Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen (LEA)
Prof. Dr. Ernst W. Mayr
Moritz Fuchs

Winter Semester 2014/15
Programming Assignment
December 02, 2014

Automata and Formal Languages

1 Programming Assignment

Your task is to write a C++/Java program that transforms a given Presburger formula into a finite automaton recognizing its solution space.

1.1 Input

Your program will receive a string `<name>` containing the name of file in the current directory. This file contains a Presburger formula which is recognizable by the grammar `PA.g` which you can download from the course homepage.
Sample program call: `pa2fa test1.txt`

1.2 Output

Your program should create a file `<name>.dotty` where `<name>` is the filename of the input file. The file should contain the minimal finite automaton recognizing the solution space of the given Presburger formula. The i-th component of the automaton should correspond to the i-th free variable (we will specify in a second what the i-th variable is).
The output file has the following form:

```
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>];` . The lines are ordered by `<source>`, 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>`. The sequence contains all final states.
- `<initial state>` is a line containing `<state>[shape=<diamond>];`, where `<state>` is the initial state of the automaton.

- `<source>`, `<target>` and `<state>` are integers representing the states of the automaton.
- `<label>` contains all input symbols on the transition between `<source>` and `<target>`. It must be recognizable by the regular expression $((0+1)^n$`<space>`$)^*$ where n is the number of free variables.
- `<list of free variables>` contains all free variables of the Presburger formula. The i-th component of the automaton you create must correspond to the i-th variable in this list. If there are no free variables in the formula print *true* if the formula is a tautology and *false* if the formula is a contradiction.

You can check your output by importing it into GraphViz[1].

1.2.1 Sample Output

The formula `(((2x-y<=2 && Ew y-4w==0) && x+y>=4) && Ez x-4z==0)` should produce the following output:

```
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
```

More examples are available on the course homepage.

[1] http://www.graphviz.org

1.3 Subtasks

The assignment can be split into several subtasks as follows:

(a) Write a transducer library and implement the algorithms you learned in the lecture (union, intersection, projection, ...).

(b) Implement a parser for the input file using ANTLR.[2]

(c) Transform the resulting AST into an automaton recognizing the solution space of the given Presburger formula.

(d) **Bonus** (will not count towards your score, but might be fun to implement/think about):

- Implement an algorithm that counts the number of solutions to the given formula. (Remember: Each solution has infinitely many encodings!)
- Add a parameter `-sample=x` to your program. The program should provide `x` distinct solutions to the given formula (e.g. in a separate file). Sample program call: `pa2fa -sample=20 test1.txt`
- Add a parameter `-max=<formula>`. Your program maximizes the given formula while only using variable assignments that satisfy the Presburger formula. Sample program call: `pa2fa -max=x+y-z test1.txt`

1.4 What to hand in

By **January 13, 2015** you have to hand in

- A compiled executable file or an executable `.jar` file.
- A zip-file containing all source files including external libraries that you used. The source code should be well-commented, in particular every class and method should come with a brief description on what it does. We should be able to compile your source code using the contents of this zip-file!
- A file description.txt containing a high level description of your program as well as compile instructions for your program ('Import into eclipse and run' does **not** count as compile instruction!)

You are allowed to work in groups of up to two students.

Please send your solution to **fuchsmo@in.tum.de** using the subject line

'AFS Programming Contest - `<Names of group members>`'

[2] http://www.antlr.org

1.5 What you are allowed to use

- ANTLR for parsing the input file using the grammar provided on the course homepage.
- All built-in functionalities of Java/C++.
- Other libraries that do not provide automata functionalities (e.g. Google Guava)

You are **not** allowed to use existing transducer/automata libraries.

1.6 Price

All submissions will be tested against a fixed set of benchmarks. The team that solves most benchmarks wins the contest. Ties will be broken using runtimes.

- **1st price: A crate of beer***
- **2nd price: A bottle of wine***
- Other participants whose submissions are recognized as reasonable: candy :)

* or some other drink of your choice.