Software-Modellierung und Verifikation
Informatik 2
Prof. J.-P. Katoen
RWTH Aachen

Priv.-Doz. T. Noll noll@cs.rwth-aachen.de
D. Klink klink@cs.rwth-aachen.de

5. Exercise sheet *Compiler Construction 2008*

Due to Wed., 18 June 2008, *before* the exercise course begins.

Exercise 5.1:

Let grammar G be given by:

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & Aa \mid bAc \mid Bc \mid bBa \\ A & \rightarrow & d \\ B & \rightarrow & d \end{array}$$

a) Check whether $G \in LR(1)$ by computing the $LR(1)$-sets of G.

b) Is $G \in LALR(1)$? Justify your answer.

Exercise 5.2:

Show that $LL(1) \not\subseteq LALR(1)$ by providing a counterexample.

Exercise 5.3: (optional)

Use ANTLR to write SPiT, a *Simple PetrI net Tool.*

a) Write a grammar that recognizes *Petri nets* in the following notation, where [. . .] denotes that . . . can be specified optionally.

   `model PetriNet;`

   `place` n_a`[:`i_a`]; place` n_b`[:`i_b`]; ...;`

   `trans` n_a`[:`$c_{a,1}$`] +` n_b`[:`$c_{b,1}$`] + ... ->` n_a`[:`$p_{a,1}$`] +` n_b`[:`$p_{b,1}$`] + ...;`

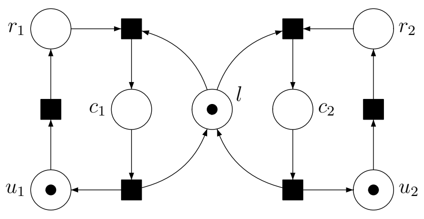
   `trans` n_a`[:`$c_{a,2}$`] +` n_b`[:`$c_{b,2}$`] + ... ->` n_a`[:`$p_{a,2}$`] +` n_b`[:`$p_{b,2}$`] + ...;`

   `...;`

   $n_a, n_b, \ldots \in \{a, \ldots, z\} \cdot \{a, \ldots, z, 0, \ldots, 9\}^*$ are the names of *places*, $i_a, i_b, \ldots \in \mathbb{N}$ are the initial number of *tokens* in the places and $c_{a,j}, c_{b,j}, \ldots \in \mathbb{N}$ and $p_{a,j}, p_{b,j}, \ldots \in \mathbb{N}$ are the number of tokens consumed and produced in each place for all transitions $j \in \{1, \ldots, n\}$.

   If the initial number of tokens is not specified, assume that the place does not contain any tokens at start. If the number of tokens consumed or produced is not specified, assume that one token is consumed or produced. Allow for line comments in Java style and for white spaces where it makes sense.

Check your grammar on the following example:



```
model PetriNet;

place l:1;  //lock

place u1:1; //uncritical section of process 1
place r1;   //process 1 requests lock
place c1;   //critical section of process 1

place u2:1; //uncritical section of process 2
place r2;   //process 2 requests lock
place c2;   //critical section of process 2

trans u1 -> r1;
trans r1 + l -> c1;
trans c1 -> u1 + l;

trans u2 -> r2;
trans r2 + l -> c2;
trans c2 -> u2 + l;
```

b) Implement a simulator, based on the ANTLR grammar, that, starting with the initial *marking* (numbers of tokens in each place), fires a transition every second and prints the current configuration to the console. If there is no transition *enabled*, i.e. there is no rule where the number of tokens on the left-hand side of the transition rule is available, stop with a *deadlock* message. If there is more than one transition that is enabled, resolve this nondeterminism by randomly selecting a transition.
You are not required to deal with technical details like catching overflows (which may occur for unbounded Petri nets).
Some ANTLR constructs that may be useful are `@header` – for package declaration and import of packages – and `@members` – for specifying fields and methods that can be used within the grammar file:

```
grammar SPiT;

@lexer::header {
    package SPiT;
}

@header {
    package SPiT;
}

@members {
    PetriNet pn = new PetriNet();
}

...
```

As main class, you may want to use the following:

```
package SPiT;

import org.antlr.runtime.*;

public class Main {

    public static void main(String[] args) throws Exception {
        if (args.length == 0)
            System.exit(0);

        SPiTLexer lexer = new SPiTLexer(new ANTLRFileStream(args[0]));
        SPiTParser parser = new SPiTParser(new CommonTokenStream(lexer));
        parser.start();
    }

}
```

Avoid cryptic and/or messy code and send it (the ANTLR and java files) to `klink@cs.rwth-aachen.de`. Use `Exercise 5.3` as subject and add your student ID numbers (Mat.nr.).