Technische Universität München **Winter term 2011/12**
I7
Prof. J. Esparza / J. Křetínský **19.12.2011**

Automata and Formal Languages – Programming Assignment: Part 2

Due 30.1.2010

Your task is to write a program that transforms a given Presburger formula into a finite automaton recognizing its solution space. Contrary to the lecture notes, the formulae are not interpreted over natural numbers but only over those in the range $0..2^\ell - 1$ for a given fixed ℓ, e.g. 32. Moreover, if you want to reuse the first part of the project recall that the msbf encoding is used.

1 Functionality

1.1 Input

The executable main method has three parameters. Firstly, a string with a name `<name>` of a file in the same (working) directory. This file contains a Presburger formula $\varphi(x_1, \ldots, x_m)$ according to the grammar `PA2.g` that contains (possibly zero) free variables $x_1, \ldots, x_m$. The second parameter is the decimal number ℓ. The third paramater is either `y` or `n`.

1.2 Output

Your program should create a file `<name>.dotty` which contains a description of the corresponding transducer over alphabet $\{0,1\}^m$ recognizing the language of length ℓ that is the solution space of $\varphi(x_1, \ldots, x_n)$, 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.

The output file has the following structure:

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

where:

- `<initial state>` is a line containing `<state>[shape=<diamond>];`
- `<final states>` is a line containing `<state>[peripheries=2];` if there is a final state `<state>`, and is empty otherwise
- `<list of edges>` is a sequence of lines each containing `<source> -> <target> [label=<label>];`
- `<source>`, `<target>`, `<state>` are decimal numbers
- `<label>` is described by the regular expression $((0+1)^m$`<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.

Further, if the third parameter is `y`, your program should *also* create a file `<name>.txt` which contains all tuples of words in the ascending order recognized by the output automaton. Please use the decimal encoding here.

1.3 Examples

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

Running your program by `pa2fa formula0 3 y` (or `java -jar pa2fa.jar formula0 3 y`) should produce a file `formula0.dotty` in the same directory with the following (or equivalent) content

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

and a file `formula0.txt` containing "`0 4`". Indeed, due to the existential quatification both `x` and `y` must be divisible by 4. Since they are in the interval 0 to $2^3 - 1 = 7$ they are either 0 or 4. Due to $x + y >= 4$, we know at least one of them is 4. Due to $2x - y <= 2$ we have that `x` is 0 and `y` is 4. Of course, this deduction is purely illustrative and the algorithm should be based on the one in the lecture notes.

Let a file `formula1` contain `Ax x==1`. Then `pa2fa formula1 1 n` should produce a file `formula1.dotty` containing

```
digraph G {
0[shape=diamond];

0 -> 0 [label=" "];
}
false
```

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 `lxhalle.in.tum.de` using the command written in `description.txt`. Otherwise, **no points will be awarded**. Your solution should be uploaded to the subdirectory `pa2fa` of your svn directory.

You shall work in the same pairs as in the first part. Hence, an oral questioning will take place after evaluating your solutions. Points will only be awarded after passing this questioning.