The $\texttt{PSPACE}$ Complexity Class

Last Updated April 19th, 2024

1 Polynomial Space

Definition 1.1. $\texttt{PSPACE}$ represents those decision problems that are decidable using a polynomial amount of space. In other words,

$$\texttt{PSPACE} = \bigcup_{k \geq 1} \texttt{SPACE}(n^k).$$

Proposition 1.2. The following inclusions hold.

$$\texttt{P} \subseteq \texttt{NP} \subseteq \texttt{PH} \subseteq \texttt{PSPACE} = \texttt{NPSPACE} \subseteq \texttt{EXPTIME}.$$

Proof. The first inclusion is Theorem 4.9 of the Complexity lecture. The second inclusion follows from the definition of $\texttt{PH}$. The equality $\texttt{PSPACE} = \texttt{NPSPACE}$ follows from Savitch's theorem, and the final inclusion follows from Lemma 1.5 of the Space Complexity lecture. All that remains to show is that $\texttt{PH} \subseteq \texttt{PSPACE}$. To this end, let $L \in \Sigma_k^p$ be given, and let $Q_1 x_1 \cdots Q_k x_k \ p(x_1, \ldots, x_k, y)$ be the predicate function associated with L. Consider the following recursive algorithm for evaluating L's predicate function.

Name: `eval`

Inputs:

1. instance y of decision problem L,
2. (possibly empty) list $L = [c_1, c_2, \ldots, c_l]$, $l \geq 1$, $c_j \in C_j = \mathrm{dom}(x_j)$, $1 \leq j \leq l$.

Output: $Q_{l+1}x_{l+1} \cdots Q_k x_k \; p(c_1, \ldots, c_l, x_{l+1}, \ldots, x_k, y)$.

If $l = \mathrm{length}(L) = k$, then return $p(c_1, \ldots, c_k, y)$.

If $l+1$ is odd, then $//Q_{l+1} = \exists$

For each $d \in C_{l+1}$,

If `eval`$(y, L + d) = 1$, then return 1.

Return 0.

Else $//Q_{l+1} = \forall$

For each $d \in C_{l+1}$,

If `eval`$(y, L + [d]) = 0$, then return 0.

Return 1.

Since predicate function p is decidable in polynomial-time, it is also decidable using a polynomial amount of space. Therefore, the base is computable in a polynomial amount of space. As for the recursive cases, notice that the required memory consists of i) at most k counters, each having $\mathrm{O}(q(|y|)$ bits and ii) list L whose size is also $\mathrm{O}(q(|y|)$ which is a bound for each of the at most k certificates that are stored in L at any given time. Therefore, the algorithm requires a polynomial amount of space. $\square$

2 Quantified Boolean Formula

Definition 2.1. A **quantified Boolean formula** is a Boolean formula of the form

$$Q_1 x_{i_1} \cdots Q_k x_{i_k} \; \phi(x_1, \ldots, x_n),$$

where each Q_j is a quantifier, each x_{i_j} a Boolean variable, $1 \le j \le k$, and $\phi(x_1, \ldots, x_n)$ is a Boolean formula.

Notes:

1. The above definition actually defines a special case of QBF's known as those in **prenex normal form**, meaning that the quantifiers are written first, followed by an unquantified Boolean formula ϕ. However, every QBF F is logically equivalent to one, call it F', that is in prenex normal form and the construction of F' can be done efficiently relative to the size of F.

2. When each variable x_i of ϕ is bounded by a quantifier Q_i, then we have what is called a **totally quantified Boolean formula (TQBF)**, and in this case it follows that every TQBF has a single evaluation of either 0 or 1. For example, the Boolean formula $x \vee y$ has four different evaluations depending on each of the four possible ways of assigning the variables x and y. However, the statment $\exists x \forall y (x \vee y)$ has the single evaluation of 1, since it is a true statement that "there exists an x (namely $x = 1$) such that, for every possible way of assigning a value to y (0 or 1), $x \vee y$ evaluates to 1". In what follows, we assume each QBF of interest is a TQBF.

