On the Polynomial Hierarchy, PSPACE, and Descriptive Complexity

Derek Aoki

June 2024

Abstract

We discuss the complexity class PSPACE, the problems that can be solved by Turing machines in polynomial space, as well as the classes composing the polynomial hierarchy, a generalization of P, NP, and coNP. We introduce the basic complexity-theoretic notions as well as the basic notions of finite model theory and close with logical characterizations of PSPACE and the polynomial hierarchy.

1 Introduction

The theory of computation in general has been studied since the 1930s, even before the advent of the electronic computer, begun with the study of what we could ”compute” with algorithmic systems. The first of these systems to be studied were Church’s lambda calculus and Turing’s Turing machines, both formalizations of the notion of an algorithm or computer program later shown to be equivalent. The first problems of interest were those of computability (what the limits of computation are as a whole), and famously this led to investigation of the halting problem.

However, once actual computers had been developed and had begun to see practical use, the natural notion of the “difficulty” or “complexity” of problems that were being attacked arose: what kind of resources do computers need to solve different “problems”? For example, what is the minimum number of steps required in an algorithm that breaks a given encryption scheme? How hard is it to verify that two graphs are isomorphic using a computer? How much memory does a computer need to be able to find the best strategy in a Go position?

This notion of verification specifically gave rise to the class (or collection) **NP**, the set of problems we can verify solutions to with a computer “easily” and the class **P**, the set of problems we can solve “easily”. The relationship between these two classes can be generalized into an entire hierarchy of classes, each of which contain problems more difficult for computers to solve, requiring more time and space. Containing this entire hierarchy is the class **PSPACE**, the set of problems that can be solved by algorithms that use fairly little computer memory. This hierarchy, called the “polynomial hierarchy”, and **PSPACE** have various interesting relationships, and both have been the subject of much study throughout the history of the field of "computational complexity".

Finally, because the objects computers deal with are necessarily finite, the computers themselves being finite, the field of "finite model theory" sees much application to complexity. Specifically, we see later that the tools of logic and model theory allow us to gain a new perspective on the capabilities of computers.

2 Complexity Preliminaries

Before we can address such questions as the difficulty of breaking encryption or playing Go, we need to rigorously define a notion of "computer" to have mathematical frame to work in. We will begin by introducing our model of computation.

2.1 Turing Machines

Rigorously, a Turing machine (which from here we will abbreviate TM) is defined as follows:

Definition 1 (Turing Machine)**.** A Turing machine is a tuple (Γ, Q, ∂) such that

1. Γ is a set of "symbols"; We assume that Γ contains a "start" symbol $\triangle$, a "blank" symbol $\square$, and the numbers 0 and 1
2. Q is a finite set of "states"; We assume Q contains the states q_{start} and q_{halt}
3. ∂ is a function from $Q \times \Gamma^k$ to $Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ (where L stands for "left", S stands for "stay", and R stands for "right")

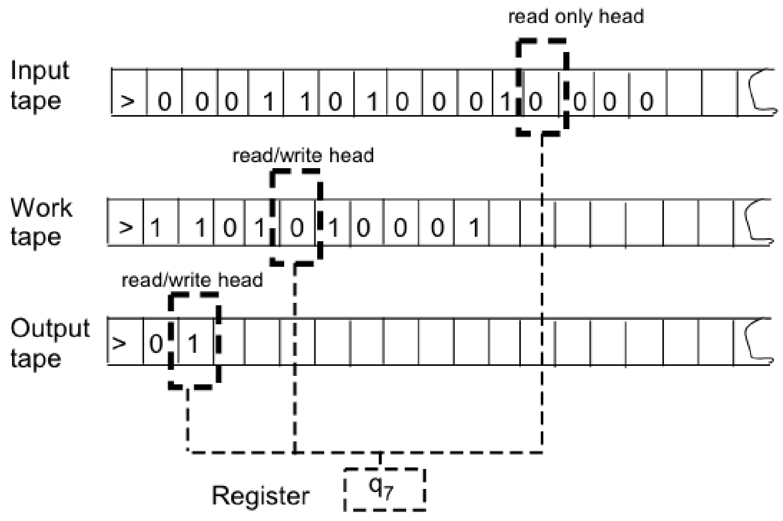
More intuitively, we can think of a Turing machine as the following:

- A "scratch pad" consisting of k tapes, each of which are sequences of symbols from Γ. One of the tapes is the "input tape", another is the "output tape", and the rest are called "work tapes". The work and output tapes start with all of the elements being the blank $\square$, and the start tape starts with the start symbol $\triangle$ at the beginning, a finite tuple of 0s and 1s, and the rest are blank. Each tape has a "tape head" that starts at the beginning of the tape. The tape head can move between symbols in the sequence, check the symbol in the sequence under them, and change the symbol under them (except for the input head).
- A finite set of "rules" that it follows. At a given time, the machine is in one of the states in Q. Given this state, it will perform its next computational step, which consists of
  1. Reading the symbols under each of the heads
  2. Changing the symbol under each of the heads (the new symbol can be the same as the old symbol, and the input tape head does not change symbols)

3. Changing the state of the TM
4. Moving each of the heads one place its tape

Each step is analogous to an application of ∂. ∂ takes in a state (an element of Q) and the symbols under each of the heads (an element of Γ^k). It outputs a new state (an element of Q), what to write under each of the tape heads on each of the work tapes (an element of Γ^{k-1}), and which direction to move each of the heads (an element of $\{L, S, R\}^k$, where L stands for left, S stands for stay, and R stands for right).

A diagram of a TM can be seen below:



We say the the finite tuple of 0s and 1s on the input tape is the "input". If the TM reaches the state q_{halt}, there will be a finite initial segment of the output tape that is only 0s and 1s; we will call this initial segment the "output". If the output is 1 on a given input, we say the TM "accepts" on that input, and likewise it "rejects" if the output is 0.

A nondeterministic Turing machine (which will from now on be refered to as an NDTM) is the same as a TM except at each step, the NDTM makes a "choice". More rigorously, instead of being a tuple (Γ, Q, ∂), it is a tuple $(\Gamma, Q, \partial_1, \partial_2)$. At each step, instead of applying ∂, the NDTM can either apply ∂_1 or ∂_2. If, on a given input, there is some finite sequence of applications of ∂_1 and ∂_2 such that the output is 1, we say that the NDTM accepts on that input. We say it rejects if it outputs 0 on all paths.

2.2 Problems and Complete Problems

Now that we have formalized the notion of a computer, we must now formalize the notion of a "problem" that a computer can solve.

Definition 2 (Language)**.** The set $\{0,1\}^n$ is the set of n-tuples of 0s and 1s. We define $\{0,1\}^*$ as $\bigcup_n \{0,1\}^n$. We can see that this is the set of possible inputs to a TM. We define a language to be a subset of $\{0,1\}^*$.

To be able to use TMs to solve problems, we must encode the problem as a language. As an example, consider the problem of deciding whether or not there is a clique in a given graph. We then encode each graph as a binary string and construct a TM that checks each set of nodes and accepts if it finds a clique. The relevant language is then the set of binary encodings of the graphs with a certain size clique.

Let M be a TM and L a language. We say M decides L if M accepts on input x if x is in L and rejects otherwise. We say that a language L is in **P** if there is a TM M that decides L and there exists some polynomial p such that M halts in at most $p(n)$ steps for inputs of length n (also called halting or running in polynomial time). Likewise, we say that a language L is in **NP** if there is an NDTM N such that N decides L and there exists some polynomial p such that M halts in $p(n)$ steps on inputs of length n. Notice the following:

Theorem 1. *Let L be a language. L is in **NP** if and only if there exists some polynomial p and some TM M that halts in $p(n)$ steps on inputs of length n such that for all binary strings $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \text{there exists } u \in \{0,1\}^{q(|x|)} \text{ such that } M \text{ accepts on input } \langle x, u \rangle$$

where $\langle x, u \rangle$ refers to some encoding of x and u as a single binary string and $|x|$ refers to the length of the binary string x.

Proof:

($\Rightarrow$) Suppose that L is decided by an NDTM $N = (\Gamma, Q, \partial_1, \partial_2)$ in polynomial time. Then there is some polynomial p such that for each $x \in L$, there is some sequence of applications of ∂_1 and ∂_2 such that N accepts in $p(|x|)$ steps. We can then construct a TM M that, given an input x and a binary string u, will "simulate" the running of N (it will apply ∂_1 if the relevant bit in u is 0 and ∂_2 if it is 1). This M fulfills our conditions.

