2nd Lt David Crow

ENG/20M

CSCE 532 Homework 2

Problem 1.37

We know is regular because we can create a finite state machine that recognizes . Because we want to check whether a binary number is divisible by , our FSM needs states (one for each remainder when dividing some number by ). Because the input string is a binary number, as long as we read from the most-significant to the least-significant bit, we know that, when we see a , our current remainder is multiplied by (that is, becomes ). When we see a , becomes .

We can then evaluate this new by transitioning to the state that corresponds to . Of course, we accept a string when we end in the state that corresponds to a remainder of .

Because we can always create an FSM (and its associated transition function) that can recognize this language (for all ), we’ve shown that this language is regular.

Problem 1.7c

:



I recognize that making the start state an accept state is redundant, but that’s okay.

Problem 1.31

If we assume that is a regular language, we know by definition that there must be some finite state machine that recognizes . We can *reverse* this FSM to create one that accepts . To reverse an FSM that recognizes , we should conduct the following procedure:

1. Reverse the direction of all arrows
2. Create a new start state and draw -transitions from to every accept state
3. Change all accept states into normal states
4. Change the original start state to an accept state

We must create an NFA (as opposed to a DFA) because an FSM that recognizes might have multiple accept states, and we must be able to begin analyzing from any of ’s accept states. In other words, we must be able to start in any of the original accept states (which are now start states) to allow for reversal.

Problem 1.33

We are assuming that, if a language is regular, then the reverse of the language is also regular. Under this assumption, we can show that is regular by showing that is regular. Here, , of course, is the language where we receive the least-significant element first and the most-significant element last. We are doing this because binary multiplication is performed from right to left.

We must take note of a few things before we can create an FSM to recognize . When multiplying a binary number by , we must consider bits carried into the next (i.e. to the left) column. Additionally, when multiplying by , we can only carry a value of , , or . We thus need only three states (one for each carry value). Computing subsequent carry values – when we know the current carry value and the incoming top-row bit – is simple math, and we only need to ensure the bottom-row value is expected. Here’s the FSM:



Because we’ve created an FSM that recognizes , we’ve shown that is regular. Thus, is also regular.

Problem 1.46c

Let be the language . Assume the complement of – that is, – is regular. Let be the pumping length given by the pumping lemma. Additionally, let . Because and , the pumping lemma tells us that can be split into three pieces such that , where, for any , the string is in . Because the pumping lemma tells us that , can only consist of s. If , then . It is clear that is not in . This is a contradiction of our original assumption, and thus is not regular. We know that regular languages are closed under complement, so we can conclude that, because , .