Example 2.2. Consider a graph $G = (A \cup B \cup C, E)$ that has three sets of vertices: $A = \{a, b, c\}$, $B = \{1, 2, 3, 4\}$, and $C = \{\alpha, \beta\}$. For each of the following predicate logic formulas, draw the smallest possible (in terms of number of edges) version of G for which the formula evaluates to true. Each statement assumes that i) $\text{dom}(x) = A$, ii) $\text{dom}(y) = B$, and $\text{dom}(z) = C$.

1. $\exists x \forall y \exists z \ ((x, y) \in E \land (y, z) \in E)$
2. $\forall x \exists y \forall z \ ((x, y) \in E \land (y, z) \in E)$

Solution.

Example 2.3. Evaluate the QBF $\forall x \exists y ((x \vee y) \wedge (\overline{x} \vee \overline{y}))$.

Solution.

Example 2.4. Evaluate the QBF $\exists x \forall y \forall z ((x \vee \overline{y}) \wedge (z \wedge x))$.

Solution.

3 PSPACE Completeness

Definition 3.1. A decision problem B is said to be `PSPACE`**-complete** iff

1. $B \in$ `PSPACE`
2. for every other decision problem $A \in$ `PSPACE`, $A \leq_m^p B$.

Definition 3.2. An instance of `Totally Quantified Boolean Formula (TQBF)` is totally quantified Boolean formula F, and the problem is to decide if F evaluates to 1.

It may not seem too surprising that `TQBF` is our first `PSPACE`-complete problem since it generalizes both `SAT` and the Polynomial Hierarchy for the following reasons.

1. Every instance $F(x_1, \ldots, x_n)$ of `SAT` is a special case of `TQBF` in that the satisfiability of F is equivalent to
$$\exists x_1 \cdots \exists x_n \; F(x_1, \ldots, x_n)$$
evaluating to 1.
2. For every $k \geq 0$, an instance y of a decision problem $L \in \Sigma_k^p$ (respectively, $L \in \Pi_k^p$) may be expressed as an instance of `TQBF` via a binary encoding of y, each of the certificate variables $x_1, \ldots, x_k$, along with a conversion of the predicate function p to a Boolean formula whose variables are the Boolean encoding variables of $y, x_1, \ldots, x_k$.
3. The recursive algorithm similar to the one presented in the proof of Proposition 1.2 may be used to evaluate a TQBF using a polynomial amount of space.

Theorem 3.3. TQBF is PSPACE-complete.

Proof. Let $L \in$ PSAPCE be given via DTM M that decides L using an amount of space that is bounded by n^k, for some constant $k \geq 1$, where $n = |x|$ is the size of input instance x of L. We must construct a TQBF F that evaluates to 1 iff $M(x) = 1$ and whose size (i.e. number of variables, quantifiers, and logic operations used) is also bounded by a polynomial with respect to n. The strategy we use is to represent as a TQBF the statement

$$\phi(c_1, c_2, t)$$

which evaluate to 1 iff configuration c_2 is reachable from c_1 in t or fewer steps, where c_1 and c_2 are possible configurations that may appear in the computation of M on input x. Thus, these configurations have a length at most $O(n^k)$ and t has a size that is no greater than 2^{dn^k}, for some constant integer $d > 0$. Thus, thinking of ϕ as a function, we would intially set $c_1 = c_{\text{init}}$, $c_2 = c_{\text{final}}$, and $t = 2^{dn^k}$, where c_{init} is an initial configuration with x placed on the tape as input, and c_{final} is a unique accepting state (say, in the accept state with the head at cell 1, all cells holding some common symbol in Γ).

Now, given arbitrary c_1, c_2, and t, We now provide a recursive formula for expressing $\phi(c_1, c_2, t)$ as a TQBF of polynomial size with respect to $n = |x|$.

