CS 154: Introduction to Automata and Complexity Theory
Spring 2009

Assignment 6 Solutions
May 26, 2009

Problem 1

(a) If M doesn't halt on w, then $L(\widehat{M}) = \emptyset$.

(b) If M does halt on w, then $L(\widehat{M}) = \{xx \mid x \in \Sigma^*\}$.

(c) We first argue the correctness of the reduction. Observe that the construction of the Turing machine $\widehat{M}$ from $<M, w>$ is easily computable by an algorithm in finite time since it only involves a minor modification of M. Next, from the solutions to parts (a) and (b), we can conclude that: $<\widehat{M}>\in L_{NC}$ if and only if $<M, w>\in L_H$. This is because when $<M, w>\in L_H$, we have by part (a) that $L(\widehat{M}) = \emptyset$ which is clearly context-free; further, when $<M, w>\notin L_H$, we have by part (b) that $L(\widehat{M}) = \{xx \mid x \in \Sigma^*\}$, which is not context-free as can be argued via the pumping lemma.

Now, we show that L_{NC} is undecidable. Assume for contradiction that L_{NC} is indeed decidable. Then we have the following decision procedure for L_H: given an instance $<M, w>$, use the above reduction to produce an instance $<\widehat{M}>$ of L_{NC}. If L_{NC} is decidable, then we will be able to determine whether or not $<\widehat{M}>\in L_{NC}$, and thereby using the reduction determine whether or not $<M, w>\in L_H$. But this would mean that L_H is decidable, giving a contradiction. Thus, our assumption that L_{NC} is decidable cannot be correct.

Problem 2

Let L denote the language implicitly described in the problem statement, i.e., the language containing all triples $<M, w, X>$ such that M running on input w will eventually write the tape symbol X somewhere on the tape.

We claim that the language L is undecidable, but it is recursively enumerable.

To show that L is undecidable, we give a reduction from L_U. Given an input instance $<M, w>$ of the language L_U, where we would like to decide whether the TM M accepts the input w, the reduction constructs a new machine $\widehat{M}$ that uses the tape alphabet $\Gamma = \{0, 1, X, B\}$ and behaves as follows: Simulate the execution of machine M on input w (using only tape symbols 0, 1, and B to perform the simulation); if the simulation halts in an final state, then write X onto the tape; on the other hand, if M halts in a non-final state, the machine $\widehat{M}$ rejects; finally, if M never halts, then the same is true of $\widehat{M}$. It is clear that there is a halting Turing machine that can construct such a machine $\widehat{M}$. Turing machine $\widehat{M}$ will write X to its tape if and only if machine M accepts the input w. This completes the reduction. We know that L_U is undecidable, meaning that no algorithm exists for L_U. It follows from the reduction that L must also be undecidable.

To show that L is recursively enumerable, we describe a procedure for L. Given an input $<M, w, X>$ for L, simply simulate machine M on input w. If the simulation of M ever writes X to its simulated tape, then accept; on the other hand, if the simulation of M halts without writing X to its tape, then reject.

Problem 3
The proof is by a reduction from L_{all}. Let Turing machine M^* be one that always accepts, regardless of its input; clearly, it is easy to construct such a Turing machine, e.g., one which has no transitions and where the initial state q_0 is also a final state. The reduction starts with an input instance $<M>$ for L_{all} and produces as output an instance $<M_1, M_2>$ of L. In particular, it sets $M_1 = M^*$ and $M_2 = M$. It is easy to verify that this reduction can be computed by a halting Turing machine in finite time. Furthermore, it is clear that $<M_1, M_2> \in L$ if and only if $L(M) = \Sigma^*$. This is equivalent to saying that: $<M_1, M_2> \in L$ if and only if $L(M) \in L_{all}$. This completes the reduction. Since we know that L_{all} is non-recursive, it follows that L must also be non-recursive.

Problem 4
We will show that L is not recursively-enumerable by providing a reduction from $L_\emptyset$ to L; note, this is sufficient to establish that L is not recursively-enuerable since we already know that $L_\emptyset$ is not recursively-enumerable.

First, let us define a Turing machine M_{none} as the Turing machine which has no transitions at all and where the initial state in *not* a final state. Clearly, this Turing machine always halts in a non-accepting state (on all inputs) and so rejects all inputs, implying that $L(M_{none}) = \emptyset$.

