CSE 547T Automata Theory March 21, 2017

Solutions for Homework 3

1. **(a)** This property is undecidable. While we can use Rice's Theorem to make this argument, we will instead prove it directly by a reduction.

   Suppose we have a TM M_R that decides regularity. We will reduce from NOT-ACC, i.e. build a TM M_{NACC} to decide NOT-ACC.

   On input x, M_{ACC} first verifies that x encodes a pair (M, w) (and rejects otherwise). It then constructs a TM M' that behaves as follows.

   On input y, M' runs M on w. If M rejects w, then M' rejects y; if M accepts w, then M' accepts y iff it has the form 0^n1^n for some $n \geq 0$, rejecting otherwise.

   Finally, M_{NACC} passes M' to M_R and returns the latter's answer.

   Claim: $(M, w) \in$ NOT-ACC iff $L(M')$ is regular.

   Pf: If M accepts w, so that (M, w) is not in NOT-ACC, then M' accepts a language that we have previously proven to be non-regular. Otherwise, if M fails to accept w, either by rejecting or by running forever, then M' runs forever; in this case, its language is $\emptyset$, which we know to be regular. Hence, the claim is proven.

   Conclude that M_R cannot exist, and so its language is not decidable.

   (b) This property of TMs is in fact decidable. Given a TM $M = (Q, \Sigma, \Gamma, q_0, \delta)$ and a constant k, whether M ever moves past its kth cells is determined entirely by the initial contents of these first k cells. The first cell is by definition initially Δ, so there are only $|\Gamma|^{k-1}$ possible sets of contents to check.

   For each possible string $\gamma \in \Gamma^{k-1}$ initially labeling cells $2..k$ of M's tape, we start M with tape contents $\Delta\gamma$ and simulate its execution. During the simulation, we keep a count of the number of moves performed by M so far.

   Observe that there are only $N = (|Q| + 2) \times |\Gamma|^k \times k$ distinct configurations (state, head position, and tape contents) of M if it never passes the kth tape cell. Hence, if the TM makes at least N moves without halting or passing this cell, it must (by the pigenonhole principle) have repeated a configuration and so is in an infinite loop.

   For a given γ, we have three possibilities:

   - M moves right past the kth cell in $< N$ moves.
   - M halts in $< N$ moves without passing the kth cell.
   - M makes N moves without halting or passing the kth cell; in this case, M will never halt or pass this cell.

   In the first case, we accept; in the other two, we discard the current γ and go on to the next one. If no γ causes us to accept, we reject.

   (c) This problem is undecidable. We will reduce from EMPTY. For convenience, let $M_\emptyset$ be a TM that trivially rejects all its inputs and so has an empty language.

   Suppose M_{EQ} is a TM deciding equality. Construct a TM M_{EMPTY} deciding emptiness as follows. M_{EMPTY} first tests that its input has the form of a TM M and rejects if not. It then constructs the encoded pair $(e(M), e(M_\emptyset))$, passes this pair to M_{EQ}, and does what it does.

Claim: $e(M) \in M_{EMPTY}$ iff $L(M) = L(M_\emptyset)$. The proof is immediate by definition of EMPTY. Conclude that M_{EQ} cannot exist, and so equality of TMs is undecidable.

2. **(a)** We claim that this language is decidable. Given a TM M and string w, we can simulate the operation of M on w while counting its moves.

   - If M ever moves its head left, reject.
   - If M accepts or rejects w without moving its head left, accept.
   - If M ever goes $|Q| \times |\Gamma|$ steps without moving its head, we know that it repeats a configuration and will therefore loop forever without moving its head. In this case, accept.

   Otherwise, we notice when M first moves its head past the end of w. At this point, its head is over a blank cell Δ, and it is in some state q. Each subsequent time that M moves its head right, it will be over a blank and in some state q'. If immediately after a rightward move, the same state q' is repeated, then the TM is in an infinite loop and so will run forever (without moving left), and we should accept. Such a repeat must occur after at most $|Q|$ rightward moves, or (if the machine does not get into an infinite loop without moving its head) at most $|Q|^2 \times |\Gamma|$ total steps.

