Descriptive Complexity

Summary Notes

Noel Arteche

February 2, 2021

Contents

Introduction

These notes were written in the context of a reading project within the Master of Logic at the University of Amsterdam, supervised by Ronald de Haan, during the month of January of 2021.

Descriptive complexity, the field studying the logical tools needed to describe computation, started in the early 70s with Fagin's theorem, and has been responsible for some important results in complexity theory. These notes are based on my reading of Neil Immerman's 1999 textbook *Descriptive Complexity*. Based on the text, I collected what I found where the most relevant definitions, theorems and proofs, and I tried to explain them in my own words, complementing them with new examples that are my own or come from exercises suggested in the book.

Section 1 presents the descriptive framework, detailing how we describe computation using (first-order) logic and what complexity and reductions look like under this perspective; this corresponds roughly to chapters 1, 2 and 3 in Immerman's book. Section 2 studies what additional logical tools on top of first-order logic are needed to capture the class $\mathbf{P}$; this corresponds to chapter 4 of the book. Section 3 covers both the basics of Ehrenfeucht-Fraissé games as an approach to prove inexpressibility, as well as the proof that $\mathbf{FO} \neq \mathbf{P}$ via Håstad's switching lemma; this corresponds roughly to the first half of chapters 6 and 13 of Immerman's book. Finally, sections 4 and 5 look into the classes $\mathbf{NP}$ and $\mathbf{PSPACE}$ under the descriptive framework, where we require the power of second-order logic; this corresponds to chapters 7 and 10 of the book.

Noel Arteche
Amsterdam
February 2, 2021

1 The descriptive framework

Descriptive complexity tries to study the formal linguistic resources needed to describe computation. It asks what expressive power is needed to describe the relation between the input and the output of a computational problem.

As a first attempt, we will take first-order logic as the basic formal linguistic framework to measure descriptive complexity.

1.1 First-order logic

In our version of first-order logic, we will be working with *vocabularies* (sets of non-logical symbols), that will be interpreted under (finite) *structures*. Eventually, structures will encode the inputs to our problems, and computation will be modelled using *queries*.

Definition 1.1 (Vocabularies)**.** A *vocabulary* $\tau = C \cup P$ is a sets of non-logical symbols, where C is a set of *constant symbols* and P contains *relation symbols* or *predicates*[1]. If we need to refer more explicitly to the elements of C and R, we may write

$$\tau = \underbrace{\{c_1, \dots, c_s\}}_{C} \cup \underbrace{\{R_1, \dots, R_r\}}_{P}$$

where $R_1, \dots, R_r$ have all some particular *arity*.

Definition 1.2 (First-order languages)**.** Let τ be a vocabulary. A *term* on τ is either a constant symbol $c \in \tau$ or a *variable symbol* from the set $V = \{x, y, z, \dots\}$ of variables.

From terms we build *atomic formulas*: either a relation symbol $R \in \tau$ of arity n applied to n terms, or an equality of the form $t = t'$, where t and t' are terms.

From atomic formulas we can then build any other formulas using the constants $\top$ and $\bot$, the usual Boolean operators $\neg, \vee, \wedge, \oplus, \rightarrow, \leftrightarrow$ and quantifying variables with $\exists$ and $\forall$. For simplicity, we take $\neg, \wedge, \exists$ as base and consider the rest as abbreviations in the usual way.

We denote by $\mathcal{L}(\tau)$ the *first-order language* on τ, that is, the set of all well-formed formulas from symbols in τ. A formula with no free variables is a *sentence*.

[1] Unlike other texts, there are no function symbols here, which are "simulated" into relation symbols.

Definition 1.3 (Structures)**.** Let $\tau = \{c_1, \dots, c_s\} \cup \{R_1, \dots, R_r\}$ be a vocabulary. A *structure* $\mathcal{A}$ for τ is a tuple

$$\mathcal{A} = (A, c_1^{\mathcal{A}}, \dots, c_s^{\mathcal{A}}, R_1^{\mathcal{A}}, \dots, R_r^{\mathcal{A}})$$

consisting of a non-empty *domain* A and an *interpretation* for the symbols in τ. The interpretation gives, to every constant symbol $c \in \tau$, an element $a \in A$, so that $c^{\mathcal{A}} = a$, and for every relation symbol $R \in \tau$ of arity n, an n-ary relation $R^{\mathcal{A}} \subseteq A^n$.

The size of $\mathcal{A}$ is the size $|A|$ if its domain, which for us will always be finite. We often abuse the notation and write $|\mathcal{A}|$. Sometimes we also write $\mathrm{Dom}(\mathcal{A})$ to refer to the set A.

We denote by $\mathsf{STRUC}[\tau]$ the set of all *finite* structures for τ.

Definition 1.4 (Satisfaction)**.** Let τ be a vocabulary, $\varphi \in \mathcal{L}(\tau)$ be a formula and let $\mathcal{A} \in \mathsf{STRUC}[\tau]$.

A *variable assignment* is a function $s : V \to A$ assigning a value to every variable symbol. For $x \in V$, we call an *x-variant* the variable assignment s' such that $s'(v) = s(v)$ for every $v \in V$ except possibly for x.

For a term t, we write $t_s^{\mathcal{A}}$ to refer to the interpretation of t under $(\mathcal{A}, s)$. That is, if t is a variable $v \in V$, then $t_s^{\mathcal{A}} = v_s^{\mathcal{A}} = s(v)$, and if t is a constant symbol $c \in \tau$, then $t_s^{\mathcal{A}} = c_s^{\mathcal{A}} = c^{\mathcal{A}}$.

Depending on whether φ has free variables or not, we will write $\mathcal{A}, s \models \varphi$ or $\mathcal{A} \models \varphi$, and say that $(\mathcal{A}, s)$ or $\mathcal{A}$ *satisfies* or *models* φ according to the usual semantics for first-order logic:

$$\begin{aligned}
&\mathcal{A}, s \not\models \bot \\
&\mathcal{A}, s \models t = t' && \text{iff } t_s^{\mathcal{A}} = t_s'^{\mathcal{A}} \\
&\mathcal{A}, s \models R(t_1, \dots, t_n) && \text{iff } ({t_1}_s^{\mathcal{A}}, \dots, {t_n}_s^{\mathcal{A}}) \in R^{\mathcal{A}} \\
&\mathcal{A}, s \models \neg\varphi && \text{iff } \mathcal{A}, s \not\models \varphi \\
&\mathcal{A}, s \models \varphi \wedge \psi && \text{iff } \mathcal{A}, s \models \varphi \text{ and } \mathcal{A}, s \models \psi \\
&\mathcal{A}, s \models \exists x \psi && \text{iff there exists an } x\text{-variant } s' \text{ of } s \text{ such that } \mathcal{A}, s' \models \psi
\end{aligned}$$

Example 1.1 (Binary strings)*.* We are particularly interested in talking about binary strings. Perhaps intuitively, one could think that structures to work with strings will have as domain the set of all strings over a given alphabet (say, the set of all binary strings: $\{0,1\}^*$). However, this is not the case: each structure will encode a single binary string, in the same way that when working with graphs, a structure encodes a single graph.

The vocabulary is $\sigma_1 = \{\leq, S\}$, containing a binary relation symbol, $\leq$, and the unary relation symbol S. Then, given a binary string, the domain of the structure will be the set of indices of the string we are representing (for a string of length n, the set $\{0, 1, \dots, n-1\}$), and a unary predicate S, which will be true if and only if there is a 1 in that position of the string. Then, a binary string like 01101 can be encoded by the following structure $\mathcal{B} = (B, \leq^{\mathcal{B}}, S^{\mathcal{B}})$:

$$\mathcal{B} = (\underbrace{\{0,1,2,3,4\}}_{B}, \underbrace{\leq}_{\leq^{\mathcal{B}}}, \underbrace{\{1,2,4\}}_{S^{\mathcal{B}}})$$

Here, the domain B contains all n indices to refer to the bits in the string, $\leq$ is interpreted as the usual less-than-or-equal relation on the natural numbers, and $S^{\mathcal{B}}$ is encoding the string 01101.

If we go on an consider a vocabulary $\sigma_2 = \{\leq, A, B\}$ to encode two binary strings $a, b \in \{0,1\}^*$, then we can write a formula that expresses the binary addition of a and b bit by bit.

Because we are going to work with finite model model theory to model computation, our structures will have a finite domain. For a domain of size n, this can be set in bijection with the set $\{0, \dots, n-1\}$, and we will use these natural numbers to refer directly to the elements in the structure[2].

In particular, this means that on any finite structure we have a notion of order, which we will want to make explicit in our vocabularies. Like in σ_1, our vocabularies will often contain the symbol $\leq$. Whenever this is the case, we will only consider structures where $\leq$ is interpreted as a total order. Hence, when picking elements from $\mathsf{STRUC}[\tau]$, we will often insist that they are *ordered* structures, meaning that $\leq$ is interpreted in the standard way. Besides, whenever this is the case, we will also assume that we have symbols $0, 1, \max \in \tau$, and assume that these are interpreted as the first, second and maximum element in the structure.

Besides, from $\leq$, we can define a predicate $\mathsf{suc}(x, y)$, meaning that y is the successor of x. On top of that, we may assume to have the predicates $\mathsf{plus}(x, y, z)$ (meaning $x + y = z$) and $\mathsf{times}(x, y, z)$ (meaning $x \cdot y = z$), as well as $\mathsf{bit}(x, i)$ (meaning that the i-th bit of the binary representation of x is 1).

In what follows, we implicitly assume that our vocabularies include the *numeric* relations and constants $(\leq, \mathsf{plus}, \mathsf{times}, \mathsf{bit}, \mathsf{suc}, 0, 1, \max)$, while the *input* relations and constants will be made explicit.

[2]Sometimes we use indices from 1 to n instead of from 0 to $n-1$, and write $[n]$ to denote the set $\{1, \dots, n\}$.

As well as requiring that structures are ordered, we will require that they have at least two elements in the domain, so that 0 and 1 are interpreted as distinct. This allows us to talk about *Boolean variables*.

We are now ready to define what a *query* is.

Definition 1.5 (Queries)**.** Let τ and σ be two vocabularies. A *query* Q is a mapping

$$Q : \mathsf{STRUC}[\sigma] \to \mathsf{STRUC}[\tau]$$

that is polynomially bounded. That is, there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $\mathcal{A} \in \mathsf{STRUC}[\sigma]$, $|Q(\mathcal{A})| \leq p(|\mathcal{A}|)$.

If Q is of the form $Q : \mathsf{STRUC}[\sigma] \to \{0, 1\}$, then we call it a *Boolean query*. We will often see Boolean queries as subsets of the set of all structures, that is, $Q \subseteq \mathsf{STRUC}[\sigma]$.

Example 1.2 (Queries on binary strings). Consider the vocabulary σ_1 of one string, as before. Then, the mapping $P : \mathsf{STRUC}[\sigma_1] \to \{0, 1\}$ that returns, for every $\mathcal{A} \in \mathsf{STRUC}[\sigma_1]$, $P(\mathcal{A}) = 1$ if and only if the bit-string encoded in $\mathcal{A}$ represents an even natural number, is a Boolean query.

Furthermore, if we also consider the vocabulary σ_2 encoding two bit-strings, then the mapping

$$S : \mathsf{STRUC}[\sigma_2] \to \mathsf{STRUC}[\sigma_1]$$

that assigns, to every structure $\mathcal{A} \in \mathsf{STRUC}[\tau_{S_2}]$ encoding two bit-strings $a, b \in \{0, 1\}^*$, a new structure $S(\mathcal{A})$ encoding the binary addition of a and b, is also a query.

We are particularly interested in queries that we can describe using first-order formulas. These are *k-ary first-order queries*: a query Q with a fixed $k \in \mathbb{N}$ that maps, to every structure $\mathcal{A}$, a structure whose domain is a first-order definable subset of A^k; whose constants are all first-order definable elements of A^k; and whose n-ary relations are first-order definable subsets of $(A^k)^n$.

Definition 1.6 (First-order query)**.** Let σ and $\tau = \{c_1, \dots, c_s, R_1, \dots, R_n\}$ be vocabularies, and let $k \in \mathbb{N}$. A *k-ary first-order query* is a query

$$Q : \mathsf{STRUC}[\sigma] \to \mathsf{STRUC}[\tau]$$

determined by first-order formulas $\varphi_0, \varphi_1, \dots, \varphi_r, \psi_1, \dots, \psi_s \in \mathcal{L}(\sigma)$ such that for every $\mathcal{A} \in \mathsf{STRUC}[\sigma]$, we have that its domain is

$$\mathrm{Dom}(Q(\mathcal{A})) = \{(a_1, \dots, a_k) \in A^k \mid \mathcal{A} \models \varphi_0(a_1, \dots, a_k)\},$$

for every constant $c_i \in \tau$,

$$c_i^{Q(\mathcal{A})} = (a_1, \dots, a_k) \in \mathrm{Dom}(Q(\mathcal{A})) \text{ (unique) such that } \mathcal{A} \models \psi_i(a_1, \dots, a_k),$$

and for every relation $R_i \in \tau$ of arity n,

$$R_i^{Q(\mathcal{A})} = \{((a_{1,1}, \dots, a_{1,k}), \dots, (a_{n,1}, \dots, a_{n,k})) \in A^{k \cdot n} \mid \mathcal{A} \models \varphi_i(a_{1,1}, \dots, a_{n,k})\}$$

When Q is a k-ary first-order query, we can write it using a notation reminiscent of the Lambda calculus as

$$Q = \lambda_{x_1,\dots,x_l}(\varphi_0, \varphi_1, \dots, \varphi_r, \psi_1, \dots, \psi_s)$$

where $x_1, \dots, x_l$ are the free variables in those formulas.

A first-order query is either *Boolean* or k-ary, for some $k \in \mathbb{N}$. We denote by $\mathbf{FO}$ the set of all first-order Boolean queries, and by $\mathbf{Q}(\mathbf{FO})$ the set of all first-order queries.

Example 1.3 (Database queries). If we were working with some database $\mathcal{D}$ of people and could define formulas $\varphi_{\mathsf{Sibling}}(x, y)$ and $\varphi_{\mathsf{Aunt}}(x, y)$, determining that x is a sibling of y and x is the aunt of y, respectively, then

$$Q_{\text{Sibling-Aunt}} = \lambda_{x,y}(\top, \varphi_{\mathsf{Sibling}}, \varphi_{\mathsf{Aunt}})$$

is a unary query, returning a structure $Q(\mathcal{D}) = (D, \mathsf{Sibling}, \mathsf{Aunt})$, where Sibling and Aunt are binary predicates. In this case, $\varphi_0 = \top$, as we do not want to filter the domain.

Example 1.4 (Strong graph product). As an example of a binary query, consider two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. We can encode them in single structure $\mathcal{G} = (V_1 \cup V_2, V_1, V_2, E_1, E_2)$. Now, we could write a query to get the strong product of the two graphs. Recall that the strong product of two graphs is $G_1 \boxtimes G_2 = (V_1 \times V_2, E_1 \boxtimes E_2)$, where the edges in $E_1 \boxtimes E_2$ are the pairs $((x, y), (x', y')) \in (V_1 \times V_2)^2$ satisfying the following formula, $\varphi_\boxtimes$:

$$\varphi_\boxtimes(x, y, x', y') : (x \neq x' \vee y \neq y') \wedge (x = x' \vee E_1(x, x')) \wedge (y = y' \vee E_2(y, y'))$$

Besides, we want the domain of the output structure to be $V_1 \times V_2$, not $(V_1 \cup V_2)^2$. Hence, we use the following formula φ_0 to filter the inputs:

$$\varphi_0(x, y) = V_1(x) \wedge V_2(y)$$

We can now take the binary query

$$Q_{\boxtimes} = \lambda_{x,y,x',y'}(\varphi_0, \varphi_{\boxtimes})$$

This query will give, for every structure $\mathcal{G} = (V_1 \cup V_2, V_1, V_2, E_1, E_2)$ encoding two graphs, a structure $\mathcal{S} = (V_1 \times V_2, E_1 \boxtimes E_2)$ encoding the strong graph product. Note that it is a *binary* query because the domain of $\mathcal{S}$ is $V_1 \times V_2$, a first-order definable subset of $(V_1 \cup V_2)^2$.

