CS 3100 – Models of Computation – Fall 2010
Notes for Lecture 24 on Reductions, NP-Completeness

How to solve hard problems?

- Solve them! If you can't, then
- Club all of them together!

This is what A_{TM} and $3SAT$ represent! If a problem in NP has a poly algorithm, so do all in the NPC-land (thru the P-time mapping reduction!)

If a problem in the undecidable (Turing complete) class have a decider, so do all!

Notes on "if" and "iff"

$A\ if\ B$ stands for $B \Rightarrow A$. Examples: $(x\%6 = 0) \Rightarrow (x\%3 = 0)$, or $(x\%3 = 0)\ \ if\ \ (x\%6 = 0)$. The set of points representing $(x\%6 = 0)$ is contained in those representing $(x\%3 = 0)$.

$A\ \ iff\ \ B$ stands for $B \iff A$ or $A \iff B$. "Iff" represents "equal truth" in some sense. Examples:

- One triangle with base B and height H has area A iff any other triangle with the same B and H has the same area
- Arithmetic induction iff complete induction (P(0) and P(n) implies P(n+1) implies for all x:P(x)) **IFF** ((for all m $\leq n$: P(m) implies P(n)) implies for all x:P(x))

(Negation law of iff.) If $A \iff B$ then $A \Rightarrow B$ and $B \Rightarrow A$. This means $\neg B \Rightarrow \neg A$ and $\neg A \Rightarrow \neg B$. Thus $\neg A \iff \neg B$.

Chapter 15, Main Idea Behind Reductions: The main idea is to reduce language A into language B through a *mapping reduction.* We will illustrate mapping reductions on the following undecidable problem: *It is undecidable whether a TM enters a specific state (or whether a program has dead code at a particular location L).* We call this the *state usage* problem and write it StateUsage sometimes.

Main Idea behind Mapping Reductions: Mapping reductions help model a problem in terms of another, such that if the latter is decidable, so is the former.

The idea is that we must find a *Turing computable* function f (for example, a C function f **that itself does not loop**) such that

$$w \in A \iff f(w) \in B$$

We capture this situation (of the existence of such an f) as $A \leq_m B$. Here are some facts:

- If B is recursive, so is A.
- If A is not recursive, then B is not (contrapositive form of the above).
- $A \leq_m B$ implies $\overline{A} \leq_m \overline{B}$. How so?
  - From the negation law of iff we have $w \notin A \iff f(w) \notin B$
  - Or $w \in \overline{A} \iff f(w) \in \overline{B}$
  - Or $\overline{A} \leq_m \overline{B}$.

StateUsage (whether some code is "Deadcode") is Undecidable:

- The input language A is A_{TM}.
- Define the output language B

$$\{\langle M, q\rangle \ \mid\ M\ visits\ state\ q\}$$

- Define the mapping reduction that takes an $\langle M, w\rangle$ and produces a $\langle M', q\rangle$ where q is a new state not used by M. *Further, we arrange M to jump to q whenever M instead would have halted in one of the states h_a (this is the description of the machine M').*

- This is a mapping reduction from the whole of A into B.
  - "There is a point x in A" means x is of the form $\langle M, w\rangle$ such that M accepts w.
  - The manner in which we produce $M^{'}$ and q, we guarantee that $M^{'}$ indeed visits q **iff** M accepts w.

  What we "hit" in B are only those instances produced by "massaging" M in the aforesaid manner. Thus we likely have a one-to-one but *into* map into B.
- Now if there is a decider for the whole of B, there must be a decider for $f(A) \subseteq B$ also.
- This means: we can decide the whole of A by taking a $\langle M, w\rangle$ and produce a $\langle M, q\rangle$ where q is a new state not in M, and asking the decider of B whether this $\langle M, q\rangle \in B$.
- We know that for $\langle M, w\rangle \notin A$, $\langle M, q\rangle \notin B$.
- Thus by knowing whether $\langle M, q\rangle \in B$ or not, we can tell whether $\langle M, w\rangle \in A$ or not, thus deciding A_{TM}.

Mapping Reduction from A_{TM} to $Regular_{TM}$: Define $Regular_{TM}$ to be the language of all TM codes such that the TM itself has a regular language. Define the following mapping reduction from A_{TM} to $Regular_{TM}$:

```
M'(x) {
 if x is of the form 0^n 1^n then goto acceptM'; // 0 power n 1 power n
 Run M on w; // This may LOOP!!
 if M accepts w, then goto acceptM';
 if M rejects w, then goto rejectM';
}
```

Now the language of `M'` is regular (actually Σ^*) iff M accepts w. Thus by testing for regularity we can solve A_{TM}.

Rice's Theorem: Every non-trivial partitioning of TM codes based on the language of the TMs is undecidable. We saw two non-trivial partitionings above: (i) based on state visitation, (ii) based on the language of the TM. *Colloquially, we are in the land of "equal oppportunity TMs." That is, "TMs" can't be discriminated based on Race, Color, or Origin.*

Chapter 17, Time Complexity and the class NP: The same mapping reduction idea works for time complexity also. In this case, we are working on looping-free TMs (deciders). For TMs that may loop, their time complexity is infinite! The time complexity of TMs is given by the number of steps they take before they halt.

Class NP: The class NP is those problems for which there are non-deterministic algorithms that "perform" in polynomial time. That is, there is an NDTM that, while taking those "magical guesses" is able to halt in a polynomial number of steps. Alternately, if a solution is given as a certificate, the solution can be checked in polynomial time. Examples of ND languages:

- $SAT = \{\Gamma \mid \Gamma\ is\ a\ Boolean\ formula\ that\ is\ satisfiable\}$
- $3SAT = \{\Gamma \mid \Gamma\ is\ a\ 3CNF\ Boolean\ formula\ that\ is\ satisfiable\}$
- $Clique = \{G \mid G\ is\ a\ graph\ with\ a\ clique\}$

Chapter 19, NP-Completeness: We don't know whether all ND languages have deterministic polynomial time algorithms. That is whether the language family NP and P are the same. To attack this annoying "gap" between P and NP, scientists have come up with the *hardest* members of NP called NP-complete.

A language L is NP-complete if

- (In NP) L is in NP
- (NP-hard) Every other language in NP can be poly-time mapping reduced to L

Note that there are undecidable NP-hard languages (e.g. Diophantines language!). Thus we must show the "In NP" part for any meaningful dialog in this space!!

The first problem shown to be NP-complete is 3SAT. This is done by showing that the "work of any NDTM can be represented as Boolean formula satisfaction."

Now for any other language (e.g. Clique), we show it NP-complete by

- Showing that Clique is in NP
- Showing that there is a mapping reduction from 3SAT to Clique