CSE 105: Automata and Computability Theory **Fall 2015**

Homework #4

Due: Wednesday, November 11th, 2015, 11:59 PM

Problem 1: FST design

Design an FST that on input a sequence of numbers, increments each number by 1. Numbers are encoded as follows:

- numbers are written in binary, with the digits listed in *reverse* order, starting with the least significan digit. E.g., 6 is encoded by the binary string "011" (possibly followed by any number of 0s.)
- The empty string is a valid representation of the number 0.
- Each number is terminated by a special character "x"

For example, an input string representing the sequence of numbers 0,2,0,9 is "0x01xx10010x". (Notice that numbers admit multiple representations, possibly containing trailing zeros.) Your transformation, beside incrementing each number by one, should satisfy the following properties:

- The string length of each number should not decrease, e.g., "1010x" (5) should map to "0110x" and not "011", even if the latter is also a valid binary representation of the number 6.
- The length should increase only if necessary, e.g., both "111x" and "1110x" should map to "0001x".
- Output symbols should be produced as soon as they can be determined from the input. So, for example, on input "11", the transformation should output "00", without waiting for the input number to be terminated by the special character "x".

Implement your FST in JFLAP and submit it as `HW41.jff` using bundleHW4. You should follow the definition of FST in the lecture notes, which can be implemented in JFLAP as a "Moore Machine", where nodes are labeled with strings (over the alphabet $\{0, 1, x\}$) and each node has three outgoings edges labeled $0, 1$ and x.

Once you save the file, you can test it by running it on some test strings with the command

```
runhaskell TestFST.hs HW41.jff 0x001x011xx1011x
```

The output should be the string `1x101x111x1x0111x`. If you get a different string, your FST is not correct. If you get an error message, then, most probably, your JFLAP file is not even a syntactically valid FST. (Check in JFLAP for missing transitions, nondeterminism, valild labels, etc.)

Problem 2: Formal Verification

Now consider the problem of incrementing the numbers by 2. This can be achieved by running the FST `HW41.jff` from problem 1 twice. Using the closure property of FSTs under function composition (described in the lecture notes, and implemented by the function `composeFST` in `FST.hs`) this gives an FST that computes the function $HW41 \circ HW41$, and increments each input number by 2 as required. You can build this FST running the program `compose.hs` provided with the starter files. This program takes an FST as input, and composes it with itself. Apply the program to your solution to proglem 1 by running the command

```
runhaskell compose.hs <HW41.jff >HW41twice.jff
```

and load the resulting FST `HW41twice.jff` in JFLAP. The FST is correct by construction, but you will notice it uses a very large number of states! You may try to simplify `HW41twice` by eliminating the some states that cannot be reached from the start state of the FST, but the result will still have a large number of states. You may think of `HW41twice` as a specification, or a fast prototype solution, correct by construction, but inefficient.

A better FST to add 2 to every number is provided by the FST `plus2FST.jff` included in the starter files. You can inspect `plus2FST.jff` in JFLAP, and you will see that it uses a much smaller number of states than $HW41 \circ HW41$. But, how can we be sure it is correct?

The goal of this problem is to develop an **automatic verification procedure** to check that two FSTs compute the same function. At the end of the problem, you will be able to check the correctness of `plus2FST.jff` by verifying that it computes the same function as `HW41twice.jff`.

The idea is the following:

- For any two FSTs T1 and T2 (over the same alphabet Σ), consider the language $L = \Delta(T1, T2)$ consisting of all input strings $w \in \Sigma^*$ such that $T1(w) \neq T2(w)$. The two FSTs are equivalent if and only if this set L is empty.

- Check that the language L is empty. For example, if you could build a DFA or NFA M for the language L, you could test if L is empty by checking that there is no path from the start state of M to a final state of M. (A program to test if a regular language is empty is given by the function `isEmptyNFA` in `NFA.hs`.)

Unfortunately, the set $L = \Delta(T1, T2)$ is not always regular (see next problem), so you cannot build a DFA for it in general. In order to address this difficulty, we consider a slighly different language $L' = \Delta(T1, T2) \cdot \Sigma^*$. Notice that $\Delta(T1, T2) \cdot \Sigma^*$ is empty if and only if $\Delta(T1, T2)$ is empty. So, we can still test that the two FSTs are equivalent by checking if L' is empty. Moreover, the language L' is regular and you can build a DFA for it.

Claim: For any two FSTs $T1, T2$ over the same alphabet Σ, the set $\Delta(T1, T2) \cdot \Sigma^*$ is regular.

Your task: Prove the claim by giving a transformation that on input any two FSTs $T1, T2$ over the same alphabet Σ, produces a DFA for the language $\Delta(T1, T2) \cdot \Sigma^*$. Implement the construction in haskell starting from the file `HW42.hs`. Submit the modified `HW42.hs` as your solution, and a brief description `HW42.pdf` explaining your construction.

You can check your solutions to problem 1 and 2 by running the program `TestHW42.hs` provided with the starter files. Inspect the program to see what it does, and then run it with the command "`runhaskell TestHW42.hs`" from the same directory containing your solutions and the automata library files.

Problem 3: Non-closure of regular languages/functions under diff

For any two functions $T1, T2 \colon \Sigma^* \to \Gamma^*$, let $\Delta(T1, T2)$ consisting of all input strings $w \in \Sigma^*$ such that $T1(w) \neq T2(w)$. Prove that the set of FST computable functions and regular languages are not closed under this operation, i.e., show that there are two FSTs T1 and T2 such that the language $L = \{w \mid T1(w) \neq T2(w)\}$ is not regular. Your solution should consists of a description of the FSTs T1 and T2, a description of the set $\Delta(T1, T2)$ using set builder notation, and a proof that the set $\Delta(T1, T2)$ is not regular.

You do not need to build T1, T2 in JFLAP or include state transition diagrams. A description of the functions computed by T1, T2, and an explaination of why these functions are FST computable is enough. Submit your solution as `HW43.pdf`.

Problem 4: Context Free Grammars

Give context free grammars for the following languages:

(a) The set of strings $w \in \{0, 1\}^*$ containing exactly the same number of 0s as 1s.

(b) The set of strings $w \in \{0, 1\}^*$ containing strictly more 0s than 1s

(c) The complement of (a), i.e., the set of strings $w \in \{0, 1\}^*$ with a different number of 0s and 1s

(d) The set of strings $w \in \{0, 1\}^*$ containing exactly twice as many 0s as 1s.

In all cases, the symbols can appear in any order, e.g., 0101011001, and not just 0000011111 or 1111100000 Submit your solutions in a file `HW44.pdf`.