Finally, note that for every first-order query

$$\begin{aligned} Q : \mathsf{STRUC}[\sigma] &\to \mathsf{STRUC}[\tau] \\ \mathcal{A} &\mapsto \lambda_{x_1,\dots,x_l}(\varphi_0, \varphi_1, \dots, \varphi_r, \psi_1, \dots, \psi_s) \end{aligned}$$

the formulas $\varphi_0, \varphi_1, \dots, \varphi_r, \psi_1, \dots, \psi_s$ must be written in $\mathcal{L}(\sigma)$. But this means that everything in the universe of τ can be rephrased in the language of σ! That is, Q has all the information needed to build a dual map

$$\hat{Q} : \mathcal{L}(\tau) \to \mathcal{L}(\sigma)$$

We skip the technical details of how to build $\hat{Q}$, but we will use it when necessary along with the following proposition.

Proposition 1.1. *Let $Q : \mathsf{STRUC}[\sigma] \to \mathsf{STRUC}[\tau]$ be a k-ary first-order query. Then, for every sentence $\varphi \in \mathcal{L}(\tau)$ and every $\mathcal{A} \in \mathsf{STRUC}[\sigma]$,*

$$\mathcal{A} \models \hat{Q}(\varphi) \Leftrightarrow Q(\mathcal{A}) \models \varphi$$

Proof. By structural induction on φ. □

1.2 Complexity theory, revisited

It is clear that the notion of queries presented in Definition 1.5 does not yet shed any light on computation. Definition 1.6 was a bit more precise, in arguing that we do not want to talk about just any query but about those that we can describe using first-order formulas. Yet, that is also unrelated to computation: as far as we know and with the tools presented so far, there might well be first-order queries, which could be described using first-order formulas, which might yet be impossible to actually compute[3].

[3]This is not the case, but we do not know at this point. This is the content of Theorem 1.2.

We now relate the idea of queries to the idea of *computable* queries. Perhaps underwhelmingly, we do so by converting a query

$$Q : \mathsf{STRUC}[\sigma] \to \mathsf{STRUC}[\tau]$$

into a classic binary function

$$f_Q : \{0,1\}^* \to \{0,1\}^*$$

and require that the latter is computable by some Turing machine.

First, we specify how to convert a structure into a binary string.

Definition 1.7 (Binary encoding of a structure)**.** Let τ be a vocabulary. We want to define a function

$$\mathsf{bin} : \mathsf{STRUC}[\tau] \to \mathsf{STRUC}[\sigma_1]$$

where σ_1 is the vocabulary used to encode a single string.

Let $\mathcal{A} \in \mathsf{STRUC}[\tau]$, and let $n = |\mathcal{A}|$. Every constant symbol $c \in \tau$ is interpreted in $\mathcal{A}$ by some natural number in $\{0, \dots, n-1\}$, so we can represent $c^{\mathcal{A}}$ simply as the binary representation of that natural number. Analogously, for every relation symbol $R \in \tau$ or arity α, we want to encode in binary the set $R^{\mathcal{A}} \subseteq A^\alpha$. Then take the Cartesian product A^α and fix some order (say, the lexicographic order). There are n^α tuples, so we write down a bit-string of length n^α, where a 1 in the i-th position means that the i-th tuple in A^α is in $R^{\mathcal{A}}$.

The *binary representation of* $\mathcal{A}$, $\mathsf{bin}(\mathcal{A})$, is simply the concatenation of the binary representation of all the constant symbols and all the relation symbols.

Note that if a vocabulary τ has s constant symbols and r relation symbols of arities $\alpha_1, \dots, \alpha_r$, then, given a structure $\mathcal{A} \in \mathsf{STRUC}[\tau]$ with $|\mathcal{A}| = n$, we have that

$$|\,\mathsf{bin}(\mathcal{A})| = \sum_{i=1}^{r} n^{\alpha_i} + s\lceil \log n \rceil$$

This is a polynomial in n. For example, if we take a structure $\mathcal{G} = (V, E)$ to represent a graph with $|V| = n$ vertices, then the size of the binary representation of $\mathcal{G}$ is $|\,\mathsf{bin}(\mathcal{G})| = n^2$, which is exactly the size of the adjacency matrix of the graph.

Definition 1.8 (Computable queries)**.** Let $Q : \mathsf{STRUC}[\sigma] \to \mathsf{STRUC}[\tau]$ be a query. We say that Q is a *computable query* if the function

$$\begin{aligned} f_Q : \{0,1\}^* &\to \{0,1\}^* \\ \mathsf{bin}(\mathcal{A}) &\mapsto \mathsf{bin}(Q(\mathcal{A})) \end{aligned}$$

is computable (that is, computable by a Turing machine).

Arguably, the descriptive approach to computation does not fundamentally change anything so far: in the most basic level, it relegates to Turing machines the task of defining computation. However, it does seem to have a benefit; namely, that we are more aware of the details of the problem instances, and pay a bit more attention to how these are encoded.

Now, how do we measure the complexity of a given query? Since each query can be converted into a binary function, and we have well-known complexity measures for these based on the resources available for Turing machines, we will again define complexity in relation to machine complexity.

Definition 1.9 (Complexity of a query)**.** Let Q be a query, and let $\mathbf{C}$ be a complexity class. If Q is a Boolean query, then we straightforwardly say that Q is *computable in* $\mathbf{C}$ if the decision problem encoded by Q is in $\mathbf{C}$. On the other hand, if Q is not a Boolean query, we say that Q is *computable in* $\mathbf{C}$ if computing the Boolean query Q_B is in $\mathbf{C}$, where

$$Q_B = \{(\mathcal{A}, i, b) \mid i\text{-th bit of } \mathsf{bin}(Q(\mathcal{A})) \text{ is } b\}$$

We denote by $\mathbf{Q}(\mathbf{C})$ the class of all queries computable in $\mathbf{C}$:

$$\mathbf{Q}(\mathbf{C}) = \mathbf{C} \cup \{Q \mid Q_B \in \mathbf{C}\}$$

Remark 1.1. Essentially, query Q is in class $\mathbf{C}$ if the problem of computing any bit of Q's output is in $\mathbf{C}$. Recall that when defining the notion of queries in Definition 1.5 we insisted that the size of the query's output should be bounded by a polynomial in the size of the input structure. Hence, if a non-Boolean query Q is in some class $\mathbf{C}$, we can compute the entire output of Q by making polynomially many calls to the algorithm that decides Q_B.

In what follows, we assume that classes $\mathbf{L}$, $\mathbf{NL}$, $\mathbf{P}$, $\mathbf{NP}$, $\mathbf{coNP}$, $\mathbf{PH}$, $\mathbf{PSPACE}$, $\mathbf{EXP}$, etc. are well-known and, in general, we assume that their definitions are originally machine-based.

In particular, we can already show how the computational power of first-order descriptions relates to basic classic complexity measures. Recall that

in Definition 1.6 we presented the class **FO**: the set of all first-order Boolean queries; it turns out, its computational power seems to be quite weak, as suggested by the following result.

Theorem 1.2. **FO** $\subseteq$ **L**.

Proof. We need to show that every first-order Boolean query $Q \subseteq \mathsf{STRUC}[\tau]$ can can be computed in deterministic logspace. Since Q is first-order and Boolean, Q is just the set of finite structures satisfying some sentence $\varphi \in \mathcal{L}(\tau)$. It is easy to check that no matter what φ actually is, it can be rewritten in prenex form as

$$\varphi : Q_1 x_1 Q_2 x_2 \dots Q_k x_k \alpha(x_1, \dots, x_k)$$

for some $k \in \mathbb{N}^*$, where every $Q_i \in \{\exists, \forall\}$.

Now we must show that there exists a logspace Turing machine M such that for every $\mathcal{A} \in \mathsf{STRUC}[\tau]$,

$$\mathcal{A} \in Q \Leftrightarrow M(\mathsf{bin}(\mathcal{A})) = 1$$

We build such an M by induction on k.

- If $k = 0$, then $\varphi = \alpha$, where α has no free variables and no quantifiers inside. We can show, by a simple but tedious induction on α's structure, that there is a logspace machine that computes whether $\mathcal{A} \models \alpha$. We omit this structural induction.

- If all first-order Boolean queries with $k - 1$ quantifiers are logspace computable, then for a formula with k quantifiers take

  $$\psi(x_1) : Q_2 x_2 \dots Q_k x_k \alpha(x_1, \dots, x_k)$$

  By induction hypothesis there exists a machine M that works for ψ as long as there are no free variables, so we loop over all possible n values that x_1 can take. On each of them, replace x_1 for that value, run M and see if it accepts. If $Q_1 = \exists$, then it should accept at least once; if $Q_1 = \forall$ then it should accept for every value. Since M runs in logspace and for every new value of x_1 the space can be reused, although we will make at most n calls to M, we will only use $O(\log n)$ space. This is a logspace computation.

This concludes the induction. Hence, for every first-order Boolean query $Q \in$ **FO**, we have $Q \in$ **L**. □

It seems like first-order Boolean queries on themselves are not very powerful for computation. However, first-order queries can be powerful enough to specify interesting reductions between problems.

1.3 First-order reductions

The last notion that we should specify in the descriptive framework is that of *reductions*. In classical complexity theory, we often talk about *polynomial-time reductions*: problem $A \subseteq \{0,1\}^*$ is polynomial-time reducible to problem $B \subseteq \{0,1\}^*$, written $A \leq_p B$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, it holds that $x \in A$ if and only if $f(x) \in B$. Similarly, depending on the properties of f, we may be talking about *logspace-reductions* ($\leq_{\log}$), etc.

Under the descriptive approach, a function like f will be a query, converting a structure encoding an instance of problem A into a structure encoding an instance of problem B. In particular, we will be interested in that these queries are first-order.

Definition 1.10 (Reductions)**.** Let $\mathbf{C}$ be a complexity class, and let $A \subseteq \mathsf{STRUC}[\sigma]$, $B \subseteq \mathsf{STRUC}[\tau]$ be Boolean queries. If the query $Q : \mathsf{STRUC}[\sigma] \to \mathsf{STRUC}[\tau]$ is an element of $\mathbf{Q}(\mathbf{C})$ such that for every $\mathcal{A} \in \mathsf{STRUC}[\sigma]$,

$$\mathcal{A} \in A \Leftrightarrow Q(\mathcal{A}) \in B$$

then Q is a $\mathbf{C}$-*many-one reduction* from A to B, written $A \leq_{\mathbf{C}} B$. If Q is a first-order query, then it is a *first-order reduction* and we write $\leq_{\text{fo}}$; if $Q \in \mathbf{Q}(\mathbf{L})$, then Q is a *logspace* reduction, and we write $\leq_{\log}$; if $Q \in \mathbf{Q}(\mathbf{P})$, then Q is the usual polynomial-time reduction ($\leq_p$).

Example 1.5 ($\mathsf{SAT} \leq_{\text{fo}} \mathsf{CLIQUE}$). Let us see how we can convert instances of SAT into instances of CLIQUE, in such a way that we can describe the process in first-order logic. That is, we want to show $\mathsf{SAT} \leq_{\text{fo}} \mathsf{CLIQUE}$. We will use the usual reduction, but instead of arguing that it is polynomial-time computable, we will show that it is a first-order query.

First we need to specify the vocabularies for these problems. For SAT, take some Boolean formula ϕ and modify it as necessary so that it is written in CNF with n variables and n clauses. The structure Φ encoding formula ϕ is

$$\Phi = (\{1, \dots, n\}, P, N)$$

where P is a binary predicate of *positive* occurrences: $(c, v) \in P$ if and only if variable v occurs as a positive literal in clause c. Predicate N is analogous but for negative occurrences. For CLIQUE we use the vocabulary $\gamma = \{E\}$ of graphs.

We now want to construct a graph $\mathcal{G}$ along with some number k such that the formula encoded in Φ is satisfiable if and only if the graph encoded

by $\mathcal{G}$ has a clique of size k. If $C = \{c_1, \dots, c_n\}$ contains the n clauses of ϕ and $L = \{v_1, \dots, v_n, \neg v_1, \dots, \neg v_n\}$ has all possible literals, then $\mathcal{G} = (V, E, k)$, such that

$$\begin{aligned} V =& (C \times L) \cup \{w\} \\ E =& \{((c_1, l_1), (c_1, l_2)) \mid c_1 \neq c_2 \text{ and } \neg l_1 \neq l_2\} \cup \\ & \{(w, (c, l)), ((c, l), w) \mid l \text{ occurs in } c\} \\ k =& n + 1 \end{aligned}$$

That is: we have a graph with a vertex w plus vertices $(c, l) \in C \times L$ for every pair of clause and literal. There is and edge between vertices (c_1, l_1) and (c_2, l_2) if and only if points come from different clauses and the literals are not the negation of each other. Besides, point w is connected to every pair (c, l) such that l does occur in c.

Now note that if $\mathcal{G}$ has a clique of size $n + 1$, then it must contain w and one point for every clause. Then, by making true those literals, we get a satisfying assignment. On the other hand, if Φ has a satisfying assignment, then this determines a $n + 1$-clique for $\mathcal{G}$: it will consist of one point per clause that is satisfied, in addition to w.

Hence, the query $Q : \Phi \mapsto \mathcal{G}$ is a many-one reduction, as

$$\Phi \in \mathsf{SAT} \Leftrightarrow Q(\Phi) = \mathcal{G} \in \mathsf{CLIQUE}$$

Clearly, the query Q is polynomial-time computable, hence the usual argument concludes that $\mathsf{SAT} \leq_p \mathsf{CLIQUE}$. We are interested in showing that Q is first-order.

This is the case, as we can encode it in the following ternary first-order query:

$$Q = \lambda_{x_1, x_2, x_3, y_1, y_2, y_3}(\varphi_0, \varphi_1, \psi_1)$$

The idea is that a vertex is encoded with a tuple (x_1, x_2, x_3), such that x_1 corresponds to the clause, x_2 corresponds to the variable and $x_3 \in \{1, 2\}$ depending on whether the literal is positive or negative. Point w will be $(1, 1, 3)$. Hence,

$$\varphi_0(x_1, x_2, x_3) : (x_3 \leq 2) \vee (x_1 = 1 \wedge x_2 = 1 \wedge x_3 = 3)$$

For the edge relation, we simply transcribe the definition of E into a formula φ_2, built from subformulas:

$$\varphi_1' : \alpha_1 \vee (\alpha_2 \wedge P(y_1, y_2)) \vee (\alpha_3 \wedge N(y_1, y_2))$$

$$\alpha_1 : x_1 \neq y_1 \wedge x_3 < 3 \wedge y_3 < 3 \wedge (x_2 = y_2 \rightarrow x_3 = y_3)$$
$$\alpha_2 : x_3 = 3 \wedge y_3 = 1$$
$$\alpha_3 : x_3 = 3 \wedge y_3 = 2$$

Then, φ_1 is the symmetric closure of φ_1':

$$\varphi_1(x_1, x_2, x_3, y_1, y_2, y_3) : \varphi_1'(x_1, x_2, x_3, y_1, y_2, y_3) \vee \varphi_1'(y_1, y_2, y_3, x_1, x_2, x_3)$$

Finally,

$$\psi_1(x_1, x_2, x_3) : x_1 = 1 \wedge x_2 = 2 \wedge x_3 = 2$$

Hence, Q is a ternary first-order query. It follows that $\mathsf{SAT} \leq_{\text{fo}} \mathsf{CLIQUE}$. □

Of course, we would like first-order reductions to work in the usual way. That is, we want first-order reductions to be closed for some classes of queries.

