Theory of Computer Science

M. Helmert, G. Röger
F. Pommerening
Spring Term 2015

University of Basel
Computer Science

Exercise Sheet 7

Due: Wednesday, April 15, 2015

Note: this exercise is a programming exercise, where you should complete existing Java code. We expect that you implement your solution on your own, that is, without using someone else's code, e.g., from the Internet. Of course you can use the Java standard library without restrictions.
We are glad to help with any technical difficulties or other problems. Please ask your tutor *with enough time to spare before the due date.*
Please do not submit your solution on paper but rather upload it in the Courses portal. If you write any test cases, also submit them together with your program (this can only have a positive effect on your marks).

Exercise 7.1 (Turing Machines; 3+2+3+2 Points)

In this exercise you should implement a simulator for a deterministic Turing machine.
You can find a basic framework for this on the lecture homepage. Implement the given methods and chose appropriate private fields for the classes. Only implement the places marked with `// TODO` in the code; do not add any additional methods, classes or public fields.

(a) Implement the following methods in the class `Tape` that represents the Turing machine's tape which is infinite in both directions and the position of the read/write head:

The constructor `Tape(word, blank)` accepts the word that is on the tape initially and the blank symbol that should be used. The read/write head initially is over the first symbol of the word.

The method `read()` should return the symbol under the read/write head.

The method `write(symbol)` should write `symbol` on the tape to the current position of the read/write head.

The methods `moveLeft()` and `moveRight()` should move the head one position to the left or right respectively.

The method `dumpAlpha()` should print out the part of the tape that is left of the current position of the read/write head (excluding the current position).

The method `dumpBeta()` should print out the part of the tape that is right of the current position of the read/write head (including the current position).

The method `usedSpace()` should return the size (the number of tape positions) of the tape that was used so far.

(b) Implement the constructor `TuringMachine(Z, Sigma, Gamma, delta, z0, blank, E)` where the parameters correspond to the normal elements of a Turing maschine. During construction make sure that the Turing machine is specified correctly: is `delta` a total function? Is `Sigma` a subset of `Gamma`? Is the `blank` in `Gamma` but not in `Sigma`? Is `E` a subset of `Z`? Throw an `InvalidSpecificationException`, if the specification is incorrect.

(c) Implement the remaining methods in the class `TuringMachine`:

`initialize(word)` should initialize the Turing machine, i.e., create the initial configuration. Make sure that only symbols from the input alphabet occur in `word`. Throw a `InvalidSpecificationException` if this is not the case.

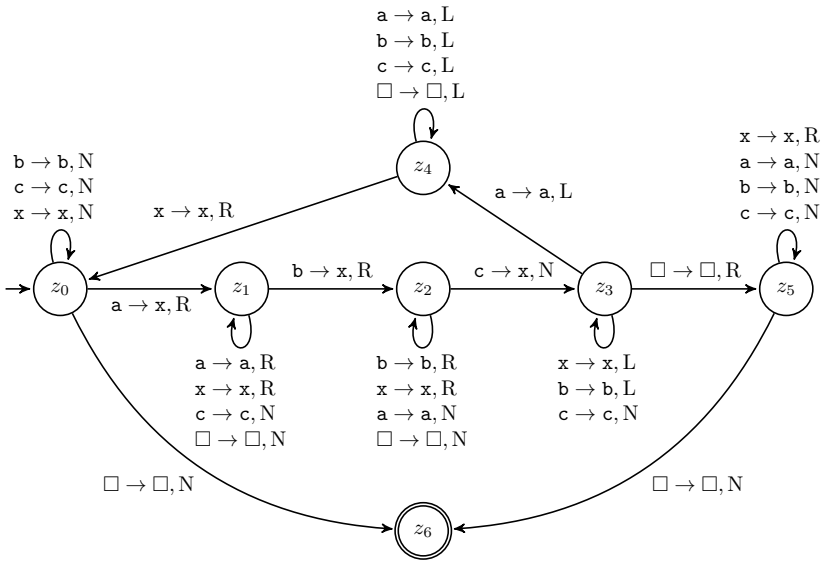
`step()` should execute one step of the Turing machine. Throw a `RuntimeException` if the machine is not initialized or is in a final state.

`run(maxSteps)` should run the Turing machine. The machine should only stop if it reaches a final state or more than `maxSteps` steps were executed. This is useful to test the machine on a word where the machine does not terminate. Do not restrict the number of steps if `maxSteps` is `null`. The method `run()` calls `run(null)` to simulate an unrestricted run.

The method `dumpConfiguration()` should print out the current configuration of the Turing machine.

The method `dumpStatistics()` should print out how many steps where simulated so far and how much space was used on the tape so far.

(d) Consider the following turing machine.



Implement the `main` method in the class `SimulateTM` that should create an object of the class `TuringMachine` representing the Turing machine above. Test different inputs for the machine, containing at least:

- the empty word, and
- a word that is not in the language accepted by the Turing machine (but not the empty word or *aabbcc*), and
- a word that is not in the language accepted by the Turing machine.

Print out the configuration in every step of the simulation. (You can do this in the method `run(maxSteps)` by default). For words that are not in the language, use a suitable value for `maxSteps` that shows that the Turing machine will never terminate. For words that are in the language, use `run()` without a parameter.

You can check your implementation with the input *aabbcc*: the Turing machine requires 29 steps and 8 tape cells for this input.