Complexity Fest Student Lecture, CPSC 568, Fall 2022

A Quick and Unorthodox Introduction to Descriptive Complexity

December 13, 2022

Lecturers: Aidan Evans and Jessie Chen

Normally in the study of complexity theory, we ask, for example, "How hard is it decide whether a number is even?"; in descriptive complexity, on the other hand, we ask, "How hard is it to describe what an even number is?". Descriptive complexity is a subfield of computational complexity which aims to study complexity classes by representing languages/problems as sentences in formal systems of logic, e.g., first-order logic. In descriptive complexity, or more broadly, the field of *finite model theory*, we discuss the *expressive* power of logics — for example, what can we express, i.e., describe, in first-order logic? Profound connections have been proved between the expressive power of certain logics and complexity classes. For example, as we'll discuss below, a logic known as *existential second-order logic* (ESO) describes exactly those problems contained in NP; thus, we say that ESO *captures* NP. This is known as Fagin's theorem.

In what follows, we first provide a brief introduction to the study of logic using model theory in Section 1 — a subfield of mathematics and one of the two preliminaries to understanding descriptive complexity, the other preliminary being computational complexity theory itself. We then introduce Fagin's theorem formally in Section 2 and give a brief overview of the proof. In 3, we then turn to discussing logics which capture some other common complexity classes in and in Section 4, we introduce the important but often inadequately explained concepts of what it means for a logic to be *order-* and *arithmetic-invariant*. We include several exercises throughout this document; Section 5 contains a challenge exercise and Section 6 contains solutions for the exercises. Finally, Section 7 lists some additional material for those whom wish to learn more.

1 Crash Course in Model Theory

1.1 Structures

We're all familiar, at least informally, with *first-order logic* (FO): statements where we use the universal ($\forall$) and existential ($\exists$) quantifiers and logical connectives ($\wedge$, $\vee$, $\neg$, etc.), i.e., quantified boolean formulas, to formalize statements. The formulas are over fixed predicates/relations, like the edge relation E we use when discussing graphs. Here are some examples stating properties of graphs:

Example 1.1. Say we wish to formalize the sentence:

"$G = (V, E)$ is fully connected."

Then, we would write:

$$\forall x \forall y Exy$$

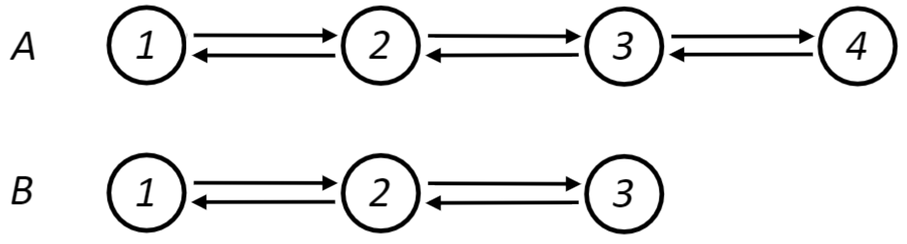

Figure 1: Graphs for Example 1.3

Example 1.2. "$G = (V, E)$ has a diameter of at most two." may be expressed as:

$$\forall x \forall y (x = y \vee Exy \vee \exists z(Exz \wedge Ezy))$$

Most of the time when we informally use first-order logic, we implicitly know by the context of our discussion what the domain of the quantifiers are, i.e., what the quantifiers "range over". This may be, for example, the natural number, the integers, or in the examples above the vertices of our graph. Model theory formalizes this using *structures*:

Definition 1.1 (Structures)**.** A structure $\mathfrak{A} = (A, R_1, R_2, \ldots)$ is a tuple where A is a set and $R_1, R_2, \ldots$ are relations/predicates on the set A. The set A is called the domain of $\mathfrak{A}$ and is typically denoted as either $|\mathfrak{A}|$ or $dom(\mathfrak{A})$. There may be no relations or infinitely many. Structures are also often referred to as *models*.

Remark 1.1. Observe that whenever we discuss graphs and write $G = (V, E)$, we are actually treating G as a structure. V is our domain and E is a relation specifying which two vertices have an edge between them.

Our structures provide an *interpretation* of our sentences. They help us take some arbitrary sentence and evaluate whether the sentence is actually "true". Truth here, however, is relative to our structure and, therefore, when we say a sentence is "true", we must always have a structure that the truth is with respect to.[1] Whenever we consider a sentence true with respect to a structure we say that the structure *satisfies* our sentence.

Definition 1.2. We say that a structure $\mathfrak{A}$ satisfies a sentence φ if and only if we evaluate φ to be true with respect to $\mathfrak{A}$.[2] We denote this as $\mathfrak{A} \models \varphi$.

Example 1.3. The FO sentence in Example 1.2 is satisfied by Graph B in Figure 1 but not Graph A.

[1]Note that the definition of what makes a sentence *true* outright, regardless of structure, is a bit more complicated. For our purposes right now, we don't need to know this, however. For more details, see [10].

[2]Again, like *truth*, the actual definition is a bit more complicated and involves recursively interpreting each symbol of the sentence. Think, how we use truth tables to evaluate whether a propositional sentence is true. For more details, see [10].

Definition 1.3 (Finite Structure)**.** In the general study of model theory, we do not have restrictions on the domain of our structures. In finite model theory, however, we require that the domains of our structures must finite sets. We refer to structures of this type as *finite structures*. From a complexity perspective, we pretty much only care about finite structures.

1.2 Second-Order Logic

Second-order logic (SO) is an extension of first-order logic where we add the ability to quantify relations. Thus, we may now use our quantifiers to act over relation symbols, like how we're used to using them to quantify variables. For the purpose of this lecture, we will first define formally an important fragment of SO known as *existential second-order logic* (ESO):[3]

Definition 1.4. ESO is an extension of first-order logic where sentences are of the form

$$\exists R_1 \ldots \exists R_k(\varphi)$$

where φ is a first-order formula and each R_i is a relation not yet bounded in the formula φ.

We then say that $\mathfrak{A} \models \exists R_1 \ldots \exists R_k(\varphi)$ if there exists relations $R_1, \ldots, R_k$ defined on the domain of $\mathfrak{A}$ such that $(\mathfrak{A}, R_1, \ldots, R_k) \models \varphi$.

In Section 2, we prove that the properties that ESO describes are exactly those in NP but, first, we need to formalize what we mean for a sentence to describe/express properties. We do this in the following subsection.

Exercises

Exercise 1.1. In this warm-up exercise, we consider how to write first-order logic and existential second-order logic expressions with some familiar examples.
Show how to express the following properties of a graph in first-order logic:

1. A graph $G = (V, E)$ has at least one vertex of degree 0.
2. Every vertex in the graph $G = (V, E)$ is connected by a path of length 3.

Show how to express the following examples in existential second-order logic:

3. A CNF formula φ is in SAT. Hint: a natural way to encode φ is to use the structure $\mathfrak{A} = (A, P, N)$, each $a \in A$ represents both a clause and a variable, P and N are relations s.t. $P(c, x)$ means variable x occurs positively in clause c and $N(c, x)$ means that variable x occurs negatively (i.e., $\neg x$) in clause c.
4. A graph $G = (V, E)$ is 3-colorable.
5. There exists a size k clique in the graph $G = (V, E)$.

[3]The literature also abbreviates ESO as $\exists$SO and SO$\exists$.

1.3 Connecting Logic to Complexity

When we say that a property is described or expressed by some sentence, what we are really talking about are properties of our structures. For example, a structure's domain could be either odd or even; thus, "having a domain with even cardinality" is a property. (We will locate which class this property falls within in Exercise 4.2.) Ultimately, whenever we prove that a logic and complexity class have the same computational power, we draw a correspondence between the structures which satisfy sentences of our logic and the inputs which an appropriate Turing machine for our complexity class accepts.

Definition 1.5 (Model Class)**.** We call the *model class* of a sentence φ inside a set of structures $\mathcal{D}$ to be the set of structures $\mathrm{Mod}_{\mathcal{D}}(\varphi) \subseteq \mathcal{D}$ which satisfy φ:

$$\mathrm{Mod}_{\mathcal{D}}(\varphi) = \{\mathfrak{A} \in \mathcal{D} \mid \mathfrak{A} \models \varphi\}$$

Definition 1.6. For a fixed sentence φ in a logic $\mathcal{L}$, the complexity of deciding membership to the set $\mathrm{Mod}_{\mathcal{D}}(\varphi)$ is known as the *data complexity* or *structure complexity*.

Remark 1.2. The terminology data complexity comes from the relationship of finite model theory to the study of relational databases: researchers are often interested in studying the expressive power of *query languages*, i.e., the languages used to query databases. Hence, sentences and formulae in logics are often also referred to as *queries*. For example, imagine a database has a bunch of graphs as its entries; then, we can have the database return all fully connected graphs with the query $\forall x \forall y Exy$.

Definition 1.7. We say that a logic $\mathcal{L}$ captures a complexity class $\mathcal{C}$ over structures $\mathcal{D}$ if

(1) The data complexity of $\mathcal{L}$ on $\mathcal{D}$ is $\mathcal{C}$. In other words, if φ is an $\mathcal{L}$-sentence, the problem of deciding for any $\mathfrak{A} \in \mathcal{D}$, whether $\mathfrak{A} \models \varphi$ is in $\mathcal{C}$.

