HW2

SJSU Students

CS254

San José State University

March 6, 2017

1. (a) $two(x) := 2$

   We know the initial functions: $zero(x) = 0$ and $S(x) = x + 1$

   Using composition we can define $two(x)$ as:

$$two(x) = S(S(zero(x)))$$

   (b) $mult(x, y) := x \cdot y$

$$g(x) := 0$$

   In our lecture we have defined the addition function $f_+(x, y) = x + y$

$$h(x, n, z) := f_+(I_3^3(x, n, z), x)$$

$$mult(x, 0) := g(x) = 0$$

$$mult(x, n + 1) := h(x, n, mult(x, n))$$

$$= f_+(I_3^3(x, n, mult(x, n)), x)$$

$$= f_+(mult(x, n), x)$$

(c) $pow(x) = 2^x$

$$g(x) := 1$$

$$h(x, z) := mult(I_2^2(x, z), 2)$$

$$pow(0) := g(x) = 1$$

$$pow(x + 1) := h(x, pow(x))$$

$$= mult(I_2^2(x, pow(x)), 2)$$

$$= mult(pow(x), 2)$$

(d) $super\_pow(x) = 2_x$

We know $2_0 = 1$ and $2_{x+1} = 2^{2_x}$

$$g(x) := 2_0 = 1$$

$$h(x, z) := h(x, z) = pow(I_2^2(x, z))$$

$$super\_pow(0) := g(x) = 1$$

$$super\_pow(x + 1) := h(x, super\_pow(x))$$

$$= pow(I_2^2(x, super\_pow(x)))$$

$$= pow(super\_pow(x))$$

2. $\mathcal{O}(T \log T)$ universal simulation:

(a) State of the simulating tape when the tape head is reading 'x':

$$\overbrace{c|o|m|p|u|t|a|t|\overline{\square\square\square\square\square\square\square\square}}^{L_3}\,\overbrace{i|o|n|a|l|c|o|m|}^{L_2}\,\overbrace{p\overline{\square\square\square}}^{L_1}\,\overbrace{|l|e|}^{L_0}\,\boxed{\text{x}}\,\overbrace{\overline{\square\square}}^{R_0}\,\overbrace{|i|t|y|\overline{\square}}^{R_1}\,\overbrace{\overline{\square\square\square\square\square\square\square\square}}^{R_2}\,\overbrace{\overline{\square}\ldots\overline{\square}}^{R_3}$$

(b) State of the simulating tape if the universal TM moved left and wrote the letter s.

Assumption: the machine moves left means tape is moving left.

$$\overbrace{c|o|m|p|u|t|a|t|\square\square\square\square\square\square\square\square}^{L_3}\overbrace{i|o|n|a|l|c|o|m|}^{L_2}\overbrace{p|l|e|s|}^{L_1}\overbrace{\square\square}^{L_0}\ \boxed{\text{i}}\ \overbrace{t|y|}^{R_0}\overbrace{\square\square\square\square}^{R_1}\overbrace{\square\square\square\square\square\square\square\square}^{R_2}\overbrace{\square\ldots\square}^{R_3}$$

3. Determine complexity class of problems based on a boolean circuit:

(a) Roughly, we can repeat the process while we find a node whose value is not known. (i) Look up in linear time its at most two predecessors, (ii) if the predecessors already have values then use these to compute and store the nodes value. Since we repeat at most n times, the whole process could be done in quadratic time.

(b) This problem is in coNP, we check foreach at most polynomial length assignment whether the two circuits agree on the output.

(c) This problem is in EXP for general k as the length of k could be proportional to the input size. So k could be a number which is exponential in the input size. If the number k is small, that is, polynomial in the input. Then we can guess k assignments and check if they all work. For this "easy" case the problem would be in NP.

(d) Given description of a boolean circuit and a number k, while we can guess k possible assignments to the input variables of the circuit and attempt to verify them, it is impossible to determine if there are only k possible assignments that make the circuit output 1 without running the circuit on all possible assignments i.e. 2^n which puts this problem in the EXP class.

4. Proof that TMSAT is complete for NQLIN under quasi-linear time reductions:

- First, we prove $TMSAT \in NQLIN$. On input $\langle \alpha, w, 1^n, 1^t \rangle$, we can non-deterministically guess a string u of length n in linear time and then simulate M_α according to this string in quasi-linear time steps to see if it accepts $\langle w, u \rangle$.
- Now, we prove $TMSAT$ is hard in $NQLIN$. Suppose $L \in NQLIN$ language, then there is some verifier M such that $x \in L \iff \exists u \in \{0,1\}^{p(|x|)}$ such that $M(x,u) = 1$ and M runs in $Q(n)$ where Q is the quasi-linear function for some constant k i.e. $(n \log^k n)$. To reduce L to $TMSAT$, we simply map every string $x \in \{0,1\}^\star$ to the tuple

$\langle \lfloor M \rfloor, x, 1^{p(|x|)}, 1^{Q(m)} \rangle$ where $m = |x| + p(|x|)$ and $\lfloor M \rfloor$ denotes the representation of M as a string. This mapping can clearly be performed in polynomial time and by the definition of $TMSAT$ and the choice of M,

$$\langle \lfloor M \rfloor, x, 1^{p(|x|)}, 1^{Q(m)} \rangle \in TMSAT \iff \exists u \in \{0,1\}^{p(|x|)}$$

s.t. $M(x,u)$ outputs 1 within $Q(m)$ steps $\iff x \in L$

5. Summary of argument for NP-completeness of knapsack problem:

   The paper referred to is: `https://people.eecs.berkeley.edu/~luca/cs172/karp.pdf`
   This paper contains proofs for NP-completeness of 21 problems one of which is the Knapsack problem. The proofs of all problems in the paper are ultimately reduced to the satisfiability problem which is known to be NP-complete by the proof provided by Cook-Levin.

   The proof for Satisfiability $\leq_p$ 3-SAT is similar to what we have in class where we introduced new variables to transform clauses from a SAT problem to clauses of size 3.

   Using the NP-completeness of 3-SAT Karp goes on to prove NP-Completeness of the Chromatic Number problem by creating a Boolean formula that provides truth assignments for k-coloring of the graph.

   In the boolean formula formulation of the chromatic number problem, we have k disjoint subsets of the graph G. We use these subsets verify exact cover.

   The proof for Exact Cover $\leq_p$ Knapsack, uses the list of subsets from exact cover as the possible list of items selected for the knapsack solution. It decides if an item is selected if that item exists in the subset.

   The hierarchy of reductions to prove that Knapsack is NP-complete is shown below:

$$\text{Satisfiability} \leq_p \text{3-SAT} \leq_p \text{Chromatic Number} \leq_p \text{Exact Cover} \leq_p \text{Knapsack}$$