Definition 1.11 (Closure under first-order reductions)**.** Let $\mathbf{C}$ be a set of Boolean queries. We say that $\mathbf{C}$ is *closed under first-order reductions* if for every pair of Boolean queries A and B, if $A \leq_{\text{fo}} B$ and $B \in \mathbf{C}$, then $A \in \mathbf{C}$.

Is it the case that first-order reductions are closed for interesting classes of queries? The following proposition, which follows easily from the fact that $\mathbf{FO} \subseteq \mathbf{L}$ (Theorem 1.2), shows that this is often the case.

Proposition 1.3. *Let $\mathbf{C}$ be a set of Boolean queries. If $\mathbf{C}$ is closed under logspace reductions, then $\mathbf{C}$ is closed under first-order reductions.*

In fact, most if not all of the classes we will encounter are closed under first-order reductions. To show this for some class $\mathbf{C}$, a first attempt will be to check that the class is closed under logspace reductions, and then invoke the previous proposition.

However, there exists a different possible scenario, that of *languages*. After all, part of our task is to relate classic complexity classes to logical languages, by showing that they define the same sets of problems. We say that a language $\mathcal{L}$ (such as first-order logic) is *closed under first-order reductions* if the set of Boolean queries definable in $\mathcal{L}$ is closed under first-order reductions.

To show whether a language is closed under first-order reductions, note that if we have queries A and B, $A \leq_{\text{fo}} B$ by reduction Q and B is expressible in $\mathcal{L}$ as formula φ_B, we have that for every structure $\mathcal{A}$,

$$\mathcal{A} \in A \Leftrightarrow Q(\mathcal{A}) \in B \Leftrightarrow Q(\mathcal{A}) \models \varphi_B \Leftrightarrow \mathcal{A} \models \hat{Q}(\varphi_B)$$

where the first implication is justified by definition of reductions, the second one by the definition of Boolean queries, and the third one by Proposition 1.1 (dual of queries). Hence, it suffices to check whether $\hat{Q}(\varphi_B) \in \mathcal{L}$. If so, then A is expressible in $\mathcal{L}$ and hence $\mathcal{L}$ is closed under first-order reductions.

A question remains open: are the usual complete problems for basic classes like **L**, **NL** or **P** also complete under first-order reductions? The answer is affirmative, as we now show.

Definition 1.12 (REACH and DREACH)**.** Let REACH be the reachability problem on graphs:

$$\mathsf{REACH} = \{(G, s, t) \mid \text{there is a path in graph } G \text{ from } s \text{ to } t\}$$

Besides, let DREACH be the deterministic version of REACH: the restriction of REACH to graphs with a unique outgoing edge for every vertex.

Theorem 1.4. *The problem* REACH *is* **NL***-complete via first-order reductions.*

Proof. Clearly $\mathsf{REACH} \in \mathbf{NL}$, as in a graph with n vertices, any path from s to t has at most length n. Hence, a nondeterministic walk starting at s can be computed in space $O(\log n)$, simply be keeping the current vertex, which is a number less o equal than n, hence representable in a bit-string of length $\log n$.

For the reduction, we need to show that for every $L \in \mathbf{NL}$, $L \leq_{\text{fo}} \mathsf{REACH}$. Let $L \in \mathbf{NL}$, $L \subseteq \mathsf{STRUC}[\sigma]$. There exists a nondeterministic logspace machine M, which uses at most $c \log n$ cells, and decides L. The idea for the reduction $Q : \mathsf{STRUC}[\sigma] \to \mathsf{STRUC}[\gamma]$ (where γ is the language of graphs) is to give, for every $\mathcal{A} \in \mathsf{STRUC}[\sigma]$, the configuration graph of M along with the start and accepting states. Hence, it will hold that

$$\mathcal{A} \in L \Leftrightarrow Q(\mathcal{A}) \in \mathsf{REACH}$$

It then suffices to show that reduction Q is first-order. The key idea is that a given configuration of M can be encoded as a k-ary tuple

$$(p, r_1, \dots, r_\alpha, w_1, \dots, w_c) \in [n]^k$$

where α is the maximum arity over the relation symbols in σ, and $k = \alpha + c + 1$. For large enough n, $p \in [n]$ encodes the machine's state and positions of the input and working tapes' heads; variables $r_1, \dots, r_\alpha$ encode a tuple in A^α, such that the machine is reading relation R in the input tape and reading a 1 if and only if $\mathcal{A} \models R(r_1, \dots, r_\alpha)$; variables $w_1, \dots, w_c$ encode the

contents of the work tape, as there are always at most $c \log n$ cells in use, and every block of $\log n$ bits encodes a natural number in $[n]$.

We can now write formula $\varphi_M(c_1, c_2)$, meaning that it is possible to go to configuration c_2 from configuration c_1, and formulas $\mathrm{Start}(c)$ and $\mathrm{Accept}(c)$, indicating that c is a start or accepting state, respectively. We skip the details of these formula encodings.

Finally, we have that the reduction is the first-order k-ary query

$$Q = \lambda_{c_1,c_2}(\top, \varphi_M, \mathrm{Start}, \mathrm{Accept})$$

We conclude that REACH is **NL**-complete via first-order reductions. □

Theorem 1.5. DREACH *is* **L***-complete via first-order reductions.*

Proof. The argument is analogous to the previous theorem. □

Another variant of the reachability problem exists is complete for **P**.

Definition 1.13 (Alternating graphs)**.** A graph $G = (V, E, A)$ is *alternating* when there is a subset $A \subseteq V$ of vertices labelled as *universal*. The reachability relation on alternating graphs is such that it is reflexive and:

- For every $v \notin A$, v reaches $w \in V$ if and only if there is an edge from v to some z such that z reaches w.
- For every $v \in A$, v reaches $w \in W$ if and only if for every outgoing edge (v, z), z reaches w.

We denote by AREACH the reachability problem on alternating graphs.

Theorem 1.6. AREACH *is* **P***-complete under first-order reductions.*

Proof. Omitted; similar to the previous two theorems. □

2 P under the descriptive framework

Since $\mathbf{FO} \subseteq \mathbf{L}$, we expect first-order languages to be too weak to express most interesting problems. In particular, it seems like **FO** cannot express "easy" problems like the ones in **P**.

For example, try to express something like the problem AREACH, presented in Definition 1.13, which is **P**-complete under first-order reductions. A first-order query expressing AREACH would need to describe the reachability relation R on alternating graphs, and this would look something like this:

$$R(x,y) : x = y \vee (\exists z(E(x,z) \wedge R(z,y)) \\ \wedge A(x) \rightarrow \forall z(E(x,z) \rightarrow R(z,y)))$$

This formula is recursive, as R appears again inside its definition, but this is not allowed! This invites us to increase the power of our logic tools. Indeed, we would like to extend first-order logic with the ability to write *inductive definitions*.

2.1 Inductive definitions

Inductive definitions are formalized using the *Least Fixed-Point* (LFP) operator. We will write R from the previous example as the following formula:

$$\varphi(P,x,y) : x = y \vee (\exists z(E(x,z) \wedge P(z,y)) \\ \wedge A(x) \rightarrow \forall z(E(x,z) \rightarrow P(z,y)))$$

where P is the predicate that we want to find, such that it "solves" the equation

$$\varphi(P,x,y) = P(x,y)$$

which will give us our desired reachability formula, R. In other words, P will be a fixed-point of φ and we want, in particular, the smallest such relation: the *least* fixed-point.

Under a given structure $\mathcal{A}$, we want to find, amongst all candidate interpretations for P, the one that is the least fixed-point. We will start by using as candidate interpretation for the predicate P the empty set. This will give us the points that satisfy the base case of the inductive definition. We denote this set by $\varphi_{\mathcal{A}}(\emptyset)$. We will then use as candidate interpretation this new set. We denote by $\varphi^r_{\mathcal{A}}(\emptyset)$ the set of tuples of $\mathcal{A}$ obtained after iterating r times starting at $\emptyset$.

Intuitively, if $|\mathcal{A}| = n$, then for our reachability relation R, iterating more that n^2 times will not make our relation bigger, as there are no further tuples to add. Besides, obtaining the least fixed-point by interation ensures that we get the smallest one, as desired. This is formalized by the following result, which needs to assume that the formula φ is *monotone*: for every two candidate interpretations S and T,

$$S \subseteq T \Rightarrow \varphi(S) \subseteq \varphi(T)$$

This is an mild requirement, as if a formula is *positive* in the new relation symbol (i.e. it appears under an even number of negations), then it is also monotone.

Theorem 2.1 (Finite Knaster-Tarski)**.** *Let $\varphi(R, x_1, \dots, x_k)$ be a monotone first-order formula, depending on a new relation symbol R of arity k. For any finite structure $\mathcal{A}$, the least fixed-point of R exists. It is equal to $\varphi^r_{\mathcal{A}}(\emptyset)$, where r is minimal, such that $r \leq |\mathcal{A}|^k$, and for every $r < r' \leq |\mathcal{A}|^k$, $\varphi^r_{\mathcal{A}}(\emptyset) = \varphi^{r'}_{\mathcal{A}}(\emptyset)$.*

Proof. We first show that the fixed-point exists, and then show that the one we obtain is the *least* one.

Because φ is monotone, that means the following inclusions hold:

$$\emptyset \subseteq \varphi^1_{\mathcal{A}}(\emptyset) \subseteq \varphi^2_{\mathcal{A}}(\emptyset) \subseteq \varphi^3_{\mathcal{A}}(\emptyset) \subseteq \dots$$

Some of these inclusions will be strict, but not forever. If they were, then there would always be new tuples to add to the relation, but this is impossible, as there are only $|\mathcal{A}|^k$ possible tuples to add and $\mathcal{A}$ is finite. Hence, for some $r \leq n^k$, $\varphi^r_{\mathcal{A}}(\emptyset) = \varphi^{r+1}_{\mathcal{A}}(\emptyset)$, that is, $\varphi_{\mathcal{A}}(\varphi^r_{\mathcal{A}}(\emptyset)) = \varphi^r_{\mathcal{A}}(\varphi)$, so we have a fixed-point of φ. We have shown that there exists an $r \leq |\mathcal{A}|^k$ such that $\varphi^r_{\mathcal{A}}(\varphi)$ is a fixed-point. In other words, fixed-points exist and can be obtained through iteration.

Do we obtain the least one in this way? Yes. Simply note that for any other fixed-point S, for every i, $\varphi^i_{\mathcal{A}}(\emptyset) \subseteq S$. We show it by induction on i. For $i = 0$, $\varphi^0_{\mathcal{A}}(\varphi) = \emptyset \subseteq S$. For the inductive case, if the inclusion holds up to i, then for $i + 1$ we have

$$\varphi^{i+1}_{\mathcal{A}}(\emptyset) = \varphi_{\mathcal{A}}(\varphi^i_{\mathcal{A}}(\emptyset)) \subseteq \varphi_{\mathcal{A}}(S) = S$$

where the last inclusion holds because φ is monotone and by induction hypothesis $\varphi^i_{\mathcal{A}}(\emptyset) \subseteq S$, and the equality holds because S is a fixed-point.

Therefore, via iteration, we can obtain the least fixed-point, and there is a minimal number of iteration that achieves it. □

This theorem ensures that every time we define a new predicate by induction through a monotone first-order formula, we will be able to find its least fixed-point, hence its right interpretation. We denote by $\mathsf{fix}_{R(x_1,\dots,x_k)}\,\varphi$ the least fixed-point of some first-order formula $\varphi(R, x_1, \dots, x_k)$, which we will allow to use as a new relation symbol in our vocabularies.

Definition 2.1 ($\mathbf{FO}(\mathsf{LFP})$)**.** We denote by $\mathbf{FO}(\mathsf{LFP})$ the set of all Boolean queries that can be described using first-order formulas allowed to contain the least fixed-point operator to create new relation symbols.

2.2 $\mathbf{P} = \mathbf{FO}(\mathsf{LFP})$

With the fix operator in hand, we can define the reachability relation on alternating graphs, and hence we can describe the computation of the $\mathbf{P}$-complete problem AREACH. It is now reasonable to conjecture that $\mathbf{P}$ and $\mathbf{FO}(\mathsf{LFP})$ are closely related. Before we make this explicit, we make sure that $\mathbf{FO}(\mathsf{LFP})$ preserves the closure under first-order reductions.

Proposition 2.2. $\mathbf{FO}(\mathsf{LFP})$ *is closed under first-order reductions.*

Proof. Use the technique explained at the end of Section 1.3. □

We are now ready to show that $\mathbf{FO}(\mathsf{LFP})$ characterizes precisely the complexity class $\mathbf{P}$.

Theorem 2.3. *Over finite ordered structures,* $\mathbf{FO}(\mathsf{LFP}) = \mathbf{P}$.

Proof. We first show that $\mathbf{FO}(\mathsf{LFP}) \subseteq \mathbf{P}$. Let $Q \in \mathbf{FO}(\mathsf{LFP})$, be a Boolean query defined by first-order formula φ (that may contain the fix operator). We need to show that $Q \in \mathbf{P}$, that is, there exists a polynomial-time Turing machine M deciding Q. This means that for every finite structure $\mathcal{A}$,

$$\mathcal{A} \in Q \Leftrightarrow \mathcal{A} \models \varphi \Leftrightarrow M(\mathsf{bin}(\mathcal{A})) = 1$$

We will show that evaluating whether $\mathcal{A} \models \varphi$ can be done in polynomial time. If φ does not contain the fix operator, then by Theorem 1.2 ($\mathbf{FO} \subseteq \mathbf{L}$), φ can be checked under $\mathcal{A}$ in logspace, hence also in poly-time. On the other hand, if φ contains the fix operator, then in order to check φ we need to obtain its least fixed-point in polynomial time. We know, by the Knaster-Tarski theorem, that $\mathsf{fix}_{R(x_1,\dots,x_k)}\,\psi = \psi_{\mathcal{A}}^{n^k}(\emptyset)$, so we can evaluate $\psi_{\mathcal{A}}^{r}(\emptyset)$ for every $r \in [n^k]$ to find the least fixed-point. Because at each time we only check whether at most n^k tuples enter the interpretation, and this is repeated n^k times, we can compute the least fixed-point of the formula under

$\mathcal{A}$ in $n^k \cdot n^k = n^{2k}$ steps. Thus φ can be evaluated under $\mathcal{A}$ in polynomial time, so $\mathbf{FO}(\mathsf{LFP}) \subseteq \mathbf{P}$.

Now, to show that $\mathbf{P} \subseteq \mathbf{FO}(\mathsf{LFP})$, note that for every $L \in \mathbf{P}$,

$$L \leq_{\text{fo}} \mathsf{AREACH}$$

as AREACH is $\mathbf{P}$-complete. Besides, with the power of inductive definitions, $\mathsf{AREACH} \in \mathbf{FO}(\mathsf{LFP})$. And we also know $\mathbf{FO}(\mathsf{LFP})$ is closed under first-order reductions. Hence, $L \in \mathbf{FO}(\mathsf{LFP})$. Therefore, $\mathbf{P} \subseteq \mathbf{FO}(\mathsf{LFP})$.