Base Case $t = 1$. In this case $\phi(c_1, c_2, t)$ may either written as the Boolean formula $c_1 = c_2$ or as the Boolean formula that evaluates to 1 iff c_2 is reachable from c_1 in a single step.

For the case of $c_1 = c_2$, since each configuration can be expressed using $O(n^k)$ Boolean variables, and we must check for pairwise equality between the variables used to encode c_1 and the variables used for c_2, it follows that $c_1 = c_2$ may be expressed with a (quantifier free) Boolean formula having size $O(n^k)$.

As for the latter case, see the proof of Cook's theorem for a list of each of the (quantifier free) Boolean formulas that must be supplied in order to check if c_2 is the next configuration after c_1. Together, these formulas have a size equal to $O(n^{2k})$.

Recursive Case $t = 2^j$, $j \geq 1$. Then for $\phi(c_1, c_2, t)$ to evaluate to 1, there must exist a middle configuration m such that for all configurations c_3 and c_4, if $c_3 = c_1$ and $c_4 = m$, of if $c_3 = m$ and $c_4 = c_2$, then $\phi(c_3, c_4, \frac{t}{2})$ must evaluate to 1. As a predicate-logic formula this may be written as

$$\exists m \forall c_3 \forall c_4 \ (((c_3 = c_1) \wedge (c_4 = m)) \vee ((c_3 = m) \wedge (c_4 = c_2)) \rightarrow \phi(c_3, c_4, \frac{t}{2})).$$

Notice that, because of the clever use of alternating quantifiers, the recursion tree has a *single branch* with depth $\log(2^{dn^k}) = dn^k$. Moreover, at each (non-leaf) level of the tree, we add $O(n^k)$ amount of quantifiers, Boolean variables, and logic operations. This is because the configurations m, c_3, and c_4 are encoded using $O(n^k)$ Boolean variables, and the four equality statements, as noted in the base case description, require at most $O(n^k)$ operations. Finally, at the bottom level of recursion when $t = 0$, $O(n^{2k})$ additional operations are required. Thus, the final formula size equals

$$O(n^k n^k) + O(n^{2k}) = O(n^{2k})$$

which is a polynomial in the size of input x. $\square$

4 More PSPACE-Complete Problems

One way of interpreting the `TQBF` problem is thinking of it as a game between two players: $\exists$ and $\forall$. Without loss of generality, we may assume that the quantifier sequence for a `TQBF` instance

$$F = \exists x_1 \forall x_2 \cdots Q_m x_m \phi(x_1, \ldots, x_m)$$

begins with $\exists$ and alternates between $\exists$ and $\forall$. The goal for player $\exists$ is to assign values to the $\exists$-variables that will ensure that formula ϕ evaluates to 1. On the other hand, for each assignment made by $\exists$, player $\forall$ tries to counter by assigning a value to the next variable that will ensure that ϕ evaluates to 0. Thus, player $\exists$ has a winning strategy iff there is some way of assigning the $\exists$ variables so that, regardless of how $\forall$ counters with its assignments to its own variables, the formula evaluates to 1.

Definition 4.1. An instance of `Formula Game` is a TQBF formula F whose quantifier sequence begins with $\exists$ and alternates between $\exists$ and $\forall$. Moreover, F is a positive instance iff it evaluates to 1, meaning that player $\exists$ can assign its variables in such a way that, regardless of how player $\forall$ assigns its variables, F will evaluate to 1.

Theorem 4.2. `Formula Game` is `PSPACE`-complete.

