Homework 4: Solutions

Problem 1.

Show that the PCP is decidable over the unary alphabet, that is, over the alphabet $\Sigma = \{1\}$.

Answer

We construct a Turing machine T that decides the PCP over the unary alphabet. Machine T works as follows. It checks if all the dominos have more symbols on their top (resp. bottom) than on their bottom (resp. top), and if this is true then T rejects, otherwise T accepts.

Lets see why this is a decider. We have two possible situations when T accepts. The first is that some domino has strings of equal length on its top and its bottom, which makes a match by itself. The second situation, is when there are at least two dominos such that the first domino has string a on top and b on bottom, with $|a| > |b|$, and the second domino has string c on top and d on bottom with $|c| < |d|$. Let m denote the difference $|a| - |b|$ of symbols in the first domino, and n denote the difference $|d| - |c|$ of the in the second domino. Number m denotes the extra symbols on the top side of the first domino, and n denotes the extra symbols on the bottom side of the second domino. A match is obtained when we take x copies of the first domino and y copies of the second domino so that $mx = ny$. A simple solution to this equation is $x = n$ and $y = m$. (Another solution is $x = LCM(m,n)/m$, and $y = LCM(m,n)/n$.)

Problem 2.

Show that the PCP is undecidable over the binary alphabet, that is, over the alphabet $\Sigma = \{0, 1\}$.

Answer

The basic idea is that you can reduce the general PCP problem to it; you can take any string in some alphabet Σ and convert it into a string in the binary alphabet {0,1}. You have to be careful about how to do this conversion, however. In particular, you have to make sure that you can find delimiters in your binary strings so you don't get confused; one easy code is to assign the string n in Σ to 01111...1 (with the 1 repeating n times).

Problem 3.

Let T_c bet the set of Turing Machines with c or fewer states. The halting problem is trivally solvable for the set of machines T_1. Can the halting problem be solved for any other T_c with $c > 1$?

Answer

We that deciding a Turing machine with c states is not possible by showing that we can reduce an arbitrary Turing machine to a 2 state Turing machine. We do this by expanding the tape alphabet, Γ.

Problem 4.

Define a ***two headed finite automaton*** (2DFA) to be deterministic finite automoton that has two read-only, bidirectional heads that start at the left-hand end of the input tape and can be independently controlled to move in either direction. The tape of a 2DFA is finite and is just large enough to contain the input plus two additional blank tape cells, one on the left-hand end and one on the right-hand end, that serve as delimeters. A 2DFA accepts its input by entering a special halt state. For example, a 2DFA can recognize the language $\{a^n b^n c^n | n \geq 0\}$.

a. Let $A_{2DFA} = \{\langle M, x\rangle | M$ is a 2DFA and M accepts $x\}$. Show that A_{2DFA} is decidable.
b. Let $E_{2DFA} = \{\langle M\rangle | M$ is a 2DFA and $L(M) = \emptyset\}$. Show that E_{2DFA} is not decidable.

Answer

a. The solution to this problem is similar to the proof of Theorem 5.8 in Sipser's Book. In order to decide whether a $2DFA$ M accepts input w, we simulate M on w for $q(|w|+2)^2$ steps, where q is the number of states of M, or until it halts. If M has halted during the simulation, we accept if it has accepted, or we reject if it has rejected. If M has not halted then we reject.

   If M hasn't halted in $q(|w|+2)^2$ steps, it means that the machine is looping and therefore it will never halt on w. We have choosen $q(|w|+2)^2$ steps for the following reason. Because M is a $2DFA$, the amount of tape available to it is limited, and is equal to $|w|+2$. Since the possible positions of the two heads on the tapes are $(|w|+2)^2$, we have that the number of possible configurations of M is $q(|w|+2)^2$. If M has not halted after $q(|w|+2)^2$ steps it must be repeating a configuration and therefore is looping.

b. The solution to this problem is similar to the proof of Theorem 5.9 in Sipser's Book. This proof is by reduction from A_{TM}. For a turing machine M and an input w we can determine whether M accepts w by constructing a certain $2DFA$ B and then testing whether $L(B)$ is empty. The language that B recognizes comprises all accepting computation histories for M on w. If we can determine whether B's language is empty, clearly we can determine whether M accepts w.

   The $2DFA$ B works in a similar way as the LBA B in the proof of Theorem 5.9. The only significant difference is in the check of the second condition. This step involves verifying that C_i and C_{i+1} are identical except for the positions under and adjacent to the head in C_i. These positions must be updated according to the transition function of M. The $2DFA$ B verifies that the updating was done properly by comparing the two configurations by placing its first head on configuration C_i, and its second head on configuration C_{i+1}. There is no need to mark the tape, since the $2DFA$ keeps track of the current position on both configurations by using the two heads.