

Homework #4

Due: Tuesday, February 17th, 2015, 11:59 PM

Problem 1: FST ReductionsConsider the following languages over $\Sigma = \{0, 1\}$:

$$\begin{aligned} A &= \{w \in \{0, 1\}^* : \text{the length of } w \text{ is a multiple of } 3\} \\ B &= \{1^{5n} : n \geq 0\} \cup \{0^{5n} : n \geq 0\} \\ C &= \{0^n 1^n : n \geq 0\} \\ D &= \{w : w \text{ has the same number of 0s and 1s}\} \end{aligned}$$

This problem has 4 parts:

- (a) Prove that $A \leq_{FST} B$ by giving an FST reduction from A to B .
- (b) Prove that $C \leq_{FST} D$ by giving an FST reduction from C to D .
- (c) Prove that $A \leq_{FST} D$ by giving an FST reduction from A to D .

For each part, draw the state transition diagram of the reduction (FST) in JFLAP, and submit your answer as file HW41a.jff, HW41b.jff, HW41c.jff. In order to draw an FST (as defined in class) in JFLAP you should select “Mealy Machine” when creating a new file. Remember to select a start state! Following the definition used in class, each transition of your FST should read one input symbol, and output a string of 0 or more symbols. (The empty string in JFLAP is displayed as λ .) You can also use JFLAP to test your automaton.

Then, do also the following:

- (d) Give an FST T and a regular language L such that either $f_T(L) = C$ or $f_T(C) = L$. (You have to guess the correct order.)

For the last part, draw the FST T and DFA M recognizing $\mathcal{L}(M) = L$, and submit them as HW41dT.jff and HW41dM.jff.

Problem 2: RLL Codes

Run Length Limited (RLL) codes are a coding technique used to avoid errors when transmitting information over a serial line or storing data on media like CD, DVD and Blu-ray discs, and other similar applications. Here we consider a simple RLL code $C \subseteq \{0, 1\}^*$ consisting of all $w \in \{0, 1\}^*$ such that all stretches of consecutive zeros in w have length at most 3. So, for example $001000111010 \in C$, but $010010100001 \notin C$. This constraint is useful to avoid synchronization errors,

where, e.g., a sequence of n zeros may erroneously read as $n + 1$ zeros. (You can read the RLL page on wikipedia http://en.wikipedia.org/wiki/Run-length_limited to learn more about the topic and why/how these codes are used.)

In this problem (and the next) you will design (and verify) efficient encoders and decodes for our simple RLL code C . Specifically, the goal is to come up with an encoding function $\text{ENCODE}: \{0,1\}^* \rightarrow \{0,1\}^*$ and a corresponding decoding function $\text{DECODE}: \{0,1\}^* \rightarrow \{0,1\}^*$, both implemented as FSTs, that can be used to map any input $w \in \{0,1\}^*$ to an encoding $\text{ENCODE}(w) \in C$ satisfying the RLL constraint, and then recover the original input from the encoding, i.e., $\text{DECODE}(\text{ENCODE}(w)) = w$.

- (a) Give an FST for the encoding function ENCODE . (There are many ways to define this function. You can solve the problem any way you want, as long as it satisfies the desired properties. But encoders that do not stretch the length of the input by much are preferable in order to conserve bandwidth.)
- (b) Give an FST for the corresponding decoding function DECODE .
- (c) Give a DFA accepting the complement of C , i.e., all strings containing 4 consecutive 0s. This will be used to verify the correctness of the encoder.

Verify that the encoding function does not violate the RLL constraint using the closure properties of FST and DFA (as proved in class, and covered in the lecture notes and haskell implementation) to build a DFA that accepts all inputs w such that your encoder from part (a) outputs a string which is accepted by the DFA in part (c). Complete the file `HW42.hs` by defining an appropriate testing function. Then, validate your encoder using the testing program `HW42test.hs`. If everything is correct, it will output `True`.

Your solution should consist of four files `HW42a.jff`, `HW42b.jff`, `HW42c.jff`, `HW42.hs` containing the FST encoder, FST decoder, DFA M from part (c), and testing function.