Proof. Since `Formula Game` is essentially the same decision problem as `TQBF`, the proof of its `PSPACE`-completeness is essentially the same as the proof that `TQBF` is `PSPACE`-complete. □

Example 4.3. Consider the formula

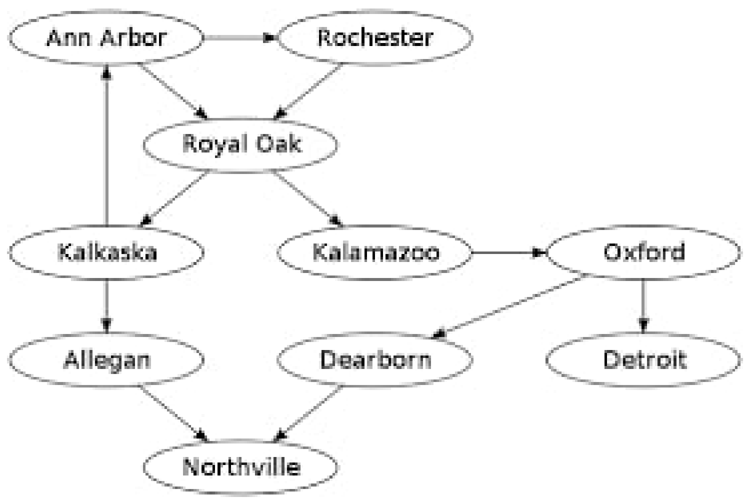
$$\exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3)) \wedge (\overline{x_2} \vee \overline{x_3})].$$

Viewed as an instance of `Formula Game`, who wins the game?

4.1 Generalized Geography

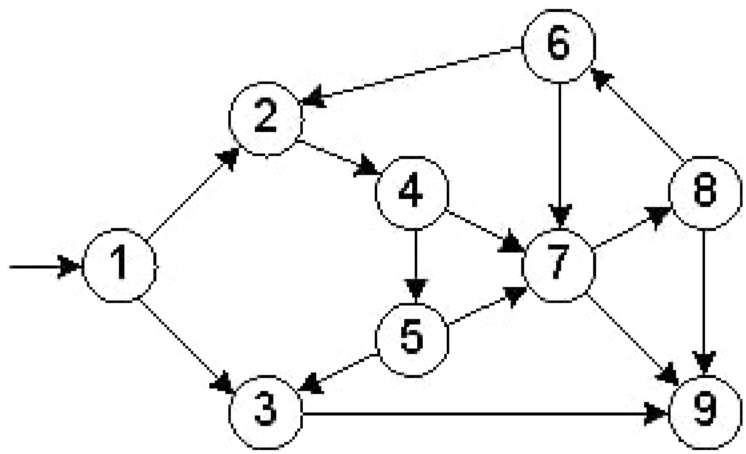
The **geography game** is a 2-player game where players take turns saying the name of a city. The game begins by starting with a city c_0. Then Player 1 must say the name of a city c_1 whose first letter equals the last letter of c_0 and for which $c_1 \neq c_0$. Player 2 must then respond by saying the name of a city c_2 whose first letter begins with the last letter of c_1, and for which $c_2 \notin \{c_1, c_0\}$. Play continues in this manner with players taking turns saying city names until a player is unable to say the name of a city c_k whose first letter equals the last letter of the previously named city c_{k-1} and for which $c_k \notin \{c_0, c_1, \ldots, c_{k-1}\}$. This player loses the game.

One way to visualize the game is by using a directed graph $G = (V, E)$ where each vertex $u \in V$ is labeled with the name of a city, denoted $c(u)$ and there is an edge from u to v iff the last letter of $c(u)$ equals the first letter of $c(v)$.



We may generalize the geography game by defining an instance of the `Generalized Geography` (`GG`) decision problem to be a pair (G, v_0), where $G = (V, E)$ is a directed graph and $v_0 \in V$ is the designated start vertex. Similar to the geography game, `Generalized Geography` is a two-player game where players take turns selecting a vertex from G. The game begins at vertex v_0. Then Player 1 must select a vertex v_1 for which $(v_0, v_1) \in E$. Player 2 must then respond by selecting a vertex v_2 for which $(v_1, v_2) \in E$ and $v_2 \notin \{v_0, v_1\}$. Play continues in this manner with players selecting vertices until a vertex v_{k-1} is selected so that the next player is unable to find a vertex v_k for which $(v_{k-1}, v_k) \in E$ and $v_k \notin \{v_0, v_1, \ldots, v_{k-1}\}$. This player loses the game. Finally, (G, v_0) is a positive instance of `GG` iff Player 1 has a winning strategy.