(2) For every property P of structures in $\mathcal{D}$ which is decidable in complexity $\mathcal{C}$, there exists an $\mathcal{L}$-sentence φ such that for every $\mathfrak{A} \in \mathcal{D}$, $\mathfrak{A} \models \varphi$ iff $\mathfrak{A}$ has property P.

Remark 1.3. Observe that in the above definition, Condition (1) proves that any property expressible in logic $\mathcal{L}$ is a property decidable in complexity class $\mathcal{C}$. Therefore, $\mathcal{L} \subseteq \mathcal{C}$. Similarly, condition (2) proves that $\mathcal{C} \subseteq \mathcal{L}$.

Condition (1) tends to be the "easier" direction to prove because Condition (2) typically requires us to encode a Turing machine's behavior in terms of our logic — a somewhat complicated and pedantic task, as we've seen with the Cook-Levin theorem.

Remark 1.4. Ideally, we would like a logic to capture classes over all finite structures but, sadly, we often can only find logics which capture classes over a subset of finite structures. This is the reason why we include in Definition 1.7 a specification of what structures $\mathcal{D}$ we care about. If a logic $\mathcal{L}$ does capture a class $\mathcal{C}$ over *all* finite structures, then we simply say that $\mathcal{L}$ captures $\mathcal{C}$.

As we'll see in Section 2, ESO captures NP over all finite structures. We'll take a look at logics which captures classes only over a subset of the finite structures in Sections 3 and 4.

Remark 1.5. Technically, in order to pass a structure $\mathfrak{A}$ into a Turing machine, we need to encode $\mathfrak{A}$ as a string. The details are beyond the scope of this document but you can imagine something akin to the following just to get a concrete idea. To encode $\mathfrak{A} = (V, E) = (\{a, b, c\}, \{\langle a, b\rangle, \langle b, c\rangle\})$ and assuming our Turing machine uses the alphabet {0, 1, #}, we could write:

```
111###00#01##01#10
```

The "`111`" informs us that we have three elements in our domain V. The "`00#01##01#10`" then encodes our relation E where each $x \in V$ is encoded with $\lceil \log |V| \rceil$ bits; thus, $a \mapsto$ `00`, $b \mapsto$ `01`, and $c \mapsto$ `10`.

By being a bit more clever, we can remove the need to encode each $x \in |\mathfrak{A}|$ in binary — and, thus, avoid introducing the extra $\log |V|$ factor required to write each element. If $n = |V| = ||\mathfrak{A}||$, we will show in Exercise 2.1 that we can encode any k-ary relation in $O(n^k)$ bits. Therefore, we can encode $\mathfrak{A}$ in $O(poly(n))$ bits.

2 Fagin's Theorem: Capturing NP with ESO

Now that we know what it means for a logic to capture a complexity class, we may *finally* discuss specific cases of this. We start with the most infamous: Fagin's theorem [11]. In 1974, for his doctoral dissertation, Ronald Fagin proved that ESO captures NP over all finite structures. At the time, this was the first result which drew a relationship between complexity classes and the expressive power of logics and kickstarted what would turn into the field of descriptive complexity.

It's worth noting, however, that this was not the first capturing result in the study of computer science and finite model theory; in what is considered the "pre-history" of descriptive complexity theory, there also exists capturing results connecting types of formal languages to the expressive power of logic. The first such result occurred circa 1960 when Büchi [4], Elgot [9], and Trakhtenbrot [32, 33] all independently proved that the regular languages are captured by *monadic second-order logic* — second-order logic where we are only allowed to have unary relations.[4]

Returning to Fagin's theorem, since proving Condition (1) of Definition 1.7, i.e., ESO $\subseteq$ NP, is relatively easy, we will provide a detailed proof of this fact. For Condition (2), i.e., NP $\subseteq$ ESO, however, we only give a proof sketch because to give a proof which goes into all the nitty-gritty details would contradict this being a "quick" introduction and some of the details are quite similar to the Cook-Levin theorem.

Theorem 2.1 (Fagin)**.** *ESO captures NP (over all finite structures).*

Proof. We first prove Condition (1) of Definition 1.7; namely, that the data complexity of ESO is NP.

Let Φ be an ESO sentence. Without loss of generality, let Φ be of the form $\exists R(\varphi)$ where R is a relation of arity k unbounded in the first-order formula φ. We want to show that there exists

[4]If you are interested in learning more about this result, we suggest to not look at the original papers because they contain outdated terminology and are, well, tough to read in general. Instead, take a look at Chapter 7 of [23].

a non-deterministic Turing machine M such that given an aribtrary finite structure $\mathfrak{A}$ as input, M can decide whether $\mathfrak{A} \models \exists R(\varphi)$ in $O(poly(||\mathfrak{A}||)$ time.

The basic idea is that we will have M non-deterministically guess each possible k-ary relation R and then along each possible computation path, check whether $(\mathfrak{A}, R) \models \varphi$. To do so, we need to prove:

(a) We can write down our k-ary relation R, in $O(n^k)$ space, where $n = ||\mathfrak{A}||$.

(b) We can check whether $(\mathfrak{A}, R) \models \varphi$ within deterministic polynomial time.

We leave (a) as an exercise; see Exercise 2.1.

For (b), first observe that because the first-order quantifiers in φ range over the set $|\mathfrak{A}|$ which has cardinality n, we only have a linear number of possibilities to check per quantifier. For example, if we have the first order sentence $\forall x Rx$, then we could check whether this was true by iterating over each $x \in |\mathfrak{A}|$ and checking whether Rx is true; if checking whether Rx takes $T(n)$ time, then this whole operation takes $O(nT(n))$ time. If we have two quantifiers, such as $\forall x \exists y (Rx \vee Ry)$, then we'd iterate over $|\mathfrak{A}|$ twice, taking $O(n^2T(n))$ time. Because there will only be a finite number of quantifiers in φ, checking whether $(\mathfrak{A}, R) \models \varphi$ will always take $O(n^cT(n))$ time for some constant c. Finally, because it takes $O(poly(n))$ bits to write down R, it will take us at most $O(poly(n))$ time to have our tape head move through R to check if x is there; thus, checking Rx will take $T(n) = O(poly(n))$ time. Thus, we can deterministically check whether $(\mathfrak{A}, R) \models \varphi$ in $O(n^c poly(n)) = O(poly(n))$ time.

All together, to check $\mathfrak{A} \models \exists R(\varphi)$, we non-deterministically write down R. By (a), this takes polynomial space and, thus, may be done in polynomial time. After, we will have one computation path for every possible R. Then, have each path accept iff $(\mathfrak{A}, R) \models \varphi$. By (b), we can have each path check whether $(\mathfrak{A}, R) \models \varphi$ in polynomial time. Therefore, if $\mathfrak{A} \models \exists R(\varphi)$, then there exists an R such that $(\mathfrak{A}, R) \models \varphi$ so one of the computation paths of M will accept and, hence, M itself will accept; if $\mathfrak{A} \not\models \exists R(\varphi)$, then there does not exist an R such that $(\mathfrak{A}, R) \models \varphi$ so all paths will reject and, thus, M will reject.

We have, therefore, proven that M accepts $\mathfrak{A}$ iff $\mathfrak{A} \models \Phi$. Thus, ESO $\subseteq$ NP.

For the direction NP $\subseteq$ ESO, we give only a proof sketch. Let P be a property of finite structures for which there exists a polynomial-time non-deterministic Turing machine M which decides whether a structure $\mathfrak{A}$ has property P. In order to show that NP $\subseteq$ ESO, we need to construct an ESO sentence Φ_P such that $\mathfrak{A} \models \Phi_P$ iff $\mathfrak{A}$ has property P.

Let $\mathfrak{A}$ be an arbitrary finite structure. All we know is that M is a non-deterministic Turing machine which can decide whether $\mathfrak{A}$ has property P in polynomial time. In order to construct Φ_P, we encode the behavior of our Turing machine as a logical sentence.

Specifically, we will have several relations defined over $|\mathfrak{A}|$, which we denote by the tuple $\overline{R}$, such that these relations will represent a computation of M on input $\mathfrak{A}$. We will then construct a first-order sentence φ_P such that $(\mathfrak{A}, \overline{R}) \models \varphi_P$ iff the relations $\overline{R}$ represent an accepting computation of M on $\mathfrak{A}$.

Note that we are essentially having our sentence search over all possible paths our machine M could take on input $\mathfrak{A}$ by encoding each path in $\overline{R}$ and checking it's correctness with φ_P.

Most of the relations in $\overline{R}$ help us determine the configuration of M at different steps of the computation on $\mathfrak{A}$. For example, for each state q that M could be in (a finite number), we'll have the predicate:

$$X_q = \{t \mid \text{at time } t, M \text{ is in state } q\}$$

Thus, X_q is passed in a timestamp t and if M is in state q, then $X_q t$ will evaluate to true. Likewise, we also have predicates to tell us the tape contents and head position.