The reduction (from $L_\emptyset$ to L) takes as input $<M>$ which is an instance of $L_\emptyset$, and produces as output an instance of L in the form of $< M_1, M_2, M_3 >$ where $M_1 = M$, $M_2 = M$, and $M_3 = M_{none}$. It is always the case that $L(M_2).L(M_3) = \emptyset$ since $L(M_3) = \emptyset$. It follows that $L(M_1) = L(M_2).L(M_3)$ if and only if $L(M_1) = L(M) = \emptyset$. This establishes the correctness of the reduction and completes the proof.

Problem 5
Let $w_1, w_2, \ldots, w_n$ and $x_1, x_2, \ldots, x_n$ be the two lists of a PCP instance over some alphabet Σ. We can convert this into an instance of the given language, i.e., grammars $< G_1, G_2 >$, as follows. Let $a_1, a_2, \ldots, a_n$ be n new terminals that are distinct from all the symbols used in the two lists of the PCP. Select $t-1$ distinct strings $z_1, z_2, \ldots, z_{t-1}$ over the same alphabet Σ as the PCP instance, and not including any of the special symbols $a_1, a_2, \ldots, a_n$. The grammars $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$ are defined as:

- $V_1 = \{S_1, A\}$ and $V_2 = \{S_2, B\}$
- $T_1 = T_2 = \Sigma \cup \{a_1, a_2, \ldots, a_n\}$
- P_1 contains the following two types of productions:
  - $S_1 \rightarrow A \mid z_1 \mid z_2 \mid \ldots \mid z_{t-1}$
  - $A \rightarrow w_i A a_i \mid w_i a_i$, for all $i = 1, \ldots, n$
- P_2 contains the following two types of productions:
  - $S_2 \rightarrow B \mid z_1 \mid z_2 \mid \ldots \mid z_{t-1}$
  - $B \rightarrow x_i B a_i \mid x_i a_i$, for all $i = 1, \ldots, n$

Let L_A and L_B be the set of sentences that can be generated from the variables A and B, respectively. Note that we ensured that $L(G_1)$ and $L(G_2)$ have at least $t-1$ strings in common, the strings $z_1, z_2, \ldots, z_{t-1}$ which cannot contain any of the special index symbols $a_1, a_2, \ldots, a_n$. Any additional strings common to the two languages $L(G_1$ and $L(G_2)$ can only come from the intersection of L_A and L_B, which is nonempty if and only if the PCP instance has a solution. (This part of the argument is exactly the same as in the proof of Theorem 9.22(a) on page 408 and is reproduced below in detail.)

Suppose first that $<G_1, G_2>$ belongs to L_t. Then the languages L_A and L_B are not disjoint, and there will be a string y belonging to both of them. By definition of the two grammars it must be the case that

$$y = w_{i_1} w_{i_2} \ldots w_{i_k} a_{i_k} \ldots a_{i_2} a_{i_1}$$

and

$$y = x_{j_1} x_{j_2} \ldots x_{j_k} a_{j_k} \ldots a_{j_2} a_{j_1}$$

for some choice of the indices $i_1, i_2, \ldots i_k$ and $j_1, j_2, \ldots j_k$. since the last part of the string y must agree in the two cases. But then it must be the case that

$$w_{i_1} w_{i_2} \ldots w_{i_k} = x_{i_1} x_{i_2} \ldots x_{i_k}$$

since the first part of y must agree in the two cases. Thus, we have a solution for the PCP instance.

Conversely, assume that the PCP instance has a solution $i_1, i_2, \ldots i_k$ such that

$$w_{i_1} w_{i_2} \ldots w_{i_k} = x_{i_1} x_{i_2} \ldots x_{i_k}$$

But is clear that the two grammars have the following derivations

$$S_1 \Rightarrow A \stackrel{*}{\Rightarrow} w_{i_1} w_{i_2} \ldots w_{i_k} a_{i_k} \ldots a_{i_2} a_{i_1}$$

and

$$S_2 \Rightarrow B \stackrel{*}{\Rightarrow} x_{i_1} x_{i_2} \ldots x_{i_k} a_{i_k} \ldots a_{i_2} a_{i_1}$$

which both give the same string. Then, the languages L_A and L_B are not disjoint, implying that $<G_1, G_2>$ belongs to L_t..

To fully establish the undecidability of L we also need to prove that the reduction from PCP to L_k can be done effectively, i.e., that the reduction is computable by a halting TM. This is easy to verify.

Problem 6

1. B (clearly a property of languages)
2. A (all answers are "YES" by definition)
3. B (since the property reduce to $\epsilon \in L(M)$, a property of languages)
4. C (not a property of recursively enumerable languages, but of pairs of languages)
5. A (just run the machine for 100 steps)