.

Esercizio 5 Scrivere un foglio di stile XSL che, dato un documento XML, restituisce il documento tale che: 1) l'elemento radice di input viene trasformato in un elemento `r`, e il suo contenuto viene ricorsivamente trasformato; 2) ogni elemento figlio dell'elemento radice di input non viene copiato, ma il suo contenuto viene ricorsivamente trasformato; 3) per ogni elemento che è figlio di un figlio dell'elemento radice viene creato in output un elemento `z`, e viene creato, come sottoelemento di tale elemento `z`, un elemento vuoto il cui nome è uguale al nome dell'elemento di input corrente. Inoltre, il contenuto dell'elemento di input viene ricorsivamente trasformato; 4) tutti gli altri elementi vengono cancellati e il loro contenuto viene ricorsivamente trasformato: 5) tutte le parti testuali del documento di input non vengono copiate in output.

Ad esempio, se il documento XML di input è il seguente:

```
<a>
  <d>testo 0
    <rad>testo 1</rad>
  </d>
  <b>
    <p>
      <w>
        <z>testo 2</z>
      </w>
    </p>
  </b>
  <c>
    <p/>
    <y>testo 3
      <d>testo 4</d>
    </y>
  </c>
  <f/>
</a>
```

il foglio di stile applicato al documento deve restituire il documento seguente:

```
<r>
  <z>
    <rad/>
  </z>
  <z>
    <p/>
  </z>
  <z>
    <p/>
  </z>
  <z>
    <y/>
  </z>
</r>
```

Esercizio 6 Siano `a`, `g`, `p`, `r`, `s` e `v` delle URI di un namespace con prefisso `m`. Scrivere un modello RDF che rappresenta le seguenti informazioni: "Le URI `a`, `g`, `p`, `r`, `s` e `v` rappresentano impiegati i cui nomi sono, rispettivamente, Anna, Guido, Patrizia, Ruggero, Stefania, Valerio. Inoltre: Anna lavora con Guido e Stefania; Patrizia e Ruggero lavorano con una persona che conosce Stefania; infine, Guido lavora con due impiegati che credono che Valerio lavori con Anna. Usare la URI `foaf:name` per esprimere il predicato "ha nome", la URI `m:lavoraCon` per esprimere il predicato "lavora con", e la URI `m:crede` per esprimere il predicato "crede".