   (b) We claim that the set of languages accepted by rightward-bound Turing machines is exactly the regular languages. We will provide constructions for both directions, i.e. FA to TM and TM to FA, along with informal justifications of why they work.

   ($\rightarrow$) Given a DFA $N = (Q, \Sigma, q_0, A, \delta)$, we can construct an equivalent TM $M = (Q', \Sigma, \Gamma', q_0', \delta')$ that never moves left as follows.

   - Let $Q' = Q \cup \{q^*\}$.
   - Let $\Gamma' = \Sigma \cup \Delta$.
   - Let $q_0' = q^*$.
   - Initially, we have that $\delta'(q^*, \Delta) = (q_0, \Delta, R)$.
   - If for $a \in \Sigma$, $\delta(q, a) = p$, then define $\delta'(q, a) = (p, a, R)$.
   - If $q \in A$, then define $\delta'(q, \Delta) = (h_a, \Delta, S)$. Otherwise, for $q \neq q^*$, define $\delta'(q, \Delta) = (h_r, \Delta, S)$.

   Informally, the TM M simulates the DFA N as follows. The TM's goal is to mimic the state transitions of the DFA exactly on a given string. Each rightward move of the TM except the first corresponds to one transition in the TM; the first move simply gets the TM's head over the input. Once the TM moves past the input (i.e. sees a trailing blank), it immediately accepts or rejects depending on whether or not it was left in an accepting state of the NFA.

   ($\leftarrow$) Given a TM $M = (Q, \Sigma, \Gamma, q_0, \delta)$, we first simplify M in several stages, then finally convert our simplified TM to an NFA. (Note that there are other ways to do this that do not involve simplifying the TM.)

   Let (q, a), for $q \in Q, a \in A$, be called a *situation* of the TM M. In situation (q, a), the machine is in state q, and the character under its head is a. We first observe that, given a starting situation (q, a), M eventually does one of four things:

   - It moves its head right in some state q'.
   - It transitions to h_a.
   - It transitions to h_r.
   - It runs forever without moving its head.

Moreover, we can compute which of these outcomes occurs by simply iterating the TM's move function, starting from situation (q, a), until the head moves right (option 1), the machine halts (options 2 or 3), or we have made $|Q| \times |\Gamma|$ moves without moving the head, in which case we have repeated a situation and so will loop forever in place (option 4). Let $\gamma(q, a)$ be a function that is q', h_a, or h_r, depending on whether the outcome is option 1, option 2, or either of options 3 or 4.

We can use γ to further compute what happens when M encounters the end of the input. By iterating the function γ starting from situation (q, Δ), we compute the behavior of M assuming it reaches the end of the input while in state q. Again, there are a limited number of outcomes:

- M eventually transitions to h_a.
- M eventually transitions to h_r.
- M eventually loops forever at a particular cell.
- M keeps moving its head to the right forever.

The first three of these possibilities are detected if iterating the function γ starting from (q, Δ) eventually yields h_a or h_r, while (as observed in part (a)) the fourth occurs if we iterate $|Q|$ times without seeing any of these outcomes (since we must then arrive at a blank cell in a state that has occurred before and so are in a cycle of situations). Define the function $\omega(q)$ to be h_a if iterating γ eventually yields h_a, or h_r otherwise.

We now use our γ and ω functions to radically simplify the behavior of M by eliminating both "S" moves and any operation past the end of the input. Define a new TM $M' = (Q', \Sigma, \Gamma', q_0', \delta')$ as follows.