We can conclude that $\mathbf{FO}(\mathsf{LFP}) = \mathbf{P}$. □

2.3 Inductive depth and iterations

When showing that the least fixed-point operator can be evaluated in polynomial time, we calculated the iteration of the inductive definition up to n^k times. However, it may well be that a certain inductive definition *closes* for a smaller number of iterations. The number of times a recursive formula has to be iterated before it closes is called its *depth*. We denote it by $\mathsf{depth}(\varphi; \mathcal{A})$. If we drop the structure, then we write $\mathsf{depth}(\varphi)$ for the maximum depth of φ over all structures of size n. If φ is an R-positive formula, where R has arity k, then clearly $\mathsf{depth}(\varphi) \leq n^k$.

Interestingly, alternative inductive definitions of the same predicate often have significantly different depths. For example, the formulas

$$\varphi(R, x, y) : x = y \vee \exists z(E(x, z) \wedge R(z, y))$$
$$\psi(R, x, y) : x = y \vee E(x, y) \vee \exists z(R(x, z) \wedge R(z, y))$$

are both defining the reflexive transitive closure on a graph. However, φ has $\mathsf{depth}(\varphi) = n$, while $\mathsf{depth}(\psi) = \lceil \log n \rceil + 1$.

Using the depth of inductive definitions, we can make finer gradations between complexity classes.

Definition 2.2 ($\mathbf{IND}[f(n)]$)**.** Let $\mathbf{IND}[f(n)]$ be the subclass of $\mathbf{FO}(\mathsf{LFP})$ such that all fixed points taken are of first-order formulas of depth $O(f(n))$.

Therefore, $\mathbf{FO}(\mathsf{LFP})$ consists of all the classes $\mathbf{IND}[f(n)]$ together, for $f(n)$ an polynomial:

$$\mathbf{FO}(\mathsf{LFP}) = \bigcup_{k \in \mathbb{N}} \mathbf{IND}[n^k]$$

Besides, we can give somewhat of a "lower bound" to these classes. Note that since the formula $\psi(R, x, y)$ defined before captures the reachability relation on graphs and $\mathsf{depth}(\psi; n) \in O(\log n)$, we have $\mathsf{REACH} \in \mathbf{IND}[\log n]$.

In addition, one can check that $\mathbf{IND}[\log n]$ is closed under first-order reductions, so using the fact that REACH is **NL**-complete, we have the following.

Proposition 2.4. $\mathbf{NL} \subseteq \mathbf{IND}[\log n]$.

It turns out that that are many properties that can be defined with inductive definitions of logarithmic-depth. The following is perhaps one of the most relevant.

Example 2.1 ($\mathsf{PARITY} \in \mathbf{IND}[\log n]$). Another interesting problem in the class $\mathbf{IND}[\log n]$ is PARITY: the set of all binary strings with an odd number of ones. We will show that PARITY can be described using an inductive definition of depth $O(\log n)$.

We inductively define predicate ρ, which takes as argument the tuple (i, j, d), so that the number of ones between indices i and j is exactly d:

$$\begin{aligned}\rho(i,j,d): \ & (i = j \wedge ((S(i) \wedge d = 1) \vee (\neg S(i) \wedge d = 0))) \vee \\ & (i < j \wedge \exists k \exists k' \exists d_1 \exists d_2 \exists l \exists l_1 \exists l_2 \\ & (i \leq k \wedge \mathsf{plus}(i, l, j) \wedge \mathsf{plus}(i, l_1, k) \wedge \mathsf{suc}(k, k') \wedge \mathsf{plus}(k', l_2, j) \wedge \\ & ((\mathsf{even}(l) \rightarrow l_1 = l_2) \wedge (\neg \mathsf{even}(l) \rightarrow \mathsf{suc}(l_1, l_2))) \wedge \\ & \rho(i, k, d_1) \wedge \rho(k', j, d_2) \wedge \mathsf{plus}(d_1, d_2, d)))\end{aligned}$$

The blue part of the formula is the base case, while the following lines define the inductive case. In red we have the recursive calls to ρ. Besides, we use predicate

$$\mathsf{even}(p) : \exists k \exists t (\mathsf{suc}(1, t) \wedge \mathsf{times}(t, k, p))$$

In the definition of ρ we would replace the recursive calls by some new relations symbol, P, and then find the least fixed-point of $\rho(P, i, j, d)$. Finally, the formula φ deciding parity will be

$$\varphi : \exists d((\mathsf{fix}_{P(i,j,d)}\, \rho)(0, \max, d) \wedge \neg \mathsf{even}(d))$$

Hence, for every $\mathcal{A} \in \mathsf{STRUC}[\sigma_1]$,

$$\mathcal{A} \in \mathsf{PARITY} \Leftrightarrow \mathcal{A} \models \varphi$$

It is left to show that $\mathsf{depth}(\varphi) = \mathsf{depth}(\rho) \in O(\log n)$. Simply note that ρ is a simple *divide and conquer* approach to the parity problem. We have $\rho^1_{\mathcal{A}}(\emptyset) = \{(i, i, d) \mid d = 1 \text{ if } S(i), \text{ otherwise } d = 0\}$, that is, all segments of length 1. If we iterate again, we will have all segments of length 1 and 2,

tagged with their corresponding number of ones. If we iterate again, we will have all segments of length 1, 2, 3 and 4. If we iterate again, we will have all segments of length up to 8... and so on. Hence, to get lo length 8 we need to iterate $\log 8 + 1 = 3 + 1 = 4$ times. For length 9, $\lceil \log 9 \rceil + 1 = 5$ times. It is easy to see that, in general, $\mathsf{depth}(\rho) = \lceil \log n \rceil + 1 \in O(\log n)$, so $\mathsf{PARITY} \in \mathbf{IND}[\log n]$. □

There is an alternative way to look at inductive definitions. Note that to calculate the least fixed-point of a formula of arity k we *iterate* at most n^k times. This polynomial iteration is made more explicit if we normalize our formulas. Given an R-positive formula $\varphi(R, x_1, \dots, x_k)$, where R has arity k, we can rewrite R in normal form as

$$\varphi \equiv (Q_1 z_1.\psi_1) \dots (Q_s z_s.\psi_s)(\exists x_1 \dots \exists x_k.\psi_s) R(x_1, \dots, x_k)$$

(see Lemma 4.20 in Immerman's book, p. 63).

Example 2.2 (Reflexive transitive closure in prenex normal form). Take the formula

$$\psi(R, x, y) : x = y \vee E(x, y) \vee \exists z(R(x, z) \wedge R(z, y))$$

defining reflexive transitive closure on a graph, which has inductive depth $\log n$. We can write this in normal form as follows. For the base case, one can rewrite

$$\psi'(R, x, y) : \forall z. \underbrace{(\neg(x = y \vee E(x, y)))}_{\psi_1} \exists z(R(x, z) \wedge R(z, y))$$

Note that for every x, y such that $x = y$ or $E(x, y)$ (i.e. pairs of points that are in the relation in the base case), the formula ψ_1 is vacuously true.

For the rest, we can take

$$\psi_2 : (u = x \wedge v = z) \vee (u = z \wedge v = y)$$
$$\psi_3 : (x = u \wedge y = v)$$

and write

$$\psi''(R, x, y) : (\forall z.\psi_1)\exists z(\forall u \forall v.\psi_2)(\exists x \exists y.\psi_3) R(x, y)$$

If we denote by $\mathcal{Q}$ the *quantifier block* $(\forall z.\psi_1)\exists z(\forall u \forall v.\psi_2)(\exists x \exists y.\psi_3)$, then

$$\psi(R, x, y) \equiv \mathcal{Q} R(x, y)$$

And, in fact, for every $\mathcal{A} \in \mathsf{STRUC}[\gamma]$,

$$\mathcal{A} \models \mathsf{fix}_{R(x,y)}\, \psi \leftrightarrow \mathcal{Q}^{\log|\mathcal{A}|} R(x,y)$$

where $\mathcal{Q}^r$ means the quantifier block repeated r times (in this case $\log|\mathcal{A}|$ times, because that is the amount of iterations needed for the inductive definition to close).

The previous example showcases that instead of writing inductive definitions with the fix operator, we can equivalently *iterate* a quantifier block polynomially many times.

Definition 2.3 ($\mathbf{FO}[f(n)]$)**.** We denote by $\mathbf{FO}[f(n)]$ the set of Boolean queries that can be described by first-order formulas with a quantifier block iterated $O(f(n))$ times.

Theorem 2.5. *For every polynomially bounded function $f(n)$,*

$$\mathbf{IND}[f(n)] = \mathbf{FO}[f(n)]$$

We omit the proof of the previous result; it follows from several normal form theorems. However, it is important to note that the theorem only holds for *polynomially bounded* functions. This is because an inductive definition can never be iterated more than polynomially many times, but quantifier blocks can. We will later see that in fact the theorem breaks for superpolynomial functions, and only the left-to-right inclusion holds for all $f(n)$.

3 Separation as inexpressibility

Until this point, we have characterized descriptively the most relevant aspects of the world of complexity up to and including polynomial time. We have the following inclusions:

$$\mathbf{FO} \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{IND}[\log n] \subseteq \bigcup_{k\in\mathbb{N}} \mathbf{IND}[n^k] = \mathbf{FO}(\mathsf{LFP}) = \mathbf{P}$$

The reason we are embarking on this descriptive endeavour is that we think this new framework can offer insights into classic complexity theory problems. That is: it can show whether certain complexity classes are distinct or not. Hence, before we keep finding descriptive analogs to other classic complexity classes, it is reasonable to ask whether the descriptive framework provides any tools to tell whether any of previous inclusions are strict.

Intuitively, such separation theorems should take the form of inexpressibility theorems. For example, our first goal will be to show that some query in $\mathbf{FO}(\mathsf{LFP})$ cannot be expressed without inductive definitions, thus showing

$$\mathbf{FO} \subsetneq \mathbf{P}$$

Note, however, that the usual techniques for first-order inexpressibility, such as the Compactness Theorem, will not work here, as we are dealing with finite structures. We will tackle this using a game-theoretic approach to first-order semantics: the *Ehrenfeucht-Fraissé games*.

3.1 The Ehrenfeucht-Fraissé games

Before we go on, it will come handy to define some notions of relation between structures. Specifically, *substructures* and *isomorphisms*.

Definition 3.1 (Substructure)**.** Let τ be a vocabulary, and let $\mathcal{A}, \mathcal{B} \in \mathsf{STRUC}[\tau]$. We say that $\mathcal{A}$ is a *substructure of* $\mathcal{B}$, written $\mathcal{A} \subseteq \mathcal{B}$, whenever

(i) $A \subseteq B$.

(ii) For every constant symbol $c \in \tau$, $c^{\mathcal{A}} = c^{\mathcal{B}}$.

(iii) For every relation symbol $R \in \tau$, $R^{\mathcal{A}} = R^{\mathcal{B}} \cap A^n$, where n is the arity of R.

Definition 3.2 (Isomorphism)**.** Let $\mathcal{A}, \mathcal{B} \in \mathsf{STRUC}[\tau]$. We say that $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic*, written $\mathcal{A} \cong \mathcal{B}$, if there exists a bijection $f : A \to B$ such that

(i) For every constant symbol $c \in \tau$, $f(c^{\mathcal{A}}) = c^{\mathcal{B}}$.

(i) For every relation symbol $R \in \tau$ of arity n, $(a_1, \dots, a_n) \in R^{\mathcal{A}}$ if and only if $(f(a_1), \dots, f(a_n)) \in R^{\mathcal{B}}$.

Proposition 3.1. *Let τ be an vocabulary and let $\mathcal{A}, \mathcal{B} \in \mathsf{STRUC}[\tau]$, such that $\mathcal{A} \cong \mathcal{B}$. Then, for every sentence $\varphi \in \mathcal{L}(\tau \setminus \{\leq\})$,*

$$\mathcal{A} \models \varphi \Leftrightarrow \mathcal{B} \models \varphi$$

Proof. Straightforward induction on the structure of φ. □

Note how we leave order out of the vocabulary in the previous proposition. This is because if we include order, finding isomorphisms becomes uninterestingly more difficult. In fact, in what follows we assume that all vocabularies exclude the order relation and numeric predicates. This will weaken our results, as we mention later in Remark 3.2.

We can now think of the following game, between two players, Samson and Delilah, over two structures $\mathcal{A}, \mathcal{B} \in \mathsf{STRUC}[\tau]$. Samson is convinced that $\mathcal{A} \ncong \mathcal{B}$, while Delilah is convinced that the two structures are indeed isomorphic. They decide to prove each other wrong in m rounds, using k pairs of pebbles. At each round, Samson chooses one of the k pairs of pebbles and one of the structures, and assigns one of the pebbles to an element in that structure. Then, Delilah places the remaining pebble in some element of the other structure. Intuitively, Delilah is trying to give, for every element Samson points at, the isomorphic image of it.

We can probably think that when playing the game, Samson has some formula φ in mind, for which he thinks $\mathcal{A} \models \varphi$ and $\mathcal{B} \not\models \varphi$. Then, we can imagine that when playing the game, the pebbles are analogous to variable assignments: he tries to assign the pebbles/variables in such a way that his formula is made true in one of the structures but not in the other under Delilah's choices.

If the formula Samson has in mind to prove Delilah wrong has more quantified variables than the number of pebbles they are playing with, then it may be the case the even if the structures are not isomorphic, Samson cannot prove so in their game. Then, when do we say that one of them has won? Well, we can basically assess whether Delilah has been successful in providing an isomorphism so far.

We can make this a bit more formal.

Definition 3.3 (Ehrenfeucht-Fraissé games)**.** Let $\tau = C \cup P$ be an unordered vocabulary, let $\mathcal{A}, \mathcal{B} \in \mathsf{STRUC}[\tau]$, and let $m, k \in \mathbb{N}$. We denote by Γ_m^k the

tuple $(\tau, \mathcal{A}, \mathcal{B}, m, k)$ describing the game presented above, played between Samson and Delilah over structures $\mathcal{A}$ and $\mathcal{B}$ using k pairs of pebbles and m moves. We call Γ_m^k an *Ehrenfeucht-Fraissé game*.

At each move $r \in [m]$, we describe the state of the game by a pair of partial functions α_r and β_r,

$$\alpha_r : C \cup \{x_1, \dots x_k\} \to A$$
$$\beta_r : C \cup \{x_1, \dots x_k\} \to B$$

where we allow some of the variables to be left unassigned, indicating that those pebbles are currently off the board.

We think of a move in the game as an update to the state of the board. If (α_r, β_r) is the state of the board after move r and at move $r+1$ Samson chooses the i-th pair of pebbles and these are assigned to elements $a \in A$ and $b \in B$ in the structures, then

$$(\alpha_{r+1}, \beta_{r+1}) = (\alpha_r[x_i \mapsto a], \beta_r[x_i \mapsto b])$$

where we take (α_0, β_0) as the initial state, when only constant have been assigned, according to the interpretations given by the structures.

The state (α_r, β_r) determines two things. Firstly, it determines *induced substructures* for $\mathcal{A}$ and $\mathcal{B}$. These are substructures $\mathcal{A}_r$, $\mathcal{B}_r$ taking as domains the sets $\mathrm{Rng}(\alpha_r)$ and $\mathrm{Rng}(\beta_r)$. Secondly, it determines a function $f_r : \alpha_r(x_i) \mapsto \beta_r(x_i)$. Then, we can say that Delilah *won round* r if f_r is an isomorphism between the induced substructures $\mathcal{A}_r$ and $\mathcal{B}_r$. Delilah *wins the game* if she wins every round. We write $\mathcal{A} \sim_m^k \mathcal{B}$ to indicate Delilah wins the game Γ_m^k. It is easy to check that $\sim_m^k$ is an equivalence relation. Whenever Delilah wins for every k or for every m, we can omit one of the indices.

With the idea of Ehrenfeucht-Fraissé games in hand, we are now closer to answer (some) inexpressibility questions. First, let us make finer gradations inside a language.

