PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACION

Complexity Theory, Semester I 2017 - IIC3242
Homework 3
Deadline: Tuesday, April 18, 2017

1 An ever cooler reduction [7 points]

For this homework we will consider a simple assembler-like programs that processes finite sequences of natural numbers. Our programs will read an input from left to right, having access to a single number from the input. At each point the program can either store the number it is reading into some variable, compare the number it is reading with the value of some variable which was previously assigned, or simply skip this number and move to the next one. In the case that the comparison fails (or the variable is not assigned), the execution halts and the program rejects the input. For the other two operations the program simply updates the variables if needed, and moves to the next number. If we read the entire input (that is, if we manage to process the final number of the input), the sequence is accepted. Additionally, the language has a built-in `OR` operator allowing it to execute either one of two programs and accept all inputs accepted by any of the two programs.

Formally, our programs are built from the following three basic operations:

1. `skip` – when reading a number the program just moves to the next one;
2. `store->x` – when reading a number store it into the variable x. If some other number was stored into x it is now overwritten;
3. `equal(x)` – if the current number the program is reading equals to the one stored into the variable x proceed to the next number. Otherwise reject.

Any basic operation is a program, called a basic program.

Basic programs can now be combined using the following composition operations:

- *Combine* – if `p` and `q` are programs, then `p/q` is a program that first executes `p` and then executes `q`. This is analogous to ordinary programming languages where you write a bunch of commands that execute one after another. Similarly as in an ordinary programming language, if the subprogram `p` was running over the input it will stop at some position (not necessarily the last), assign some variables (using `store->x`) and then pass those onto `q`. The subprogram `q` now starts where `p` stopped, but it has access to the values that `p` assigned to its variables. (Also see the examples below.)
- *Disjunction* `OR` – if `p` and `q` are programs, then `p OR q` is a program that executes either `p` or `q`. You can think of this as a non-deterministic choice as to which one to execute.

Brackets can be used freely in our programs to make disjunctions and combinations easier to read.

Examples:

1. The program `store->x/skip/equal(x)` accepts all sequences of precisely three numbers such that the first number in the sequence equals the last one; e.g. it will accept the sequence $[1, 4, 1]$, but not $[1, 2, 2]$, nor $[1, 1, 1, 1]$.
2. The program `store->x/skip/equal(x) OR skip` accepts all sequences of either a single number, or of precisely three numbers, where the first number in the sequence equals the third one. It will accept $[1]$ and $[1, 1, 1]$, but not $[1, 1, 1, 1]$.
3. The program `(store->x/skip OR skip/store->x)/equal(x)` accepts all sequences of precisely three numbers where either the first one equals the third one, or the second one equals the third one. This is done by allowing to store into `x` either the second or the firs number in the (sub)program inside the brackets, and then execute the final comparison. Now we accept both the sequence $[1, 4, 1]$ and $[1, 2, 2]$.

Consider now the problem:

$$\text{ASSEMBLER} = \{\langle p, w\rangle |\ p \text{ is a program and } w \text{ is a sequence of numbers accepted by } p\}.$$

- Show that ASSEMBLER is NP-hard (5.5 points).
- Argue why the problem is NP-complete. For this you do not need a very formal proof, but you need to be able to explain why is the non-deterministic guess bounded, where you assume that reading/storing/comparing a number takes one step (1 point).
- Show that when the `OR` operator is not allowed, the problem is in PTIME. (0.5 pt).

Hint: As promised there are no hints. Or are there?