- $Q' = Q \cup \{q_0'\}$.
- $\Gamma' = \Sigma \cup \{\Delta\}$.
- $\delta'(q_0', \Delta) = (\gamma(q_0, \Delta), \Delta, R)$.
  (If $\gamma(q_0, \Delta)$ happens to be h_a or h_r, we can trivially simulate M by a DFA that accepts Σ^* or $\emptyset$.)
- For $a \in \Sigma$, $\delta'(q, a) = (\gamma(q, a), a, R)$.
- For $q \neq q_0'$, $\delta'(q, \Delta) = (\omega(q), \Delta, S)$.

Intutively, the TM M' makes an initial rightward move onto the input string, then moves rightward one cell at each step until it either accepts, rejects, or reaches the blank at the end of the input (at which point it immediately accepts or rejects). Our definitions of functions γ and ω ensure that the state changes of M' on each rightward move match what would occur in M, and that the behavior at the end of the input matches what M would do.

We make one further simplification of M' to create a machine $M^* = (Q^*, \Sigma, \Gamma', q_0', \delta^*)$ to avoid having to simulate accepance by the TM before reading the entire input string.

- $Q^* = Q' \cup \{q^y\}$.
- For $a \in \Sigma$, $\delta^*(q^y, a) = (q^y, a, R)$.
- $\delta^*(q^y, \Delta) = (h_a, \Delta, S)$.
- If for $a \in \Sigma$, $\delta'(q, a) = (h_a, a, R)$, then we define

$$\delta^*(q, a) = (q^y, a, R).$$

- δ^* otherwise includes the same moves as δ'.

M^* modifies M' so that, if the M' would transition to state h_a while its head is still over the input, M^* instead transitions to state q^y, moves the head to the first blank after the end of input, and *then* accepts.

Finally, we describe the translation of M^* to an NFA $N = (Q_N, \Sigma, q_0^N, A_N, \delta_N)$.

- $Q_N = Q^* \cup \{h_a, h_r\}$.
- $q_0^N = \delta^*(q_0', \Delta)$.
- For $a \in \Sigma$, if $\delta^*(q, a) = (p, a, R)$, then $\delta_N(q, a) = \{p\}$.
- For each $q \in Q$, if $\delta^*(q, \Delta) = (h_a, \Delta, S)$, then $\delta_N(q, \varepsilon) = \{h_a\}$.
- $A_N = \{h_a\}$.

Intuitively, the NFA N simulates the behavior of the TM M^* move-for-move. The initial ε-move from q_0^N matches the TM's move onto its first input character, while subsequent moves that consume a character exactly match the TM's move function.

The only subtlety is how we define the accepting state of N and its associated transitions. By construction, M^* can accept only at the end of its input. The NFA N nondeterministically guesses when it has reached the end of its input and, if the input string seen so far would be accepted by the TM, transitions by a ε move to an accepting state h_a (this is the fourth bullet point above). However, there is *no* self-loop on h_a, so if more input is read after the NFA moves to h_a, it crashes (and hence does *not* accept). If M^* moves to state h_r (which it may do at any point, not just the end of input), so does N, at which point it either crashes on the next input or (if at end of input) stays in h_r and so does not accept.

Each step in our construction preserves the language of the machine in the previous step, so $L(N) = L(M^*) = L(M') = L(M)$.

3. Let L be an infinite RE language. Then there exists an enumerator TM M that enumerates the strings of L in some order. We will construct a TM M' that enumerates an infinite subset $L' \subseteq L$ in *canonical* order, which implies that L' is recursive.

   The TM M' internally contains a tape on which to simulate M, plus two more tapes: an *output tape* (since it is an enumerator) and a *storage tape* containing a unary integer (initially $0^0 = \varepsilon$). M' internally simulates M and observes the contents of its output tape. Each time M writes a new string x to its output, M' compares the length of x to the value 0^i on its storage tape. If $|x| > i$, then M' writes $0^{|x|}$ to its storage tape and appends x to its output tape; otherwise, both storage and output tapes are unchanged.

