CS331: Homework #8

Due on April 11, 2013 at 11:59pm

Professor Zhang 9:00am

Josh Davis

Problem 1

Let $A \leq_L B$ mean that $A \leq_T B$ with additional condition that the oracle Turing machine M^B that solves A queries the oracle for B only once, at the very last step.

Prove that $A \leq_L B$ if and only if $A \leq_m B$.

Proof. To prove this, we will show that each side implies the other.

Part One If $A \leq_L B$, then $A \leq_m B$.

Assume $A \leq_L B$. This means there is an oracle TM M^B that can be queried once and only at the end execution.

Let's construct a TM N that decides $A \leq_m B$.

$N =$" On input w:

1. Perform some computation using w
2. Simulate M^B on newly computed value"

This is equivalent to a many-one reduction using oracle TM M^B because invoking M^B acts just as a computed function that maps A to B. Thus it is simply a computed function.

Part Two If $A \leq_m B$, then $A \leq_L B$.

Assume that $A \leq_m B$. This means there is a computable function f that reduces A to B. Also this means there are two deciders, M_1 and M_2 that decide the languages for A and B respectively.

To show what we'd like to prove, we will construct a new TM M using this function that operates exactly like a one-time use oracle executed last.

Let's construct a TM N that decides $A \leq_L B$.

$N =$" On input w:

1. Perform some computation on w using M_1
2. Compute $f(w)$
3. Simulate M_2 on newly computed value"

This works because the computed function f is just a TM that computes f. Once we use the computed function on the input after manipulation we then simulate M_2 using the result of $f(w)$. This will leave whatever M_2 leaves on the tape after execution. This exactly how a oracle TM functions.

Since we have proven each side, we have shown that $A \leq_L B$ if and only if $A \leq_m B$. Thus our proof is complete.

□

Problem 2

Describe two different Turing machines, M_1 and M_2 such that when started on any input, M_1 outputs $\langle M_2 \rangle$ and M_2 outputs $\langle M_1 \rangle$.

The two TMs are quite similar to the SELF program in that the first TM will print the encoding of the second machine. The second TM will then use the Recursion Theorem to compute what the first one is. We can define the two TM's like so:

$M_1 = P_{\langle M_2 \rangle}$

$M_2 =$" On any input, just ignore it

1. Retrieve own description $\langle M_2 \rangle$
2. Compute $q(\langle M_2 \rangle)$.
3. Print newly computed TM and halt."

This works because M_1 just uses **Lemma 6.1** and then M_2 just computes what M_1 must be and prints it.

Problem 3

Prove that no universal corruptor exists.

Proof. We will show that no universal corruptor exists by proof by contradiction.

Assume that a universal corruptor does exist. Let this corruptor be the function f. This function when given any TM, it will construct a TM that behaves differently. Formally this means $L(M) \neq L(f(M))$.

Given that this function exists, there should be no such TM that violates this corruptability.

Let's create a new TM, C that tries to violate this:

$C =$" On input w:

1. Obtain using the recursion theorem, our own description, $\langle C \rangle$.
2. Compute $f(\langle C \rangle)$ to obtain a new description of a TM, $UNTOUCHABLE$.
3. Simulate $UNTOUCHABLE$ on w."

Since f is universal, there cannot exist a TM that has the same language as the output of f.

Knowing this we can see that this is a contradiction for any universal corruptor f. If f is in fact universal, then there is no TM that has the same language as the output of f. Yet C and the new TM $UNTOUCHABLE$ have the same language because C simulates $UNTOUCHABLE$. This means that $UNTOUCHABLE$ is in fact uncorruptable and thus untouchable.

□

This proof follows the same idea that is presented in **Theorem 6.8**. Which shows that for any such transforamtion of a TM description, there exists some TM whose behavior is unchanged by the transformation.

Problem 4

Let $SELF_{TM} = \{\langle M \rangle : L(M) = \{\langle M \rangle\}\}$. Prove that neither $SELF_{TM}$ nor $\overline{SELF_{TM}}$ is Turing-recognizable.

We will use the same idea that is presented in **Theorem 5.30** by using recursion theorem to prove the unrecognizability of a language.

Proof. First we will show that $SELF_{TM}$ is not Turing recognizable. We do so by showing that A_{TM} is reducible to $\overline{SELF_{TM}}$. The reducing function f works as follows:

```
$F =$" On input $\langle M, w \rangle$ where $M$ is a TM and $w$ a string:
  1. Construct a new TM, $M'$:
     $M' =$" On any input:
       (a) Retrive own description using Recursion Theorem, $\langle M' \rangle$.
       (b) If $M$ accepts $w$ and $\langle M \rangle \neq \langle M' \rangle$, accept"
  2. Output $\langle M' \rangle$.
```

Here, M' accepts only when M accepts w and the two encodings of M and M' are equal. Conversely, if M doesn't accept, then we will never accept. Thus f reduces A_{TM} to $\overline{SELF_{TM}}$.

Now to show that $\overline{SELF_{TM}}$ is not Turing-recognizable we will give a similar reduction from A_{TM} again to the complement of $\overline{SELF_{TM}}$, which is just $SELF_{TM}$. The following TM G computes the reducing function g:

```
$G =$" On input $\langle M, w \rangle$ where $M$ is a TM and $w$ a string:
  1. Construct a new TM, $M'$:
     $M' =$" On any input:
       (a) Retrive own description using Recursion Theorem, $\langle M' \rangle$.
       (b) If $M$ accepts $w$ and $\langle M \rangle = \langle M' \rangle$, accept"
  2. Output $\langle M' \rangle$.
```

The difference between these two functions f and g is in the if statement on the inner TM. Depending on the language we are reducing to, we just ensure that it is a valid reduction function by following $w \in A \iff f(w) \in B$.

Thus we have shown that neither $SELF_{TM}$ nor $\overline{SELF_{TM}}$ is Turing-recognizable and conclude our proof. □

Problem 5

Prove that the class P is closed under union and complementation.

Part One Prove that P is closed under union.

Proof. To prove that P is closed under union, we assume that there are two languages L_1 and L_2 with M_1 and M_2 as TMs that decide them.

Assume that M_1 runs in polynominal time, $O(n^x)$ as well as M_2, $O(n^y)$.

We will construct a new TM M that runs both M_1 and M_2 and show that it still runs in polynomial time.

$M =$" On input w:

1. Run M_1 on w.
2. Run M_2 on w.
3. If one of the TMs accepted, accept, else reject."

The runtime of M can be determined as $O(n^x) + O(n^y)$. Asymptotically, this is equal to the following: $O(n^z)$ where $z = max(x, y)$.

Thus we can see that P is closed under union. □

Part Two Prove that P is closed under complementation.

Proof. To prove that P is closed under complementation, we assume that there is a language L with a TM M that decides it.

Assume that M runs in polynominal time, $O(n^x)$.

We will construct a new TM N that runs M and we will show that it still runs in polynomial time.

$N =$" On input w:

1. Run M on w.
2. If M accepted, reject, else accept."

The construction of N is such that it decides the complement of the language that M decides. This TM also runs in $O(n^x)$, which means it still runs in polynomial time.

Thus we can see that P is closed under complementation.

□