Definition 3.4 (Quantifier rank)**.** Let $\mathcal{L}$ be a language (such as first-order logic), and let φ be a formula in this language. We denote by $q(\varphi)$ the *quantifier rank* of φ, the number of nested quantifier in φ.

The quantifier rank of a formula, along with its number of variables, lets us define the following subclasses inside a language.

Definition 3.5. Let $\mathcal{L}$ be a language (such as first-order logic), and let $k, m \in \mathbb{N}$. We denote by $\mathcal{L}^k$ the subset of formulas of $\mathcal{L}$ containing occurrences of only k variables, $x_1, \dots, x_k$. We denote by $\mathcal{L}_m$ the subset of formulas of $\mathcal{L}$ with quantifier rank at most m. Finally, we denote by $\mathcal{L}^k_m$ the intersection $\mathcal{L}^k \cap \mathcal{L}_m$.

Definition 3.6 ($\mathcal{L}$-equivalence)**.** Given a vocabulary τ on language $\mathcal{L}$ and two structures $\mathcal{A}, \mathcal{B}$ for it, we say that $\mathcal{A}$ and $\mathcal{B}$ are $\mathcal{L}^k_m$*-equivalent*, written $\mathcal{A} \equiv^k_m \mathcal{B}$, if they agree on all sentences of $\mathcal{L}^k_m(\tau)$:

$$\mathcal{A} \equiv^k_m \mathcal{B} \text{ if and only if for all sentence } \varphi \in \mathcal{L}^k_m(\tau),\ \mathcal{A} \models \varphi \Leftrightarrow \mathcal{B} \models \varphi$$

This lets us state and prove the fundamental theorem of Ehrenfeucht-Fraissé games: Delilah wins game Γ^k_m if and only if the structures are $\mathcal{L}^k_m$-equivalent. To prove that, we first need a lemma.

Lemma 3.2. *Let τ be a first-order vocabulary, and let $r \in \mathbb{N}$. There are only finitely many inequivalent first-order sentences of quantifier rank r. That is, there are only finitely many inequivalent sentences in $\mathcal{L}_r(\tau)$.*

Proof. For this proof, we assume that $\mathcal{L}_r(\tau)$ contains sentences only. Let us first make explicit what it means for sentences to be equivalent. We say that $\varphi, \psi \in \mathcal{L}_r(\tau)$ are *equivalent*, written $\varphi \approx \psi$, if for every pair of structures $\mathcal{A}, \mathcal{B} \in \mathsf{STRUC}[\tau]$, $\mathcal{A} \models \varphi$ if and only if $\mathcal{B} \models \psi$. It is straightforward to check that $\approx$ is indeed an equivalence relation. Then the lemma states that for every $r \in \mathbb{N}$, the quotient set $\mathcal{L}_r(\tau)/_\approx$ is finite.

We prove the lemma by induction on r. For $r = 0$, we have finitely many atomic formulas; say a of them. Each of these can be true or false, so we can take them as propositional variables. We use them alongside $\neg$ and $\vee$ to build new formulas, so these are just propositional formulas with up to a propositional letters. Hence, we cannot build more formulas than Boolean functions on a variables. Since there are $2^{(2^n)}$ possible Boolean functions over n variables, we have that $\mathcal{L}_0(\tau)/_\approx$ has $\sum_{n=0}^{a} 2^{(2^n)}$ equivalence classes, hence finitely many inequivalent formulas.

Now assume $\mathcal{L}_r(\tau)/_\approx$ is finite up to r, and show it is also finite for $r+1$. In particular, suppose $\mathcal{L}_r(\tau)/_\approx$ has k equivalence classes. Now, for every class $[\varphi] \in \mathcal{L}_{r+1}(\tau)/_\approx$, we can take as representative some other formula $\varphi' \approx \varphi$ in prenex form. Similarly, we characterize each of the k equivalence classes of $\mathcal{L}_r(\tau)/_\approx$ by a prenex formula. Then, $\varphi' = Qx\psi$, where $q(\psi) = r$, so $\psi \in \mathcal{L}_r(\tau)$, and ψ is also in prenex form. Since there are only k inequivalent ψ's, there can only be at most $2k$ inequivalent formulas taking the shape of φ', so $\mathcal{L}_{r+1}(\tau)/_\approx$ is finite. This concludes the induction.

For every $r \in \mathbb{N}$, $\mathcal{L}_r(\tau)/_{\approx}$ is finite, so there are only finitely many inequivalent formulas of rank r. This was to show. □

We can now prove the fundamental theorem.

Theorem 3.3 (Fundamental theorem of Ehrenfeucht-Fraissé games)**.** *Let τ be an unordered vocabulary, let $\mathcal{A}, \mathcal{B} \in \mathsf{STRUC}[\tau]$, and let $k, m \in \mathbb{N}$. Then,*

$$\mathcal{A} \sim_m^k \mathcal{B} \text{ if and only if } \mathcal{A} \equiv_m^k \mathcal{B}$$

Proof. For the forward direction, we proceed by induction on m. For $m = 0$, if $\mathcal{A} \sim_0^k \mathcal{B}$, then f_0 is an isomorphism between the induced substructures $\mathcal{A}_0$ and $\mathcal{B}_0$. That is, for every sentence $\varphi \in \mathcal{L}(\tau)$,

$$\mathcal{A}_0 \models \varphi \Leftrightarrow \mathcal{B}_0 \models \varphi$$

In particular, $\mathcal{L}_0^m(\tau) \subseteq \mathcal{L}(\tau)$, so for every sentence $\psi \in \mathcal{L}_0^m(\tau)$,

$$\mathcal{A}_0 \models \psi \Leftrightarrow \mathcal{B}_0 \models \psi$$

Now we show by a simple induction on the structure of ψ that $\mathcal{A} \models \psi$ if and only if $\mathcal{B} \models \psi$. Note that $q(\psi) = 0$, which means ψ cannot be a quantified formula. For the base cases:

- If $\psi = R(c_1, \dots, c_n)$, then it follows immediately by the fact that $c_1^{\mathcal{A}}, \dots, c_n^{\mathcal{A}} \in \mathrm{Rng}(\alpha_0) = A_0$.
- If ψ is $c = c'$, it again follows immediately for the same reason.

For the inductive cases:

- If $\psi = \neg\chi$, then the induction hypothesis gives us that $\mathcal{A} \models \chi$ iff $\mathcal{B} \models \chi$, hence $\mathcal{A} \not\models \psi$ iff $\mathcal{B} \not\models \psi$, as desired.
- If ψ is a disjunction, then it follows immediately from the induction hypothesis.

This concludes the structural induction. Therefore, for every sentence $\psi \in \mathcal{L}_0^m(\tau)$, $\mathcal{A} \models \psi$ if and only if $\mathcal{B} \models \psi$. That is, $\mathcal{A} \equiv_0^k \mathcal{B}$. This concludes the proof for base case.

For the inductive case, suppose $\mathcal{A} \sim_m^k \mathcal{B}$ implies $\mathcal{A} \equiv_m^k \mathcal{B}$ up to m, and show the contrapositive for $m + 1$. If $\mathcal{A} \not\equiv_{m+1}^k \mathcal{B}$, that means there exists a sentence $\varphi \in \mathcal{L}_{m+1}^k(\tau)$ for which $\mathcal{A}$ and $\mathcal{B}$ disagree. In particular, let φ be the in prenex form and assume without loss of generality that $\mathcal{A} \models \varphi$ and

$\mathcal{B} \not\models \varphi$. Note that $q(\varphi) = m + 1 \geq 1$, so φ cannot be an atomic formula. It also cannot be $\varphi = \neg\psi$, as then it is not prenex. Similarly, φ cannot be a disjunction. Hence, $\varphi = \exists x\psi(x)$, there exists an $a \in A$ such that $\mathcal{A} \models \psi(a)$, but for every $b \in B$, $\mathcal{B} \not\models \psi(b)$. That is: $\mathcal{A} \not\equiv_m^k \mathcal{B}$. Since $q(\psi) = m$, by the contrapositive on the induction hypothesis, we have $\mathcal{A} \not\sim_m^k \mathcal{B}$, which means that Samson has a winning strategy for all m-move games, and we know that Delilah loses at move $m + 1$, so $\mathcal{A} \not\sim_{m+1}^k \mathcal{B}$.

For the backwards direction of the theorem, we will use Lemma 3.2 alongside and induction on m. For $m = 0$, it is trivially the case that $\mathcal{A} \equiv_0^k \mathcal{B}$ implies $\mathcal{A} \sim_0^k \mathcal{B}$.

Suppose $\mathcal{A} \equiv_m^k \mathcal{B}$ implies $\mathcal{A} \sim_m^k \mathcal{B}$ up to m, and show it for $m + 1$. Suppose $\mathcal{A} \equiv_{m+1}^k \mathcal{B}$. Note that, by the lemma, there are only finitely many inequivalent sentences in $\mathcal{L}_m^k(\tau)$. Imagine $\varphi_1, \dots, \varphi_n$ are the ones $\mathcal{A}$ satisfies. Then it is also the case that $\mathcal{A} \models \varphi_1 \wedge \cdots \wedge \varphi_n$ and trivially $\mathcal{A} \models \exists x(\varphi_1 \wedge \cdots \wedge \varphi_n)$. Since this formula is a sentence in $\mathcal{L}_{m+1}^k(\tau)$, by assumption, $\mathcal{B} \models \exists x(\varphi_1 \wedge \cdots \wedge \varphi_n)$ and so $\mathcal{B} \models \varphi_1, \dots, \mathcal{B} \models \varphi_n$, and so $\mathcal{A} \equiv_m^k \mathcal{B}$. By induction hypothesis $\mathcal{A} \sim_m^k \mathcal{B}$, and in the $m + 1$-th move, Delilah can just choose the right witness for whatever Samson chooses. □

This theorem is a useful tool to prove our first simple inexpressibility theorem.

Theorem 3.4 (Expressibility of CLIQUE_k)**.** *Let $k \in \mathbb{N}$, let CLIQUE_k denote the Boolean query on graphs containing those for which a k-clique exists, and let γ be the vocabulary of graphs without order. In the language $\mathcal{L}(\gamma)$, CLIQUE_k is first-order expressible with k variables, but not with $k - 1$ variables:*

$$\mathsf{CLIQUE}_k \in \mathcal{L}^k(\gamma) \setminus \mathcal{L}^{k-1}(\gamma)$$

Proof. It is easy to show that $\mathsf{CLIQUE}_k \in \mathcal{L}^k(\gamma)$. Simply note that he existence of a k-clique is expressed by the following formula κ:

$$\kappa : \exists x_1 \dots \exists x_k \left(\mathsf{distinct}(x_1, \dots, x_k) \wedge \bigwedge_{\substack{i,j \in [k] \\ i \neq j}} E(x_i, x_j) \right)$$

How do we show that k variables are strictly necessary? Suppose for contradiction that there was a formula κ' on $k - 1$ variables that also defined CLIQUE_k. We can use the fundamental theorem to argue that if such a formula existed, then it would follow that graphs with $k - 1$ vertices could have a k-clique!

Consider the structures $\mathcal{K}_k$ and $\mathcal{K}_{k-1}$, encoding the complete graphs of sizes k and $k-1$. Now see that $\mathcal{K}_k \sim^{k-1} \mathcal{K}_{k-1}$. In other words, when playing over these two graphs with $k-1$ pairs of pebbles, no matter what choice Delilah makes to answer Samson's, she wins. Therefore, by the fundamental theorem, $\mathcal{K}_k \equiv^{k-1} \mathcal{K}_{k-1}$, that is, they agree on all formulas contained in $\mathcal{L}(\gamma)^{k-1}$. In particular, since $\kappa' \in \mathcal{L}(\gamma)^{k-1}$ and $\mathcal{K}_k \models \kappa'$, we have $\mathcal{K}_{k-1} \models \kappa'$. But then $\mathcal{K}_{k-1}$ has a clique of size k, which is impossible, as the graph only has $k-1$ vertices. Contradiction. We can conclude that such a κ' cannot exist. □

Remark 3.1 (What was *not* proved). Based on the previous theorem, we have $\mathsf{CLIQUE}_k \in \mathbf{FO}$, hence $\mathsf{CLIQUE}_k \in \mathbf{P}$. Unfortunately, this does *not* imply $\mathsf{CLIQUE} \in \mathbf{P}$! Though it might seem intuitive to think so, we are talking about CLIQUE_k here, while $\mathsf{CLIQUE} = \bigcup_{k\in\mathbb{N}} \mathsf{CLIQUE}_k$. One could think that given some graph $\mathcal{G}$ with n nodes, one can construct the formula

$$\kappa : \bigvee_{k=1}^{n} \kappa_k$$

such that $\mathcal{G} \models \kappa$ if and only if $\kappa \in \mathsf{CLIQUE}$. This is *apparently* a Boolean first-order query, so from Theorem 1.2 ($\mathbf{FO} \subseteq \mathbf{L}$) it should follow $\mathsf{CLIQUE} \in \mathbf{L}$? The answer is negative, as κ is not a first-order query. A first-order Boolean query is defined by some fixed formula that works for every input structure. This is not the case with κ: depending on the value of n, the formula will be bigger or smaller (it will have more or less disjuncts). So this formula does not work for any graph, thus it does not define CLIQUE in general.

3.2 First-order inexpressibility

The fundamental theorem in itself is not powerful enough to prove the type of first-order inexpressibility results we are after, but it immediately gives us the following theorem, which brings us closer.

Theorem 3.5 (Methodology Theorem)**.** *Let τ be a first-order vocabulary, and let $Q \subseteq \mathsf{STRUC}[\tau]$ be a Boolean query. In order to show that query Q is not first-order expressible, it suffices to show, for every $n \in \mathbb{N}$, a pair of structures $\mathcal{A}_n$ and $\mathcal{B}_n$ such that*

(*i*) *$\mathcal{A}_n \in Q$ and $\mathcal{B}_n \notin Q$*

(*ii*) *$\mathcal{A}_n \sim_n \mathcal{B}_n$*

Proof. Suppose for contradiction that query Q is expressed by first-order formula φ, yet the conditions of the theorem hold. In particular, for the quantifier rank of φ, $q(\varphi)$, there exists a pair of structures $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \models \varphi$, $\mathcal{B} \not\models \varphi$ and $\mathcal{A} \sim_{q(\varphi)} \mathcal{B}$. From the latter, the fundamental theorem gives $\mathcal{A} \equiv_{q(\varphi)} \mathcal{B}$, implying that $\mathcal{A}$ and $\mathcal{B}$ must agree on all formulas of quantifier rank $q(\varphi)$. But this is not the case, as we just said that $\mathcal{A} \models \varphi$ and $\mathcal{B} \not\models \varphi$. Contradiction; such a φ cannot exist. □

We can use this method to prove the first meaningful inexpressiblity result.

Theorem 3.6. *The property of a graph being acyclic is not first-order expressible.*

Proof. We use the methodology theorem. For every natural number n, we consider the pair of structures $\mathcal{A}_n$ and $\mathcal{B}_n$ as in Figure 1.

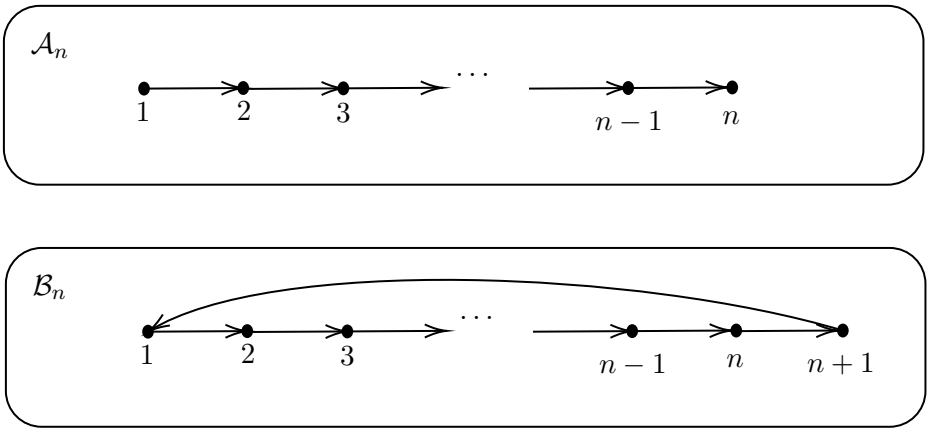

Figure 1: The graphs $\mathcal{A}_n$ and $\mathcal{B}_n$ from Theorem 3.6.