There's a small nuance we left out of the above definition: namely, if our computation takes a polynomial number of steps, it's not at first glance clear how we ensure that all timestamps can be passed in to our relation because our relation can only have a finite arity — even though the number of timestamps we need to handle is unbounded. We leave this for Exercise 2.2.

Our first-order sentence φ_P will then be of the form

$$\textsc{Start} \wedge \textsc{Computation} \wedge \textsc{End}$$

where

- START ensures that our relations correctly represent the starting configuration a Turing machine should be in; i.e., we're in our initial state, etc.
- COMPUTATION ensures that at each timestamp, only those cells which should have been changed were and that we made a valid transition from one state to the next. (This is the most detailed of the three clauses.)
- END finally ensures that we eventually end up in an accepting state.

After a "little" more work, we eventually get that M accepts $\mathfrak{A}$, i.e., $\mathfrak{A}$ has property P, iff $\mathfrak{A} \models \Phi_P$. Thus, $NP \subseteq ESO$.

Therefore, we have proven that ESO captures NP so we finally have that NP = ESO. Thus, we conclude our "brief" explanation of Fagin's theorem.

□

Remark 2.1. For all the details of Fagin's theorem, several sources are available containing a proof of it. Notably, the original source [11] is worth checking out for *all* the details. Textbooks [13, 23, 22, 27] also all contain proofs using slightly different methods — and of various degrees of difficulty to understand.

Remark 2.2. From [14], if instead of Turing machines, we consider a model of computation known as non-deterministic RAM machines, then we actually know of a precise relationship between ESO and $O(n^k)$ computation. Namely, for each $k \geq 1$, NRAM-TIME($O(n^k)$) = ESO(k-arity, $k\forall$, fun); that is, ESO with functions, only at most k first-order universal quantifiers, and each relation and function has arity at most k. We don't know of such tight bounds for Turing machines. (Also, note that this model of computation differs from "TMs with random-access", defined in Section 4.2.2.)

Exercises

Exercise 2.1. Show that we can write down our k-ary relation R in $O(n^k)$ space, where $n = ||\mathfrak{A}||$. Note that arity is the number of arguments or operands taken by the relation. *Hint:* First try to encode the familiar edge relation E using n^2 bits by thinking about the adjacency matrix representation.

Exercise 2.2. Our quantified relations, X_q for each state q, need to have a fixed finite arity. They take a tuple of variables as input; this tuple represents an encoding of what step of the computation we are at, i.e., the timestamp. Our computation takes $O(n^k)$ time and, therefore, has $O(n^k)$ timestamps. Prove that we can encode all $O(n^k)$ timestamps as a constant length tuple which we can then pass into X_q, thus, proving that X_q has a fixed finite arity.

Exercise 2.3. Construct the END part of our first-order sentence φ_P.

3 Capturing Other Classes

3.1 coNP and PH

As a corrolary to Fagin's theorem, we get:

Corollary 3.1. *Universal second-order logic (USO), SO with only universal second-order quantifiers, captures coNP over all finite structures.*

Say that a problem P in NP is captured by ESO sentence Φ. Complementing P surmounts to simply negating Φ; thus, $\overline{P} = \neg\Phi$. Since $\neg\exists \equiv \forall\neg$, we now have a USO sentence capturing $\overline{P}$.

Theorem 3.2. *SO captures PH over all finite structures.*

Proof. This proof is similar to Fagin's theorem and, therefore, we omit the proof here. □

3.2 Fragments of SO

We now turn to discussing some of the complexity classes within NP. In this subsection, we focus our attention on P followed by a brief discussion of L and NL. In 1982, Immerman [19, 21], Vardi [34], and Libchak [24] all independently proved that FO extended with a special *least-fixed point* operator captures P over all *ordered* finite structures; this has become known as the Immerman-Vardi theorem. Other extensions of FO were later shown to capture L and NL. Instead of approaching the task of capturing L, NL, and P by constructing logics which *extend* FO as most introductory texts do, we instead follow the work of Grädel [12] and look at logics which *restrict* SO, i.e., fragments of SO.

Remark 3.1. In this section, all of the capturing results only hold over all ordered finite structures. We postpone discussion of what this means until Section 4 but we note that if a logic $\mathcal{L}$ captures a class $\mathcal{C}$ over all *ordered* finite structures, then we write that $\mathcal{C} = \mathcal{L}(<)$ or that $\mathcal{L}(<)$ captures $\mathcal{C}$. We also note that $\mathcal{L} \subseteq \mathcal{L}(<)$.

3.2.1 Capturing P

Definition 3.1 (Propositional Horn Formula)**.** A propositional formula is in *Horn form* if it is of the form $\bigwedge_{i=1}^{i} C_i$ where each clause C_i is a disjunction of literals containing *at most* one positive literal, i.e., each clause has at most one variable without a negation. Thus, each C_i is either of the form $x_1 \vee \neg x_2 \vee \cdots \vee \neg x_j$ or $\neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_j$.

Remark 3.2. Horn formulae are named after logician Alfred Horn whose work on studying formulae of this type in 1951 [18] led to Horn formulae's significant role in the fields of computational logic, automated theorem proving, and logic programming languages, such as Prolog.

Definition 3.2 (ESO-Horn)**.** ESO-Horn is a fragment of ESO. An ESO-Horn sentence is of the form:

$$\exists R_1 \ldots \exists R_i \left(\forall \overline{x} \left(\bigwedge_j C_j \right) \right)$$

such that each C_j is either of the form $\alpha \vee \beta_1 \vee \cdots \vee \beta_m$ or $\beta_1 \vee \cdots \vee \beta_m$ where α is a positive literal using one of the existentially quantified relations $(R_1, \ldots, R_i)$ and each β_k is either a negative literal using one of $R_1, \ldots, R_i$ or is a first-order formula not containing any $R_1, \ldots, R_i$. In other words, the first-order part is in Horn form with respect to the quantified relations.

Definition 3.3 (SO-Horn)**.** Like ESO-Horn but now we can use any second-order quantifiers, not just existential ones.

We now aim to prove that SO-Horn captures P over all ordered finite structures. First, we discuss two lemmas:

Lemma 3.3. *ESO-Horn $\subseteq$ P.*

Proof. To show that ESO-Horn $\subseteq$ P, we show that for any sentence $\psi \in$ ESO-Horn, we can decide whether a finite model satisfies ψ in P.

W.l.o.g., let $\psi = \exists R_1 \ldots \exists R_i(\forall \overline{x}(\wedge_j C_j))$, and A be the domain of the structure $\mathfrak{A}$. First, notice that we can expand the first-order part of the sentence to remove the universal quantifiers $\forall x_j$. More specifically, we can replace each universal quantifier $\forall x_k$ with conjunctions plugging in each $x \in A$ for x_k. For example, $\forall x(Rx)$ with the domain $A = \{a, b, c\}$ would become $Ra \wedge Rb \wedge Rc$. Since the conjunctions are taken over elements in A, the replacement leaves us with a polynomial-sized formula in $|A|$.

Then, for each clause in the first-order formula that does not contain any relation R_m ($1 \leq m \leq i$), we can evaluate the clause by interpreting it using $\mathfrak{A}$. By a similar argument in Theorem 2.1 (b), this evaluation can be completed in polynomial time. Then one can simplify the first-order formula by removing clauses evaluated to be true and any repeated clauses. Or if there is a clause evaluated to be false, we can reject ψ directly — since the whole sentence will evaluate to false.

After the simplification, the rest of the first-order part is quantifier-free and polynomially bounded in size of $|A|$. And each atom $R_i\overline{c}$ can be bijectively mapped to a propositional variable;

thus, the sentence takes the form of a propositional Horn formula. For example, let $A = \{a, b, c\}$; the remaining expression

$$\exists R_1 \exists R_2((R_1 a \vee \neg R_1 b) \wedge (\neg R_2 c \vee \neg R_1 b))$$

could be mapped to the propositional formula

$$(w_1 \vee \neg w_2) \wedge (\neg w_3 \vee \neg w_2)$$

This propositional formula will be satisfiable iff we can actually construct Rs which satisfy the first-order part of ψ.

Because the propositional formula is in Horn form, we can determine satisfiability in polynomial time (see Exercise 3.1). Our propositional formula has length $O(poly(|A|))$ so our computation wil take $O(poly(poly(|A|))) = O(poly(|A|))$.

Thus, we can decide the set of finite models of ψ in polynomial time so ESO-Horn $\subseteq$ P. $\square$

Lemma 3.4. *SO-Horn collapses to ESO-Horn. In other words, any SO-Horn sentence may be rewritten as an equivalent ESO-Horn sentence.*

Proof. This proof is beyond the scope of this lecture; see [12] and Chapter 3 of [13] for details. At the end of the day, we can do this because P is closed under complementation. $\square$

Theorem 3.5 (Grädel)**.** *SO-Horn captures P over all ordered finite structures; i.e., P = SO-Horn(<).*

Proof. First, by Lemmas 3.3 and 3.4, we know that SO-Horn $\subseteq$ P. While we still haven't discussed "order" yet, know that this result also holds over all ordered finite structures; thus, we may write that SO-Horn(<) $\subseteq$ P. This satisfies Condition (1) of Definition 1.7.