Problem 3: Formal Verification

In the previous problem (part (c)) you verified that your encoder is safe, in the sense that it never produces strings that violate the RLL property. For the encoding/decoding process to be correct, we also need to check that running the decoder on the output of the encoder, retrieves the original input string, i.e., composing the encoder and decoder gives the identity function $\text{DECODE}(\text{ENCODE}(x)) = x$. We have already proved that FSTs are closed under function composition, i.e., given FSTs ENCODE and DECODE , we can build a new FST $T_1(x) = \text{DECODE}(\text{ENCODE}(x))$. In order to check this is the identity function, we need a way to compare two FSTs, say $T_1(x)$ and the FST computing the identity function $T_0(x) = x$.

Prove the following closure property:

Closure Property: For any two FST-computable functions $f_0, f_1: \Sigma^* \rightarrow \Gamma^*$, the set of inputs $\{x \in \Sigma^*: f_0(x) \neq f_1(x)\}$ where the two functions take different values is a regular language.

You should prove the property by giving a transformation that on input any two FSTs T_0, T_1 , produces a DFA for the language described in the closure property. Implement the construction in haskell starting from the file `HW43.hs`, and submit it as part of your solution. You can

test your encoder/decoder functions and implementation of the closure property using the test program `HW43test.hs` provided on the course web page. Calling `HW43test fst1.jff fst2.jff fst3.jff` prints True if `fst2(fst1(w)) = fst3(w)` for all inputs strings w . You can test your encoder/decoder by creating an FST `id.jff` that computes the identity function, and then call `HW43test HW42a.jff HW42b.jff id.jff`.

Submit `HW43.hs` as your solution, and a brief description `HW43.pdf` explaining your construction.

Problem 4

For each of the languages listed below, explain why the proposed choice of string w will *not* lead to a convincing pumping-lemma proof that the language isn't regular. In each case, the alphabet is $\Sigma = \{0, 1\}$.

We also use the following notation: For an integer $n \geq 0$, let $n_{\beta=2}$ be the string that is the big-endian binary representation of n . For example, $0_{\beta=2} = 0$, $6_{\beta=2} = 110$, and $105_{\beta=2} = 1101001$.

- (a) With the notation $n_{\beta=2}$ from above, let L be the language $\{k_{\beta=2} \mid k \text{ is prime}\} = \{10, 11, 101, 111, \dots\}$.

Given $p > 0$ by the adversary, let n be the smallest prime strictly larger than p , and set $w = n_{\beta=2}$.

- (b) Let L be the same language as in part (a).

Given $p > 0$ by the adversary, set $w = 0^{\lceil p/2 \rceil} 10^{\lceil p/2 \rceil}$.

- (c) Let L be the language $\{w \mid w \text{ contains at least as many 0s as 1s}\}$.

Given $p > 0$ by the adversary, set $w = 0^p 1^p$.

- (d) With the notation $n_{\beta=2}$ from above, let L be the language $\{k_{\beta=2} \mid k \text{ is power of } 2\} = \{1, 10, 100, \dots\}$.

Given $p > 0$ by the adversary, set $n = 2^p$ and $w = n_{\beta=2} = 10^p$.

Submit your solutions as `HW44.pdf`.

Problem 5

Consider the two grammars G and G' , with respective start variables S and S' , and defined respectively by the rules on the left and right hand sides:

$$\begin{array}{ll}
 S \rightarrow \mathbf{aSc} \mid T & S' \rightarrow \mathbf{aS'c} \mid A'T' \mid T'B' \mid T'C' \\
 T \rightarrow \mathbf{aTb} \mid \varepsilon & T' \rightarrow \mathbf{aT'b} \mid \varepsilon \\
 & A' \rightarrow \mathbf{aA'} \mid \mathbf{a} \\
 & B' \rightarrow \mathbf{bB'} \mid \mathbf{b} \\
 & C' \rightarrow \mathbf{cC'} \mid \mathbf{c}
 \end{array}$$

- (a) For each of the following strings, state whether the string is generated by G and whether it is generated by G' :

ε ac $aabc$ $abbcc$ $bccab$

If a string is generated by one of the grammars, give a derivation showing how; if a string is not generated by one of the grammars, explain why not.

- (b) What is the language generated by the grammar G ? By the grammar G' ?

Express both of your answers in set-builder notation, with as simple a property as you can find for the included strings.

Submit your solutions as `HW45.pdf`.