($\Leftarrow$) We can construct the NDTM N' that nondeterministically writes down a binary string of length at most $p(n)$ (∂_1 writes down a 1, ∂_2 writes down a 0) and then deterministically runs M on the string (∂_2 will be the same as ∂_2 after writing down the string). ■

Definition 3 (Polynomial-Time Reduction)**.** We say that a language L is polynomial-time Karp reducible to a language L', denoted $L \leq_p L'$, if there is some polynomial-time TM M such that M outputs an element of L' if and only if the input is an element of L. Define $f : \{0,1\}^* \to \{0,1\}^*$ such that a string x maps to the output of M on input x, denoted $M(x)$. f is called a polynomial-time reduction from L to L'. More generally,

Definition 4 (NP-Completeness)**.** Consider the set of languages **NP** and let $L \subseteq \{0,1\}^*$ be a language. We call L **NP**-hard if for every language L' in **NP**, L' is reducible to L. We call L **NP**-complete if it is **NP**-hard and is in **NP**.

We will now discuss a central example of an **NP**-complete problem.

Definition 5 (Boolean Formulae)**.** We define boolean formulae by induction on their structure:

1. If x_1 is a "literal", then $\phi = x_1$ is a formula
2. If x_1 is a literal, then $\phi = \neg x_1$ is a formula
3. If ϕ and ϕ' are formulae, then $\psi = \phi \wedge \phi'$ is a formula
4. If ϕ and ϕ' are formulae, then $\psi = \phi \vee \phi'$ is a formula
5. If ϕ is a formula, then $\neg\phi$ is a formula.

A formulae with literals $x_1, x_2 \dots x_n$ is denoted $\phi(x_1, x_2 \dots x_n)$. An input to this formula is a n-tuple of 0s and 1s that can be evaluated in the following way:

1. If $\phi(x_1) = x_1$, then $\phi(u) = 1$ if and only if $u_1 = 1$.
2. If $\phi(x_1) = \neg x_1$, then $\phi(u) = 1$ if and only if $u = 0$
3. If $\psi(x_1, \dots x_n, y_1, \dots y_m) = \phi(x_1, \dots x_n) \wedge \phi'(y_1, \dots y_m)$, then $\psi(u_1, \dots u_{n+m}) = 1$ if and only if $\phi(u_1, \dots u_n) = 1$ and $\phi'(u_{m-n}, \dots u_m) = 1$.
4. If $\psi(x_1, \dots x_n, y_1, \dots y_m) = \phi(x_1, \dots x_n) \vee \phi'(y_1, \dots y_m)$, then $\psi(u_1, \dots u_{n+m}) = 1$ if and only if $\phi(u_1, \dots u_n) = 1$ or $\phi'(u_{m-n}, \dots u_m) = 1$.
5. If $\psi(x) = \neg\phi(x)$, then $\psi(u) = 1$ if and only if $\phi(u) = 0$.

We call a formula $\phi(x_1, \dots x_n)$ if there exist $u_1, \dots u_n$ such that $\phi(u_1, \dots u_n) = 1$.

Theorem 2 (Cook-Levin)**.** *Define some binary encoding of boolean formulae. We define a formula to be a CNF (conjunctive normal form) formula if it is of the form* $\bigwedge_n \left(\bigvee_m x_{nm} \right) = (\phi_{11} \vee \dots \phi_{1m}) \wedge \dots (\phi_{n1} \vee \dots \phi_{nm})$ *where each* ϕ *is of the form* x_i *or* $\neg x_i$. *We define SAT as the set of encodings of satisfiable CNF formulae. SAT is* ***NP****-complete.*

Proof:

Clearly, for each satisfiable CNF formula has an input of polynomial length that satisfies it and can be verified in polynomial time. Thus, SAT is in **NP**. We defer the proof of hardness to Lemma 2.12 in [1].

Lastly, we will define the notation $O(f(n))$ for convenience later.

Definition 6. Let f, g be functions. We say that $g(n) = O(f(n))$ if there exists a constant $c > 0$ such that $g(n) \leq cf(n)$ for sufficiently large natural n.

3 Traditional Characterizations

We will now introduce the primary complexity classes (collections of languages) of interest to us here: **PH** and **PSPACE**.

3.1 The Polynomial Hierarchy

We will first define the relevant notion of an "oracle". Intuitively, an oracle or Turing oracle for a language L can be thought of as a magic box added to a TM or NDTM M that can tell M whether or not a string x is in L. More rigorously, we have the following:

Definition 7 (Oracle Turing Machine)**.** We define an oracle TM with an oracle for a language L, denoted M^L, as a TM with an extra tape called "oracle tape" and an extra state called a "query state". M^L can write on its oracle tape as normal, and can at any time enter its query state. After this, if the binary string written on the query tape is in L, M^L will write 1 on the query tape and 0 otherwise. In effect, M^L can decide whether or not a string of length n is in L in $O(n)$ steps.

We will now introduce a relevant complexity class:

Definition 8 (**coNP**)**.** Let L be a language. Denote by $\overline{L}$ the complement of L, $\{0,1\}^* \setminus L$. We define the complexity class **coNP** as $\{\overline{L} \subseteq \{0,1\}^* : L \text{ is in } \mathbf{NP}\}$.

A common first thought is that **coNP** is equal to **NP** because for every language L in **NP**. We can consider the polynomial time NDTM $N = (\Gamma, Q, \partial_1, \partial_2)$ that decides L and then consider the NDTM that runs N and then rejects if N accepts and rejects if N rejects. It seems like N' should decide $\overline{L}$, but it does not. This is because, if there is a sequence of applications of ∂_1 and ∂_2 that rejects on an input $x \in L$ (which there may be, since we only assume that N will accept for *some* sequence of choices), then N' will accept x on that sequence of choices. Thus, x will be in the language decided by N' and N' will not decide $\overline{L}$.

We now consider the polynomial hierarchy as a whole, which is a generalization of **NP** and **coNP**.

Definition 9 (The Polynomial Hierarchy)**.** Let $\Sigma_0^p = \Pi_0^p = \mathbf{P}$. Inductively define $\Sigma_{i+1}^p = \{L \subseteq \{0,1\}^* : \text{there exists an polynomial-time oracle NDTM } M^A \text{ for some language } A \text{ in } \Sigma_i^p \text{ that decides } L\}$ and $\Pi_{i+1}^p = \{\overline{L} : L \text{ in } \Sigma_{i+1}^p\}$. We define $\mathbf{PH} = \bigcup_i \Sigma_i^p$.

Noticing that $\Sigma_1^p = \mathbf{NP}$ and $\Pi_1^p = \mathbf{coNP}$, we trivially have the following diagram of inclusions:

$$\begin{tikzcd}
\Sigma_{i+1} & & & & \Pi_{i+1} \\
& & \cdots & & \\
\Sigma_2 & & & & \Pi_2 \\
\Sigma_1 \arrow[u] \arrow[urrrr] & & & & \Pi_1 \arrow[u] \arrow[ullll] \\
\Sigma_0 & = & \mathrm{P} \arrow[ull] \arrow[urr] & = & \Pi_0
\end{tikzcd}$$

We can now introduce the following theorem, which will later be a useful reformulation of our previous definition:

Theorem 3. *A language L is in Σ^p_{i+1} if and only if there is a polynomial-time TM M and a polynomial p such that the following holds:*

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{p(|x|)} \forall u_2 \in \{0,1\}^{p(|x|)} \dots Q_i u_i \in \{0,1\}^{p(|x|)}$$
$$\textit{such that } M \textit{ accepts on input } \langle x, u_1, u_2, \dots u_i \rangle$$

where each Q_n is an existential quantifier if n is odd and universal otherwise.

Proof:

We will proceed by induction on i. Our base case is more or less Theorem 1.

Assume that the theorem is true for some i. We will now show it is true for $i+1$. We can see that L being in Σ^p_{i+1} is equivalent to the existence of a polynomial-time oracle NDTM N^A for some A in Σ^p_i. By a similar argument to Theorem 1, this is equivalent to the following:

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \text{ such that } M^A \text{ accepts on input } \langle x, u \rangle$$

where q is a polynomial and M^A is a polynomial-time oracle TM. Let $f(x,u)$ be the number of queries that M^A makes on input $\langle x, u \rangle$. We can see that an oracle for A is also in effect an oracle for $\overline{A}$. Also, because the time to enter a query is linear, the sum of the lengths of the inputs on the query tape must be polynomially bounded in terms of $|x|$, and so the lengths of each input on the query tape must also be polynomially bounded by some $t(|x|)$. Clearly, $t(|x|) \cdot f(x,u)$ for u of sufficiently bounded size must also be polynomially bounded.