For the other direction, we prove that P $\subseteq$ ESO-Horn(<) and, thus, P $\subseteq$ SO-Horn(<). We sadly must omit the nitty-gritty details of this direction but here's the idea: by taking a careful look at Fagin's theorem when we don't need to worry about non-determinism, we see that the ESO sentence constructed is actually an ESO-Horn(<) sentence. Again, this is only because we are now dealing with deterministic TMs as opposed to non-deterministic TMs. Thus, given a problem in P, we can express it as an ESO-Horn(<) sentence. This satisfies Condition (2) of Definition 1.7.

Therefore, ESO-Horn captures P over ordered finite structures. $\square$

3.2.2 Capturing NL and L

We may similarly construct fragments of SO which capture NL and L over all ordered finite structures. In the case of NL, we restrict the FO part to be of *Krom form*, i.e., 2CNF, with respect to the quantified relations; essentially, the relationship between SO-Krom and propsitional 2CNF formulae is analagous to the relationship between SO-Horn and propositional Horn formulae. We formalize the logic of ESO-Krom in the exercises below.

For L, things get a bit more complicated. Namely, Grädel's original work in 1992 ([12]) constructed a logic which captures coSL, the complement of symmetric logspace,[5] over all ordered finite structures. The first-order part of this logic was of a form where we knew satisfiability could be tested in coSL. Because we now know from Reingold's result in 2004 ([28]) that L = SL = coSL, the fragment of SO constructed by Grädel, therefore, captures L over all ordered finite structures.

Exercises

Exercise 3.1. Prove that the satisfiability problem for propositional Horn formula (HORN-SAT) is P-complete. This provides a motivation for considering why seconder-order Horn logic (SO-HORN) captures P. In fact, it has been well-established that HORN-SAT can be solved in linear time [6].

Definition 3.4 (ESO-Krom)**.** ESO-Krom is a fragment of ESO. An ESO-Krom sentence is of the form:

$$\exists R_1 \ldots \exists R_i \left(\forall \overline{x} \left(\bigwedge_j C_j \right) \right)$$

such that each C_j is a disjunction of at most two literals using the $R_1, \ldots, R_i$ relations and of a first-order formula that does not contain $R_1, \ldots, R_i$. In other words, the first-order part is in Krom/2CNF form with respect to the quantified relations.

Exercise 3.2. Prove that ESO-Krom $\subseteq$ NL. Hint: Note that 2SAT is NL-complete.

4 Invariance and Small Complexity Classes

We finally turn our attention to complexity classes which are captured by logics only over subsets of all finite structures, e.g., over all *ordered* finite structures. We focus our attention here on logics which captures extremely small complexity classes; specifically, we discuss classes which are known to be *proper* subsets of L. For the most part, we omit proofs in this section due to a desire to keep this a "quick" introduction; where applicable, we provide basic ideas or point to sources providing proofs — this has turned Section 4.2.2 into something of a reference page.

4.1 Defining Invariance

We carve out subsets of the finite structures by making our logics *invariant* with respect to certain relations. For example, let $\mathcal{L}$ be a logic; we denote order-invariant $\mathcal{L}$ by $\mathcal{L}(<)$.[6] $\mathcal{L}(<)$ extends $\mathcal{L}$ by

[5]We suggest taking a look at `https://complexityzoo.net/Complexity_Zoo:S#sl` for a concise definition of SL.

[6]Sadly, there isn't a consistent notation for invariance in the literature. Most texts use $\mathcal{L}(<)$, others $<$-inv-$\mathcal{L}$. Some texts don't even specify; e.g., [22] just uses FO to denote arithmetic-invariant FO even though "vanilla" first-order logic and arithmetic-invariant first-order logic are two different things. We use $\mathcal{L}(<)$ because it is the most concise for our purposes.

giving $\mathcal{L}$ access to an arbitrary total/linear ordering relation $<$. Informally, imagine that we were to take the domain of our structure and put the elements in some arbitrary line; the ordering relation takes two elements as input a, b and returns true iff a comes before b in this line. Importantly, we don't actually know what the specific ordering is; hence, why we call the logic order *invariant*: all we know is that there is an ordering and that $<$ allows us to query this order. We can't actually assume that $<$ is some specific ordering. The relation $<$ is what we call a *numerical predicate*; we now formalize this concept so that we may discuss numerical predicates more generally:

Definition 4.1 (Embedding)**.** Let $\mathfrak{A}$ be a finite structure such that $||\mathfrak{A}|| = n$ and let $[n] = \{x \in \mathbb{N} \mid 1 \leq x \leq n\} = \{1, 2, \ldots, n\}$. An *embedding* g is a bijection $g : |\mathfrak{A}| \to [n]$. In other words, g is a function which uniquely maps the elements of our structure's domain to the first n non-zero natural numbers.[7]

Remark 4.1. In what follows, we need to distinguish the numerical symbols we use in our logical sentences from the same symbols we typically use to make claims about the natural numbers. For example, we will need to specify whether the symbol "$<$" is being used as a predicate in our logic or whether it's the "less than" symbol over the natural numbers. Unless the context is obvious, we will use "$<^g$" to denote the predicate symbol in our logic and we will use "$<^{\mathbb{N}}$" to denote the "less than" symbol over the natural numbers. We will likewise use the same notation for other numerical predicates we come across, e.g., $+$ and $\times$.

Definition 4.2 (Order-Invariant)**.** A logic $\mathcal{L}$ is order-invariant, denoted $\mathcal{L}(<)$ if in addition to the predicates and symbols normally available, we may also use an additional binary predicate $<^g$ in our sentences.

Let $\mathfrak{A}$ be a finite structure. For $a, b \in |\mathfrak{A}|$, we say that $a <^g b$ iff $g(a) <^{\mathbb{N}} g(b)$ where g is an arbitrary embedding. We call $<^g$ a total/linear order.[8]

Remark 4.2. Observe that when we work with database queries, we automatically have an arbitrary ordering present: the memory address of our pieces of data. This is why researchers tend to be interested in order-invariant queries. We know we'll have an ordering but we have no specifics about that ordering because the operating system manages the memory for us.

Example 4.1. Consider the FO($<$) sentence φ:

$$\exists x \exists y (x < y)$$

For a finite structure $\mathfrak{A}$, we say that $(\mathfrak{A}, <) \models \varphi$ iff there exists $a, b \in |\mathfrak{A}|$ such that $a <^g b$ for an arbitrary embedding g. (Note that we need to specify "$<$" on the lefthand-side of "$\models$" because if we didn't, we technically would not have the ability to interpret the $<$ symbol in our sentence.)

Observe that φ is satisfied for $||\mathfrak{A}|| \geq 2$ because irrelevant of which embedding is chosen, one element of $\mathfrak{A}$ will be mapped to 1 and the other to 2; thus, we have two elements a, b such that

[7]Note that convention for $[n]$ also is not standard. Some texts, such as [23], use $[n] = \{1, 2, \ldots, n\}$ while others, such as [29], use $[n] = \{0, 1, \ldots, n-1\}$. While this doesn't necessarily change the expressive power of our logic, it may lead to describing properties with slightly different sentences.

[8]Technically, $<$ is a *strict total order*; cf. `https://en.wikipedia.org/wiki/Total_order`.

$a <^g b$. If $||\mathfrak{A}|| = 1$, then the only element in $|\mathfrak{A}|$ would map to 1 and, therefore, we'd only have that $1 <^{\mathbb{N}} 1$, which is false.

Definition 4.3 (Arithmetic-Invariant)**.** A logic $\mathcal{L}$ is arithmetic-invariant, denoted $\mathcal{L}(+, \times)$ if in addition to the predicates and symbols normally available, we may also use two additional ternary (3-ary) predicates $+^g$ and $\times^g$.

Let $\mathfrak{A}$ be a finite structure. For $a, b, c \in |\mathfrak{A}|$, we say that

- $+^g abc$ if and only if $g(a) +^{\mathbb{N}} g(b) = g(c)$
- $\times^g abc$ if and only if $g(a) \times^{\mathbb{N}} g(b) = g(c)$

for an arbitrary embedding g. For shorthand and to ease readability, we write $a +^g b = c$ to represent $+^g abc$, and likewise for $\times^g$.

Definition 4.4 (Arb-Invariant)**.** A logic $\mathcal{L}$ is arb-invariant, denoted $\mathcal{L}(Arb)$ if in addition to the predicates and symbols normally available, we may use *any* arbitrary numerical predicate definable over the natural numbers. For example, we could have a unary predicate Odd^g which tells us for some $a \in |\mathfrak{A}|$ whether $g(a)$ is an odd number.

Exercises

Exercise 4.1. Prove that ESO $=$ ESO($<$).

Exercise 4.2. In this exercise, we will prove that whether our input structure has an even cardinality, i.e. EVEN($\mathfrak{A}$), is in FO($+, \times$). To do that, show that

1. EVEN $\in$ FO($<, +$).
2. FO($<, +$) $\subseteq$ FO($+, \times$).

This will show that describing what it means for something to be even may be expressed in FO($+, \times$) and may also be decided by a really small circuit class defined in Section 4.2.2 below.

4.2 Capturing Results

We now discuss some interesting results and observations about the relationship among first-order logic extended with various numerical predicates and the classes of languages they capture.