That is, $\mathcal{A}_n$ is an ascending chain of n points, and $\mathcal{B}_n$ is an ascending chain of $n+1$ points that goes back from $n+1$ to 1. The graph $\mathcal{A}_n$ is acyclic, while $\mathcal{B}_n$ is not. Now note that when playing over n rounds with n pairs of pebbles (it does not make sense to play with more), all points in $\mathcal{A}$ will be chosen, while in $\mathcal{B}_n$ a point is always left unchosen. Hence the induced substructure of $\mathcal{B}_n$ is acyclic, as there is always a gap. Besides, it is an ascending chain of n elements, so $\mathcal{A}_n \sim_n^n \mathcal{B}_n$. If we were to play with $k < n$ pebbles, then it is also clear that Delilah could still always match Samson's moves, so $\mathcal{A} \sim_n \mathcal{B}$. The methodology theorem applies, telling us that there is not fisrt-order formula defining the property of a graph being acyclic. □

A straightforward consequence is the fact that reachability is also not first-order expressible.

Theorem 3.7. *The reachability relation is not first-order expressible.*

Proof. If there was a first-order formula R expressing reachability, then

$$\forall x \forall y (x \neq y \rightarrow \neg(R(x,y) \wedge R(y,x))$$

would define the property of being acyclic. But then acyclicity would be first-order expressible, which contradicts the previous theorem. □

Remark 3.2 (The absence of ordering). Though tempting, it does not follow from the previous theorems that $\mathbf{FO} \subsetneq \mathbf{L}$. The inexpressibility results proven so far apply to the language of first-order logic *without ordering*. Though they are definitely going in the right direction, they do not settle the separation between $\mathbf{FO}$ and $\mathbf{FO}(\mathsf{LFP})$. We will now develop heavier machinery to tackle this goal.

3.3 Beyond games: $\mathbf{FO} \subsetneq \mathbf{P}$

As made clear by the previous remark, it is not possible to use the previous inexpressibility results to show that $\mathbf{FO} \neq \mathbf{P}$, as those theorems only hold for the language without ordering.

In order to show a separation that applies to $\mathbf{FO}$, we need something more elaborate. Recall the problem PARITY, which we worked with in Example 2.1, which is in $\mathbf{IND}[\log n]$. That lower bound can be improved slightly, but not much. In fact, it is not possible to express PARITY without inductive definitions. In other words: $\mathsf{PARITY} \notin \mathbf{FO}$. We can use this to separate $\mathbf{FO}$ from $\mathbf{P}$. Proving this, however, is rather intricate. We do it in three steps:

1. Show that $\mathbf{FO} = \mathbf{AC}^0$, the class of Boolean circuits of polynomial size and constant depth.
2. Prove Håstad's switching lemma[4].
3. Use the switching lemma to show that $\mathsf{PARITY} \notin \mathbf{AC}^0$.

[4] We follow Immerman's proof, which is copied verbatim from Beame, who wrote nicely a counting argument by Razborov, who in turn simplified Håstad's original proof, which first appeared in his 1986 PhD thesis.

3.3.1 $\mathbf{FO} = \mathbf{AC}^0$

The class $\mathbf{AC}^0$ contains the problems that can be decided with Boolean circuits of polynomial size and constant depth. In order to make this more fair, we let the gates in these circuits have unbounded fan-in.

It turns out that the reason $\mathbf{FO}$ is so weak is that it is exactly the same as the very weak circuit class $\mathbf{AC}^0$.

Theorem 3.8. $\mathbf{FO} = \mathbf{AC}^0$.

Proof. The tricky part is the forward inclusion: $\mathbf{FO} \subseteq \mathbf{AC}^0$. Let τ be a vocabulary, let $Q \in \mathbf{FO}$, and let φ be the first-order formula defining Q. We want to convert φ into a polynomial-size constant-depth circuit. We do this by "grounding" φ into a propositional formula.

On inputs of size n, we ground φ in the following way, by substituting the atomic formulas by propositional ones. For every relation symbol $R \in \tau$ of arity k, consider the propositional variables $r_{(a_1,\dots,a_k)}$, for every $(a_1, \dots, a_k) \in [n]^k$. For equalities, we consider the variables $e_{(a,a')}$, for every pair $(a, a') \in [n]^2$. Whenever an atomic variable occurs in φ, we substitute it by the appropriate combination of propositional variables. We do the same with ordering and numeric predicates. Then, on an input structure $\mathcal{A} \in \mathsf{STRUC}[\tau]$ of size n, determine what the value of the propositional variables should be looking at the interpretation given in $\mathcal{A}$, and serve that bit-string as input to the circuit encoding the grounded formula for inputs of size n. Because $Q \in \mathbf{FO}$, φ can be checked in logspace, hence also in poly-time, and so it cannot be an very big formula in n. Now, the grounded version of φ is only polynomially-bigger, so it is still polynomial in size. Besides the grounded φ has constant depth, because adding more propositional variables as n increases does not make the circuit deeper (only wider). Therefore, $Q \in \mathbf{AC}^0$.

For the backwards inclusion ($\mathbf{AC}^0 \subseteq \mathbf{FO}$), simply note that a Boolean circuit can be easily translated into a first-order formula by having a predicate represent the input bit-string. □

3.3.2 Håstad's switching lemma

The switching lemma states rather intricately something quite intuitive: given a Boolean formula over n variables, if we randomly assign values to a significant number of the n variables and leave there rest unassigned, chances are that the formula will already evaluate to either zero or one without any further assignments.

We formalize this in two results. First, the proper lemma (Lemma 3.10), will tell us that the fraction of restrictions that will leave a big decision tree for the restricted formula is very small. Then, we will be able to show (Proposition 3.11) that if the amount of variables left unassigned in a constant-depth circuit is less than some number depending on the size and depth of the circuit, then there will be a restriction that turns the formula into a constant.

Let us first develop the notation to make this more precise. Given a Boolean function $\phi : \{0,1\}^n \to \{0,1\}$, often assumed to be written as a DNF formula, a *restriction* $\rho : \{x_1, \dots, x_n\} \to \{0, 1, \star\}$ is a partial assignment to the variables in ϕ, such that the variables left unassigned are denoted with $\star$. We write $\phi|_\rho$ to refer to the formula ϕ after instantiating the partial assignment given by restriction ρ. We denote by $R(r,n)$ the set of all restrictions on n variables leaving r variables unassigned. Observe that $|R(r,n)| = \binom{n}{r} \cdot 2^{n-r}$.

In addition, given a Boolean formula ϕ in DNF, we denote by $T(\phi)$ the *binary decision tree of* ϕ, defined as follows. If $\phi = C_1 \vee \dots \vee C_m$, then the root of $T(\phi)$ is $T(C_1)$, which is just a binary decision tree for the clause C_1. This will have a single leaf labeled with 1, while the rest of the leaves are labeled 0, and represent each a restriction ρ, to which we append $T(\phi|_\rho)$. We denote by $h(T(\phi))$ the *height* (or *depth*) of $T(\phi)$.

Before we state and prove the lemma, we need a technical claim that will help us later. Let $\mathrm{Stars}(k,s)$ denote the set of sequences of subsets of $[k]$ such that the cardinalities of the subsets add up to s. We can give the following upper-bound to the size of this set.

Lemma 3.9. $|\mathrm{Stars}(k,s)| \leq (k/\ln 2)^s$.

Proof. Let γ be such that $(1+1/\gamma)^k = 2$, and prove that $|\mathrm{Stars}(k,s)| \leq \gamma^s$. By induction on s. For $s = 0$, the inequality holds trivially, by assuming that the empty sequence counts. Assuming it holds for $s' < s$, to prove it for s, let $\sigma \in \mathrm{Stars}(k,s)$. Then, $\sigma = (\sigma_1, \sigma')$, and if $|\sigma_1| = i \leq s$, then

$\sigma' \in \operatorname{Stars}(k, s-i)$. There are $\binom{k}{i}$ choices for σ_1, so

$$\begin{aligned}
|\operatorname{Stars}(k,s)| &= \sum_{i=1}^{\min\{k,s\}} \binom{k}{i} \cdot |\operatorname{Stars}(k,s-i)| \\
&\leq \sum_{i=1}^{k} \binom{k}{i} \cdot \gamma^{k-i} \qquad \text{(by induction hypothesis)} \\
&= \gamma^k \sum_{i=1}^{k} \binom{k}{i} \cdot \frac{1}{\gamma^i} \\
&= \gamma^s \cdot \left((1+1/\gamma)^k - 1\right) = \gamma^k
\end{aligned}$$

Now, since $1 + x < e^x$ for every $x > 0$, we have that

$$1 + \frac{1}{\gamma} < e^{\frac{1}{\gamma}}$$

and hence $(1+1/\gamma)^k < e^{k/\gamma}$. Taking logarithms on both sides and noting that the left-hand side of the inequality is 2, we have

$$\ln 2 < \frac{k}{\gamma} \ln e$$

hence $\gamma < \frac{k}{\ln 2}$. Combining this with the inequality proven by induction, we get the inequality stated in the lemma. □

We are now ready to state and prove Håstad's lemma. The idea formalized is that the fraction of restrictions for which the resulting decision tree is very deep is very small. In particular, is says that when assigning at least six sevenths of the input variables, the resulting formula is very likely to be quite straightforward to decide.

Lemma 3.10 (Håstad's switching lemma)**.** *Let ϕ be a DNF formula of width k over n variables (i.e. each clause has at most k literals). Let $p < 1/7$, and let $r = pn$. Then, for all $s \geq 0$,*

$$\frac{|\{\rho \in R(r,n) \mid h(T(\phi_\rho)) \geq s\}|}{|R(r,n)|} < (7pk)^s$$

Proof. Let $R = \{\rho \in R(r,n) \mid h(T(\phi_\rho)) \geq s\}$. We want to show that $\frac{|R|}{|R(r,n)|} < (7pk)^s$. We do this by showing that there is an injective function from R to

a set that we can upper-bound in size, hence giving an upper-bound on the size of $\frac{|R|}{|R(r,n)|}$, which will later imply the inequality stated in the lemma.

That is, we claim that there exist an injective function

$$\alpha : R \to R(r-s,n) \times \text{Starts}(k,s) \times \{0,1\}^{[s]}$$

This can be achieved as follows. Suppose $\phi = C_1 \vee \cdots \vee C_m$. For every $\rho \in R$, suppose C_i is the first clause in ϕ that is not made zero under ρ. In other words, $C_i|_\rho \neq 0$, and for all $j < i$, $C_j|_\rho = 0$. Now take the first branch (under some ordering of the tree, like the lexicographic order) of $T(\phi|_\rho)$ that has length at least s, and call it b. Let V_i be the set of variables left unassigned in $C_i|_\rho$, and let a_i be an assignment to the variables in V_i that makes $C_i|_\rho = 1$. If b ends before all the variables in V_i have been assigned, then let $b_i = b$ and shorten a_i, so that is only assigns values to the same variables as b. Otherwise, let b_i be the initial segment of b that assigns values to the variables in V_i.

Now consider the set $\sigma_i \subseteq [k]$ including those x such that the x-th variable in V_i is set by a_i. It happens that σ_i is nonempty, and from σ_i and C_i we can reconstruct a_i.

If $b \neq b_i$, then $b - b_i$ is still a path in $T(\phi|_{\rho b_i})$, so we take the next clause C_j, $i < j$, such that $C_j|_{\rho b_i} \neq 0$ and repeat the process above, taking a segment b_j of the branch and an assignment a_j. We keep doing this until the branch is used up. We will have split b into segments, $b = b_i, b_j, \dots$ and we will have an assignment $a = a_i, a_j, \dots$ assigning values to all the variables that show up in branch b.

Finally, define a map $\delta : [s] \to \{0,1\}$ such that $\delta(x) = 1$ if and only if a and b agree on the value they assign to variable x. Then, let the image of ρ be

$$\alpha(\rho) = (\rho a, (\sigma_i, \sigma_j, \dots), \delta)$$

The function α is injective, in that from $\alpha(\rho)$ we can reconstruct ρ as follows: C_i is the first clause that evaluates to 1 under ρa. From C_i and σ_i we reconstruct a_i, and using δ we can reconstruct the segment b_i of the branch. We can repeat this process until we get the original ρ. Hence, α is injective.

Now that we have an injective function, we have

$$|R| \leq |R(r-s,n)| \cdot |\,\text{Starts}(k,s)| \cdot 2^s$$

and thus

$$\frac{|R|}{|R(r,n)|} \leq \frac{|R(r-s,n)|}{|R(r,n)|} \cdot |\,\text{Starts}(k,s)| \cdot 2^s$$

Now note that because $|R(r,n)| = \binom{n}{r} \cdot 2^{n-r}$, we have

$$\frac{|R(r-s,n)|}{|R(r,n)|} = \frac{\binom{n}{r-s} \cdot 2^{n-(r-s)}}{\binom{n}{r} \cdot 2^{n-r}} = \frac{\binom{n}{r-s}}{\binom{n}{r}} \cdot 2^s \leq \left(\frac{2r}{n-r}\right)^s$$

Combining this with the upper-bound $|\operatorname{Stars}(k,s)| \leq (k/\ln s)^s$ from the previous lemma, we get

$$\begin{aligned}\frac{|R|}{|R(r,n)|} &\leq \left(\frac{2r}{n-r}\right)^s \cdot \left(\frac{k}{\ln 2}\right)^s \cdot 2^s \\ &= \left(\frac{4rk}{(n-r)\ln 2}\right)^s = \left(\frac{4pk}{(1-p)\ln 2}\right)^s\end{aligned}$$

where the last equality is obtained when replacing $r = pn$. For $p < 1/7$, the above expression is less than $(7pk)^s$. This was to show. □

We can now use the switching lemma to show that a circuit that is small can be made constant by restricting some of its variables. This is where the "switching" aspect of the lemma gets into action.

Proposition 3.11. *Let C be a circuit of size s, depth d and unbounded fan-in. If we choose $r \leq n/(14^d(\log s)^{d-1}) - (\log s - 1)$, then there is a restriction $\rho \in R(r,n)$ for which $C|_\rho$ is constant.*

Proof. We assume that the circuit is rearranged in alternating layers of AND and OR gates, such that the input level has all OR gates. Similarly, we assume that both the input bits $x_1, \dots, x_n$ as well as their negations $\neg x_1, \dots, \neg x_n$ are available and only used in the first level.

At the first layer, there are at most s OR gates, each taking as inputs up to n literals. Hence, we have at most s DNFs with clauses of width $k = 1$. Now, we know that if we assign values to a lot of the variables, a lot of these gates will be easily decided. More formally, for each OR gate G on the first layer, we can apply Håstad's switching lemma with $p = 1/14$ and $r_1 = pn = n/14$ and we get

$$|\{\rho \in R(r_1,n) \mid h(T(G_\rho)) \geq \log s\}| < 2^{-\log s} \cdot |R(r_1,n)|$$

In other words, most of the assignments to $13/14$-ths of the variables make the clauses easy to decide. Adding over all input OR gates, of which there are at most s, we have that the number of restrictions ρ for which $h(T(G|_\rho)) \geq \log s$ for some G is at most

$$s \cdot |\{\rho \in R(r_1,n) \mid h(T(G1_\rho)) \geq \log s\}| < s \cdot 2^{-\log s} \cdot |R(r_1,n)| = |R(r_1,n)|$$

In particular, since it is less than $|R(r_1, n)|$, that means there is a restriction $\rho_1 \in R(r_1, n)$ such that all gates on the input level are turn into decision trees of height less than $\log s$.