Lastly, by the induction hypothesis, there must be some polynomial-time TM K such that the following holds:

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{r(|x|)} \exists u_2 \in \{0,1\}^{r(|x|)} \dots Q_i u_i \in \{0,1\}^{r(|x|)}$$
$$\text{such that } K \text{ accepts on input } \langle x, u_1, \dots u_i \rangle$$

We will now construct a polynomial-time TM M that take inputs of the form $\langle x, u_1, \dots u_{i+1} \rangle$ with $u_1 \in \{0,1\}^q(|x|)$ and $u_n \in \{0,1\}^{t(|x|)f(x,u_1)}$ for $n > 1$. It will run exactly as M^A does, except instead of answering queries immediately, it will simulate the nth query with input y by simulating K on input $\langle y, v_2^n, v_3^n \dots \rangle$, where v_n^m is the $mt(|x|)$th digit through the $(m+1)t(|x|) - 1$th digit of u_n. This construction works analogously in the other direction, so we have shown the conditions to be equivalent. ■

Theorem 4. *For a given i, consider the following statements:*

1. $\Sigma_i^p = \Pi_i^p$ *for* $i > 0$
2. $\Sigma_i^p = \Sigma_{i+1}^p$
3. $\Sigma_i^p = \mathbf{PH}$ *(ie, the polynomial hierarchy collapses to the ith level).*

We have $1 \Rightarrow 2$ *and* $2 \Leftrightarrow 3$.

Proof:

($1 \Rightarrow 2$) Let $i > 0$ such that $\Sigma_i^p = \Pi_i^p$. Let L be a language in Σ_{i+1}^p. Then there exists a polynomial-time TM M such that for a binary string x, $x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{p(|x|)} \forall u_2 \in \{0,1\}^{p(|x|)} \dots Q_{i+1} u_i \in \{0,1\}^{p(|x|)}$ such that M accepts on input $\langle x, u_1, \dots u_{i+1} \rangle$ for some polynomial p. Thus, the language L' defined by $u_1 \in L' \Leftrightarrow \forall u_2 \in \{0,1\}^{p(|x|)} \dots Q_{i+1} u_i \in \{0,1\}^{p(|x|)}$ such that M accepts on input $\langle u_1, \dots u_{i+1} \rangle$. Then L' is clearly in Π_i^p. Thus, there is some polynomial-time oracle TM $M^{L'}$ such that $x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{p(|x|)}$ such that $M^{L'}$ accepts on input $\langle x, u_1 \rangle$. Thus, L is in Π_i^p, so $\Sigma_i^p = \Pi_i^p = \Sigma_{i+1}^p$.

($2 \Rightarrow 3$) Clearly every L in $\mathbf{PH}$ is an element of some Σ_n^p for $i > n$. By the polynomial-time oracle NDTM definition of the polynomial hierarchy, we clearly have $\Sigma_i^p = \Sigma_n^p$ for $n > i$, so L is in Σ_i^p.

($3 \Rightarrow 2$) Clearly, $\Sigma_i^p \subseteq \Sigma_{i+1}^p \subseteq \mathbf{PH} \subseteq \Sigma_i^p$, so $\Sigma_i^p = \Sigma_{i+1}^p$. ■

Definition 10 (Completeness in the Polynomial Hierarchy)**.** Define **PH**-, Σ_i^p-, and Π_i^p-hardness as in Definition 4, via polynomial-time reductions, and define **PH**-, Σ_i^p-, Π_i^p-completeness analogously as well.

We have the following results about complete problems in the polynomial hierarchy:

Theorem 5. *We define a quantified boolean formula of the form $\exists u_1 \forall u_2 \dots Q_i u_i \phi(u_1, \dots u_i)$, with Q_i alternating between existential and universal and each u_n a tuple of literals, to be true if and only if there exists a tuple $(u_1, u_3 \dots)$ of 0s and 1s such that for all tuples $(u_2, u_4 \dots)$, $\phi(u_1, u_2, u_3 \dots) = 1$. Defining an encoding of such a boolean formula into binary, we now define a language $\Sigma_i^p SAT$ of the encodings of formulas of this form that are true. $\Sigma_i^p SAT$ is Σ_i^p-complete.*

Proof: We will demonstrate the case in which $i = 2$, as the argument easily generalizes by induction. We begin by showing that Σ_2^pSAT is in Σ_2^p. We can see that if, given an oracle for SAT, an NDTM can decide Σ_2^pSAT in polynomial time, then Σ_2^pSAT is in Σ_2^p. We will now construct the oracle NDTM M^{SAT}. Given a quantified boolean formula of the form $\exists u_1 \forall u_2 \phi(u_1, u_2)$, it will start by nondeterministically writing down a "guess" v_1 for the value of u_1 and then use the oracle for SAT to determine whether or not $\neg\psi(u_2)$ has a satisfying assignment where $\psi(u_2) = \phi(v_1, u_2)$ for all u_2. We can then see that M^{SAT} decides Σ_2^pSAT in polynomial time. We now show hardness. Let L be some language in Σ_2^p. Then there is some polynomial-time TM M such that for all x, $x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{p(|x|)} \forall u_2 \in \{0,1\}^{p(|x|)}$ such that M accepts on input $\langle x, u_1, u_2 \rangle$. By the hardness of SAT, this is clearly equivalent to finding tuples u_1 and u_2 that satisfy some formula $\phi(x, u_1, u_2)$. Thus, we have our reduction and Σ_2^pSAT is Σ_2^p-complete. ■

Theorem 6. *There exists a **PH**-complete language if and only if the polynomial hierarchy collapses to the ith level for some i.*

Proof:

($\Rightarrow$) Assume there exists a language L that is **PH**-complete. Then there is some Σ_i^p such that $L \in \Sigma_i^p$. If $i = 0$ it is clearly true that $\mathbf{PH} = \mathbf{P}$ because all languages in **PH** are polynomial-time reducible to a language that can be decided in polynomial time. If $i > 0$, then L must be decidable by a polynomial-time oracle NDTM M^A. For each B in **PH**, we can then construct the polynomial-time oracle NDTM N^A that reduces B to L in polynomial time and then simulates M^A. We can clearly see that B is in Σ_i^p, so $\mathbf{PH} \subseteq \Sigma_i^p$. Thus, $\mathbf{PH} = \Sigma_i^p$.

($\Leftarrow$) By Theorem 5, there is a complete language L_i for Σ_i^p for each i. Thus, if $\mathbf{PH} = \Sigma_i^p$, L_i is complete for **PH**. ■

3.2 PSPACE

"Space" is the formalization of the notion of a computer's memory usage. We now introduce the second of our key complexity classes:

Definition 11 (PSPACE)**.** We define the collection of languages **PSPACE** as the set of languages decided by some TM M such that on an input of length N, M uses only the first $p(n)$ cells of its tape.

First, we will introduce the important relationship:

Theorem 7. $\mathbf{PH} \subseteq \mathbf{PSPACE}$.

Proof:

Let L be a language in Σ_i^p for some i. Then there is some polynomial-time oracle TM M such that for every x, $x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{p(|x|)} \ldots Q_i u_i \in \{0,1\}^{p(|x|)}$ such that M accepts on input $\langle x, u_1 \ldots u_i \rangle$. We can thus construct the TM N that on input x iterates over assignments of u_1 through u_i and simulates N on input $\langle x, u_1 \ldots u_n \rangle$, tracking only the previous assignment tried. Each simulation of M will take polynomially many steps, and thus polynomially many cells, and tracking the previous assignment will also take polynomially many cells. This works because we can reuse the cells used for checking each time. ■

We will now define the notion of completeness for **PSPACE**:

Definition 12 (PSPACE-Complete)**.** A language L is defined to be **PSPACE**-hard if for every language A in **PSPACE**, $A \leq_p L$. It is **PSPACE**-complete if it is **PSPACE**-hard and is in **PSPACE**.

We will now generalize the satisfiable quantified boolean formula language from Theorem 5:

Theorem 8. *A quantified boolean formula is a formula of the form $Q_1x_1 \dots Q_nx_n\phi(x_1, \dots x_n)$ for some boolean formula ϕ and with each Q_n a quantifier. We define it to be "true" in the natural way. An example of a "true" formula is the formula $\exists x_1 \forall x_2 x_1 \wedge x_2$. Fix an encoding of all quantified boolean formulae. We define the language TBQF as the set of encodings of true quantified boolean formulas. TBQF is **PSPACE**-complete.*

Proof:

We first demonstrate that TBQF is in **PSPACE**. We now define a TM M that, given as an input an encoding of some formula $Q_1x_1 \dots Q_nx_n\phi(x_1, \dots x_n)$, checks the satisfiability of $\psi(x_2, \dots x_n) = \phi(u_1, x_2, \dots x_n)$ for $u = 1$ and $u = 0$. Checking each will take polynomial space and tracking what has been tried will take polynomial space, so TBQF is in **PSPACE**. We defer the proof of hardness again to [1] Theorem 4.11. ■

Corollary 1. If **PH** = **PSPACE**, then TBQF is **PH**-complete, which by Theorem 6 implies that the polynomial hierarchy collapses.

4 Logic Preliminaries

Complexity theory, as mentioned before, is the study of the resources required for a computer to solve a problem. We will now present a reframing of traditional complexity-theoretic notions in terms of finite model theory.

