Luke Jaffe
Theory of Computation
Due 03/23/16

Problem Set 7

Problem 1.
a.) Consider the function f(<M,w>) that, given a Turing Machine description M and an input w, returns the number k of the farthest number of tape square that is reached by the Turing Machine during its operation, or -1 if the number of squares reached by the Turing Machine is unbounded. Prove this function is not computable by showing that if it were, an undecidable problem could be decided.

If we could decide this problem, then we could decide the halting problem. Let us construct a Turing Machine A which computes f(<M,w>). Then, let us construct a Turing Machine B which takes as input <A,M,w>. We will show how machine B solves the halting problem. This machine runs A on <M,w>, rejecting on output -1, and accepting on output > -1. B accepts on output > -1 because if we know the number k of the farthest tape square that is reached on M, then we know that M halts on w. This is because the space-limited halting problem is decidable. B rejects on output = -1 because if the number of squares reached by M is unbounded, then M does not halt on w. This is because we can never reach a halting state if the machine progresses indefinitely, i.e. "reaches" an unbounded number of squares.

b.) Interpret the above result in the context of real (and Turing-complete) programs. What useful bound is not computable in general?

The amount of memory a computer program will use during its execution cannot be computed.

Problem 2.
Prove that the following language is undecidable: {<M,w>: at some point processing w, Turing Machine M takes the sequence of consecutive actions ←,→, ←,→, write 'b', write 'a'}. You may assume an arbitrarily large alphabet.

Let M_C be the Turing Machine which decides {<M,w>: at some point processing w, Turing Machine M takes the sequence of consecutive actions ←,→, ←,→, write 'b', write 'a'}. We can use M_C to decide A_{TM} = {<M,w> : M is a TM accepting w}.

Let us construct a new TM, M_{ATM} as follows: On input <M,w'> create a Turing Machine M_1 which accepts (only) on input w after performing the sequence of consecutive actions ←,→, ←,→, write 'b', write 'a', and otherwise rejects. Then, M_1 runs M.

Now, run M_C on M_1. TM_{ATM} accepts if M_C accepts, and rejects if M_C rejects. If M_C accepted, $L(M_1)$ = {w}, so M accepts w. If M_C rejected, $L(M_1)$!= w, so M rejects w.

M_{ATM} decides A_{TM}, which is undecidable; therefore the given language is undecidable.

Problem 3.
Identify which of the following languages Rice's Theorem declares undecidable. If it is undecidable, briefly explain how it satisfies both properties necessary for Rice's Theorem to hold. If it is not, identify which property does not hold.

a.) {<M>: M is a Turing Machine and L(M) is deterministic context-free}

Rice's Theorem declares this language undecidable. First, “M is a Turing Machine and L(M) is deterministic context-free” is a nontrivial property. That is because there are Turing Machines which recognize a language with this property, but not all Turing Machines recognize a language with this property. For example, there exist Turing Machines which recognize non-deterministic context-free languages. This property is clearly a property of the language, because it specifically describes a characteristic of the language. That is to say, if two TMs have the same language, both have this property or neither do.

b.) {<M>: M is a Turing Machine and L(M) U L(M) = Σ*}

Rice's Theorem declares this language undecidable. The property can be simplified to M is a Turing Machine and L(M) = Σ*, because L(M) U L(M) = L(M). The property is non-trivial, because there are Turing Machines which recognize Σ*, and Turing Machines which do not. For example, a TM with language {a, b, c} does not recognize Σ*. We know this is a property of the language, because the language of L(M) is explicity given as Σ*.

c.) {<M>: M is a Turing Machine and M accepts the input “foo”}

Rice's Theorem declares this language undecidable. First, “M is a Turing Machine and M accepts the input “foo”” is a nontrivial property. That is because there are Turing Machines which recognize a language containing “foo”, but not all Turing Machines recognize a language containing “foo”. This property is clearly a property of the language, because it specifically discusses the contents of the language. Explicitly, if two TMs have the same language, those languages must both either contain “foo” or not contain “foo”.

d.) {<M>: M is a Turing Machine with no useless states (states reachable with no input)}

Rice's Theorem does **not** declare this language undecidable. That is because being “a Turing Machine with no useless states” is not a property of the language. That is to say, we could have two different TMs that recognize the same language, but one might exhibit this property, while the other does not. For example, two Turing Machines could both have the language {“bar”}, but one might have any number of states that it never reaches, while the other reaches all of its states.

Problem 4.
lg* n (“log star” or “iterated binary logarithm”) is the number of times that it is necessary to take the log2 (lg) of the input until the result is 1 or less. For example, $\lg^* 2^{65536} = 5$, because $\log_2 (2^{65536}) = 65536$, $\log_2 (65536) = 16$, $\log_2 16 = 4$, $\log_2 4 = 2$, $\log_2 2 = 1$. lg* n is a very slow-growing function, no more than 5 in practice. But show that because the Busy Beaver function Σ(n) grows faster than any computable function, lg* Σ(n) must grow faster than any computable function as well.

Structure of proof:
Suppose that lg* Σ(n) does not grow faster than some computable function. <what goes here?> Then Σ(n) also does not grow faster than some computable function. But we know that Σ(n) does grow faster than any computable function. Therefore we have a contradiction, proving lg* Σ(n) must grow faster than any computable function.

To get some credit despite unfinished proof:

The asymptotic complexity class of the busy beaver dwarfs that of the lg* function, such that is must grow faster.

Problem 5.
A language is prefix-free if there are no two strings in the language such that one is a prefix of the other (w and wx are both in the language for $w \in \Sigma^*$, $x \in \Sigma^+$). Reduce from the Post Correspondence Problem to prove the undecidability of {<G>: G is a CFG and L(G) is prefix-free}. (Hints: Find a proof to imitate; and be sure to argue the "iff" that shows your construction is correct.)

We need to describe a CFG which is prefix-free for all legal orderings of tops and bottoms, which also solves the PCP. This also implies that for any legal ordering of tops and bottoms, the two strings produced are the same. Since one string must be strictly shorter than another to count as a prefix, this is not a problem. For any PCP problem, we can construct our CFG as follows:

$S \rightarrow T \mid B$

$T \rightarrow t_0\varepsilon, t_1 0T, \ t_2 12T, \ldots, t_n(n-1)nT$

$B \rightarrow b_0\varepsilon, b_1 01B, b_2 23B, \ldots, b_n nB$

With such a CFG, the only way to way to guarantee that all strings produced by the grammar are prefix-free is to solve the PCP. In fact, the dominoes must be placed in order starting from domino 1, and the sequence must end with domino 0. Using any other ordering, the PCP is not solved. To show the iff part: if any other ordering is used, a string will be added to the language which is a prefix of another string in the language. This is guaranteed by the non-terminal at the end of all rules except for the empty rule. Therefore, the language is prefix-free iff the PCP is solved with the dominoes.