For each of these trees, consider the branches that lead to zero: they have length at most $\log s$ and they can be written as a conjunction of literals, representing the decisions taken to reach those leaves. Then, the gate can be expressed as a conjunction of the negation of these clauses. And by De Morgan's laws, this can be converted into a CNF of width $\log s$. In short, the input layer, formed by DNFs, has *switched* into a layer of CNFs. This is where the "switching" in the switching lemma comes from.

Now the second layer, which contains AND gates, is basically formed by CNFs, whose clauses are the CNF coming from the switched first layer i.e. the second layer has CNFs of width at most $\log s$. If G is now a gate in the second layer, we can apply the switching lemma again, this time with width $k = \log s$, $p = 1/(14 \log s)$ leaving $r_2 = pr_1 = r_1/(14 \log s)$ variables unassigned, and get

$$|\{\rho \in R(r_2, r_1) \mid h(T(G_{\rho_1})) \geq \log s\}| < 2^{-\log s} \cdot |R(r_2, r_1)|$$

By the same argument as before, there exists a restriction $\rho_2 \in R(r_2, r_1)$ under which every gate at level two switched to a DNF formula of width at most $\log s$.

We keep repeating this process through all d levels, ending with a restriction $\rho = \rho_1 \rho_2 \dots \rho_d$ such that $h(T(C|_\rho)) < \log s$. Note that the number of variables left unassigned in ρ is

$$r_d = \frac{n}{14^d (\log s)^{d-1}}$$

so after taking any branch b in $T(C|_\rho)$, we will have that in $C|_{\rho b}$ there will be have at least

$$r = r_d - (\log s - 1) = \frac{n}{14^d (\log s)^{d-1}} - (\log s - 1)$$

variables left unassigned.

Therefore, we can conclude that given a circuit C, there exists a restriction that makes the circuit constant while leaving r variables or less unassigned. This was to show. □

3.3.3 PARITY $\notin$ **FO**

The previous proofs were difficult and technical, but they directly imply our impossibility result.

Theorem 3.12. *Constant-depth, unbounded fan-in Boolean circuits need exponential size to compute* PARITY.

Proof. Suppose C is a size-s, depth-d unbounded fan-in circuit computing PARITY for inputs of size n. Suppose we want to take a restriction of C that leaves only one variable unassigned. Furthermore, assume that $1 \leq n/(14^d(\log s)^{d-1}) - (\log s - 1)$. Then by the previous proposition, there exists a restriction $\rho \in R(1,n)$ such that $C|_\rho$ is constant. But this cannot be, because PARITY is sensible to any bit-flip, so it cannot be made constant by fixing some of its inputs. Hence, it must be that $1 \not\leq n/(14^d(\log s)^{d-1}) - (\log s - 1)$. Then we have the following inequalities:

$$\begin{aligned} 1 &> \frac{n}{(14^d(\log s)^{d-1})} - (\log s - 1) \\ \log s &> \frac{n}{14^d(\log s)^{d-1}} \\ (\log s)^d &> \frac{n}{14^d} \\ \log s &> \frac{n^{1/d}}{14} \\ s &> 2^{\frac{1}{14}n^{1/d}} \end{aligned}$$

We conclude that C is exponential in size. This was to show. □

Recall that $\mathbf{AC}^0$ contains problems that are solvable by constant-depth *polynomial-size* circuits. Hence,

Corollary 3.13. $\mathsf{PARITY} \notin \mathbf{AC}^0 = \mathbf{FO}$.

Now recall from Example 2.1 that $\mathsf{PARITY} \in \mathbf{IND}[\log n] \subseteq \mathbf{FO}(\mathsf{LFP}) = \mathbf{P}$. Hence, $\mathsf{PARITY} \in \mathbf{P}$ but $\mathsf{PARITY} \notin \mathbf{FO}$. This is what we have been after.

Corollary 3.14. $\mathbf{FO} \subsetneq \mathbf{P}$.

This result further limits the power of first-order queries. Until now, we supposed that **FO** was weak based on complexity conjectures such as $\mathbf{L} \neq \mathbf{P}$. Now we can tell for sure: there are problems that are easy to compute that cannot be described using first-order logic alone. This further justifies the use of inductive definitions to capture **P**: they are strictly necessary! Unfortunately, this separation alone does not yet prove any separations between the in-between classes. In order words, it does not settle anything about the relations between **L**, **NL** and **P**.

4 NP under the descriptive framework

Now that we proved $\mathbf{FO} \subsetneq \mathbf{P}$, we know that we will not be able to characterize the class **NP** using first-order formulas only. Besides, under the widely-believed $\mathbf{P} \neq \mathbf{NP}$, we do not expect to be able to express all **NP** problems using only first-order logic with inductive definitions. It is therefore time to increase our linguistic resources.

In 1974, Ronald Fagin did so, and gave a characterization of **NP** using existential second-order logic. The fact that jumping to second-order logic is necessary to capture **NP** gives further evidence that $\mathbf{P} \neq \mathbf{NP}$.

We start by introducing second-order logic as an extension to what we already know, and show how some well-known **NP** problems can be easily stated using second-order formulas. We then prove Fagin's theorem, look into how **NP**-complete problems are preserved under first-order reductions, and we finish by extending our results to the Polynomial Hierarchy, showing that it is exactly the class of problems characterized by the full power of second-order logic.

4.1 Second-order logic

Second-order logic is a natural extension of first-order logic. Now, in addition to the usual quantification over the elements of our domain, we can also quantify over predicates. That is, we can quantify over relations on that domain.

Definition 4.1 (Second-order languages)**.** Let $\tau = C \cup P$ be a first-order vocabulary. In second-order logic, we consider a set of variables

$$V = \underbrace{\{x, y, z, \ldots\}}_{V_1} \cup \underbrace{\{R^1, R^2, \ldots, S^1, S^2, \ldots\}}_{V_2}$$

where V_1 is the set of first-order variables and V_2 contains *second-order variables*. These are tagged with a number, indicating their arity.

A *term* is still just a first-order term (a constant symbol $c \in \tau$ or a first-order variable from V_1), while second-order variables applied on terms work as atomic formulas.

A formula is just built from terms using the same quantifiers and operators as in first-order logic, with the difference that now quantifiers can also take a second-order variable. We will implicitly distinguish between first-order and second-order quantifiers, though the symbols will be the same.

We denote by $\mathcal{S}(\tau)$ the set of all well-formed second-order formulas from vocabulary τ.

Definition 4.2 (Second-order satisfaction)**.** Let τ be a first-order vocabulary. Structures remain the same as in first-order logic. Hence, let $\mathcal{A} \in \mathsf{STRUC}[\tau]$. A *second-order assignment* is a function $s : V \to \bigcup_{n\in\mathbb{N}} A^n$ assigning first-order variables to elements in A and second-order variables X^n to elements in A^n. The interpretation of a second-order variable is the same a for first-order variables: $R^{n\mathcal{A}}_s = s(R^n) \subseteq A^n$.

We can then extend first-order semantics with two new cases:

- If $\varphi = R^n(t_1, \dots, t_n)$, then
$$\mathcal{A}, s \models R^n(t_1, \dots, t_n) \text{ iff } ({t_1}^{\mathcal{A}}_s, \dots, {t_n}^{\mathcal{A}}_s) \in s(R^n) \subseteq A^n$$
- If $\varphi = \exists R^n \psi$, then
$$\mathcal{A}, s \models \exists R^n \psi \text{ iff there is an } R^n\text{-variant } s' \text{ of } s \text{ for which } \mathcal{A}, s' \models \psi$$

The notion of first-order queries admits an immediate generalization.

Definition 4.3 (**SO** and **SO-E**)**.** We denote by **SO** the set of all second-order Boolean queries. We denote by **SO-E** the set of all second-order Boolean queries using only existential second-order quantifiers[5].

Example 4.1 ($\mathsf{3COL}, \mathsf{SAT} \in$ **SO-E**). We can already write well-known **NP** problems into second order-formulas. For example, the query $\mathsf{3COL} \subseteq \mathsf{STRUC}[\gamma]$ asking whether a graph is 3-colorable can be written as

$$\exists R^1 \exists Y^1 \exists B^1 \forall x((R(x) \vee Y(x) \vee B(x)) \wedge \forall y(E(x,y) \to \\ \neg((R(x) \wedge R(y)) \vee (Y(x) \wedge Y(y)) \vee (B(x) \wedge B(y))))$$

More interestingly, SAT has a very succinct representation as a second-order formula, using the vocabulary from Example 1.5:

$$\exists S^1 \forall x \exists y((P(x,y) \wedge S(y)) \vee (N(x,y) \wedge \neg S(y)))$$

Hence $\mathsf{3COL}, \mathsf{SAT} \in$ **SO-E** $\subseteq$ **SO**. □

4.2 Fagin's theorem: NP = SO-E

Given that, as in the previous example, we can write well-known **NP**-complete problems into second-order logic, it is not surprising that **NP** is precisely the class of existential second-order Boolean queries. This is exactly the content of Fagin's theorem. We now prove this in some detail.

The right-to-left inclusion is rather straightforward.

[5]There may still be universal quantifiers, but only on first-order variables.

Proposition 4.1. **SO-E** $\subseteq$ **NP**.

Proof. We need to show that every existential second-order Boolean query can be computed in non-deterministic polynomial time. Let τ be a vocabulary, let $Q \subseteq \mathsf{STRUC}[\tau]$ be such that $Q \in$ **SO-E**, and let $\varphi = \exists R_1^{r_1} \dots \exists R_k^{r_r} \psi \in \mathcal{S}(\tau)$ be the second-order formula defining Q.

We show that there exists a non-deterministic polynomial-time Turing machine M such that for every $\mathcal{A} \in \mathsf{STRUC}[\tau]$,

$$\mathcal{A} \models \varphi \Leftrightarrow M(\mathsf{bin}(\mathcal{A})) = 1$$

The machine is simple: given $\mathcal{A} \in \mathsf{STRUC}[\tau]$, $|\mathcal{A}| = n$, the machine M simply "guesses" non-deterministically the right interpretations for the relations $R_1^{r_1}, \dots, R_k^{r_r}$. That is, it writes, for every $R_i^{r_i}$, a bit-string of length n^{r_i}, which is simply the binary representation of a correct interpretation for this symbol (using the representation defined in Definition 1.7). This takes time $\Theta(\sum_{i=1}^{k} n^{r_i})$, which is polynomial in the input structure's size, n. Now, given the right interpretations in binary for the second-order variables, as well as the binary representation of $\mathcal{A}$ received as input, checking ψ is simply checking a first-order formula, and we know **FO** $\subseteq$ **L**, so this can be done in polynomial time. Hence, **SO-E** $\subseteq$ **NP**. □

The other direction of the theorem is a bit trickier, as it requires to encode a computation into a second-order formula.

Proposition 4.2. **NP** $\subseteq$ **SO-E**.

Proof. Let query $Q \subseteq \mathsf{STRUC}[\tau]$ be in **NP**. That means there exists a non-deterministic Turing machine M computing Q in n^k steps for some k. Then, we build a second-order existential sentence φ such that for every $\mathcal{A} \in \mathsf{STRUC}[\tau]$, $\mathcal{A} \models \varphi$ if and only if $M(\mathsf{bin}(\mathcal{A})) = 1$.

The main idea is that this formula will use a second-order existential quantifier to find the sequence of non-deterministic choices that makes M accept, if there is such a sequence. That is: at every computation step $i \in [n^k]$, machine M chooses non-deterministically which of the two transition functions to use. Let $\delta_i \in \{0, 1\}$ encode that choice. We will encode the sequence of choices $\delta_1, \dots, \delta_{n^k}$ into a k-ary second-order variable Δ^k. We can do this because working on base n, we can encode every number in $[n^k]$ using a tuple of length $\log_n n^k = k$. Then we consider predicate Δ^k, such that for every $i \in [n]^k$, $\Delta^k(i)$ is true if and only if $\delta_i = 1$.

We now need some extra book-keeping to do: we still need to specify the details of the machine into the formula, but we leave most of the work to the

existential quantifiers. Since the machine runs in time n^k, it also does not use more that n^k cells of memory. We can represent the Turing machine's trace by a $n^k \times n^k$ matrix C, where position $C(i,j) = s$, containing the symbol s written in cell j at time i. If, in addition, the machine's head is at that cell at that step, we suppose it contains the pair (q,s), where q is the machine's current state. If the machine uses symbols from Σ and has as set of states Q, then each cell of the matrix can take one of the elements in $\Sigma \cup (Q \times \Sigma)$. This set is finite, so we can enumerate the elements. Say there are $|\Sigma \cup (Q \times \Sigma)| = c$ possible contents for the cells. Then we represent the machine's trace matrix C using c predicates $C_1^{2k}, \dots, C_c^{2k}$. Each of these predicates takes two tuples $i, j \in [n]^k$, each representing a number between 1 and n^k, such that $C_s(i,j)$ is true if and only if the trace matrix contains the s-th content symbol on cell j at step i.

Then, the formula φ encoding M will be

$$\varphi : \exists \Delta^k \exists C_1^{2k} \dots \exists C_c^{2k} \mu$$

where μ is a first order formula making sure that the given predicates correctly compute the query. That is,

$$\mu = I \wedge S \wedge T \wedge A$$

Here I checks the input (that content of the tape at the first step is $\mathsf{bin}(\mathcal{A})$). Formula S makes sure that no two things are simultaneously written on the same cell,

$$S : \bigwedge_{\substack{i,j \in [n]^k \\ s \neq s'}} (\neg C_s(i,j) \vee \neg C_{s'}(i,j))$$

Formula T encodes M's transition function, making sure that for every step i, the next row in the matrix follows from the i-th one following choice $\Delta^k(i)$. Finally, A checks that state n^k is the accepting state.

Clearly, φ expresses that there exists an accepting computation of M on input $\mathcal{A}$. Hence, for every $\mathcal{A} \in \mathsf{STRUC}[\tau]$, $M(\mathsf{bin}(\mathcal{A})) = 1$ if and only if $\mathcal{A} \models \varphi$. Therefore, $Q \in \mathbf{SO\text{-}E}$. This was to show. □

We then just proved both directions of Fagin's theorem.

Theorem 4.3 (Fagin's theorem)**.** $\mathbf{NP} = \mathbf{SO\text{-}E}$.

Remark 4.1 (Certificate definition of **NP**). Today, we often define **NP** as the class of problems for which short certificates of membership exist, instead of

referring to non-deterministic polynomial time. The idea of the certificate-based definitions was in part inspired by Fagin's theorem. For that reason, the result might not be as striking now as it was in 1974. For example, we know that SAT or 3COL are in **NP** because they have short certificates (an assignment or a 3-coloring, respectively), and this is specially obvious from the second-order formulas in Example 4.1. In a way, second-order syntax makes the structure of these certificates more explicit than when me simply talk about a binary string bounded in length by some polynomial, and, perhaps, that extra structure could be helpful in proving separations.

4.3 NP-completeness under first-order reductions

An alternative approach to proving $\mathbf{NP} \subseteq \mathbf{SO\text{-}E}$ in Fagin's theorem would have been to use the same technique as for Theorem 2.3 ($\mathbf{FO}(\mathsf{LFP}) = \mathbf{P}$): show that both **SO-E** and **NP** are closed under first-order reductions, give some complete problem for **NP** under first-order reductions and finally show that this problem is also in **SO-E**.