4.1 Structures

We start by defining the basis on which our reframing will be built:

Definition 13 (Vocabularies)**.** A vocabulary $\tau = \langle f_1^{a_1}, \dots f_r^{a_r}, c_1, \dots c_s, R_1^{r_1}, \dots R_t^{r_t}\rangle$ is a tuple of function symbols, constant symbols, and relation symbols. Each function symbol f_n has an "arity" a_n, and each relation symbol R_n has an arity r_n.

Standard examples include the vocabulary of groups $\langle \cdot^2, 1\rangle$ where $\cdot^2$ is the binary operation and 1 is the identity element, the vocabulary of directed graphs with a "start" node and "terminal" node $\langle s, t, E^2\rangle$ where s is the start node, t is the terminal node, and E^2 is the edge relation, and the vocabulary of a database that stores whether or not a person is male, paternal relationships, and sibling relationships $\langle M^1, P^2, S^2\rangle$ with M^1 being the "male" relation, P^2 being the paternal relation, and S^2 being the sibling relation. We now define the notion of a structure on a given vocabulary:

Definition 14 (Structures)**.** We define a structure with vocabulary $\tau = \langle f_1^{a_1}, \dots f_r^{a_r}, c_1, \dots c_s, R_1^{r_1}, \dots R_t^{r_t}\rangle$ as a tuple $\mathcal{A} = \langle |\mathcal{A}|, f_1^{\mathcal{A}}, \dots f_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \dots c_s^{\mathcal{A}}, R_1^{\mathcal{A}}, \dots R_t^{\mathcal{A}}\rangle$ such that $|\mathcal{A}|$ is a set, each $f_n^{\mathcal{A}}$ is a function from $|\mathcal{A}|^{a_n}$ to $|\mathcal{A}|$, each $c_n^{\mathcal{A}}$ is an element of $|\mathcal{A}|$, and each $R_n^{\mathcal{A}}$ is a subset of $|\mathcal{A}|^{r_n}$. $|\mathcal{A}|$ is called the universe of $\mathcal{A}$.

Examples of structures are the group $\mathbb{Z}_5 = \langle\{0,1,2,3,4\}, +, 0\rangle$ where $+$ is the standard addition modulo 5. We also have the graph below:

[insert digraph diagram]

As a final example, consider the relational database from before:
$\langle\{Alice, Jim, Bob, Rachel\}, \{(Jim), (Bob)\}, \{(Alice, Jim), (Alice, Bob)\}, \{(Jim, Bob), (Bob, Jim)\}\rangle$.

From here, structures will have finite universes and vocabularies will consist of finitely many relation symbols and constant symbols.

We will now introduce our first "descriptive" notion:

Definition 15 (First-Order Logic)**.** For a vocabulary τ we define the first-order language $\mathcal{L}(\tau)$ as the formulas built from the relations and constants of τ; the logical relation $=$; the boolean relations $\wedge$, $\vee$, and $\neg$; variables from the set $VAR = \{x, y, z \dots\}$; and the quantifier $\exists$. If two formulae φ and φ' are equivalent, we write $\varphi \equiv \varphi'$.

We call a variable in a formula "bound" if it is within the scope of a quantifier $(\exists x)$ and "free" if it is not.

Let V be some finite subset of VAR and $\mathcal{A}$ a structure. We call a mapping $i : V \to |\mathcal{A}|$ an interpretation into $\mathcal{A}$. For convenience, we let $i(c) = c^{\mathcal{A}}$ for constant symbols c.

We now define our notion of "truth":

Definition 16. Let $\mathcal{A}$ be a structure and i an interpretation of relevant free variables into $\mathcal{A}$. We define a formula as true in $(\mathcal{A}, i)$ by induction on the structure of formulae:

1. $(\mathcal{A}, i) \models x_1 = x_2 \Leftrightarrow i(x_1) = i(x_2)$
2. $(\mathcal{A}, i) \models R_n(x_1, \dots x_{a_n}) \Leftrightarrow (i(x_1), \dots i(x_{a_n})) \in R_n^{\mathcal{A}}$
3. $(\mathcal{A}, i) \models \neg\phi \Leftrightarrow (\mathcal{A}, i) \models \phi$ is not true
4. $(\mathcal{A}, i) \models \varphi \wedge \varphi' \Leftrightarrow (\mathcal{A}, i) \models \phi$ and $(\mathcal{A}, i) \models \varphi'$
5. $(\mathcal{A}, i) \models \varphi \vee \varphi' \Leftrightarrow (\mathcal{A}, i) \models \varphi$ or $(\mathcal{A}, i) \models \varphi'$
6. $(\mathcal{A}, i) \models \exists x\varphi \Leftrightarrow$ there exists some $x \in |\mathcal{A}|$ such that $(\mathcal{A}, i, a/x) \models \varphi$

where $(i, a/x)(y) = \begin{cases} y \text{ if } y \neq x \\ a \text{ if } y = x \end{cases}$.

Lastly, we write $\mathcal{A} \models \varphi$ if $(\mathcal{A}, \emptyset) \models \varphi$.

There are several things to note about this definition. Firstly, it is an "inductive" one in the same sense as Definition 5, and this inductive construction will be integral to some of our proofs. Secondly, some might think that the use of $=$ here is circular. However, it does not denote a binary relation on $|\mathcal{A}|$ but instead the naive "standard" equality.

We can now introduce several useful shorthands:

1. $(\forall x)\varphi \equiv \neg(\exists x)\neg\varphi$
2. $y \neq z \equiv \neg(y = z)$
3. $\varphi \rightarrow \varphi' \equiv \neg\varphi \wedge \varphi'$
4. $\varphi \leftrightarrow \varphi' \equiv \varphi \rightarrow \varphi' \wedge \varphi' \rightarrow \varphi$
5. $(\exists! x)\varphi \equiv (\exists x)\varphi \wedge (\forall y)\varphi \leftrightarrow y = x$.

Note that, given a lack of parentheses, we are implicitly performing the $\neg$ operator first, then $\wedge$ and $\vee$, and finally $\rightarrow$ and $\leftrightarrow$. As an example, the formula $\neg R(a) \rightarrow R(b) \wedge R(c) \leftrightarrow R(d)$ is equivalent to $[(\neg R(a)) \rightarrow R(b)] \wedge [R(c) \leftrightarrow R(d)]$. A "sentence" is a formula without free variables, so it must be either "true" or "false" for a given structure.

An example of a first-order formula in the language of graphs is the following:

$$\varphi_{undirected} \equiv (\forall x)(\forall y)\neg E(x,x) \wedge E(x,y) \leftrightarrow E(y,x)$$

which says that a given graph is undirected. Another is the following inductively defined formulae:

$$\begin{aligned}\varphi_{dist(1)} &\equiv x = y \vee E(x,y)\\ \varphi_{dist(n)} &\equiv (\exists z)(\varphi_{dist(n-1)}(x,z) \wedge \varphi_{dist(n-1)}(z,y))\end{aligned}$$

Each $\varphi_{dist(n)}$ says that the distance between two given nodes is n.

For convenience's sake, we will say that every universe is of the form $0, 1, \ldots n$ for some n. We will require structures to be ordered (and vocabularies will have the symbol $\leq$), and they will implictly have at least two distinct elements, have at least two distinct constants called 0 and 1, have the "successor" relation $SUC(x,y)$ defined by $SUC(x,y) \equiv [x < y] \wedge \neg[(\exists z)(x \neq z) \wedge (y \neq z) \wedge (x < z) \wedge (z < y)]$, and have the BIT relation where $BIT(i,j)$ is equivalent to the jth bit of the binary representation of i being 1. Note that BIT lets us define standard arithmetic such as addition and multiplication by encoding the standard algorithm in a first-order formula.

4.2 Queries

Now we will introduce the way in which we will model computation via logic.

Definition 17 (Queries)**.** A query I is a mapping from $STRUC[\sigma]$ to $STRUC[\tau]$ for vocabularies σ, τ that is "polynomiall bounded" (there is some polynomial p such that $||I(\mathcal{A})|| \leq p(||\mathcal{A}||)$). A "boolean" query I_b is a mapping from $STRUC[\sigma]$ to $\{0,1\}$ for a vocabulary σ. It may also be considered as a subset of $STRUC[\sigma]$ (ie, the subset for which $I_b(\mathcal{A}) = 1$).

We care about not just queries in general but specific kinds of queries. A central example is that of first-order boolean queries, which are maps $I_\varphi : STRUC[\sigma] \to \{0,1\}$ with an associated sentence φ such that $I_\varphi(\mathcal{A}) \mapsto 1$ if and only if $\mathcal{A} \models \varphi$. We are also concerned with general first-order queries.