   Because L is infinite, we observe that, after running the enumerator M for any number of steps k, it will take only finitely many more steps to emit a string $x \in L$ of length greater than any seen so far. Hence, M' never finishes, and so it enumerates an infinite language. Moreover, the strings enumerated by M' are emitted in strictly increasing order by size and hence are in canonical order with no repetitions. Finally, note that the head for the output tape of M' need never backtrack or overwrite an already written cell, so M' is a proper enumerator. We conclude that M' is a canonical-order enumerator for its language $L' = L(M')$, and so L' is an infinite recursive language.

4. First, let's simplify the description of the problem. A tile can be uniquely described by the (signed) difference between the number of 0's on its top and bottom strings. Hence, we describe tile $[\alpha \mid \beta]$ by the value $|\alpha| - |\beta|$. The unary PCP problem can now be rephrased as follows:

   > Given a set of signed integers $v_1 \ldots v_n$, can we add these integers together (perhaps with repetition) so as to sum to 0?

Clearly, if any integer v_i is itself 0, then we can make 0. Moreover, if the input includes integers a and $-b$, where $a, b > 0$, then we can make 0 by adding b copies of a and a copies of b, obtaining $ab - ab = 0$. Finally, if all the v_i's are non-zero and of the same sign, we cannot make zero. Hence, we can easily decide whether an instance of unary PCP is solvable.

5. We first consider how to represent the memory of the RAM on the TM, then describe how to implement its operations.

   Our TM has a *memory tape* that represents the contents of the RAM's memory. The tape contains one or more memory cells, each containing a signed number (in the base of your choice); each cell is followed by a separator character #. At any time, the number of cells is equal to the highest address ever read or written by the RAM so far.

   To seek to cell a, we reset the memory tape's head to the left end, then scan to the right until we have passed a separators. (A separate address tape can be used to keep a count of separators seen while seeking.) If we encounter a blank on the tape, then we write "0#" as often as needed until the tape contains $a + 1$ memory cells.

   To write a number n into memory cell a, we seek to its beginning as described above, then erase the number present in the cell and replace it with n. The cell's length may need to be adjusted if n is not the same length as the number it replaces; in this case, we move the non-blank contents of the tape following cell a to the right to accommodate n if it is longer, or move these contents to the left if n is shorter. As we saw for the universal TM, moving a finite amount of tape contents left or right to resize a cell can easily be implemented with marking, finite memory, and an extra counter tape to track how far to move the contents.

   We now consider the overall TM simulation of a RAM with a fixed program. The TM maintains its input tape, its memory tape, a "program counter" (PC) tape, and a finite number of auxiliary tapes as needed to perform arithmetic on numbers. Initially, the input tape contains the input value (a number), the memory tape is empty, and the PC tape contains 0. The TM first copies its input value to RAM address 0 as described above, then begins the RAM's execution loop.

   The loop proceeds as follows: first, the TM reads the current value of the PC into its finite control. (The PC cannot exceed the fixed length of the RAM's program; if it does, or becomes negative, the TM should crash.) The PC references a particular instruction, which is part of the finite control. The TM then executes the instruction as follows.

   - On "`STR` n a", the TM seeks to address a on the storage tape and writes n, as described above.
   - On "`ADD` a_1 a_2 a_3", the TM first seeks to the cells at addresses a_1 and a_2 on the storage tape and copies their contents to two work tapes t_1 and t_2. It then increments the value on t_1 while decrementing the value on t_2 toward 0, until the latter goes to 0. Finally, it stores the value on tape t_2 back to cell a_3 of the storage tape.
   - On "`BR` a j", the TM seeks to the start of cell a on the memory tape and determines the sign of the value stored there. (For reasonable representations of signed integers, this involves reading only a single symbol.) If the value is negative, Th TM replaces the value on the PC tape by $j - 1$.
   - On "`HLT`", the TM seeks to the start of cell 0 on the memory tape, erases its input tape, and replaces its contents with the cell's contents. Finally, it accepts.

   In all cases other than halting, the TM finishes the current instruction by incrementing the value on the PC tape by one.