4.2.1 A Hierarchy of Predicates

Theorem 4.1. *FO $\subsetneq$ FO($<$) $\subsetneq$ FO($+, \times$) $\subsetneq$ FO(Arb)*

Proof. That all of the above are subsets follows from the fact that we can, for example, express the relation $<$ in FO($+, \times$), see Exercise 4.2.

The fact that they are all *proper* subsets is beyond the scope of this lecture. The common method for proving such results is to use what are known as Ehrenfeucht-Fraïssé games; see Chapters 3 and 5 of [23] for details. $\square$

Remark 4.3. This may seem counter-intuitive but by restricting our structures to only those where we have, for example, a total ordering, we are actually extending the expressive power of our logic. In other words, by only caring about a subset of all finite structures, we are able to take advantage of this subset's properties in order to express more about it. Because the set of all ordered finite structures is still countably infinite like the set of all inputs to a Turing machine, we can still draw a bijection between our the set of inputs a Turing machine accepts and each ordered finite structure which satisfies a corresponding logical sentence.

Remark 4.4. A wealth of research has also been conducted on extending FO with other numeral predicates, including algebraic characterizations of these classes using what are known as *syntactic monoids* and *semigroup theory*. For an introduction to the subject, see Straubing's textbook [31].

4.2.2 Capturing with FO($+$, $\times$) and FO(Arb)

Definition 4.5 (TM with random-access)**.** A Turing machine with *random-access* is a Turing machine with a special *query tape* which allows us to "index into" the input tape. Specifically, we may write a number x in binary on the query tape and then upon entering a special *query state*, the head of our TM jumps to the x^{th} cell of our input tape.

Observe that if our input is of length n, on a TM with random-access, we can jump to the last bit of our input (the $(n-1)^{th}$ cell) in only $O(\log n)$ time because it only takes $O(\log(n))$ space, and thus $O(\log(n))$ time, to write $n-1$ in binary on our query tape.

Definition 4.6 (DLOGTIME)**.** DLOGTIME is the class of problems decidable by a deterministic Turing machine with random-access in $O(\log n)$ time.

Definition 4.7 (The Logarithmic-time Hierarchy (LH))**.** LH is to DLOGTIME what PH is to P.

Definition 4.8 (Uniform Circuit Class)**.** A circuit class is *$\mathcal{C}$-uniform* if for each family of circuits $\{C_0, C_1, ...\}$ deciding a problem, there must exist a Turing machine with complexity $\mathcal{C}$ such that on input of length k outputs an encoding of circuit C_k. Thus, we are no longer be able to just choose each circuit C_i with no restrictions; observe that this prevents uniform circuit classes from containing undecidable problems, like the non-uniform circuit classes we are familiar with.

Definition 4.9 (Unbounded Fan-in Gates)**.** A logic gate has is an *unbounded fan-in* gate if it can handle any number of inputs. This is in contrast to *bounded* fan-in gates which only accept a constant number of inputs — like how the circuits we've discussed in class only use gates which have at most two inputs arguments.

Definition 4.10 (Non-Uniform AC^0)**.** Let P be some property. If P can be decided by a circuit family such that for each input x of length k, there exists a polynomial-sized *bounded-depth* circuit with *unbounded fan-in* AND and OR gates which decides whether x has property P, then we say that P is in the complexity class non-uniform AC^0.

Definition 4.11 (DLOGTIME-uniform AC^0)**.** The DLOGTIME-uniform version of non-uniform AC^0, i.e., the same type of circuits as non-uniform AC^0 but the circuit families for each problem must be generated by a DLOGTIME Turing machine.

Theorem 4.2 (What FO($+$, $\times$) Captures)**.** *FO($+$, $\times$) $=$ LH $=$ DLOGTIME-uniform AC^0.*

Proof. See [3] and [22]. □

Theorem 4.3 (What FO(Arb) Captures)**.** *FO(Arb) $=$ non-uniform AC^0.*

Proof. See [20, 16, 2, 26]. □

Theorem 4.4 (LH Does **Not** Collapse)**.** *The log-time hierarchy (LH) is proper; i.e., it does not collapse.*

Proof. See [17, 35]. This corresponds to proving lower-bounds for DLOGTIME-uniform AC^0 circuits; for the analogous lower bounds on non-uniform AC^0 circuits, see [30]. □

Theorem 4.5. *FO$(+, \times) \subsetneq$ ALOGTIME.*

Proof. See [5] (uses Theorem 4.4). □

Remark 4.5. ALOGTIME is the class of problems solvable in alternating logarithmic time by TMs with random-access. Note that ALOGTIME $\subseteq$ L; therefore, FO($+$, $\times$) is really small.

We actually don't know whether ALOGTIME is a proper subset of NP or PH. Also, if a class $\mathcal{C} \subsetneq$ ALOGTIME, then $\mathcal{C} \subseteq$ FO($+$, $\times$). Therefore, this is the smallest class for which we don't know of a separation from NP or PH.

4.2.3 Capturing with FO($<$)

Definition 4.12 (Star-Free Regular Languages)**.** The star-free regular languages (SF) are a subset of the regular languages. Namely, a regular languages is *star-free* (and hence in SF) if it can be described by a regular expression constructed using the letters of the alphabet, the empty set symbol, union, intersection, complementation, and concatenation but with no use of the Kleene star.

Theorem 4.6. *FO($<$) = SF*

Proof. See Straubing's textbook [31]; originally, [25]. □

4.3 The Undecidability of Invariance

Often, when discuss connections between a logic $\mathcal{L}$ and complexity class $\mathcal{C}$, it is desirable that (1) $\mathcal{L}$ captures $\mathcal{C}$ over *all* finite structures and (2) $\mathcal{L}$ is an *effective* logic:

Definition 4.13 (Effective Logic)**.** Let $\mathcal{L}$ be a logic. Let σ be the set of symbols used to construct $\mathcal{L}$-sentences; let σ^* be the set of strings using symbols from σ. We say that $\mathcal{L}$ is *effective* if membership to the set $\{x \in \sigma^* \mid x \text{ is a } \mathcal{L}\text{-sentence}\}$ is decidable — i.e., we have a TM which can tell us whether a string $x \in \sigma^*$ is a sentence of $\mathcal{L}$. Otherwise, we say that $\mathcal{L}$ is *non-effective*.[9]

[9] cf. [15] and Lindström's Second Theorem in Chapter 13 of [8].

Remark 4.6. Observe that FO and ESO are both *effective* logics because given a string of symbols φ — where φ uses the symbols: $\forall$, $\exists$, R_1, x, y, $\wedge$, etc. — we can decide whether φ is an FO-sentence by making sure that each quantfier has a variable after it, that each left parenthesis has a matching right parenthesis, etc. For example, we can obviously construct an algorithm to tell us that "$\forall x \exists y(R_1 x \wedge \neg R_2 y)$" is a sentence in FO but that "$\forall(x \neg \wedge R_1))$" is not.

Theorem 4.7. *FO$(<)$ is non-effective.*

Remark 4.7. To prove Theorem 4.7, we would prove that given a string φ which uses the symbols of FO plus the "$<$" symbol, we cannot decide whether φ is a FO$(<)$ sentence. Note that we can clearly decide if a string is an FO sentence which simply *uses* the "$<$" symbol. The undecidability, however, comes from the fact that our sentence must be $<$-invariant (order-invariant); i.e., we can't just use "$<$" however we want. This is typically proved by using a many-one reduction to *Trakhtenbrot's Theorem*:

Theorem 4.8 (Trakhtenbrot)**.** *Membership to the set of FO-sentences satisfiable by some finite structure is undecidable. Formally, the problem of determining whether an FO-sentence is in the set*

$$\{\varphi \mid \varphi \textit{ is an FO-sentence and there exists a finite structure } \mathfrak{A} \textit{ s.t. } \mathfrak{A} \models \varphi\}$$

is undecidable.

Proof. See Chapter 9 of [23] for details. □

Remark 4.8. Almost all invariant logics are non-effective. (All of the ones we have seen so far are.) There are a few known exceptions: for example, order-invariant FO with only unary relations is decidable. This follows from its relationship to the regular languages and some algebraic details — both of which are beyond the scope of this lecture.

Remark 4.9. When a logic is effective, it is sometimes referred to as *syntactic* because we can decide sentence membership based solely on the syntax of the string. When a logic is non-effective, it is referred to as *semantic*; we need to actually know the meaning assigned to the the symbols of a string in order to decide membership.

Conjecture 4.1 (Gurevich)**.** *There do not exist effective logics which capture P and NP $\cap$ coNP over all finite structures.*

Corollary 4.9. *If P does not have an effective logic which captures it over all finite structure, then P $\neq$ NP.*

Remark 4.10. In 1985, Gurevich conjectured that P and NP $\cap$ coNP do not have effective logics [15]. This has been an open question ever since and arguably *the main* question in the field of descriptive complexity. The only logics known to capture P are non-effective logics which capture over ordered finite structures, e.g., ESO-Horn$(<)$. (ESO-Horn$(<)$ is non-effective for the same reason FO$(<)$ is.)