Definition 18 (First-Order Queries)**.** We define a query $I : STRUC[\sigma] \to STRUC[\tau]$ where $\tau = \langle c_1, \ldots c_s, R_1^{r_1}, \ldots R^{r_t} \rangle$ to be a first-order k-ary query if it is "defined" by some tuple of formulae $\varphi_0, \ldots \varphi_t, \psi_1, \ldots \psi_s$ from $\mathcal{L}(\sigma)$; ie, the universe and relations of $I(\mathcal{A})$ are defined by the following:

1. The universe is first-order definable: $I(\mathcal{A}) = \{(b_1, \ldots b_k) : \mathcal{A} \models \varphi_0(b_1, \ldots b_k)\}$. We often take $\varphi_0 \equiv$ true for simplicity.

2. Each $R_n^{I(\mathcal{A})}$ is first-order definable: $R_n^{I(\mathcal{A})} = \{((b_1^1, \ldots b_k^{r_n}), \ldots (b_k, \ldots b_k^{r_n}) \in |I(\mathcal{A})| : \mathcal{A} \models \varphi_n\}$.

3. Each $c_n^{I(\mathcal{A})}$ is first-order definable: each $c_n^{I(\mathcal{A})}$ is the unique $(b_1, \ldots b_k) \in |I(\mathcal{A})|$ such that $\mathcal{A} \models \psi_n(b_1, \ldots b_k)$.

An important vocabulary is that of binary strings, $\tau_{bin} = \langle S^1 \rangle$, because of course TMs are computing queries $I : STRUC[\tau_{bin}] \to STRUC[\tau_{bin}]$ and boolean queries $I_b : STRUC[\tau_{bin}] \to \{0,1\}$. However, as we have seen with our examples of graphs, it is interesting to study queries and boolean queries with domains that are not the collection of binary strings, such as the collection of finite graphs or finite relational databases. We will now define the query $bin_\tau : STRUC[\sigma] \to STRUC[\tau_{bin}]$, the encoding of structures as binary strings. We can define this map as follows:

Let $\mathcal{A} = \langle |\mathcal{A}|, c_1^{\mathcal{A}}, \ldots c_s^{\mathcal{A}}, R_1^{\mathcal{A}}, \ldots R_t^{\mathcal{A}} \rangle$ be in $STRUC[\tau]$. Let $n = ||\mathcal{A}||$ We encode $R_j^{\mathcal{A}}$ as a binary string of length n^{r_j} where the relevant bit is 1 if and only if the relevant tuple is in $R_j^{\mathcal{A}}$. We can analogously encode $c_j^{\mathcal{A}}$ as a string of length $\log(n)$. We define the encoding $bin_\tau(\mathcal{A})$ as the concatenation of the encodings of the relations followed by the encodings of the constants. Note that the size of the structure (n) can be recovered from the encoding. Our "numeric" relations such as $\leq$, SUC, and BIT can also be easily recovered. The reader should also note the following result:

Theorem 9. *For every vocabulary τ, the encoding query bin_τ and its inverse are first-order queries.*

Proof:

This is left to the reader. Note that addition, multiplication, and exponentiation are first-order.

Definition 19 (Computing Queries)**.** Let $I_b : STRUC[\tau] \to \{0,1\}$ be a boolean query. We say that a TM M computes I_b if M accepts on input x if and only if $x = bin_\tau(\mathcal{A})$ for some $\mathcal{A}$ such that $I_b(\mathcal{A}) = 1$. We say it computes I_b in polynomial time if there is some polynomial p such that it computes I_b in $p(n)$ steps on inputs of the form $bin_\tau(\mathcal{A})$ such that $||\mathcal{A}|| = n$. From here, all previously introduced complexity classes will consist only of their languages that correspond to queries.

5 Second-Order Logic and PH

We finally come to characterizations of our main complexity classes of interest. We will first introduce the language that captures the polynomial hierarchy.

5.1 Second-Order Logic

Recall the structure of first-order formulae. Earlier, we only had first-order variables in formulae, but we now also have a collection of second-order variables, with interpretations defined in the natural way. We have the following:

Definition 20. Second-order formulae over a vocabulary τ consist of formulae constructed via the standard operators as well as the additional operator $(\exists R^a)\varphi$. We define $(\mathcal{A}, i) \models (\exists A^r)\varphi \Leftrightarrow$there exists some relation R of arity a such that $(\mathcal{A}, i, R/X) \models \varphi$.

Consider the language of graphs τ_g. We will now define a formula in second-order logic that says that a given graph has a clique:

$$\varphi_{clique} \equiv (\exists R^1)(R(x) \wedge R(y) \wedge x \neq y) \to E(x,y)$$

Note that in the above example, the relation quantified over had arity one; we in effect quantified over subsets of the universe. We can also in effect quantify over functions, as seen below:

$$\varphi_{function} \equiv (\forall z)F(x,y) \wedge F(x,z) \to y = z$$

We can now define a formula that says that a given function is injective:

$$\varphi_{inj} \equiv \varphi_{function} \wedge (\forall z)F(x,z) \wedge F(y,z) \to x = y$$

Finally, we can define our formula that says that a given graph has a clique of size k:

$$\varphi_{clique} \equiv (\exists F^2)(\forall xy)\varphi_{inj}(F) \wedge ((x \neq y \wedge (F(x,z) \to y < k) \wedge (F(y,z) \to z < k)) \to E(x,y))$$

Note that any second-order formula is equivalent to a formula with all second-order quantifiers in the front. If all of those quantifiers are of the form $\exists$ (ie, are existential), the formula is an existential second-order formula. Let SO be the collection of boolean queries $I_\varphi : STRUC[\tau] \to \{0,1\}$ with associated second-order sentence φ such that $I(\mathcal{A}) = 1$ if and only if $\mathcal{A} \models \varphi$, and similarly define $SO\exists$ for existential second-order φ. We have the following theorem:

5.2 Logical Characterizations of PH

Theorem 10 (Fagin). *$SO\exists$ =* **NP** *(boolean queries that can be described in existential second-order logic are exactly those that can be computed by polynomial-time NDTMs).*

This is fairly shocking. One would not expect a natural "descriptive" (logical) complexity class to be exactly the same as a complexity class defined via computation. We will begin the proof with the following result:

Lemma 1. We can compute first-order boolean queries in logarithmic space (ie, for a vocabulary τ and first-order boolean query I_b, there is a TM M such that on an input structure $\mathcal{A} \in$ such that $||\mathcal{A}|| = n$, M uses $O(\log(n))$ cells.)

Proof:

Let I_φ be a boolean query determined by the first-order sentence φ with k quantifiers. We will proceed by induction on k.

If $k = 0$, then φ is quantifier-free. φ is then a finite boolean combination of "atomic" formulae (formulae of the form $R(x_1, \ldots x_n)$, $x = y$, $x \leq y$, $SUC(x, y)$, or $BIT(x, y)$). It is fairly easy to see that a TM can compute n and $\lceil log(n) \rceil$ in $c \log(n)$ space. Thus, because of the form of $bin_\tau(\mathcal{A})$, it can find whether or not each atomic formula is true in logarithmic space. It is then not hard to see that we can compute the truth value of the boolean combination in logarithmic space.

We will now verify the induction step. Assume that, for every first-order boolean query I_b defined by a first-order sentence ϕ with $k - 1$, there is some TM M_ϕ that computes I_b in $c_{phi} \log(n)$ cells for some constant c_ϕ. Let I_b be a boolean query defined by a first-order sentence ψ of the form $\psi \equiv (\forall x)\phi(x)$ for some first-order sentence $\phi(x)$ with at most $k - 1$ quantifiers. Then we define the TM M_ψ that runs M_ϕ on input $\phi(x)$ and tracks only the last value of x tried. This will clearly run in logarithmic space. A similar argument applies when ψ is of the form $\psi \equiv (\exists x)\phi(x)$. Thus, we have completed the proof. ■

Theorem 11. *$SO\exists \subseteq$* **NP**.

Proof:

Let I_b be a boolean query defined by $\phi \equiv (\exists R_1^{r_1}) \ldots (\exists R_k^{r_k})\varphi$, an existential second-order sentence over the vocabulary τ. We wish to build a polynomial-time NDTM that accepts on input x if and only if $x = bin_\tau(\mathcal{A})$ for some $\mathcal{A} \models \phi$.

Let $\mathcal{A} \in STRUC[\tau]$ and let $n = ||\mathcal{A}||$. We define an NDTM M_ϕ that first nondeterministically writes down a binary string of length $\displaystyle\sum_i^k n^{r_i}$ as a "guess" for each R_i. We know that we can check whether or not $\mathcal{A} \models \varphi(R_1, \ldots R_k)$ in logarithmic space.

