6.840 **Dan Ports**
Theory of Computation drkp@mit.edu
2004/11/18

Problem 5-1

Theorem 1. *Any language* $A \in \mathsf{P} \setminus \{A = \varnothing, A = \Sigma^*\}$ *is* NP*-complete if* $\mathsf{P} = \mathsf{NP}$*.*

Proof. Let A be a language in P that is not $\varnothing$ or Σ^*. Note that there are therefore strings $\aleph \in A$ and $\beth \notin A$.

We give a reduction from SAT to A to show that A is NP-complete. Suppose ϕ is an instance of SAT. Our reduction is as follows:

1. Solve ϕ. This can be done in polynomial time, since $\mathsf{P} = \mathsf{NP}$.
2. If ϕ is solvable, let $\psi_\phi = \aleph$. If it is not, let $\psi_\phi = \beth$.
3. Output ψ_ϕ.

The reduction runs in polynomial time, and if ϕ is solvable it outputs $\aleph \in A$, and $\beth \notin A$ if not. Thus, SAT is reduced to A, and A is NP-complete. So every language in $\mathsf{P} \setminus \{A = \varnothing, A = \Sigma^*\}$ is NP-complete if $\mathsf{P} = \mathsf{NP}$. $\square$

6.840 **Dan Ports**

Theory of Computation `drkp@mit.edu`

2004/11/18

Problem 5-2

Theorem 2. *For a CNF-formula ϕ with m variables and c clauses, there is a* NFA *with $O(cm)$ states that accepts all non-satisfying assignments, represented as m-length Boolean strings.*

Proof. Let ϕ be given as above. We give a NFA N that accepts non-satisfying assignments of ϕ. The NFA contains c "sub-machines", corresponding to each clause of ϕ. For each clause $\mathfrak{c}$, we create a state $\Game_{\mathfrak{c},i}$ corresponding to the variable x_i ($1 \leq i \leq m$), plus an accept state $\Game_{c,m+1}$. We connect $\Game_{\mathfrak{c},i}$ to $\Game_{\mathfrak{c},i+1}$ with a $\mathtt{0}$ transition if variable x_i appears non-negated in clause $\mathfrak{c}$, with a $\mathtt{1}$ transition if variable x_i appears negated in clause $\mathfrak{c}$, and with both if x_i does not appear in $\mathfrak{c}$. We then create a start state that is connected by a ϵ-transition to all of the states $\Game_{\mathfrak{c},1}$.

This uses $m+1$ states per each of the c clauses, plus the start state, so the number of states is $O(cm)$.

This NFA gives the correct result. It accepts if the sequence of assignments given by the input does not satisfy one of the clauses in ϕ, and therefore does not satisfy ϕ; it rejects otherwise. $\square$

Corollary 3. NFA*s cannot be minimized in polynomial time unless* $\mathsf{P} = \mathsf{NP}$.

6.840 **Dan Ports**
Theory of Computation drkp@mit.edu
2004/11/18

Problem 5-3

Theorem 4. $EQ_{\mathsf{REX}} \in \mathsf{PSPACE}$

Proof. $\square$

6.840 **Dan Ports**
Theory of Computation drkp@mit.edu
2004/11/18

Problem 5-4

Theorem 5. $GM \in \mathsf{PSPACE}$

Proof. Go-moku is clearly a combinatorial game, we can use the standard approach of inspecting the game tree to identify winning strategies. Note that polynomial space is required to check whether any player has won in a given configuration, since we can simply test all of the possible 5-lines individually. We need only bound the depth of the game tree.

Lemma 6. *At most n^2 moves can be made from any board position.*

Proof. Markers can never be removed after they are placed. The board is $n \times n$, so it has n^2 spaces. Each player places a marker on every move. It is therefore not possible to make more than n^2 moves, even if the board starts empty. □

Corollary 7. *The maximum depth of the game tree is polynomial in n.*

Proof. Immediate from Lemma 6. □

A representation of the board position is polynomial in n, since we simply need to store the state of each of the n^2 squares. So we can store the game history in space polynomial in n, since the size of this is simply the size of the board position multiplied by the depth of the game tree. We need only keep a representation of the game history when testing all possible outcomes in the game tree to check whether there is a winning strategy, so polynomial space is required. Thus $GM \in \mathsf{PSPACE}$. □

6.840 **Dan Ports**
Theory of Computation `drkp@mit.edu`
2004/11/18

Problem 5-5

Theorem 8. *HAPPY-CAT* $\in$ P.

Proof. The game is combinatorial, so we can generate and scan the game tree to determine whether the Cat player has a winning strategy. The game tree is exponential in size, but we do not need to consider all of it:

$O\left(n^2\right)$ possible game states must be examined. Each of the two players can be on any of the n vertices of the graph, and either of the two players may be next up to move. So there are $2n^2$ game states. Recall that the game is defined so that it ends in a draw if both players are in the same position and it is the same player's turn to move as a previous game state. Thus, no game state needs to be examined twice when traversing the game tree, since this gives an immediate draw. So inspecting the game tree to determine whether the cat player has a winning strategy takes $O\left(n^2\right)$ time, and *HAPPY-CAT* $\in$ P. $\square$

6.840 **Dan Ports**
Theory of Computation drkp@mit.edu
2004/11/18

Problem 5-6

a)

Definition 9. *EQ-FORMULA* $= \{\phi_1, \phi_2 \mid \phi_1$ *and* ϕ_2 *are equivalent formulas*$\}$

Lemma 10. $\overline{\textit{EQ-FORMULA}} \in \mathsf{NP}$

Proof. Let the certificate be an assignment of variables such that ϕ_1 and ϕ_2 differ. We can compute in polynomial time the values of ϕ_1 and ϕ_2 for this assignment and thereby verify the certificate. $\square$

Corollary 11. *EQ-FORMULA* $\in \mathsf{coNP} \subseteq \mathsf{PSPACE}$

Theorem 12. *MIN-FORMULA* $\in \mathsf{PSPACE}$

Proof. Given a formula ϕ, we give a procedure for determining whether there is a shorter formula ϕ' equivalent to ϕ: for every formula $\eth$ shorter than ϕ, we check whether $\langle \phi, \eth \rangle \in$ *EQ-FORMULA*. If any such $\eth$ exists, we reject; otherwise, we accept. Since *EQ-FORMULA* $\in \mathsf{PSPACE}$, each check can be done using polynomial space in $|\phi|$. Indeed, the entire procedure requires only polynomial space, since we simply need to store on the tape the current formula $\eth$ being processed, process it, then find the next $\eth$ and overwrite the tape. So *MIN-FORMULA* $\in \mathsf{PSPACE}$ $\square$

b) It is true that a NTM can verify that $\phi \notin$ *MIN-FORMULA* by guessing the smaller equivalent formula ϕ'. However, it cannot necessarily do so *in polynomial time.*