CS388T: ANSWER TO HOMEWORK 4

VISHVAS VASUKI

1. Question

Prove that 2-SAT is NL complete under logspace reductions.

1.1. Solution.

Acknowledgement. I referred to [1] in solving this problem.

Suppose that we are given a 2-SAT formula, F. Construct the following directed graph G: The verteces in G are the variables in F and their negations. The edge (a,b) exists if and only if one of the clause $(\sim a \vee b)$ exists in F.

Lemma 1.1.1. *A 2-CNF forumula F is unsatisfiable if and only if there is a path from a variable to its negation in the graph described above.*

Proof. The graph G corresponding to the formula F captures 'implication' relationships between variables. So, an edge between the literals (a,b) represents the fact that, according to the formula F, a implies b or b implies a. Furthermore, implication relationships are transitive.

$a \Rightarrow \sim a$ and $\sim a \Rightarrow a$ can never be true. Hence, if a path exists between a variable and its negation, the formula is unsatisfiable.

If no such path exists, any assignment which preserves the implication relationships satisfies F. Such assignments can be found by repeatedly doing the following:

- Select a node n in G which has not yet been assigned a truth value.
- Assign the value true to n and all nodes reachable from n. Assign the value false to the negation of these nodes.

□

Lemma 1.1.2. *2-SAT is in NL.*

Proof. 2-UNSAT, the complement of the language corresponding to the decision problem 2-SAT, can be solved by solving reachability on the graph described above. (This follows from the previous lemma.)

It also follows that 2-SAT can be solved by solving UNREACHABILITY on the graph corresponding to F. (We need to check that there is no path from any variable to its negation.) Due to the Immermann Szelepcsenyi theorem, we know that UNREACHABILITY can be solved in non-deterministic log-space. Hence, 2-SAT is in NL. □

Theorem 1.1.3. *2-SAT is NL complete.*

Proof. We have already shown that 2-SAT is in NL.

We will now reduce UNREACHABILITY to 2-SAT. Consider an acyclic graph G. Every edge (a,b) in G is equivalent to the implication $a \Rightarrow b$, which is equivalent

to the clause ($\sim a \vee b$). The set of the resulting clauses constitutes a 2-CNF. The arguments to the UNREACHABILITY problem consist of a graph, a start node and an end node. Corresponding to these, we can add clauses s and $\sim s$.

From a lemma we proved earlier, we know that the end node is unreachable from the start node if and only if the corresponding 2-SAT is satisfiable. Thus, we have reduced UNREACHABILITY to 2-SAT. This reduction can be accomplished in log-space.

As UNREACHABILITY is coNL complete, considering the above reduction, 2-SAT is coNL complete. But, according to the Immermann Szelepcsenyi theorem, NL=coNL. Hence, 2-SAT is NL complete. □

2. Question

Problem 8.4.4: Generic complete problems. Show that all languages in TIME(f(n)) reduce to { M; x : M accepts x in $f(|x|)$ steps }, where $f(n) > n$ is a proper complexity function. Is this language in TIME(f(n)).

Repeat for nondeterministic classes and for space complexity classes.

2.1. **Solution.**

Lemma 2.1.1. *Every $L \in TIME(f(n))$ can be reduced to { M; x : M accepts x in $f(|x|)$ steps }.*

Proof. First, let us consider the language { M; x : M accepts x in $f(|x|)$ steps }. Let us call this language H_f. H_f accepts only strings of the form M;x, and that too, only if the turing machine corresponding to the string M accepts the string x in $f(|x|)$ steps. (If, instead of TIME(f(n)), we were defining generic complete problems for classes like P and NP, this description would need to be changed slightly.)

If $L \in TIME(f(n))$, then, there exists a deterministic turing machine which decides L in $f(|x|)$ steps. Let the string representation of this turing machine be M. A string x belongs to L if and only if M accepts it in $f(|x|)$ steps.

Every input x to M can be changed to an input of the form M:x. A universal turing machine, equipped with a yardstick to keep track of computation time, can then simulate M and check if M would have accepted x in $f(|x|)$ steps. The universal turing machine then accepts the string M:x if and only if M would have accepted x in $f(|x|)$ steps.

Hence, every decision problem about every $L \in TIME(f(n))$ can be reduced to some instance of the decision problem H_f.

Note that the yardstick is necessary because inputs to H_f can include representations of turing machines which require more than f(n) (even infinite) steps on the string x. □

Remark 2.1.2. $H_f \notin TIME(f(n))$. This can be proved by the diagonalization technique, as was done in the proof of Lemma 7.2 in [1]. However, we note that $H_f \in TIME(f(n))^3$, as was shown in Lemma 7.1 in [1].

Lemma 2.1.3. *Every $L \in NTIME(f(n))$ can be reduced to { M; x : M accepts x in $f(|x|)$ steps }.*

Proof. If $L \in NTIME(f(n))$, then, there exists a non-deterministic turing machine which decides L in $f(|x|)$ steps. Let the string representation of this turing machine be M. A string x belongs to L if and only if M accepts it in $f(|x|)$ steps.

Every input x to M can be changed to an input of the form M:x. A universal non-deterministic turing machine, equipped with a yardstick to keep track of computation time, can then simulate M and check if M would have accepted x in $f(|x|)$ steps. The universal non-deterministic turing machine then accepts the string M:x if and only if M would have accepted x in $f(|x|)$ steps.

Note: Non-determinism can be simulated, just as it was in the proof reduction used in Cook's theorem.

