Computability and Logic

HW 8

Due: Friday, May 1

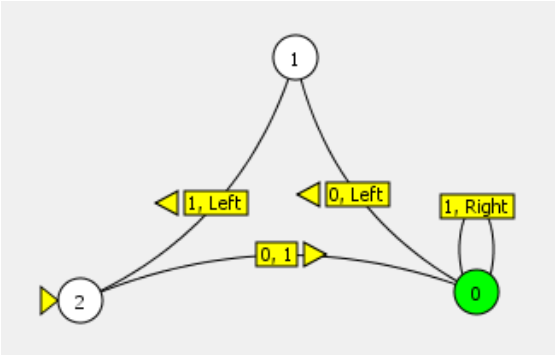
1. (see discussion of bounded minimization Min[R] in slides) Prove that the bounded maximization function Max[R] is recursive for any recursive relation R. Show that your definition works.
2. In the proof that all Turing-computable functions are recursive, we made several simplifying assumptions:

    - The function to be computed has only 1 argument
    - The Turing-machine starts in state 1
    - The Turing-machine has transitions defined for every symbol for every state other than 0, and no transitions defined for state 0 (so, whenever it does halt, it halts in state 0).

    Let's see if and how we can loosen these assumptions while retaining the result that all Turing-computable functions are recursive.

    a. Where and how does the proof need to be modified to deal with functions that have more than 1 argument? Be specific.
    b. Show that if a function is computable using a Turing-machine that starts in a state other than 1, then it is computable by a Turing-machine that starts in state 1.
    c. Show that if a function is computable using a Turing-machine for which it is not true that it has transitions defined for every symbol for every state other than 0, and no transitions defined for state 0, then it is computable by a Turing-machine that does have transitions defined for every symbol for every state other than 0, and no transitions defined for state 0.

    For b and c, show how you would transform the following Turing-machine that does not satisfy the simplifying assumptions into a functionally equivalent Turing-machine that does, in a way that makes it clear that this transformation process can be done for any Turing-Machine.



3. We saw that having a 2-sided infinite tape does not give the Turing-machine any more computational power than a one-sided infinite tape Turing Machine has. What would be the smartest/quickest way to prove that a Turing-machine does not gain any power either by using a 2-dimensional representational system (i.e. some kind of machine that is able to move around, reading and writing symbols on a 2-dimensional sheet with squares to place symbols in)? I am not looking for a detailed proof; just a high-level proof strategy for how you would try to show this.
4. Use the Turing-machine software to create a '2-shift' Turing-Machine: a Turing-machine that, when started on $[n_1, n_2, \ldots n_k]$ for some unknown k, shifts everything 2 places to the right without ever moving to the left of the starting square. Your head should end up on the leftmost 1 (i.e. 2 squares to the right of the square where it started). Make sure to use the quadruple definition.
5. Show that the last, ext, conc, pre, and sub cryptographic functions used in the arithmetization of FOL are recursive.