We can see that a TM that runs in $O(\log(n))$ space has non-blank symbols on at most $c \log(n)$ cells on each of its $k - 1$ non-input tapes, can have each of the non-input tape heads on $c \log(n)$ different cells, can have the head on the input tape on n different cells, and can be in s different states if it halts on every input, it can only ever be in a total of

$4^{(k-1)c\log(n)} \cdot 4^{(k-1)c\log n} \cdot n \cdot s$ "configurations" (the first term is the cells used, the second and third are head positions, and the last is the state), so it must run in $s4^{(k-1)c\log(n)} = s(\log(n))^{2(k-1)c} = sn^{(k-1)c}$ time. We can thus check whether or not $\mathcal{A} \models \varphi(R_1, \ldots R_k)$ in polynomial time with a TM and thus in polynomial time with an NDTM. We conclude that $I_b \in \mathbf{NP}$ and thus $SO\exists \subseteq \mathbf{NP}$. ■

We now proceed with the proof of Theorem 10, given the inclusion in Theorem 11. We must show that an **NP** query is in $SO\exists$. Let $N = (\Gamma, Q, \partial_1, \partial_2)$ be a nondeterministic polynomial-time TM that decides some language L in time n^k on inputs of length n. We wish to construct a sentence of the form $\phi \equiv (\exists C_1^{2k} \ldots C_g^{2k} \Delta^k)\varphi$ that says "there exists an accepting computation $\overline{C}, \Delta$ of N".

Fix some n. Intuitively, we will construct an n^k by n^k matrix $\overline{C}$ of elements of $(Q \times \Gamma) \cup \Gamma$, where the element in the $\overline{s}$th row and $\overline{t}$th column is the symbol γ from Γ in cell $\overline{s}$ at time $\overline{t}$ if the head is not on that cell and (q, γ) where q is the state at time $\overline{t}$ during an accepting computation. Let $\Sigma = \{\sigma_1, \ldots \sigma_g\} = (Q \times \Gamma) \cup \Gamma$ be an enumeration of the possible contents of a cell of the matrix. We wish to define relations $C_1^{2k}, \ldots C_g^{2k}$ such that $C_i(\vec{s}, \vec{t})$ is true if and only if cell $\overline{s}$ contains σ_i at time $\overline{t}$ during said accepting computation, where $\vec{s}, \vec{t}$ are elements of $\{0, \ldots n-1\}^k$ that encode values from 0 to $n^k - 1$. At each step of the accepting computation, N will apply either ∂_1 or ∂_2, which we will encode using the relation Δ^k, which is intuitively described as $\Delta(\vec{t})$ if and only if N applies ∂_1 at step $\overline{t}$. We now write the formula φ that says that given relations $C_1, \ldots C_g, \Delta$ code such an accepting computation. This sentence consists of 4 parts:

$$\varphi \equiv \alpha \wedge \beta \wedge \eta \wedge \zeta.$$

α will say that column 0 of the computation correctly codes the input, β will say that $C_i(\vec{s}, \vec{t}$ and $C_j(\vec{s}, \vec{t})$ cannot both be true for $i \neq j$, η says that column $\overline{t} + 1$ is the result of an application of the function ∂_1 or ∂_2 corresponding to $\Delta(\vec{t})$ applied to column $\overline{t}$, and ζ says that the last column of the computation correctly codes N accepting. Construction of β is fairly trivial, so we have the following:

(ζ) Let σ_7 be the element of Σ that corresponds to $\langle q_{halt}, 1\rangle$, the "accept" state. We can define the relation $\varphi_{max}(x_1, \ldots x_k)$ that says that $(x_1, \ldots x_k)$ is the maximal element of $\{0, \ldots n-1\}^k$ fairly easily in first-order logic using the ordering relation $\leq$. Without loss of generality, we can assume that N moves the output tape head all the way to the left before it accepts, so we can define $\zeta \equiv (\exists x)\varphi_{max}(x) \wedge C_7(0, x)$.

(α) Without loss of generality, let σ_0 be 0 and let σ_1 be 1. We will construct α, which will be of the form $\bigwedge \varphi_j$. As an example, suppose τ contains a relation of arity 1 R_1^1, the first relation in the tuple. Then one of the φ_j will be of the form

$$\begin{aligned}\varphi_j \equiv &(\vec{t} = 0 = s_1 = \ldots s_{k-1} \wedge s_k \neq 0 \wedge R_1(s_k) \to C_1(\vec{s}, \vec{t})) \wedge \\ &(\vec{t} = 0 = s_1 = \ldots s_{k-1} \wedge s_k \neq 0 \wedge \neg R_1(s_k) \to C_0(\vec{s}, \vec{t})),\end{aligned}$$

which says that a given cell is 1 if R_1 contains s_k and 0 otherwise.

(η) Let $\vec{max}$ encode the maximum element of the k-tuples under the canonical ordering, which we have shown is first-order definable, let $(a_{-1}, a_0, a_1, \delta) \to [N]b$ mean that the triple of cell contents a_{-1}, a_0, a_1 lead to cell b via move δ of N (move 1 is an application of ∂_1, and move 0 is ∂_2), let $\neg^\delta$ be $\neg$ if $\delta = 1$ and the empty symbol if $\delta = 0$, and define

$$\eta_1 \equiv (\forall \vec{t})(\forall \vec{s})(\vec{t} \neq \vec{max}) \wedge (\vec{0} < \vec{s} < \vec{max}) \wedge \left(\bigwedge_{(a_{-1},a_0,a_1,\delta)\to[N]b} (\neg^\delta \Delta(\vec{t}) \vee \neg C_{a_{-1}}(\vec{s}-1, \vec{t}) \vee \neg C_{a_0}(\vec{s}, \vec{t}) \vee \neg C_{a_1}(\vec{s}+1, \vec{t}) \vee C_b(\vec{s}, \vec{t}+1)) \right).$$

Finally let $\eta \equiv \eta_0 \wedge \eta_1 \wedge \eta_2$, where η_0 and η_2 encode the same information when $\vec{s} = \vec{0}$ and $\vec{max}$ respectively.

Recall that the definition of the classes in the polynomial hierarchy are founded on **NP**. Because of this, a logical characterization of **NP** and each Σ_i^p follows with comparably much less difficulty:

Theorem 12. *A boolean query $I_b : STRUC[\tau] \to \{0, 1\}$ is defined by a formula of the form $\phi \equiv (\exists \vec{R_1})(\forall \vec{R_2}) \ldots (Q_i \vec{R_i})\varphi$ for some first-order φ and tuples of relations $\vec{R_j}$ if and only if the corresponding language $\{bin_\tau(\mathcal{A}) : I_b(\mathcal{A}) = 1\}$ is in Σ_i^p.*

Proof:

($\Rightarrow$) This direction is not particularly difficult. Let $\vec{R_j}$ have length l_j and let the mth relation in $\vec{R_j}$ have arity $r_{j,m}$. Then we define the polynomial $p(n) = \sum_{m \leq l_j} n^{r_{j,m}}$. We can see that strings of length $p(n)$ are "guesses" for each relation quantified in ϕ, and we can also see that we can verify our "guess" in polynomial space, so it is not difficult to see that on an input structure $\mathcal{A}$, so by Theorem 3 we can see that the language corresponding to I_b is in Σ_i^p.

($\Leftarrow$) We will proceed by induction on i. Our base case has already been demonstrated in Fagin's theorem.

Our induction hypothesis is that for all $i \leq k$ for some k, the theorem is true for Σ_i^p. Let N^L be a polynomial-time oracle NDTM that decides a Σ_{k+1}^k query I_b for some language $L \in \Sigma_k^p$ corresponding to a query I_φ defined by a sentence φ. Note that an oracle for the language corresponding to I_φ is the same as an oracle for the language corresponding to $I_{\neg\varphi}$, the query defined by $\neg\varphi$, which is of the form $(\forall \vec{R_2}) \ldots (Q_i \vec{R_i})\phi$ for some first-order ϕ. We can construct a similar formula to the one in Fagin's theorem to simulate the running of an oracle NDTM, where β will also incorporate $\neg\varphi$ for queries, relying on the fact that the inverse of the binary encoding query is first-order. This sentence will be of the form $\Phi \equiv (\exists \vec{R_1})[\alpha \wedge \beta \wedge \eta \wedge \zeta]$. We can see that $\neg\varphi$ will occur only "positively" (in the scope of an even number of applications of the $\neg$ operator), so we can "pull out" the second-order quantification in $\neg\varphi$, yielding a sentence of the form $x(\exists \vec{R_1})(\forall \vec{R_2}) \ldots (Q_i \vec{R_i})\psi$ for some first-order ψ. Thus, we are done. ■

In this proof, we also saw the natural logical characterization of Π_i^p for every i.

We finally come to our characterization of **PH** itself:

Theorem 13. $\mathbf{PH} = SO$

Proof: Let I_b be a query corresponding to a language L in **PH**. Then $L \in \Sigma_i^p$ for some i, so there is some sentence of the form $\phi \equiv (\exists \vec{R_1}) \dots (Q_i \vec{R_i}) \varphi(\vec{R_1}, \dots \vec{R_i})$ where each $\vec{R_n}$ is that determines I_b. ϕ is clearly second-order so I_b is in SO.