Hence, every decision problem about every $L \in NTIME(f(n))$ can be reduced to some instance of the decision problem NH_f. □

Remark 2.1.4. $NH_f \notin NTIME(f(n))$. (In claiming this, I assume non deterministic equivalents of the time hierarchy theorems.)

Lemma 2.1.5. *Every $L \in SPACE(f(n))$ can be reduced to $\{$ M; x : M accepts x using only $f(|x|)$ space$\}$.*

Proof. If $L \in SPACE(f(n))$, then, there exists a deterministic turing machine which decides L using only $f(|x|)$ space. Let the string representation of this turing machine be M. A string x belongs to L if and only if M accepts it using only $f(|x|)$ space.

Every input x to M can be changed to an input of the form M:x. A universal turing machine, equipped with a yardstick to keep track of the space used, can then simulate M and check if M would have accepted x it using only $f(|x|)$ space. The universal turing machine then accepts the string M:x if and only if M would have accepted x using only $f(|x|)$ space.

Hence, every decision problem about every $L \in SPACE(f(n))$ can be reduced to some instance of the decision problem SH_f.

□

Remark 2.1.6. As log (f(n)) space is sufficient for the space yardstick, $SH_f \in SPACE(f(n))$.

3. Question

Problem 10.4.6: A strong nondeterministic Turing machine is one that has three possible outcomes: "yes", "no" and "maybe". We say that such a machine decides L if this is true: If $x \in L$, then all computations end up with "yes" or "maybe", and at least one with "yes". If $x \notin L$, then all computations end up with "no" or "maybe", and at least one with "no". Show that L is decided by a strong nondeterministic machine if and only if $L \in NP \cap coNP$.

3.1. **Solution.**

Lemma 3.1.1. *If L has a strong NDTM (non deterministic turing machine), then $L \in NP \cap coNP$.*

Proof. If L has a strong NDTM, we know that, if $x \in L$, then all computations end up with "yes" or "maybe", and at least one with "yes". (It is implicit in the question that all these computations terminate in polynomial time.) This NDTM can be trivially modified so as to replace all the "maybe" answers with a "no" answer, which satisfies the requirement for $L \in NP$.

If L has a strong NDTM, we know that its negation, L' also has a strong NDTM. (One only needs to interchange "yes" and "no" answers in the original strong

NDTM.) By the same argument used earlier in this proof, $L' \in NP$. Hence, it follows that $L \in coNP$. □

Lemma 3.1.2. *If $L \in NP \cap coNP$, then L has a strong NDTM (non deterministic turing machine).*

Proof. If $L \in NP \cap coNP$, then for any x, there is either an acceptance-certificate or a rejection-certificate, but not both. An acceptance certificate can be used to verify x's membership in L in polynomial time. A rejection certificate can be used to verify x's non-membership in L in polynomial time. For any language in $L \in NP \cap coNP$, the length of the certificate is bounded by some polynomial of the input size.

Thus, one can conceive of the following NDTM:

- For any input x to the NDTM, guess both an acceptance certificate and a rejection certificate. (Just generate, using the different computation paths, all possible strings under a certain length determined by $|x|$ and L.)
- Using the certificates generated, verify x's membership and non-membership in L.
- If one of the certificates prove without doubt x's membership or non-membership, terminate with the appropriate answer ("yes" and "no" respectively). Otherwise, terminate with the result "maybe".

Thus, we have a strong NDTM for every $L \in NP \cap coNP$. □

4. Question

Problem 11.5.18: Show that, if $NP \subseteq BPP$ then RP=NP. (That is, if SAT can be solved by randomized machines, then it can be solved by randomized machines with no false positives, presumably by computing a satisfying truth assignment as in Example 10.3.)

4.1. **Solution.**

Lemma 4.1.1. *If $NP \subseteq BPP$ then SAT can be solved by an algorithm in RP.*

Proof. SAT is in NP. Its certificate, an assignment to all variables, can be checked by a deterministic algorithm in polynomial time. Let us call this algorithm ALG-CHECKER.

If $NP \subseteq BPP$, SAT has an algorithm in BPP which solves SAT. Let us call this algorithm the BPP Oracle. The BPP oracle runs in polynomial time, as it belongs to BPP.

Now, consider the following RP algorithm (ALG-RP-SAT) to solve SAT (Let F be the input formula.):

- While there is a variable, v, in F which has not been assigned a value, do the following:

  Assign the value True to v. Copy and modify F by replacing all instances of v in F with True.

  Ask BPP oracle if the modified formula is satisfiable. If BPP oracle says "yes", then proceed with the next iteration of this loop. Otherwise, do the following.

  Assign the value False to v. Copy and modify F by replacing all instances of v in F with False.

Ask BPP oracle if the modified formula is satisfiable. If BPP oracle says "yes", then proceed with the next iteration of this loop. Otherwise, reject F.

- Once all variables in F have been assigned a value, use ALG-CHECKER to check if the resulting assignment certifies that F is satisfiable. If the answer is yes, then accept F. Otherwise, reject F.

Note that ALG-RP-SAT terminates in polynomial time, as every step described above can be completed in polynomial time. Furthermore, ALG-RP-SAT guarantees that there are no false positieves. The BPP oracle can be set up to repeat n times, so as to make its error probability exponentially small, so that for all satisfiable formulae, ALG-RP-SAT is correct atleast half the time.

So, ALG-RP-SAT is in RP.

□

Lemma 4.1.2. *If $NP \subseteq BPP$ then $NP = RP$.*

Proof. We already know that $RP \subseteq NP$. As SAT is NP complete, and because ALG-RP-SAT solves SAT, we see that $NP \subseteq RP$. Hence, $NP = RP$. □

References

[1] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, November 1993.