CS221: Computational Complexity Prof. Salil Vadhan

Lecture 11: **PSPACE**-completeness and the Polynomial Hierarchy

10/16 *Scribe: Alex Healy*

Contents

Introduction

Recall that **AP** = **PSPACE**, and so we will often prove that a language is **AP**-complete in order to prove that it is **PSPACE**-complete. Games between two players are particularly well-suited to the alternation model since, for instance, the question of whether black has a winning strategy in Go[1] is the same as asking whether there exists a move for black, such that for all moves for white, there exists a move for black, ..., there exists a move for black that is a winning move. A consequence of this is that many **PSPACE**-completeness results involve competitive games between two players.

1 PSPACE-complete Problems

Quantified Boolean Formula

The first **PSPACE**-complete language we will consider is QUANTIFIED BOOLEAN FORMULA, which captures the power of alternation in a very natural way:

$$\mathrm{QBF} = \{\text{3-CNF } \varphi(x_1,\ldots,x_n) \ : \ \exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \varphi(x_1,\ldots,x_n) = 1\}$$

Theorem 1 QBF *is* **PSPACE**-*complete.*

Proof Sketch: We will show that QBF is **PSPACE**-complete by showing that it is **AP**-complete.

It is not hard to see that QBF $\in$ **AP**; if we use the relation characterization of **AP**, then we simply define a polynomial-time relation $R(\varphi, \vec{y})$ which is 1 if $\varphi(\vec{y}) = 1$ and 0 otherwise.

Now it remains to show that QBF is **AP**-hard. Given a language $L \in$ **AP**, there is an associated polynomial-time relation $R(x, y)$ (again using the relation characterization of **AP**). Given x, we want to construct a 3-CNF formula φ with the property that

$$\exists y_1 \forall y_2 \cdots Q_n y_n R(x,y) = 1 \Longleftrightarrow \exists y_1 \forall y_2 \cdots Q_m y_m \varphi(\vec{y}) = 1$$

[1] Unlike chess, black plays first in Go.

We can replace the relation R with a 3-CNF φ much in the same way as we replaced a Boolean circuit with a 3-CNF in the Cook-Levin theorem. In particular, given a relation $R(x, y)$ we can construct a 3-CNF formula $\varphi_R(\vec{x}, \vec{y}, \vec{z})$ such that

$$R(x, y) = 1 \iff \exists \vec{z} \varphi_R(\vec{x}, \vec{y}, \vec{z}) = 1$$

Once we have such a relation, we obtain a reduction to QBF simply by hardwiring the given instance $\vec{x}$ into the formula φ_R. ($\vec{y}$ and $\vec{z}$ remain quantified as part of the QBF instance produced.)

How did we get φ_R from R? Recall that this is accomplished by first transforming R into a Boolean circuit which has the values from $\vec{x}$ and $\vec{y}$ as inputs. Then, just as in the Cook-Levin theorem, we may introduce new variables which represent the values of the function at each gate in the circuit. The vector $\vec{z}$ will include all the input values to the circuit as well as all these intermediate values at each gate, in such a way that there is a $\vec{z}$ where $\varphi_R(\vec{x}, \vec{y}, \vec{z}) = 1$ if, and only if, there is an assignment of the inputs of the boolean circuit and a valid computation on that input (both described by $\vec{z}$) that produces 1 as the output. Finally, the bulk of the effort of the Cook-Levin theorem was in showing that the disjunction of 3 literals is rich enough to decide whether the output of a gate is consistent with its inputs. In particular, deciding whether the 3-CNF that results by taking the conjunction of all such disjunctions is equivalent to deciding whether the circuit has a satisfying assignment. Applying this same reasoning to in this case proves the result. $\square$

(Generalized) Geography

The game Geography is played as follows. Two players alternate turns naming cities such that the first letter of the city is the same as the last letter of the previously named city. A city may not be named more than once, and a player wins when his opponent fails to mention a valid city during his turn. For example:
Player 1: BOSTO**N**
Player 2: **N**AIROB**I**
Player 1: **I**STANBU**L**
Player 2: **L**ONDO**N**
Player 1: "I give up!"
(*i.e.,* Player 1 knows of no cities starting with "N", other than "Nairobi" which cannot be replayed, so he looses.)

