Technische Universität München **Winter term 2010/11**
I7
Prof. J. Esparza / J. Křetínský **9.12.2010**

Automata and Formal Languages – Programming Assignment

Due 30.1.2010

Your task is to write a program in Java/C++ that transforms a given Presburger formula into a finite automaton recognizing its solution space. All referred files are accessible on the web of the course.

1 Functionality

1.1 Input

The executable main method has one parameter, namely a string with a name `<name>` of a file in the same (working) directory. This file contains a Presburger formula $\varphi(x_1, \ldots, x_n)$ according to the grammar `PA2.g` that contains (possibly zero) free variables $x_1, \ldots, x_n$.

1.2 Output

Your program should create a file `<name>.dotty` which contains a description of a *canonic* finite automaton over alphabet $\{0,1\}^n$ recognizing the solution space $\varphi(x_1, \ldots, x_n)$ and a list of the free variables in alphabetical order. The ith component of the alphabet corresponds to the ith free variable. For instance, if the formula contains free variables z and x, then a letter 01 corresponds to reading 0-bit of x and 1-bit of z.

A canonic finite automaton for a language L is the minimal DFA for L where, moreover, we have the following requirement on the names of the states. Since the alphabet $\{0,1\}^n$ can be ordered in the obvious increasing manner (e.g. $\{0,1\}^2$ is ordered as $00 < 01 < 10 < 11$), there is a unique breadth-first-search run through the automaton graph starting from the inital state. We require that when a state s is found as the ith state, then $s = i$ (starting with the inital state being 0).

The output file has the following structure:

```
digraph G {
<list of edges>
<list of final states>
<initial state>
}
<list of free variables>
```

where:

- `<list of edges>` is a sequence of lines each containing `<source> -> <target> [label=<label>];` and is ordered by `<source>` and edges from the same `<source>` are ordered by `<target>`
- `<list of final states>` is a sequence of lines each containing `<state>[peripheries=2];` sorted by `<state>`
- `<initial state>` is a line containing `<state>[shape=<diamond>];`
- `<source>`, `<target>`, `<state>` are decimal numbers
- `<label>` is described by the regular expression $((0+1)^n$`<space>`$)^*$ and contains all letters between `<source>` and `<target>` in the increasing order
- `<list of free variables>` is a sequence of letters with no spaces if it is nonempty, `true` if it is empty and the fomula is a tautology, and `false` if it is empty and the fomula is a contradiction.

You can check that your solution is displayed properly by running `dotty` on your output file.

1.3 An example

The file `formula0.txt` contains `(((2x-y<=2 && Ew y-4w==0) && x+y>=4) && Ez x-4z==0)`

Running your program on the input file, i.e. `pa2fa formula0.txt` (or `java -jar pa2fa.jar formula0.txt`) should produce a file `formula0.txt.dotty` in the same directory with the following content

```
digraph G {
0 -> 1 [label="00 "];
0 -> 2 [label="01 10 11 "];
1 -> 2 [label="01 10 11 "];
1 -> 3 [label="00 "];
2 -> 2 [label="00 01 10 11 "];
3 -> 3 [label="00 "];
3 -> 4 [label="01 "];
3 -> 5 [label="10 11 "];
4 -> 4 [label="00 01 "];
4 -> 5 [label="10 11 "];
5 -> 4 [label="01 "];
5 -> 5 [label="00 11 "];
5 -> 6 [label="10 "];
6 -> 5 [label="00 01 "];
6 -> 6 [label="10 11 "];
4[peripheries=2];
0[shape=diamond];
}
xy
```

1.4 Another example

On the input file `formula3.txt` with `!(Ax Ez -2x+3z <= 4 -> Ex Az 3z-2x>4)` the output file `formula3.txt.dotty` contains

```
digraph G {
0 -> 0 [label=" "];
0[peripheries=2];
0[shape=diamond];
}
true
```

2 What to hand in?

By **January 30** you have to hand in the following:

- a compiled executable file or executable .jar file;
- zipped source codes (.zip, .rar, .tar,. . . , or as a part of .jar); the code should be well structured, easily readable and properly commented and documented, in particular every class and method should be (apart from comments in the code) immediately preceded by a comment on what it does;
- a file `description.txt` containing the command used to compile your files in order to get the executable; further, the structure of your source files is very briefly discussed here, i.e. which file implements what; it may also contain any other comments from your side if necessary.

Note that the sources must be compilable on `halle.in.tum.de` using the command written in `description.txt`. Otherwise, **no points will be awarded**. Your solution should be uploaded to an svn directory that we will asssign to you shortly.

3 Points

You are allowed to work in pairs. Hence, an oral questioning will take place after evaluating your solutions. Points will only be awarded after passing this questioning. Moreover, the source codes will be checked by the standard tools for plagiarism.

2/3 of points will be awarded for correct solution to formulae similar to the ones published as “Basic examples” within the time limit of 5 seconds.

1/3 of points will be awarded for correct solution to formulae similar to the ones published as “Advanced examples” depending on the time required, however, not exceeding 1 minute.