Linguaggi per il Web – appello del 11/1/2013

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:

$$S \to ABCd$$
$$A \to abA \mid bBC \mid cC$$
$$B \to baA \mid cbB \mid aC$$
$$C \to cC \mid a$$

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:

$$S \to Sb \mid ABCd$$
$$A \to abA \mid aBC \mid abcC$$
$$B \to baA \mid baBC \mid BcC$$
$$C \to Cc \mid CBa \mid aa$$

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

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` 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 `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 oppure una costante di tipo stringa oppure un identificatore di variabile; (ii) una istruzione `System.out.println` il cui argomento può essere una costante intera oppure una costante di tipo stringa oppure un identificatore di variabile; (iii) una assegnazione, il cui lato destro può essere una costante intera oppure una costante di tipo stringa oppure un identificatore di variabile oppure una invocazione di metodo; (iv) una sequenza 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 variabile oppure una costante intera oppure una costante di tipo stringa oppure una invocazione di metodo; 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 = p(0, "ciao"); }
}
```

```
int metodo2(String x, int y) {
  paperino="ciao";
  { z=met4();
    {  y=met3(x,z,-1);
       z=metodo("ciao ciao",z,1); }
    System.out.println(z); }
  return metodo2(x,z);
}
```

```
String metodo3(int x, String y, int z) {
  a123=z;
  {  paperino=123;
     { x=met1(x,abc12,0,a,y); y="ciao"; }
  }
  System.out.println("abc xyz");
  return "ciao ciao";
}
```

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 r [
  <!ELEMENT r (a+, (b,c)?, (d|e)*)>
  <!ELEMENT a (#PCDATA)>
  <!ELEMENT b ((c|d)*, (a|e)+)>
  <!ELEMENT c (d,e)>
  <!ELEMENT d (#PCDATA|e)*>
  <!ELEMENT e EMPTY>
  <!ATTLIST r val CDATA #REQUIRED>
  <!ATTLIST a attr CDATA #IMPLIED>
]>
```

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) e che contenga tutti gli elementi dichiarati nella DTD.

Esercizio 4 Data la DTD (eventualmente corretta) del precedente esercizio, 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 copiato e il suo contenuto viene ricorsivamente trasformato; 2) ogni elemento figlio dell'elemento radice di input viene trasformato in un elemento `x`, aggiungendo un attributo di nome `elemName` e valore uguale al nome dell'elemento di input corrente. Inoltre il contenuto dell'elemento viene ricorsivamente trasformato; 3) ogni elemento che è figlio di un figlio dell'elemento radice viene copiato in output, aggiungendo un attributo di nome `inputLevel` e valore `3`. Inoltre, il contenuto dell'elemento viene eliminato; 4) nessuna parte testuale del documento di input viene copiata in output.

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

```
<z>
  <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>
    <p>testo 5
      <d>testo 6</d>
    </p>
    <p/>
  </c>
  <z/>
</z>
```

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

```
<z>
  <x elemName="d">
     <rad inputLevel="3"/>
  </x>
  <x elemName="b">
    <p inputLevel="3"/>
  </x>
  <x elemName="c">
    <p inputLevel="3"/>
    <y inputLevel="3"/>
    <p inputLevel="3"/>
    <p inputLevel="3"/>
  </x>
  <x elemName="z"/>
</z>
```

Esercizio 6 Siano `b`, `c`, `d`, `r`, `s` e `t` delle URI di un namespace con prefisso `p`. Scrivere un modello RDF che rappresenta le seguenti informazioni: "Le URI `b`, `c`, `d`, `r`, `s` e `t` rappresentano persone i cui nomi sono, rispettivamente, Beatrice, Carlo, Dario, Renata, Simona, Tommaso. Inoltre: Simona crede che Renata conosca Carlo e Dario; Renata conosce sia Beatrice che Tommaso; infine, Carlo conosce una persona che conosce Dario e una persona che conosce Beatrice. Usare la URI `foaf:name` per esprimere il predicato "ha nome" e la URI `foaf:knows` per esprimere il predicato "conosce".