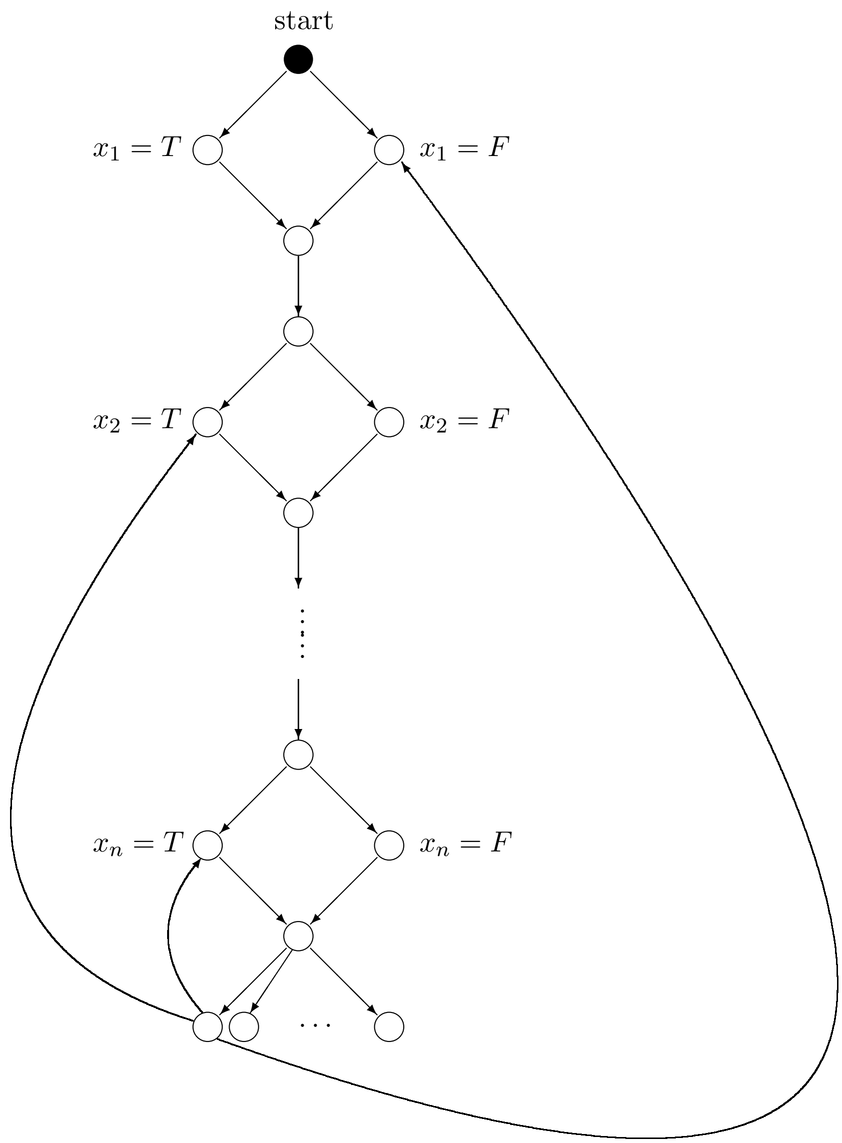
In fact, we will study a generalization of Geography, defined as follows: An instance of Generalized Geography is a directed graph G (where the vertices represent cities in the original game) and a start vertex, such that $(u, v) \in E$ if, and only if, a player may name v after u. Players alternate picking subsequent vertices in a (directed) walk on G, and no vertex may be chosen more than once. A player wins if his opponent has no legal moves at some point during the game. From this we define a language:

$$\text{GGEO} = \{G \ : \ \text{Player 1 has a winning strategy}\}$$

Theorem 2 GGEO *is* **PSPACE**-*complete.*

Proof: First we note that GGEO $\in$ **AP** $=$ **PSPACE**. This can be seen as follows: Use $\exists$ states to nondeterministically guess the moves of player 1 and use the $\forall$ to nondeterministically guess the moves of player 2; finally, use the polynomial-time computation to verify that player 1 wins. Observe that (a) there can be at most a polynomial number of moves and that (b) the outcome can be determined in polynomial time. This is not the case with some games, such as Chess or Go which can take super-polynomially many moves or whose rules are complicated enough as to require super-polynomial time to determine the winner.

