CPSC 421 - Homework 8

Tristan Rice, q7w9a, 25886145

1. Consider a fixed language …

1.a Does the same argument work for "Turing machines" replaced with "Turing machines with oracle L"?

No, it doesn't work for Turing machines with oracle L. The set of languages over Σ^* is uncountably infinite. Since an oracle L could accept all languages in Σ^*, the set of languages a Turing machine with oracle L is uncountably infinite.

1.b Give an L such that the set of languages accepted by Turing machines with oracle L is equal to set of languages accepted by Turing machines.

For the sets to be equal L must be computable by a Turing machine. Thus you can pick a simple language that's easily computable.

Thus, we can just do $L = \Sigma^*$ which means that the oracle will return true always since all inputs to the Turing machine must be $\in \Sigma^*$. Thus, it's trivially computable by a standard Turing machine and thus the set of languages accepted by Turing machine with oracle L is equal to the set of languages accepted by Turing machines.

1.c

Let L be the set of languages describing Turing machines that do not halt. Since halting is undecidable, a Turing machine can not accept L. A Turing machine with oracle L can accept L since an oracle can compute anything.

2. Exercise 6.4 of [Sip]

Let $A'_{TM} = \{\langle M, w\rangle | M$ is an oracle TM and $M^{A_{TM}}$ accepts $w\}$. Show A'_{TM} is undecidable relative to A_{TM}.

$A_{TM} = \{\langle M, w\rangle | M$ is a TM and M accepts $w\}$.

We must show that there does not exist a Turing machine with an oracle for A_{TM} that recognizes A'_{TM}.

Let w be a definition of a Turing machine. A_{TM} cannot decide the halting problem. Thus, if we have A'_{TM} be a Turing machine that accepts if w halts and rejects if it doesn't, we know that A'_{TM} is undecidable relative to A_{TM}, since the halting problem is undecidable.

3. Prove the assertion regarding replacing $a_1 \vee \ldots \vee a_l$ by a 3CNF with $l-2$ clauses at the top of page 311 of Sip.

Proof by induction:

Base case $l < 4$:

This is trivially in 3CNF and doesn't need anything else.

Base case $l = 4$:

We can replace

$$a_1 \vee a_2 \vee a_3 \vee a_4$$

with

$$(a_1 \vee a_2 \vee z) \wedge (6\ z \vee a_3 \vee a_4)$$

.

Since z can be anything that satisfies this, we can construct all the cases such that the statement is true.

$$(0 \vee 0 \vee 0) \wedge (6\ 0 \vee 0 \vee 0) = 0$$
$$(1 \vee 0 \vee 0) \wedge (6\ 0 \vee 0 \vee 0) = 1$$
$$(0 \vee 1 \vee 0) \wedge (6\ 0 \vee 0 \vee 0) = 1$$
$$(0 \vee 0 \vee 1) \wedge (6\ 1 \vee 1 \vee 0) = 1$$
$$(0 \vee 0 \vee 1) \wedge (6\ 1 \vee 0 \vee 1) = 1$$
$$(1 \vee 1 \vee 0) \wedge (6\ 0 \vee 0 \vee 0) = 1$$
$$(1 \vee 0 \vee 0) \wedge (6\ 0 \vee 1 \vee 0) = 1$$
$$(1 \vee 0 \vee 0) \wedge (6\ 0 \vee 0 \vee 1) = 1$$
$$(0 \vee 1 \vee 0) \wedge (6\ 0 \vee 1 \vee 0) = 1$$
$$(0 \vee 1 \vee 0) \wedge (6\ 0 \vee 0 \vee 1) = 1$$
$$(0 \vee 0 \vee 1) \wedge (6\ 1 \vee 1 \vee 1) = 1$$
$$(1 \vee 1 \vee 0) \wedge (6\ 0 \vee 1 \vee 0) = 1$$
$$(1 \vee 1 \vee 0) \wedge (6\ 0 \vee 0 \vee 1) = 1$$
$$(1 \vee 0 \vee 0) \wedge (6\ 0 \vee 1 \vee 1) = 1$$
$$(0 \vee 1 \vee 0) \wedge (6\ 0 \vee 1 \vee 1) = 1$$
$$(1 \vee 1 \vee 0) \wedge (6\ 0 \vee 1 \vee 1) = 1$$

This covers all 16 possible cases of the original statement.

Inductive step.

$n > 4.$

Assume we have statements in the form

$$... (... \vee z_r) \wedge (6\ z_r \vee a_{r-2} \vee z_{r+1}) \wedge (6\ z_{r+1} \vee ...) ...$$

If a_{r-2} is true, it doesn't matter what values of z there are for this clause to hold true.

If there are no true a values, it's possible for $z_r = 1, z_{r+1} = 0$ and this clause to be false as it should be.

If a_{r-2} is false, that means either $z_r = 0 \vee z_{r+1} = 1.$ If that holds, this clause is true.

If the true a value is to the right of this clause, $z_{r+1} = 1, z_r = 1$ and the inverse if it is to the right. That causes this clause to be true.

Thus by induction, all clauses on either sides of a a can be satisfied. If all values of a is false, there is no selection such that the entire statement is satisfied.

4. Consider a non-deterministic Turing machine …

4.a

$f(x_1) = x_1$ is accepted by M since $f(true) = true.$ It is likewise accepted by M' since $f(false) = false).$

4.b

f is not accepted by M since no matter what you provide to f it returns $false$. It is accepted by M' since it will always return false.

4.c

f is accepted by M since no matter what input is provided it always returns $true$. It isn't accepted by M' since it will never return $false$.

4.d

Since $UNSAT$ always returns $false$ that means it's accepted by M', but not by M since M requires it to evaluate to $true$.