

Homework 6

Due: April 6, 2005

Vinod Vaikuntanathan

Reading: Sipser, Sections 6.1, 6.3**Problem 1:** (Counter Machine Decidability) Show that the acceptance problem for 1-counter machines is decidable, by describing a procedure that correctly decides it.

(Hint: Develop a way of detecting situations in which the counter machine must be in a loop.)

Solution 1:

A piece of notation: Let the counter machine be M and the input word be w . The configuration of the counter machine M at any point of time is described by a pair (q, c) where q is the state in the finite state control of M and c is the counter value.

The following procedure $D(M, w)$ decides the acceptance problem for 1-counter machines:

1. Run the counter machine M on input w .
2. If M enters the accept state, accept.
3. If M enters a reject state, reject.
4. (*detect a loop*) If M repeats a configuration (q, c) without consuming any input, output reject.
5. (*detect a loop*) If M enters a configuration (q, c_1) and later enters a configuration (q, c_2) without consuming any input, and for all intermediate configurations (q', c) (between (q, c_1) and (q, c_2)), the counter value $c > c_1$, then reject.

Lemma 1 *If the decider D (as given above) rejects, then M does not accept w .***Proof:** There are 3 situations where D outputs “reject”.

- (Instruction 3) M enters a reject state. Then, clearly M does not accept w .
- (Instruction 4) M repeats a configuration (q, c) without consuming any input. The machine is at configuration (q, c) at some point, it goes through configurations $(q, c) = C_0, C_1, \dots, C_k = (q, c)$. It is easy to see that, $C_{k+1} = C_1, C_{k+2} = C_2$ etc. (All this is true only if the machine did not consume any input when going through the configurations C_1, \dots, C_k .) Thus, M loops, if this happens, and therefore, M does not accept w .
- (Instruction 5) M enters a configuration (q, c_1) and later enters a configuration (q, c_2) without consuming any input, and for all intermediate configurations (q', c) (between (q, c_1) and (q, c_2)), the counter value $c > c_1$.

Let the configuration $C_1 = (q, c_1)$. The machine goes through configurations C_2, \dots, C_{k-1}, C_k and reaches the configuration $C_{k+1} = (q, c_2)$, where $c_2 > c_1$. Since, in all intermediate configurations, the counter value $c > c_1 > 0$, the machine will do exactly the same thing as it did in the previous loop. More formally, consider the configurations $C_i = (q', c')$ and $C_{i+1} = (q'', c'')$, where $i \leq k$. By induction on i and k , we can prove that $C_{k+i} = (q', c''')$ and $C_{k+i+1} = (q'', c''')$ and $c'''' - c''' = c'' - c'$. That is, the states that the machine M is in, in configurations C_i and C_{k+i} are the same, for all i . Moreover, the counter value changes the same way in C_{k+i} as it did in C_i . Thus, the counter keeps increasing and the machine goes through the same sequence of states, on each loop. Therefore, M does not accept w .

Induction Argument(Sketch): First of all, by induction on i , we establish that the state in the configuration C_{k+i} is the same as the state in C_i . The base case is clear, since the state in $C_1 = (q, c_1)$ is by definition the same as the state in $C_{k+1} = (q, c_2)$. Now, suppose the state in configuration C_{k+j} is the same as the state in C_j for some j . The counter value in C_j is larger than c_1 , and in particular, it is non-zero. The counter value in C_{k+j} is larger than c_2 , which is larger than c_1 . Thus, the counter in C_{k+j} is non-zero too. This, together with the fact that the machine does not read a new input symbol, means that the machine will make the same transition in both C_j and C_{k+j} . Thus, the states in C_{j+1} and C_{k+j+1} are the same. If the machine increases (resp. decreases) the counter in configuration C_j it increases (resp. decreases) the counter in C_{k+j} too. Thus, if $C_j = (q', c')$ and $C_{j+1} = (q'', c'')$, then $C_{j+1} = (q', c''')$ and $C_{k+j+1} = (q'', c''')$, where $c''' - c''' = c'' - c'$.