Next we will show that GGEO is **PSPACE**-hard. Given a quantified 3-CNF formula $\varphi(x_1, \ldots, x_n)$ we will construct an instance of GGEO that looks as follows:



Player 1 gets to chose the value of x_1 (corresponding to a nondeterministic existential choice, $\exists x_1$). Then player 2's move is forced, as is player 1's subsequent move. Next player 2 gets to choose the value of x_2 (corresponding to a nondeterministic universal choice $\forall x_2$), and so on. Intuitively, player 1 is trying to satisfy the boolean formula and player 2 is trying to make it unsatisfied. Without loss of generality, we may assume that n is even, so player 2 chooses x_n. Player 1's move is then forced. Now, the graph branches into a collection of vertices, each of which corresponds to a clause in the 3-CNF formula. At this point, player 2 is trying to choose a term that is unsatisfied (and hence, as we shall see, force player 1 to lose). Each node corresponding to a clause in the 3-CNF is connected to the *negation* of the 3 literals in that clause; for instance, the clause node that is shown above corresponds to the clause $(x_1 \vee \neg x_2 \vee \neg x_n)$. This way, player 1 has a valid move if and only if the clause was satisfied. Hence, player 1 has a wining strategy if and only if the original quantified boolean formula is true.

■

1.1 Other PSPACE-complete Languages

There are many other examples of **PSPACE**-complete problems such as deciding whether there is a winning strategy for Go (where a bound is placed on the number of moves), certain generalizations of Chess, Checkers and recognizing context-sensitive grammars (as shown on problem set 2).

1.2 Alternating Space

We have already seen how $\mathbf{AP} = \mathbf{PSPACE}$, and more generally how alternating time is equivalent to deterministic space (with the same bound). But what is the relationship between alternating space and deterministic time? Although we won't prove it in this course, we have the following theorem.

Theorem 3 $\mathbf{AL} = \mathbf{P}$

The proof of this is obtained by showing that MONOTONE CIRCUIT VALUE is **AL**-complete. Roughly, this is done by using OR gates to make the decisions in the existential ($\exists$) states, and the AND gates to make the decisions in the universal ($\forall$) states of an **AL** machine. By a padding argument, we get that alternating space $O(f(n))$ is equal to deterministic space $2^{O(f(n))}$ so, for example, $\mathbf{APSPACE} = \mathbf{EXP}$.

2 The Polynomial Hierarchy

The *Polynomial hierarchy*, **PH**, is complexity theory's analog to the *arithmetic hierarchy* (or Kleene hierarchy) in recursion theory.

In particular, we define the class $\mathbf{\Sigma_k P}$ to be the languages that are decidable by polynomial-time algorithms where every path of computation has at most $k-1$ alternations between $\exists$ and $\forall$, starting with $\exists$.

Similarly, $\mathbf{\Pi_k P}$ is the class of languages that are decidable by polynomial-time algorithms where every path of computation has at most $k-1$ alternations between $\exists$ and $\forall$, starting with $\forall$.

The following basic observations follow almost immediately from these definitions:

$$\mathbf{\Sigma_1 P} = \mathbf{NP} \quad \text{and} \quad \mathbf{\Pi_1 P} = \textbf{co-NP}$$

$$\mathbf{\Sigma_k P} = \mathbf{co\text{-}\Pi_k P} \quad \text{and} \quad \mathbf{\Pi_k P} = \mathbf{co\text{-}\Sigma_k P}$$

$$\mathbf{\Sigma_k P} \subseteq \mathbf{\Pi_{k+1} PP} \quad \text{and} \quad \mathbf{\Pi_k P} \subseteq \mathbf{\Sigma_{k+1} P}$$

$$\mathbf{PH} = \bigcup_k \mathbf{\Sigma_k P} \subseteq \mathbf{AP} = \mathbf{PSPACE}$$

For example, define the following Circuit Minimization language:

$$\text{CKT-MIN} = \{(C, k) \ : \ \text{there is a circuit } C' \text{ of size at most } k \text{ such that } C' \equiv C\}$$

Deciding this language is computationally equivalent to determining the size of the smallest circuit that computes a given function. CKT-MIN $\in \Sigma_2\mathbf{P}$ since it asks whether there *exists* a small circuit which, *for all* inputs, computes the same function as C.

A related problem is the following Minimum Circuit:

$$\text{MIN-CKT} = \{C \ : \ C \text{ is the smallest circuit computing its function}\}$$

MIN-CKT $\in \Pi_2\mathbf{P}$ since it is asking whether, *for all* circuits C' smaller than C, there *exists* an input x such that $C(x) \neq C'(x)$. Another related problem which has only been settled recently, despite being open for many years, is that DNF Minimization language is $\Sigma_2\mathbf{P}$-complete.