4.4 Non-Probabilistic Weak Verifiers for NP

On a past pset, we proved that NP may also be characterized as the class of problems for which there exists a polynomial-size certificate which we can be verified by a deterministic log-space TM. In this subsection, we prove an even stronger claim: we prove as a corollary of Fagin's theorem that we can in fact use DLOGTIME-uniform AC^0 circuits to verify a polynomial-sized certificate for NP.[10]

Theorem 4.10. *NP is the class of problems for which on there exists a polynomial-size certificate c which can be verified using DLOGTIME-uniform AC^0 circuits.*

Proof. Let P be a problem in NP. By Fagin's theorem, we know that there exists an ESO-sentence Φ such that $\mathfrak{A} \in P$ iff $\mathfrak{A} \models \Phi$. W.l.o.g., let $\Phi = \exists R(\varphi)$ where φ is an FO formula and R has arity k.

Let $\mathfrak{A}$ be arbitrary; let $n = ||\mathfrak{A}||$. If $\mathfrak{A} \in P$, then $\mathfrak{A} \models \exists R(\varphi)$. Therefore, there exists a relation R such that $(\mathfrak{A}, R) \models \varphi$. By Exercise 2.1, we know that R may be written down using $O(n^k) = O(poly(n))$ bits. This will be our polynomial-sized certificate.

In the original proof of Fagin's theorem in Section 2, we proved that $(\mathfrak{A}, R) \models \varphi$ could be checked with complexity P. In that case, we essentially proved that our certificate R could be verified in poly-time. We now know that FO $\subseteq$ FO$(+, \times)$ = DLOGTIME-uniform AC^0 by Theorems 4.1 and 4.2. Thus, because φ is an FO-sentence, we can check whether $(\mathfrak{A}, R) \models \varphi$ with a complexity of DLOGTIME-uniform AC^0.

The other case, when $\mathfrak{A} \notin P$ is similar. Therefore, our input $\mathfrak{A}$ is in P iff there exists a certificate R which may be verified using DLOGTIME-uniform AC^0 circuits. $\square$

Remark 4.11. Observe that this is quite surprising when one remembers that DLOGTIME-uniform AC$^0 \subsetneq$ P. Therefore, despite DLOGTIME-uniform AC^0 being a proper subset of P, both classes in some sense have the same power when it comes to verifying polynomial-sized certificates.

In fact, by being a little more careful in our analysis, we can even prove that problems in NP have polynomial-sized certified which can be verified in coNLOGTIME. (coNLOGTIME is to DLOGTIME what coNP is to P.) This is really small!

Remark 4.12. By following the same procedure as done in the above theorem on the SO fragments in Section 3.2, we can also represent L, NL, and P in terms of polynomial-sized certificate verification. Each class's certificate may be verified by the corresponding fragment of first-order logic being existentially quantified over in its SO fragment. Thus, for example, certificates for P may be verified by the first-order Horn sentences in ESO-Horn$(<)$.

[10]Check out the PCP theorems for a probabilistic analog to weak verifiers.

5 Challenge Exercise

This following exercise relies upon intuition from all of the previous sections.

Definition 5.1 (ESO-Horn'). ESO-Horn' is a fragment of ESO. An ESO-Horn' sentence is of the form:

$$\exists R_1 \dots \exists R_i \left(\forall \overline{x} \left(\bigwedge_j C_j \right) \right)$$

such that each C_j is either of the form $\alpha \vee \beta_1 \vee \cdots \vee \beta_m$ or $\beta_1 \vee \cdots \vee \beta_m$ where α is a positive literal and each β_k is a negative literal. In other words, the first-order part is in Horn form with respect to the relations.

Exercise 5.1. Prove that if P $=$ ESO-Horn'($<$), then P = NP.

Hint 5.1. How does ESO-Horn' differ from ESO-Horn?

Hint 5.2. Thinking about the standard verifier-based definition for NP may help.

6 Solutions to Exercises

6.1 Section 1 Solutions

Exercise 1.1. Warm-up exercise for first-order logic and existential second-order logic expressions.
1. $\exists x \forall y (\neg Exy)$.
2. $\forall x \forall y \exists z_1 \exists z_2 (Exz \wedge Ez_1 z_2 \wedge Ez_2 y)$.
3. If a CNF formula $\varphi \in$ SAT, then there exists some satisfying assignment of variables for φ. Using the structure mentioned in the hint, the following sentence asserts that $\varphi \in$ SAT.

$$\exists S \{\forall x \exists y [P(x,y) \wedge S(y)) \vee (N(x,y) \wedge \neg S(y)]\} \tag{1}$$

Here, S is the set of variables assigned true in the satisfying assignment.
4. If a graph $G = (V, E)$ is 3-colorable, then there exists three disjoint sets R, Y, B s.t. $V = R \cup Y \cup B$ and for every edge $(x, y) \in E$, x and y belong to different disjoint sets. Thus, we can define R, Y, B as unary relation variables and write the following two-part sentence that describes 3-colorability.

$$\exists R \exists Y \exists B \begin{pmatrix} \forall x \begin{bmatrix} (R(x) \wedge \neg Y(x) \wedge \neg B(x)) \\ \vee(\neg R(x) \wedge Y(x) \wedge \neg B(x)) \\ \vee(\neg R(x) \wedge \neg Y(x) \wedge B(x)) \end{bmatrix} \\ \wedge \\ \forall x \forall y, Exy \to \neg \begin{bmatrix} (R(x) \wedge R(y)) \\ \vee(Y(x) \wedge Y(y)) \\ \vee(B(x) \wedge B(y)) \end{bmatrix} \end{pmatrix} \tag{2}$$

One can check for themselves that the first bracket tests the constraint that $V = R \cup Y \cup B$ while R, Y, B are disjoint. And the second bracket tests that for every edge $(x, y) \in E$, x and y belong to different disjoint sets.
5. Let $U \subseteq V$ s.t. $|U| = k$. We want to test whether there exists a set C that gives the nodes of a clique and a binary relation M that is a one-to-one mapping between C and U. This motivates us to write the following sentence:

$$\exists C \exists M \begin{pmatrix} \forall x \forall y [M(x,y) \to (C(x) \wedge U(y))] \\ \wedge \forall x [C(x) \to \exists! y (M(x,y) \wedge U(y))] \\ \wedge \forall y [U(y) \to \exists! x (M(x,y) \wedge C(y))] \\ \wedge \forall x \forall y [C(x) \wedge C(y) \to E(x,y)] \end{pmatrix} \tag{3}$$

Here, $\exists! y (M(x,y) \wedge U(y))$ is an abbreviation for there is exactly one y s.t. $M(x,y) \wedge U(y)$ is true. Thus, the first three parts of the sentence tests the existence of a one-to-one mapping between C and U. And the last part tests whether C represents a clique.

6.2 Section 2 Solutions

Exercise 2.1. Show that we can write down our k-ary relation R in $O(n^k)$ space, where $n = ||\mathfrak{A}||$. Note that arity is the number of arguments or operands taken by the relation. *Hint:* First try to encode the familiar edge relation E using n^2 bits by thinking about the adjacency matrix representation.

Proof. First we address the hint, in an a graph with n vertices, we could have n^2 possible edges. An adjacency matrix M has n^2 entries where entry $M_{ij} = 1$ if there's an edge between vertex v_i and vertex v_j and M_{ij} if there's no edge. M will have n rows an n columns where each entry only contains a 0 or 1; thus, we can write down M by writing the first row using n bits, followed by the second row using n bits, and so on until we've written down all n rows. This will leave us with a n^2 length bit string where each consecutive string of n bits encodes a row.

Given a k-ary relation R, for each argument in R, we have n choices, since $n = ||\mathfrak{A}||$. Thus, for k arguments, we have n^k possible inputs — all of which either evaluate to true or false on R. Like we did for our binary edge relation E, we can then write down an n^k length bit string which encodes whether $R\overline{a}$ for some $\overline{a} \in |A|^k$. Furthermore, because we'll need to specify the arity in our encoding, we first write k in unary, followed by a separator symbol such as $\#$, and then our n^k bits encoding R. Thus, we can encode any k-ary relation in $O(n^k)$ space. $\square$

Exercise 2.2. Our quantified relations, X_q, need to have a fixed finite arity. They take a tuple of variables as input; this tuple represents an encoding of what step of the computation we are at, i.e., the timestamp. Our computation takes $O(n^k)$ time and, therefore, has $O(n^k)$ timestamps. Prove that we can encode all $O(n^k)$ timestamps as a constant length tuple which we can then pass into X_q, thus, proving that X_q has a fixed finite arity.

Proof. We know that our computation take time $O(n^k)$ where k is constant. Thus, we have $O(n^k) < n^{k+1}$ timestamps. For each timestamp, each individual argument to a predicate has n possibilities since $||\mathfrak{A}|| = n$. The idea is we'll have each argument to X_q be a digit of our timestamp encoded in base n. When we encode our n^{k+1} timestamps in the base n, each timestamp will always have a finite length of $\log_n(n^{k+1}) = k + 1$. Therefore, we can encode each timestamp as a tuple of length $k + 1$ and, thus, pass each timestamp into a relation with arity $k + 1$. $\square$

Exercise 2.3. Construct the END part of our first-order sentence φ_P.

