Computational complexity 2020/2021

Assignment 1

In Brief

In this assignment your job is to:

1. Write an interpreter for nondeterministic Turing machines
2. Construct a Turing machine for the language $\{ww \mid w \in \{1,2\}^*\}$
3. Write a program that translates two-tape Turing machines into equivalent one tape Turing machines **in a determinism preserving way.**

You should submit your assignments on moodle. It is not yet available, but we are working on that.

Variant of Turing Machines for this task

In this task we consider nondeterministic Turing machines. We represent states of the machines as strings and the letters from the tape and input alphabet as numbers. In particular:

- 0 denotes the blank symbol;
- `start` denotes the initial state;
- `accept` and `reject` denote the accepting and rejecting states.

The machine's description is given as a list of its transitions. Each transition is written in a separate line in the following format (see example below).

<current_state> <currently_seen_letter> <target_state> <letter_to_write> <direction>

The machine accepts its input by entering the accept state.
The machine rejects the input word by either:

- entering the reject state
- getting stuck -- entering a configuration for which there is no applicable transition
- looping forever

The Turing machine does not have to have/use the `reject` state.
The machine first writes the output letter and then moves its head.
The machine cannot move left in the first position -- if it tries to, it simply stays in place.
The machine can output blanks.

For example the following machine recognises the language of palindromes over the alphabet $\{1,2\}$:

```
start 1 golong1 0 R
start 2 golong2 0 R
start 0 accept 0 S
golong1 1 golong1 1 R
golong1 2 golong1 2 R
golong1 0 checklast1 0 L
golong2 1 golong2 1 R
golong2 2 golong2 2 R
golong2 0 checklast2 0 L
checklast1 1 goback 0 L
checklast1 0 accept 0 S
checklast2 2 goback 0 L
checklast2 0 accept 0 S
goback 1 goback 1 L
goback 2 goback 2 L
goback 0 start 0 R
```

Part 1. An interpreter

For this part your job is to write an interpreter, such that

./interpreter <path_to_turing_machine> <steps>

inputs a word and outputs either “YES” or “NO” depending whether the Turing machine has an accepting run on this word of length of at most <steps>.

For example:

./interpreter palindrome.tm 100
11222211
should output YES

./interpreter palindrome.tm 100
122122
should output NO

./interpreter palindrome.tm 5
11222211
should output NO

Your program will be tested on instances with configuration trees of size at most 250,000 (measured as the sum of lengths of all the configurations), so it is ok to copy the cofigurations.

You can write your solution in any of the following languages: Python, C++, Java, Haskell, Prolog. (If you would like to use a different one -- ask us to add it).

The interpreter can be an executable created by a Makefile, or an executable script e. g. in Python or Bash.

You should submit your solution as a compressed archive.

Part 2. A Turing machine for squares

For this part you should submit a description of a Turing machine which recognises the language of squares: $\{ww \mid w \in \{1, 2\}^*\}$.

Your solution will be tested on words of length up to 20, with a step limit of 10,000. For such instances, the size of its configuration tree should not exceed 250,000.
If the input word is a palindrome, your Turing machine should have an accepting run that fits in this limit.

For this part you should submit a single square.tm file.

Part 3. Two tape to one tape translation.

A two tape Turing machine is very much like a one tape Turing machine -- it has one state, two tapes -- each with its own head. The heads of such machines can move independently from one another.

Its input configuration is as follows:

- state is `start;`
- the first tape contains the input word followed by blanks;
- the second tape contains only blanks.

A transition of such a machine can be represented as follows:

<state> <let1> <let2> <target_state> <out_let1> <out_let2> <dir1> <dir2>

For example, the following is a two-tape machine for recognising the language of all the palindromes:

```
start 0 0 accept 0 0 S S
start 1 0 go 3 0 S S
start 2 0 go 4 0 S S
go 1 0 go 1 0 R S
go 2 0 go 2 0 R S
go 3 0 go 3 0 R S
go 4 0 go 4 0 R S
go 0 0 copy 0 0 L S
copy 1 0 copy 1 1 L R
copy 2 0 copy 2 2 L R
copy 3 0 go2 3 1 L R
copy 4 0 go2 4 2 L R
go2 1 0 go2 1 0 R S
go2 2 0 go2 2 0 R S
go2 3 0 go2 3 0 R S
go2 4 0 go2 4 0 R S
go2 0 0 check 0 0 L L
check 1 1 check 1 1 L L
check 2 2 check 2 2 L L
check 3 1 accept 3 1 S S
check 4 2 accept 4 2 S S
```

For this task your job is to write a program that translates a two-tape turing Machine into a one tape turing machine **while preserving determinism.**

./translate <path_to_a_two_tape_turing_machine>

should output a description of an equivalent one-tape machine.

If the input machine is deterministic (every configuration has at most one applicable transition), then the output machine **should also be deterministic**.

The number of transitions in the output machine should depend polynomially on the number of transitions in the input machine.

Your solution will be tested on instances with at most 3 letters -- including blank -- and at most 10 states -- including start, accept and reject (if present). For such instances your solution should output no more than 250,000 transitions which use no more than 50 letters -- including blank. (Note that the example two-tape Turing Machine uses 5 letters, which means that it won't be used for testing).

If the original machine has an accepting run of length at most 200, for an input of size at most 20, then the machine created by your program should have an accepting run on the same word of length at most 100,000.

If you have an intuitive solution (for any part) that does not fit in the limits let us know -- we might increase them.