Let I_b be a query defined by a second-order sentence $\phi = (\exists \vec{R_1}, \dots R_i)(\forall T_1, \dots T_j)\varphi(R_1, \dots R_i, T_1, \dots T_j)$. Without loss of generality, let $i \geq j$. Then we have $\phi \equiv (\exists R_1)(\forall T_1) \dots (\forall T_i)\psi(R_1, \dots R_i, T_1, \dots T_j)$, so I_b is in Σ_i^p and thus the corresponding language is in **PH**. ■

6 Fixed Point Operators and PSPACE

Just as we logically characterized the polynomial hierarchy, we wish to logically characterize **PSPACE**. The key to this will be the "transitive closure" operator we add to the grammar of second-order logic.

6.1 The Transitive Closure Operator

Let R^2 be a relation of arity 2 on the set $\{0, \dots n-1\}$. The transitive closure of R, denoted $TC(R)$, is the set $\{(a,b) : \exists n, a_1, a_2, \dots a_n$ such that for all $i, R(a_i, a_i + 1)\}$. The languages in the class $SO(TC)$ are the languages corresponding to boolean queries determined by sentences formed with the standard second-order operators along with the operator $[TC_{\vec{X},\vec{X'}}\varphi](\vec{Y}, \vec{Y'})$, where $\vec{X}, \vec{X'}, \vec{Y}, \vec{Y'}$ are each tuples of second-order variables of the same "type" (of length n for some n and the ith variable in each has arity r_i for constants r_i) and φ is some formula. We define interpretations for second-order variables analogously to those for first-order ones; ie, we have some collection of sufficient second-order variables $SOV = \{X_1^{r_1}, \dots\}$, each with an associated arity, and an interpretation I is a function from a subset $\{X_1^{r_1}, \dots X_n^{r_n}\} \subset SOV$ to $(\mathcal{P}(|\mathcal{A}|))^{r_1} \times \dots (\mathcal{P}(|\mathcal{A}|))^{r_n}$ where $\mathcal{P}(|\mathcal{A}|)$ denotes the powerset of the universe of a structure $\mathcal{A}$. This is to say that it takes in second-order variables of given arities and maps them to relations of those arities. An interpretation of a tuple of variables is simply an interpretation of the set of variables in the tuple. We have the following definition:

Definition 21 (Transitive Closure)**.** Let $\varphi(\vec{X}, \vec{X'})$ be a formula with $\vec{X}, \vec{X'}$ of the same type, $\mathcal{A}$ a structure, and $\vec{Y}, \vec{Y'}$ tuples of second-order variables of the same type. Define $\mathcal{J}_{\vec{X}}$ as the set $\{J(\vec{X}) : J$ is an interpretation of $\vec{X}$ on $\mathcal{A}\} = \{J(\vec{X'}) : J$ is an interpretation of $\vec{X'}$ on $\mathcal{A}\}$. Define $\mathcal{B}_{\vec{X},\vec{X'}}$ as $\{(\vec{R}, \vec{R'}) \in \mathcal{X} : \mathcal{A} \models \varphi(\vec{R}, \vec{R'})\}$. Let ψ be a formula of the form $\psi(\vec{X}, \vec{X'}, \vec{Y}, \vec{Y'}) \equiv [TC_{\vec{X},\vec{X'}}\varphi](\vec{Y}, \vec{Y'})$. For some interpretation I of $\{\vec{X}, \vec{X'}, \vec{Y}, \vec{Y'}\}$, we say that $\mathcal{A} \models \psi$ if and only if $(I(\vec{Y}), I(\vec{Y'})) \in TC(\mathcal{B}_{I(\vec{X}),I(\vec{X'})})$.

Definition 22 (First-Order Reductions)**.** Let $I : STRUC[\tau] \to \{0,1\}$ and $I' : STRUC[\sigma] \to \{0,1\}$ be boolean queries. A first-order reduction from I to I' is a first-order query I_r such that $I'(I_r(\mathcal{A})) = 1$ if and only if $I(\mathcal{A}) = 1$. We define completeness and hardness via first-order reductions in the natural way (reductions between queries corresponding to languages in a class).

The reader will note that in general, the classes we are concerned with are all "closed under first-order reductions" (for example, in **NP**, if a query I is reducible to an **NP** query I', then I corresponds to a language in **NP**). We have the following important lemma:

Lemma 2. Consider the set of languages **PSPACE**. If there exists a query I that is complete for **PSPACE** via first-order reductions that is also in $SO(TC)$, then for every query I' corresponding to a language in **PSPACE**, I' is in $SO(TC)$.

Proof:

Let I_φ be a **PSPACE** query defined by an $SO(TC)$ sentence φ that is complete for **PSPACE** via first-order reductions and let be a TM M that can decide I_φ in $q(n)$ space for some polynomial q. Consider another **PSPACE** query $I_{\varphi'}$ defined by some sentence φ' and a reduction I_r of arity k from $I_{\varphi'}$ to I_φ defined by the formulae $\varphi_0, \dots \varphi_t, \psi_1, \dots, \psi_s$. Then we can define

$$\begin{aligned}\Phi \equiv &(\exists R_1^{kr_1})(\exists R_2^{kr_2})\dots(\exists R_t^{kr_t})(\exists! c_{1,1} \dots c_{1,k})\dots(\exists! c_{s,1}\dots c_{s,k})\\ &\left(\bigwedge_{j\le t}[(\forall x_1, \dots x_{kr_j})R_j(x_1, \dots x_{kr_j}) \leftrightarrow \varphi(x_1 \dots x_{kr_j}) \bigwedge_{i \le r_j} \varphi_0(x_i, \dots x_{i+k})]\right) \wedge \\ &\left(\bigwedge_{m\le s}[(\forall y_1, \dots y_k)(y_1 = x_{m_1} \wedge \dots y_k = x_{m_k}) \leftrightarrow \psi_m(y_1, \dots y_k) \wedge \varphi_0(y_1, \dots y_k)]\right) \wedge \varphi.\end{aligned}$$

The boolean query defined by Φ is basically defining $I_r(\mathcal{A})$ and then checking whether or not $I_r(\mathcal{A}) \models \varphi$. Because φ is in $SO(TC)$, so is Φ, and we can see that $\Phi \equiv \varphi$ simply by the construction of our reduction. Thus, I_φ is in $SO(TC)$. ■

We will now introduce the following problem: Fix an n. Define a k-local graph on vertex set $\{0,1\}^n$ as a graph such that, for every vertex u, there is some unique "next" vertex v such that the ith bit of v is determined by bits $i-k, i-k+1 \dots i+k$ of u. We can see that we can encode this in a tuple of 2^{2k+1} bits. So that we can encode this "next" relation in a unary (1-ary) relation, we will force $k \le \lfloor \log(n)/2 - 1 \rfloor$ so that $2^{2k+1} \le n$. We define the vocabulary $\tau_\ell = \langle R^1, S^1, T^1 \rangle$ where on a given structure $\mathcal{A}$ of universe size n, $R^{\mathcal{A}}$ encodes the transition relation above, $S^{\mathcal{A}}$ encodes the "start" node, and $T^{\mathcal{A}}$ encodes the "terminal" node. We wish to define the query I_{reach} that says that there is a path from $S^{\mathcal{A}}$ to $T^{\mathcal{A}}$ in the k-local graph determined by $R^{\mathcal{A}}$. Let A^1 be a second-order variable representing a "current position" in the graph. Then to move to the next node, for each $i \le n$, check the relevant $\lfloor \log(n) \rfloor$ bits in R to determine the ith bit of the "next" position. Note that $\lfloor \log(n) \rfloor$ is first-order definable because it is the largest r such that $BIT(max, r)$ holds, where max is the greatest element in the universe (which we showed earlier is first-order definable). We define the following formula:

$$\alpha(i, \omega, A) \equiv (\exists k)(\forall j)[(k = \lfloor \log(n)/2 - 1) \wedge (j \leq \lfloor \log(n) \rfloor) \wedge (A(i - k + j) \leftrightarrow BIT(\omega_j))]$$

which says that the binary representation of ω encodes the relevant bits to determine the ith bit of A. We can then define the formula below,

$$\delta(A, A') \equiv (\forall i)(\exists \omega)[\alpha(i, \omega, A) \wedge (A'(i) \leftrightarrow R(\omega))],$$

which says that A is connected to A' in the graph defined by R. Finally, we define $\phi_{reach} \equiv [TC_{A,A'}\delta](S, T)$ to be the sentence defining I_{reach} (this is a slight abuse of notation, but single second-order variables here denote tuples with 1 element).