Proof. As defined in Theorem 2.1, END ensures that we end up in an accepting state at the end of the computation. One way to achieve this is by forbidding the Turing machine M to end up in a rejecting state. Then since M is defined to be a non-deterministic Turing machine that decides whether $\mathfrak{A}$ has property P, when it does not end up in a rejecting state, we know that it will end up in an accepting state.

Let F_r be the set of rejecting states for M, where F_r is a finite set since the set of states in M is finite for a standard Turing machine. Since M only rejects if it transitions to some state $q \in F_r$,

Algorithm 1 HORN-SAT Search (V, C)

```
$S \leftarrow \varnothing$
**if** $\forall c \in C, \exists(\neg x) \in c$ for some $x \in V$ **then**
    **for** $x \in V$ **do**
        assign $x = 0$, append the assignment to $S$
    **end for**
    output $S$
**else**
    **for** $c \in C$ **do**
        **if** $c$ is an empty clause **then**
            output FALSE
        **else if** $c$'s only literal is a positive literal $p$ **then**
            assign $p = 1$, append the assignemnt to $S$
            **while** $\exists c \in C$ that contains $\neg p$ **do**
                remove $\neg p$ from $c$
            **end while**
        **end if**
    **end for**
    assign every unassigned variable $0$
    output $S$
**end if**
```

we may write the sentence for END as

$$\forall\, \bar{t} \left(\bigwedge_{q \in F_r} \neg X_r \bar{t} \right) \tag{4}$$

where $\bar{t}$ is a tuple of variables used as an abbreviation for the $k+1$ variables which we use to represent the timestamps (see Exercise 2.2). Observe that $\mathfrak{A} \models$ END iff M never enters a rejecting state, as desired. □

6.3 Section 3 Solutions

Exercise 3.1. Prove that HORN-SAT is P-complete.

Proof. First, we show that HORN-SAT is in P. We show this by constructing a polynomial time algorithm that searches for a satisfying assignment for a given Horn formula. The algorithm outputs the assignment if a satisfying assignment exists and returns false otherwise. The algorithm is given in Algorithm 1. V is the set of all variables in the given Horn formula, and C is the set of all clauses in the formula.

To understand the algorithm, consider the following. If every clause in C contains at least one negative literal, then assigning all variables False gives an obvious satisfying assignment for the formula. Else, by the definition of the Horn formula, if a clause contains no negative literal, it must contains exactly one positive literal p. To satisfy the clause, we assign it to be True, and subsequently can remove any $\neg p$ literal from all clauses without changing the satisfiability of the formula. Any empty clauses from the simplified formula results from having all negative literals being assigned False values and thus cannot be satisfied. The rest of the non-empty clauses can have at least one negative literal being assigned True and thus can be satisfied.

The algorithm at most needs to iterate through all clauses on each literal in each clause, and thus runs in polynomial time.

Next, we show that HORN-SAT is P-hard by reducing a P-complete problem the Circuit Value Problem (CVP) to it. Let $G = \{g_1, g_2, \dots, g_n\}$ be the set of gates in some Boolean circuit C. In particular, let g_n be the output gate. Denote the value of the gates by their output, then each g_i has value either 0 or 1.

For each gate g_i, we introduce two variables p_i, n_i to construct a Horn formula. We set $p_i = g_i$ and $n_i = \neg g_i$. We construct a Horn formula as the conjunction of the following clauses. For each gate g_i:

- If $g_i = 1$, include the following clauses: $(p_i) \wedge (\neg n_i)$
- If $g_i = 0$, include the following clauses: $(\neg p_i) \wedge (n_i)$
- If g_i is the $\wedge$-gate with inputs from g_j, g_k, include the following clauses: $(\neg p_i \vee p_j) \wedge (\neg p_i \vee p_k) \wedge (\neg p_j \vee \neg p_k \vee p_i)$
- If g_i is the $\vee$-gate with inputs from g_j, g_k, include the following clauses: $(\neg p_i \vee \neg n_j \vee \neg q_k) \wedge (\neg p_j \vee p_i) \wedge (\neg p_k \vee p_i)$
- If g_i is the $\neg$-gate with input from g_j, include the following clauses: $(\neg p_i \vee \neg p_j) \wedge (\neg n_j \vee p_i)$
- If $i = n$, include the clause (p_n)

First, observe that the clauses described above all have at most one positive literal. So the conjunction of them forms a Horn formula. By induction on i, one can show that for any satisfying assignment S of the Horn formula, $p_i \in S$ has the value of g_i and $n_i = \neg g_i$, and since the clause p_n is satisfied, $g_n = 1$, the output of the circuit is 1. On the other hand, also by induction on i, one can show that when C outputs 1, for each i, defining $p_i = g_i$ and $n_i = \neg g_i$ gives a satisfying assignment. Thus, the Horn formula is satisfiable iff the output of C is 1.

And translating each gate into some constant number of clauses can be done by some polynomial time reduction. Thus, HORN-SAT is P-hard.

Given that HORN-SAT is in P and P-hard, it is also P-complete. □

Exercise 3.2. Prove that ESO-Krom $\subseteq$ NL. Hint: Note that 2SAT is NL-complete.

Proof. This is similar to Lemma 3.3 and, therefore, we do not go into the details. The only difference is that we end up with a propsitional formula in 2CNF form (a.k.a. Krom form) and then use 2SAT to solve for satisfiability in NL. □

6.4 Section 4 Solutions

Exercise 4.1. Prove that ESO = ESO($<$).

Proof. To prove that ESO = ESO($<$), we show that the linear ordering relation $<$ can be defined as some existentially quantified predicate. By definition, a (strict) linear ordering $<$ on a set A must satisfy the following constraints:

(1) Not $a < a$ (irreflexivity)

(2) If $a < b$ then not $b < a$ (asymmetry)

(3) If $a < b$ and $b < c$ then $a < c$ (transitivity)

(4) If $a \neq b$ then $a < b$ or $b < a$ (connectivity)

where $a, b, c \in A$. Moreover, it is obvious that the asymmetry property follows naturally from transitivity and irreflexivity. Suppose for contradiction that $a < b$ and $b < a$ holds, then by transitivity, $a < b < a \Rightarrow a < a$, which contradicts irreflexivity. Thus, to define a linear ordering relation, we only need to test the three properties irreflexivity, transitivity, and connectivity. Let L be some relation, consider the following sentence:

$$(\forall x \neg L(x,x)) \wedge (\forall x \forall y \forall z (L(x,y) \wedge L(y,z) \rightarrow L(x,z)) \\ \wedge (\forall x \forall y ((x \neq y) \rightarrow (L(x,y) \vee L(y,x)))) \tag{5}$$

One can check for themselves that the three parts of the sentence tests the three properties respectively.

Then, for any ESO sentence ψ, we can define an existentially quantified predicate $\exists L$ as above and *restrict* the first-order part of the sentence to only care when L is a linear order. For example, let $\psi = \exists R(\varphi)$ is an ESO(<) sentence with first-order part φ; the following holds:

$$(\mathfrak{A}, <) \models \exists R(\varphi) \text{ iff } \mathfrak{A} \models \exists{<}\exists R((5) \wedge \varphi)$$

where "(5)" is replaced by equation (5) above in order to specify what "<" is. This preserves invariance because we have no idea exactly what ordering "<" will take on; (5) simply makes sure that "<" is *some* linear order.

Thus, ESO($<$) $\subseteq$ ESO. The other direction is obvious because adding a relation symbol does not reduce the expressivity of ESO. Thus, ESO = ESO($<$).

□

Exercise 4.2. Prove that EVEN($\mathfrak{A}$) can be decided by AC^0 circuits. More specifically, show that

1. EVEN $\in$ FO($<, +$).

2. FO($<, +$) $\subseteq$ FO($+, \times$).

Proof. 1. Let $\mathfrak{A}$ be some structure with domain A. In FO$(<,+)$, we have two additional relation symbols $<,+$, where $<$ acts as a linear ordering and $+$ defines the additive relation among elements in A. Thus, the following expression tests EVEN($\mathfrak{A}$) for $\mathfrak{A}$:

$$[\neg\exists x(x=x)] \vee \exists x \exists y[(x+x=y) \wedge \neg\exists z(y<z)] \tag{6}$$

The first part of the expression $\neg\exists x(x=x)$ tests whether A is empty. And the second part of the expression tests whether the largest element y in A can be expressed as $y=x+x$ for some element x. Note that by linear ordering, we implicitly introduce a bijection between A and the set of positive integers $[|A|]=\{1,\ldots,|A|\}$, e.g. if $A=\{a,b,c\}$, we can introduce $a \to 2, b \to 1, c \to 3$ s.t. $b<a<c$. So the last element in the ordering will be mapped to $|A|$, the cardinality of A. Then, $y=x+x$ requires the cardinality to be even.

Thus, the expression tests whether $|A|=0$ (mod 2), i.e. EVEN($\mathfrak{A}$).

