Technische Universität München **Winter term 2011/12**
I7
Prof. J. Esparza / J. Křetínský **30.11.2011**

Automata and Formal Languages – Programming Assignment

Due 21.12.2011

Your task is to write a program in Java/C++ that executes operations on automata recognizing fixed-length languages. 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 program according to the fllowing syntax:

```
<program> ::=<command> | <command> \n <program>
<command> ::=<variable> = <expression> ;
<variable> ::=a | b | ··· | z
<expression> ::=union <variable> <variable> |
                intersection <variable> <variable> |
                negation <variable> |
                join <variable> <variable> |
                product <variable> <variable> |
                project <variable> <number> |
                section <variable> <number> <variable> |
                read <file>
<number> ::=integer
<file> ::=string
```

The variables represent minimal deterministic automata for fixed-length languages over the alphabets $\{0,1\}^m$ for various m called *dimension*; as usual, we identify $(\Sigma^m)^\ell$ with $(\Sigma^\ell)^m$, where ℓ is the *length* of the language. Therefore, we can regard the automata as recognizing m-tuples of integers using ℓ bits in the msbf encoding. The command = stands for evaluating the expression and assigning the result to the variable. Let $L(e), \ell(e), m(e)$ denote the language recognized by the automaton corresponding to the expression e, its length and its dimension, respectively. Since the operations are defined on languages of the same length, whenever two arguments u, v of an operation have different lengths, i.e. $\ell(u) \neq \ell(v)$, the shorter one needs to be padded by zeros on the left first.

- $L(\text{union } u\ v)$ contains the words of $L(u)$ and the words of $L(v)$ if $m(u) = m(v)$ and raises an exception otherwise;
- similarly for $L(\text{intersection } u\ v)$;
- $L(\text{negation } u) = (\{0,1\}^{\ell(u)})^{m(u)} \setminus L(u)$;
- $L(\text{join } u\ v) = \{(x_1, \ldots, x_{m-1}, y_2, \ldots, y_n) \mid \exists x_m = y_1 : (x_1, \ldots, x_m) \in L(u), (y_1, \ldots, y_n) \in L(v)\}$;

- $L(\text{product } u\ v) = \{(x_1, \ldots, x_m, y_1, \ldots, y_n) \mid (x_1, \ldots, x_m) \in L(u), (y_1, \ldots, y_n) \in L(v)\}$;
- $L(\text{project } u\ i) = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n) \mid (x_1, \ldots, x_n) \in L(u)\}$,
  note that the resulting automaton must be minimal;
- $L(\text{section } u\ i\ v) = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n) \mid \exists x_i \in L(v) : (x_1, \ldots, x_n) \in L(u)\}$,
  note that the result is non-empty only for automata v over Σ^m with $m = 1$;
- $L(\text{read}f)$ denotes the language of an automaton in a file f in the following format:

```
<dimension>
<length>
digraph G {
<initial state>
<final state>
<list of edges>
}
```

where:

- `<dimension>`, `<length>` are decadic integers m and ℓ
- `<initial state>` is a line containing `<state>`$[shape =< diamond >]$;
- `<final state>` is a line containing `<state>`$[peripheries = 2]$;
- `<list of edges>` is a sequence of lines each containing `<source>` -> `<target>` $[label =$ `<label>`$]$;
- `<source>`, `<target>`, `<state>` are decimal integers
- `<label>` is described by the regular expression $((0+1)^m$`<space>`$)^*$ and contains all letters between `<source>` and `<target>`

The automaton on the input is guaranteed to be minimal, accepting language of a fixed length ℓ over the alphabet $\{0, 1\}^m$, but it may not contain transitions to the trap state.

1.2 Output

Your program should create a file `<name>.txt` which contains all tuples of words in the ascending order recognized by the automaton corresponding to the variable that has been assigned a value in the last line (command) of the program. Please use the decadic encoding on the output.

1.3 An example

Consider the following files:

- `input` contains

  ```
  a = read f1 ;
  b = read f2 ;
  c = section a 2 b ;
  d = union c b ;
  e = product d c ;
  ```

- `f1` contains

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

- `f2` contains

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

Thus a recognizes pairs $(3, 1)$ and $(2, 2)$, while b recognizes numbers 0 and 1. Therefore, c recognizes only 3 and d recognizes $0, 1, 3$. Hence, the file `input.txt` will contain the following:

```
0 3
1 3
3 3
```

2 What to hand in?

By **December 21** 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 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 the solutions to the second assignment. Points will only be awarded after passing this questioning. Moreover, the source codes will be checked by the standard tools for plagiarism.

Points will be awarded for correct solutions to examples similar to the one above and those published on the web page. Point awards are subject to correctness of the result and the time performance.