Theorem 14. $SO(TC) =$ **PSPACE**

Proof:

For the inclusion **PSPACE**, because of Lemma 2, it suffices to show that I_{reach} is complete for **PSPACE** via first-order reductions. To do this, we will introduce a somewhat fundamental tool.

Recall that a Turing machine is defined by a "finite set of instructions". It is not difficult to see that these can be encoded in a finite binary string. Consider, then the following problem:

Fix an encoding of Turing machies as binary strings. Let M_s be the encoding of a TM M, x a string, and r_s a binary encoding of a number r. Call the language composed of strings of the form $\langle M_s, x, r_s \rangle$ such that M accepts on input x in r steps U_{PSPACE}. We claim that U_PSPACE is **PSPACE**-complete. Inclusion is not difficult to see; we can construct a TM that simply takes M_s and x and simulates M on x, tracking the number of steps simulated, and accepting if and only if the simulated M accepts in r simulated steps, which will in total take polynomial space. We can then see that given a **PSPACE** language L decided by a TM N in $p(n)$ space for a polynomial p, that our reduction from L to U_{PSPACE} is defined by $x \mapsto \langle N_s, x, p(|x|) \rangle$. It is left to the reader to show that this reduction is first-order.

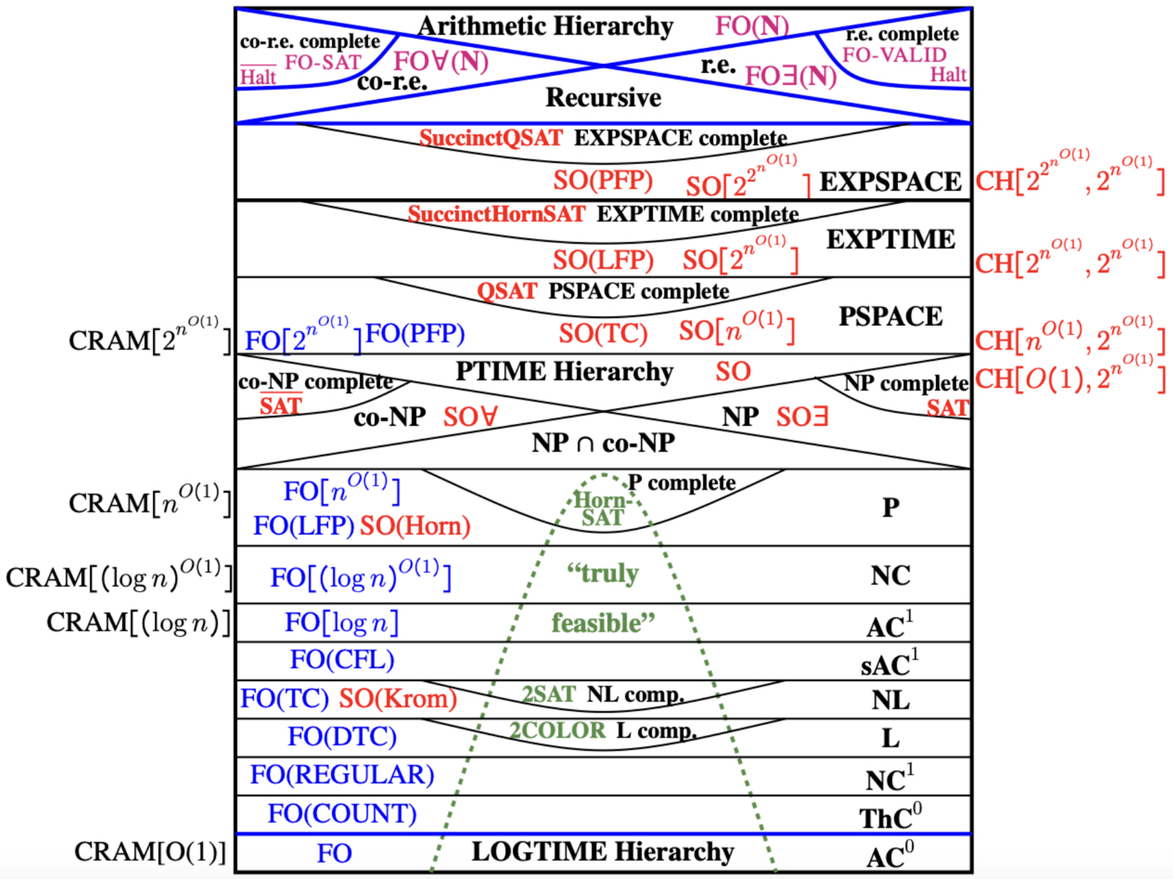
Because first-order reducibility is trivially transitive, we must now simply show that U_{PSPACE} is first-order reducible to I_{reach}. The reader can derive the first-order reduction themselves or check Proposition 10.27 of [2].

For the inclusion $SO(TC) \subseteq$ **PSPACE**, we only need to show that we can verify the formula of the form $[TC_{A,A}\delta(S, T)]$ in polynomial space, which is not difficult to see given that δ is first-order and thus can be verified in logarithmic space. ■

7 Conclusion

Open complexity-theoretic problems relevant to the material discussed here are of course whether or not the polynomial hierarchy collapses to any level, whether or not there are any **PSPACE**-complete problems, whether or not **PSPACE** = **PH**, and whether or not there are any classes "between" **PH** and **PSPACE** (classes strictly contained in/containing one, which would imply that they are unequal). The famous specific example is of course whether or not the polynomial hierarchy collapses to the 0th level, or in other terms, $\mathbf{P} = \mathbf{NP}$.

Of course, the complexity classes discussed in this paper are far from all of those of interest to complexity theorists. Some classes of most interest include the class of languages that can be decided in logarithmic space by TMs, called **L**, the class of languages decided by logarithmic space NDTMs, called **NL**, the class of languages decided by exponential time ($O(2^n)$ steps) TMs, called **EXPTIME**, and the class of languages decided by exponential space TMs ($O(2^n)$ tape cells used), called **EXPSPACE**. We of course have logical characterizations of each of these classes as well as many more. A relevant image is Immerman's view of the world of complexity, including each computational and logical characterization of the classes, seen below:



We do study these logical characterizations of complexity classes as pure mathematics, but we also do care about what they tell us about complexity-theoretic problems. For example, given the result that $\mathbf{P} = FO(LFP)$, where $FO(LFP)$ is first-order logic with an added operator similar to transitive closure and related to induction, we have the result that $\mathbf{P} = \mathbf{NP}$ if and only if $SO\exists = FO(LFP)$. It seems like this must be false, and this does seem a promising method of attack on various probems of this type. Another example relevant to us is the general problem of whether or not our logical characterizations of Σ_i^p and Σ_{i+1}^p are equivalent, which also seems intuitively false, but of course given that the problems are still open we have not actually proven this. Finally, a very convenient rephrasing of the problem of whether or not **PSPACE** is equal to **PH** is the question of whether or not second-order logic gains any extra expressive power over structures with the "numeric" relations from earlier. While proving these equalities or inequalities would not necessarily demonstrate that given problems are in or not in specific classes, they do give us results about existence or nonexistence of, say, "efficient" (polynomial-time) algorithms for **NP**-complete problems or noninclusion of **PSPACE**-complete problems in **PH**. More on specific logics that capture complexity classes can be found in the survey [3] or the book [2].

Of course, given that we can reduce many of our complexity-theoretic problems to problems of finite model theory, we investigate the "expressive power" (what problems can be expressed in the logics) of the logics that capture complexity classes. An example of this is the use of Ehrenfeucht-Fraisse games (the uses of which can be seen in [2]), which we can use to derive results about the expressivity of first-order logic, or the use of $MSO\exists$ (the fragment of $SO\exists$ that has only monadic, or arity 1, relation variables) games, which are similar to EF games except they have an initial "coloring" stage that naturally better distinguishes structures. We also have Ajtai-Fagin games, and we can use them to prove "lower bounds" (lower limits of expressivity) for $MSO\exists$.

There are a great many logical characterizations of **PSPACE**, which can be found in [4], [5], and again of course [2]. Some of the most notable include characterizations of **PSPACE** with certain "choice" operators, the "partial fixed point" operator, the "least fixed point" operator, and first-order and second-order logic with various bounds on logical resources such as quantifier block sizes. For general complexity, [1] is a good first text.

Acknowledgements

The author would like to thank Sawyer Dobson for helpful critiques, Simon Rubinstein-Salzedo for support; and Paul Aoki and Allison Woodruff for helpful conversations.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[2] Neil Immerman. Descriptive complexity. In *Graduate Texts in Computer Science*, 1999.

[3] Neil Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.

[4] Flavio Ferrarotti, Jan Van den Bussche, and Jonni Virtema. Expressivity within second-order transitive-closure logic. *CoRR*, abs/1804.05926, 2018.

[5] David Richerby. Logical characterizations of pspace. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic*, pages 370–384, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.