Example 4.4. Below is an example of an instance of `Generalized Geography`. Is this a positve instance of `GG`?



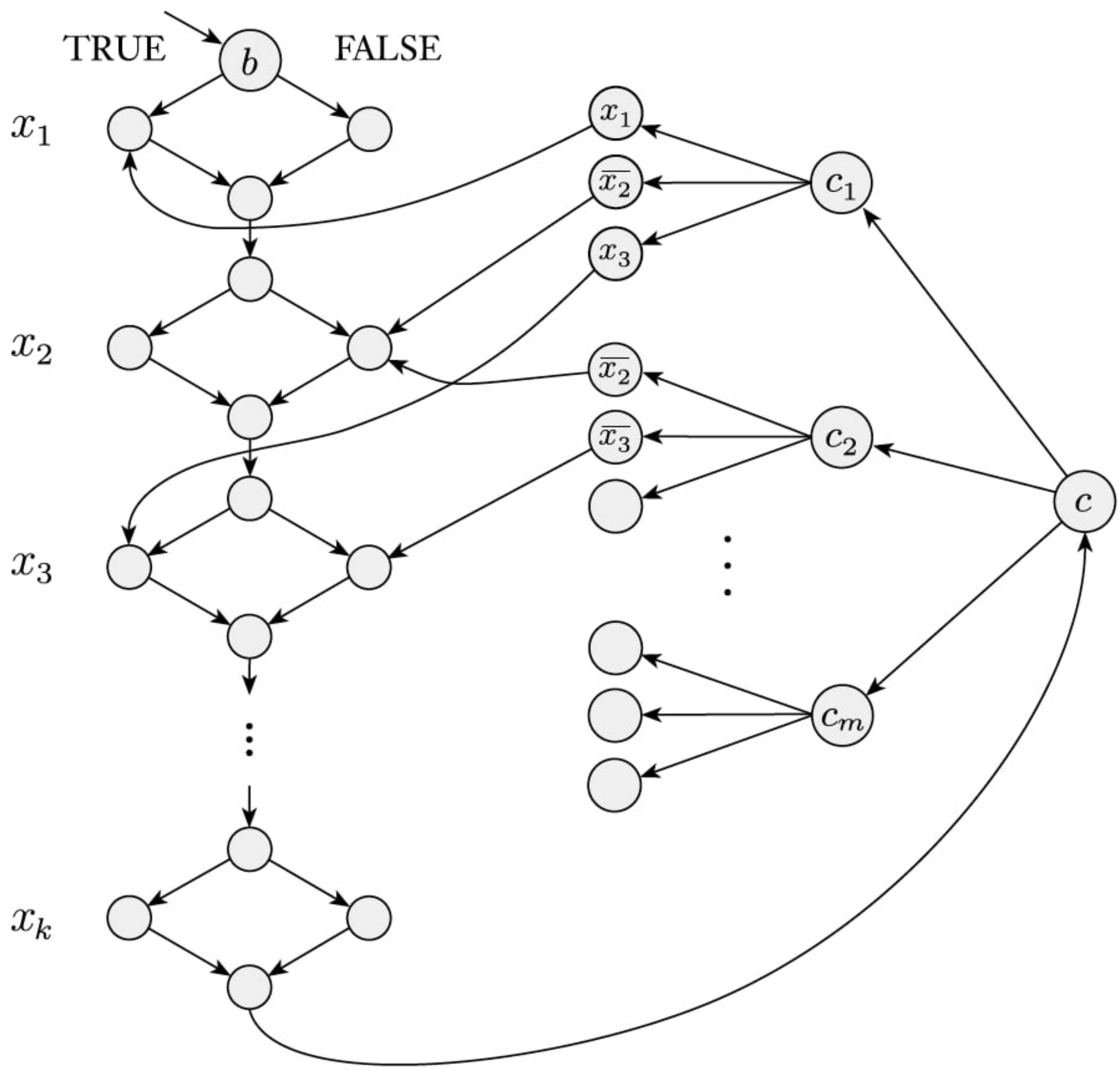
Theorem 4.5. `Generalized Geography` is `PSPACE`-complete.