■

Lemma 2 *If the machine M does not accept w , the decider D (as given above) rejects.*

Proof: If M does not accept w , M either rejects w or loops on input w . The former case is handled by Instruction 3 of the decider D .

Suppose M on input w loops. Then, eventually, M consumes the last input it is ever going to consume, so thereafter, it does infinitely many steps without consuming any more input. Let α be this infinite, input-free suffix of the computation.

Suppose that a (q, c) configuration is never repeated in α . Since there are only finitely many states and we never repeat such a pair, it must be that the value of the counter is unbounded.

Pick any counter value c taken on in α . Then in α , all counter values larger than c must also arise. So, consider the *last* configuration in α in which the counter takes on each value $c, c+1, c+2, \dots$. Call these configurations $C_c, C_{c+1}, C_{c+2}, \dots$. (These last occurrences must occur in increasing order in α .) By Pigeonhole principle, in two of these configurations, the state must be the same. i.e, there are two configurations of the form $C_{c_1} = (q, c_1)$ and $C_{c_2} = (q, c_2)$ such that $c_2 > c_1$. By our choice of the configurations C_{c_1} and C_{c_2} , all intermediate configurations have counter value strictly greater than c_1 . This gives us the condition of Instruction 5, and the decider will reject in this case. ■

Problem 2: (From Sipser Problem 6.19.)

Recall the Post Correspondence Problem discussed in class and in Section 5.2 of Sipser. Show that PCP is decidable relative to A_{TM} , the acceptance problem for ordinary Turing machines.

Solution 2: We construct an oracle TM that decides PCP as follows:

$T^{A_{TM}}$ = "On input $\langle d_1, d_2, \dots, d_k \rangle$ (dominos),

1. Construct TM M as follows:

M = "On input x ,

1. Try all domino combinations in parallel.
2. If a match is found, accept."
2. Query the oracle to determine whether $\langle M, 0 \rangle$ in A_{TM} .
3. If the oracle answers YES, accept; if NO, reject."

If $\langle d_1, d_2, \dots, d_k \rangle$ have a match, then M will eventually find it and accept all strings (i.e., $L(M) = \Sigma^*$). If there is no match, however, then M will never accept any string. Thus, T can use its oracle in step 2 to test M on any string (we picked 0), and this tells us whether the dominos are in PCP or not.

Problem 3: (From Sipser Exercise 6.4.)

Let $A'_{TM} = \{ \langle M, w \rangle \mid M \text{ is an oracle Turing machine and } M^{A_{TM}} \text{ accepts } w \}$. Thus, A'_{TM} can be thought of as the "acceptance problem for oracle TMs relative to the acceptance problem for ordinary TMs".

Show that A'_{TM} is not decidable relative to A_{TM} . That is, even with an oracle for the ordinary acceptance problem, the relative version of the acceptance problem still cannot be decided!

(Hint: Use a diagonalization argument like the one used to prove undecidability of A_{TM} .)

Solution 3: Suppose there exists a oracle machine T with oracle access to A_{TM} that decides Z , then we can construct D (an oracle TM with oracle access to A_{TM}) as follows:

D = "On input $\langle M \rangle$, where M is an oracle TM,

1. Run $T^{A_{TM}}$ on input $\langle M, \langle M \rangle \rangle$.

2. If T accepts, reject; otherwise accept."

Now D is a oracle TM with oracle access to A_{TM} that accepts the oracle machine $\langle M \rangle$ iff $M^{A_{TM}}$ does not accept $\langle M \rangle$. In other words,

$$D^{A_{TM}} \text{ accepts } \langle M \rangle \text{ iff } M^{A_{TM}} \text{ does not accept } \langle M \rangle.$$

Now, if we set M to D in the above, we get the following contradiction:

$$D^{A_{TM}} \text{ accepts } \langle D \rangle \text{ iff } D^{A_{TM}} \text{ does not accept } \langle D \rangle.$$

Thus, we conclude that Z must be undecidable.