2. To show that FO$(<,+) \subseteq$ FO$(+,\times)$, we show that FO$(<) \subseteq$ FO$(+)$, then subsequently, FO$(<,+) \subseteq$ FO$(+,\times)$. From the explanation in part (1) of this question, we see that when introducing the relation symbols $<,+$, we also implicitly introduce a bijective mapping between a set of integers and the domain A. Intuitively, we can define $<$ via $+$ because we know that $1<3$ can also be seen from $1+2=3$ where $2>0$. Thus, we can define linear order with $+$ as following:

$$x<y \iff \exists z(x+z=y) \tag{7}$$

Then FO$(<) \subseteq$ FO$(+)$, and subsequently, FO$(<,+) \subseteq$ FO$(+,\times)$. Since EVEN $\in$ FO$(<,+)$, it follows that EVEN $\in$ FO$(+,\times)$. □

6.5 Challenge Exercise Solution

Exercise 5.1. Prove that if P = ESO-Horn'(<), then P = NP.

Proof.
Remark 6.1. Recall that an ESO-Horn' sentence is of the form:

$$\exists R_1 \ldots \exists R_i \left(\forall \overline{x}\left(\bigwedge_j C_j\right)\right)$$

such that each C_j is either of the form $\alpha \vee \beta_1 \vee \cdots \vee \beta_m$ or $\beta_1 \vee \cdots \vee \beta_m$ where α is a positive literal and each β_k is a negative literal. In other words, the first-order part is in Horn form with respect to the relations.

ESO-Horn' differs from ESO-Horn because each α and β have slightly different constraints. In ESO-Horn, α can only contain quantified relations, our Rs, and β only has to be a negative literal if one of the quantified relations is used in it.

Assume that P = ESO-Horn'(<). Let $P \in$ NP be arbitrary. We will show that $P \in$ P.

Because P is in NP, by Fagin's theorem and the fact that ESO = ESO($<$) (see Exercise 4.1), we know that there exists an ESO($<$) sentence Φ_P such that

$$\text{input } \mathfrak{A} \text{ has property } P \text{ if and only if } (\mathfrak{A}, <) \models \Phi_P \tag{1}$$

for an arbitrary ordering relation $<$. Without loss of generality, let Φ_P be of the form

$$\exists R(\varphi_P)$$

where φ_P is an FO($<$) formula. Therefore, there exists an R such that $(\mathfrak{A}, <, R) \models \varphi_P$.

Remark 6.2. From our discussion in Section 4.4, we know that the FO-part of an ESO sentence essentially represents an encoding of a verifier for NP certificates while the existentially quantified predicates represent the certificate existing. We can then deduce as a corollary to Fagin's theorem that NP may be characterized as having poly-sized certificates verifiable by DLOGTIME-uniform AC^0 circuits. Our standard definition of NP as having poly-sized certificates verifiable in poly-time demonstrates, however, that we can "loosen" the first-order part — we don't *need* it to be so small.

Therefore, because

- φ_P is an FO($<$) sentence
- FO($<$) $\subseteq$ DLOGTIME-uniform AC^0 $\subseteq$ P,
- and P $=$ ESO-Horn'($<$) by assumption,

it follows that there exists an ESO-Horn'($<$) formula φ'_P such that

$$\begin{array}{lll} & (\mathfrak{A}, <, R) \models \varphi_P \text{ iff } (\mathfrak{A}, <, R) \models \varphi'_P & \\ \therefore & (\mathfrak{A}, <) \models \exists R\varphi_P \text{ iff } (\mathfrak{A}, <) \models \exists R\varphi'_P & \text{by definition} \\ \therefore & \mathfrak{A} \text{ has property } P \text{ iff } (\mathfrak{A}, <) \models \exists R\varphi'_P & \text{by (1)} \end{array}$$

We will now prove that $\exists R\varphi'_P$ is an ESO-Horn'($<$) sentence. Without loss of generality, say φ'_P is of the form

$$\exists R'(\varphi''_P)$$

where φ''_P is an FO sentence of the appropriate form specified in our definition of ESO-Horn'. Observe that $\varphi'_P = \exists R'(\varphi''_P)$ is an ESO-Horn'($<$) sentence which uses R so φ''_P will also use R. By definition, $\exists R \exists R'(\varphi''_P)$ will still be an ESO-Horn'($<$) sentence because the constraints on φ''_P do not differentiate between our the quantified relations (R') and the non-quantified ones (R) so we can "requantify" R while preserving that the sentence is an ESO-Horn'($<$) sentence. Therefore, $\exists R\varphi'_P$ is an ESO-Horn'($<$) sentence.

Remark 6.3. In the original definition of ESO-Horn($<$), "requantifying" R does not preserve that the sentence is an ESO-Horn($<$) sentence — which is why the proof of this exercise doesn't unconditionally prove that P $=$ NP.

Because $\exists R\varphi'_P$ is an ESO-Horn'($<$) sentence and that P $=$ ESO-Horn'($<$) by assumption, it follows that we can check whether $(\mathfrak{A}, <) \models \exists R\varphi'_P$ in deterministic polynomial-time. Because

$$\mathfrak{A} \text{ has property } P \text{ iff } (\mathfrak{A}, <) \models \exists R\varphi'_P$$

it follows that we can check if an input $\mathfrak{A}$ has property P in deterministic polynomial-time. Therefore, $P \in \mathrm{P}$, allowing us to conclude that NP $\subseteq$ P so P $=$ NP. $\square$

7 Additional Material

For a very comprehensive introduction to finite model theory for computer scientists, we highly suggest taking a look at Libkin's textbook *Elements of Finite Model Theory* [23]. Also, check out the textbook *Finite Model Theory and Its Applications* by Grädel et al. [13] — most notably Chapter 3 by Grädel. We also suggest taking a look at Immerman's textbook *Descriptive Complexity* [22] for a more detailed look on descriptive complexity theory, as opposed to finite model theory as a whole; although, we highly suggest taking a look at Libkin first because it is easier to comprehend and then referencing Immerman for more details. We also found [29] to be a very helpful survey paper when it came to understanding order invariance. For more information on connections to formal language theory and algebra, specifically semigroup theory, see Straubing's textbook *Finite Automata, Formal Logic, and Circuit Complexity* [31]. Papadimitriou's textbook on computational complexity [27] also contains a nice introduction to first- and second-order logic written for computer scientists as well as a proof of Fagin's theorem, which the readers may find to be another helpful explanation. For connections to database theory, see [1] — notably, Part E.

For a formal introduction to first-order logic and (non-finite) model theory see Enderton's textbook *A Mathematical Introduction to Logic* [10]. For a more advanced treatment of model theory and finite model theory aimed towards mathematicians, see [8] and [7] respectively.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[2] D. A. M. Barrington, K. Compton, H. Straubing, and D. Thérien. Regular languages in nc1. *Journal of Computer and System Sciences*, 44(3):478–499, 1992.

[3] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within nc1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[4] J. R. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6), 1960.

[5] S. R. Buss. The boolean formula value problem is in alogtime. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 123–131, 1987.

[6] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984.

[7] H.-D. Ebbinghaus and J. Flum. *Finite model theory*. Springer Science & Business Media, 1999.

[8] H.-D. Ebbinghaus, J. Flum, W. Thomas, and A. S. Ferebee. *Mathematical logic*, volume 1910. Springer, 1994.

[9] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961.

[10] H. B. Enderton. *A mathematical introduction to logic*. Elsevier, 2001.

[11] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.

[12] E. Grädel. Capturing complexity classes by fragments of second-order logic. *Theoretical Computer Science*, 101(1):35–57, 1992.

[13] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, S. Weinstein, et al. *Finite Model Theory and its applications*. Springer, 2007.

[14] E. Grandjean. Universal quantifiers and time complexity of random access machines. *Mathematical systems theory*, 18(1):171–187, 1985.

[15] Y. Gurevich. Logic and the challenge of computer science. Technical report, 1985.

[16] Y. Gurevich and H. R. Lewis. A logic for constant-depth circuits. *Information and Control*, 61(1):65–74, 1984.

[17] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20, 1986.

[18] A. Horn. On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[19] N. Immerman. Relational queries computable in polynomial time. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 147–152, 1982.

[20] N. Immerman. Languages which capture complexity classes. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 347–354, 1983.

[21] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1):86–104, 1986.

[22] N. Immerman. *Descriptive complexity*. Springer, 1999.

[23] L. Libkin. *Elements of finite model theory*, volume 41. Springer, 2004.

[24] A. Livchak. Languages for polynomial-time queries. *Computer-based modeling and optimization of heat-power and electrochemical objects*, page 41, 1982.

[25] R. McNaughton and S. A. Papert. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

[26] B. Molzan. Expressibility and nonuniform complexity classes. *SIAM Journal on Computing*, 19(3):411–423, 1990.

[27] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[28] O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), sep 2008.

[29] N. Schweikardt. A short tutorial on order-invariant first-order logic. In *International Computer Science Symposium in Russia*, pages 112–126. Springer, 2013.

[30] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 61–69, 1983.

[31] H. Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhäuser, 1994.

[32] B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Sibirskii Matematicheskii Zhurnal*, 3(1):103–131, 1962.

[33] B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *American Mathematical Society Translations*, 2(59):23–55, 1966.

[34] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.

[35] A. C.-C. Yao. Separating the polynomial-time hierarchy by oracles. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 1–10. IEEE, 1985.