Proof. `GG` is in `PSPACE` via an algorithm similar to the one used to show that PH is in `PSPACE` (see Proposition 1.2).

We now describe a reduction from `Formula Game` to `GG`. Given an instance $F = \exists x_1 \forall x_2 \cdots \exists x_k \phi(x_1, \ldots, x_k)$ of `Formula Game`, we assume without loss of generality that

1. $Q_1 = Q_3 = \cdots = Q_k = \exists$, where k is odd and
2. ϕ is written in conjunctive normal form for which each disjunction has three literals (i.e. ϕ is an instance of `3SAT`).

Then mapping reduction $f(F) = (G, b)$ maps F to an instance of `GG`, an abstract example of which is shown below.



The idea is that the left half of G has a stacked sequence of k diamond subgraphs, one for each variable x_i. When i is odd (respectively, even), Player 1 (respectively Player 2) must select either the left-corner or the right-corner vertex of diamond x_i. Selecting the left (respectively, right) corner is analgous to assigning variable x_i the value 1 (respectively, 0). Also, since k is odd, Player 1 will select vertex c, while Player 2 then has the option of selecting one of the nodes $c_1, c_2, \ldots, c_m$, where c_i corresponds with clause i of ϕ.

Now suppose that Player 1 has a winning strategy for `Formula Game` instance F. Then Player 1 applies this strategy to (G, b) by choosing the appropriate corner of each diamond that it controls/assigns. Then, after Player 1 has selected vertex c, then regardless of which c_i is selected by Player 2, there will exist at least one literal node, say l_j of c_i that evaluates to 1, meaning that the player who controls/assigns variable x_j selected a corner that corresponds with l_j being assigned 1. Without loss of generality, assume $l_j = x_j$ and Player 1 controls/assigns x_j. Then Player 1 selected the left-corner node n of diamond j since x_j was assigned 1. Moreover, n is the only node that is reachable from literal node x_j in clause c_i. Hence, Player 2 loses since n was already selected by Player 1. Therefore, if F is a positive instance of `Formula Game`, then $f(F)$ is also a positive instance of `GG`, since Player 1 has a winning strategy.

Now suppose Player 1 does *not* have a winning strategy for instance F of `Formula Game`. In this case, regardless of the assignments made by Player 1, Player 2 may assign its variables values in such a way that at least one clause c_i will not be satisfied by the assignment α formed by both players. Player 2 then selects c_i after Player 1 selects c. Now it's Player 1's turn. Without loss of generality, suppose Player 1 selects $\overline{x_j} \in c_i$. Then, since x_j is assigned 0 by α, the left-corner node of diamond x_j has yet to be selected. Thus, Player 2 may select that node. But then that node points only to the bottom node of the diamond, which has already been played, and so Player 1 loses. Therefore, if F is a negative instance of `Formula Game`, then $f(F)$ is also a negative instance of `GG`, since Player 2 has a winning strategy. Therefore f is a valid polynomial-time mapping reduction from `Formula Game` to `GG`. $\square$

Example 4.6. Consider the formula

$$\exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3)) \wedge (\overline{x_2} \vee \overline{x_3})].$$

Viewed as an instance of `Formula Game`. Apply the reduction described in the proof of Theorem 4.5.