We now look into the topic of first-order reductions and complete problems for **NP**. Given that the theory of **NP**-completeness under the usual polynomial-time reductions has been so successful, do first-order reductions work equally well in this class?

We first make sure that we have first-order closure. That is, first-order reductions keep working in the intended way in this new class.

Proposition 4.4. *The class* **NP** (*and hence also* **SO-E**) *is closed under first-order reductions.*

Proof. The class **NP** is closed under logspace reductions, so by Proposition 1.3, **NP** is closed under first-order reductions. By Fagin's theorem, **SO-E** is also closed under first-order reductions. □

The most important **NP**-complete language, as shown by the Cook-Levin theorem in 1973, is SAT. This problem remains complete under the weaker first-order reductions.

Theorem 4.5. SAT *is* **NP***-complete under first-order reductions.*

Proof. We already know that $\mathsf{SAT} \in \mathbf{NP}$. We show that for every other query $Q \in \mathbf{NP}$, $Q \leq_{\text{fo}} \mathsf{SAT}$. From Proposition 4.2 ($\mathbf{NP} \subseteq \mathbf{SO\text{-}E}$) we know that there exists a second-order formula characterizing Q. Using the same notation as in the proof of Fagin's theorem, that formula is of the form

$$\varphi : \exists \Delta^k \exists C_1^{2k} \dots \exists C_c^{2k} \mu$$

Then, we simply make the predicates "explicit" by introducing propositional variables that represent them. In particular, we introduce n^k propositional variables $d_1, \dots, d_{n^k}$ to represent predicate Δ^k, as well as variables $p_{1,(1,1)}, \dots, p_{c,(n^k,n^k)}$ to represent predicates $C_1^{2k}, \dots, C_c^{2k}$ on all possible inputs. Then go through μ and interchange each occurrence of the second-order variables with the corresponding combination of propositional ones, as well as numeric predicates with $\top$ and $\bot$, after evaluating them on the structure. For every structure $\mathcal{A}$, this process yields a propositional formula that is satisfiable if and only if $\mathcal{A} \models \varphi$, hence if and only if $\mathcal{A} \in Q$. It can be checked that all of this can be described in first-order logic. Representing the propositional formula into the language of Example 4.1, we have that this is a $n^k + 1$-ary first-order query. Thus, $Q \leq_{\text{fo}} \mathsf{SAT}$. This was to show. □

4.4 The Polynomial Hierarchy

In the same way that Fagin's theorem characterizes the class **NP** as *existential* second-order logic, it is intuitive to suppose that this characterization can be generalized to nearby classes. Intuitively, it should hold that **coNP** is precisely universal second-order logic, and in fact such generalization can be extended to the entire Polynomial Hierarchy. That is, we conjecture (and prove) that $\mathbf{PH} = \mathbf{SO}$.

Unlike for other complexity classes, we cannot prove $\mathbf{PH} = \mathbf{SO}$ using complete problems and first-order closure. After all, **PH** is conjectured not to have complete problems, as otherwise the hierarchy would collapse.

Let us recall that complexity classes in these notes are taken to be defined originally based on machines. Hence, each level Σ_i^p of the polynomial hierarchy is defined as the class $\bigcup_{c\in\mathbb{N}} \Sigma_i\mathbf{TIME}[n^c]$, where $\Sigma_i\mathbf{TIME}[n^c]$ represents the class of problems computable by alternating Turing machines starting on existential states that run in time n^c and alternate at most $i-1$ times.

The main insight is given by the following lemma.

Lemma 4.6. *For every $k \in \mathbb{N}^*$, $Q \in \Sigma_k^p$ if and only if Q can be defined by a second-order formula of the form*

$$\exists R_1 \forall R_2 \dots Q_k R_k \varphi$$

where Q_k is either $\forall$ or $\exists$, depending on whether k is odd or even.

Proof. By induction on k. For $k = 1$, if $Q \in \Sigma_1^p$, then Q is decided by a polynomial-time alternating Turing machine that starts at an existential

state and never alternates. That is exactly a polynomial-time non-deterministic Turing machine. Indeed, $\Sigma_1^p = \mathbf{NP}$. So by Fagin's theorem, Q can be defined as a second-order existential formula. The backwards direction follows similarly: if Q is defined by a second-order existential formula, a structure $\mathcal{A}$ models the formula if and only if there exists a correct interpretation for the existential predicates. Say c is the maximum arity amongst them. Then the interpretation, written in binary, takes space $O(n^c)$. The idea is that this bit-string encodes the transition choices made by the machine. Hence the machine runs in time $O(n^c)$ and goes only through existential states. So $Q \in \Sigma_1^p$.

For the inductive step, assume the lemma holds up to k and show it for $k+1$. Suppose $Q \in \Sigma_{k+1}^p$. Then, there exist an alternating Turing starting in an existential state, alternating for k times and accepting in n^c steps, for some constant c. Then, as before, the existential transition choices of the first block make the interpretation for an existential predicate, and then we introduce a universal predicate as long as the number of steps the machine runs on the first universal block. Then, by induction hypothesis, there exists an alternating second-order formula with k alternations that we can append to this quantifier prefix, giving the desired second-order formula. The backwards direction follows from the same reasoning as in the base case. □

We now show that the full power of second-order logic characterizes $\mathbf{PH}$.

Theorem 4.7. $\mathbf{PH} = \mathbf{SO}$.

Proof. If $Q \in \mathbf{PH}$, then $Q \in \Sigma_i^k$ for some k, hence by the previous lemma Q is second-order expressible, hence $Q \in \mathbf{SO}$, so $\mathbf{PH} \subseteq \mathbf{SO}$. On the other hand, if $Q \in \mathbf{SO}$, then on the formula defining Q, when written in prenex from, one can easily readapt quantifiers, the formula and the arity of predicates so that the formula takes the form of a k-alternating formula for some k, hence by the previous lemma $Q \in \Sigma_k^p$, thus $Q \in \mathbf{PH}$, $\mathbf{SO} \subseteq \mathbf{PH}$. It follows that $\mathbf{PH} = \mathbf{SO}$. □

Remark 4.2 ($\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ in the descriptive framework). Since $\mathbf{PH} = \mathbf{SO}$ and we know that $\mathbf{P} = \mathbf{NP}$ if and only if the polynomial hierarchy collapses at level zero, we have that $\mathbf{P} = \mathbf{NP}$ if and only if $\mathbf{P} = \mathbf{SO}$, that is, if and only if $\mathbf{FO}(\mathsf{LFP}) = \mathbf{SO}$. In other words, the question $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ can be rephrased in descriptive terms as: *Can all second-order Boolean queries be expressed as first-order Boolean queries containing* (*monotone*) *inductive definitions?*

5 PSPACE under the descriptive framework

We have seen that $\mathbf{PH} = \mathbf{SO}$, and we conjecture that $\mathbf{PSPACE} \neq \mathbf{PH}$, so, perhaps unintuitively, second-order logic alone does not seem to help us in describing **PSPACE**-complete problems.

Descriptive complexity gives several characterizations of **PSPACE**. Some of these rely on ideas and classes that we have not covered here, but we still provide two alternative characterizations of the class, one using an extension of first-order logic, and one using an extension of second-order logic.

First, we make sure that the usual **PSPACE**-complete problems under polynomial-time reductions are also complete under first-order reductions. We then look into *iterative definitions*, an extension of inductive definitions that drops the monotonicity requirement in order to capture polynomial space. Finally, we look into how second-order logic can describe **PSPACE**.

5.1 PSPACE-completeness under first-order reductions

The most famous **PSPACE**-complete problem is probably TQBF: the set of true quantified Boolean formulas (QBF), a natural extension of SAT. This problem remains complete under first-order reductions.

Theorem 5.1. TQBF *is* **PSPACE***-complete under first-order reductions.*

Proof. We know that TQBF is complete under polynomial-time reductions, so $\mathsf{TQBF} \in \mathbf{PSPACE}$. We now show that for every other query $Q \subseteq \mathsf{STRUC}[\tau]$ such that $Q \in \mathbf{PSPACE}$, $Q \leq_{\text{fo}} \mathsf{TQBF}$. We use the fact that $\mathbf{PSPACE} = \mathbf{AP}$, the class of problems solved by alternating Turing machines in polynomial time.

Say M is an alternating Turing machine deciding Q in time n^k. We can rearrange M into a normal form where M writes down its transition choices and then executes them deterministically. If $c = c_1, \dots, c_{n^k}$ is the sequence of choices, then we can imagine a deterministic machine D taking as input c and $\mathcal{A} \in \mathsf{STRUC}[\tau]$,

$$M(\mathsf{bin}(\mathcal{A})) = 1 \Leftrightarrow \exists c_1 \forall c_2 \dots Q_{n^k} c_k D(c, \mathsf{bin}(\mathcal{A})) = 1$$

We can now easily see D as a non-deterministic Turing machine, and hence $L(D) \in \mathbf{NP}$. As a result, $L(D) \leq_{\text{fo}} \mathsf{SAT}$, so there exists a first-order query f such that

$$D(c, \mathsf{bin}(\mathcal{A})) = 1 \Leftrightarrow f(\mathcal{A}) \in \mathsf{SAT}$$

If $d_1, \dots, d_s$ are the additional Boolean variables of $f(\mathcal{A})$, then

$$M(\mathsf{bin}(\mathcal{A})) \Leftrightarrow \exists c_1 \forall c_2 \dots Q_{n^k} c_{n^k} \exists d_1 \dots \exists d_s f(\mathcal{A}) \in \mathsf{TQBF}$$

concluding that $Q \leq_{\text{fo}} \mathsf{TQBF}$. □

Let us look at a different example, related to graphs.

Definition 5.1 (GEOGRAPHY)**.** Let $G = (V, E)$ be a graph, and let $s \in V$ be a fixed node in the graph. The game *Geography* is played between Player P and Player Q over graph G starting at s. For the first move, Player P chooses node s. Then, Player Q has to choose a node $v_1 \in V$ adjacent to s: $(s, v_1) \in E$. Then, Player P chooses a node $v_2 \in V$, different form s and v_1, adjacent to v_1, and so on. One of the players loses whenever they are stuck: they cannot choose a further node, because there are no adjacent nodes, or because they have already been visited before.

We denote by GEOGRAPHY the problem containing the graphs with a starting node such that Player P has a winning strategy for Geography starting at that point.

Theorem 5.2. GEOGRAPHY *is* **PSPACE***-complete under first-order reductions.*

Proof. GEOGRAPHY is in **PSPACE**: there is a winning strategy for Player P if and only if there *exists* a first move (s) such that *for any move* Q makes, there *exists* a move for P, such that *for every* move Q makes... Q ends up being stuck. Clearly, GEOGRAPHY can be easily encoded into a QBF, and **PSPACE** is closed under first-order reductions, so GEOGRAPHY $\in$ **PSPACE**.

Now we show that GEOGRAPHY is complete by showing that TQBF $\leq_{\text{fo}}$ GEOGRAPHY. Suppose Φ is a QBF, which we arrange so that it is written in PCNF (prenex conjunctive normal form). Then we will build a graph over which a winning strategy for Geography exists if and only if Φ is true.

We show how to build this graph with an specific example. Take

$$\Phi : \exists x_1 \forall x_2 \exists x_3 ((x_1 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3))$$

and build the graph as in Figure 2. At each diamond, the player makes a choice as to what value to assign to one of the variables, and when they reach the clauses, player Q will try to choose one that is made false, in the sense that all the literal nodes have been used already. Therefore, $\Phi \in$ TQBF if and only if the graph built is in GEOGRAPHY.

With some technical work, we can extend the vocabulary of Example 1.5 to encode quantifiers for Φ, while the vocabulary for the graph will be the

usual vocabulary $\gamma = \{s, E\}$ we have been using. We can suppose that each vertex in the graph is represented as a triple $(i, j, k) \in [n]^3$. In particular, for the vertices in a diamond, these will be

$$(i, j, k) \in [n] \times \{1, 2, 3, 4\} \times \{1\}$$

where i refers to the variable x_i being played over, the index j indicates which one of the four vertices of the diamond we are referring to, and the third index is dummy.

For the clause nodes, we represent them as

$$(i, j, k) \in [n] \times \{1\} \times \{2\}$$

where i refers to the clause, j is fixed and $k = 2$, indicating that this is a clause node.

Under this representation, writing the query describing the graph is easily done in first-order logic, albeit tediously. We omit the formulas. This will be a ternary first-order query. Hence, TQBF $\leq_{\text{fo}}$ GEOGRAPHY. We conclude that GEOGRAPHY is **PSPACE**-complete. □

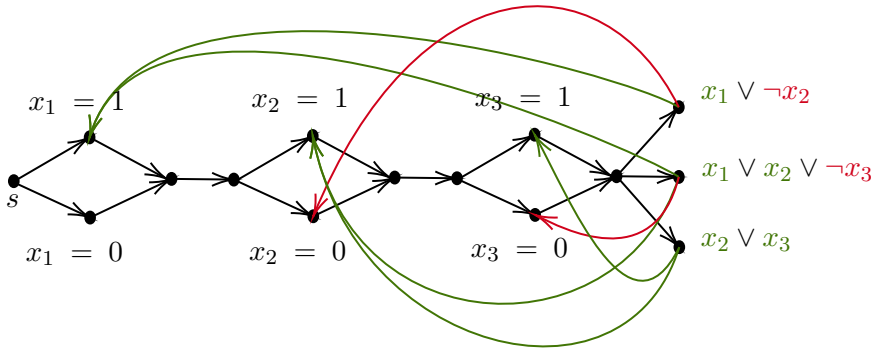

Figure 2: Graph construction for a Geography game on Φ.

Despite their different aspect, the two previous problems belong to the same "flavour" of **PSPACE**-complete problems; they consist of some sequence of decisions, quantified alternatingly. The other flavour, more useful for our descriptive purposes, revolves around reachability problems on

exponential-size graphs. For example, a polynomial-space Turing machine has a configuration graph of exponential size, implicitly defined by its transition function, and deciding whether the machine accepts on a certain input amounts to deciding whether there is a path from the start to the accepting state in this graph.

Definition 5.2 (TMREACH)**.** We denote by TMREACH the problem of deciding whether a deterministic Turing machines accepts a certain input x in space $|x|^k$, for some k fixed in the input.

Theorem 5.3. TMREACH *is* **PSPACE***-complete under first-order reductions.*

Proof. First, TMREACH $\in$ **PSPACE**. Note that if M runs in polynomial space, so that on inputs of size n it uses $s(n) = n^k$ cells of memory, then the transition graph of M on some input x has size $2^{O(n^k)}$, has an starting vertex s and an accepting vertex a, and deciding whether $M(x) = 1$ amounts to showing whether a is reachable from s in the configuration graph, which is an exponential-size implicitly described graph. Hence, the algorithm is the same as for DREACH: if there is a path from s to a, then this has length at most $2^{O(n^k)}$. We keep a counter up to that number, in a bit-string of length $O(n^k)$, to make sure that the walk never runs for longer than that. Then, at every step, we compute M's next step from the current state using its transition function. This algorithm is deterministic and runs in polynomial space, as it only needs to keep a step counter that is never bigger than polynomial in size.

To show completeness, the reduction is simple: given $Q \in$ **PSPACE**, take the machine M that decides Q in polynomial space, and then run the algorithm above on its transitions graph. The first-order formulas describing the reduction have the machine's transition function hardwired in them. Hence, $Q \leq_{\text{fo}}$ TMREACH. □

5.2 Capturing PSPACE with first-order logic

So far, the strongest extension of first-order logic we have defined in **FO**(LFP), which is exactly **P**. That operator imposes a constraint in our definitions: monotonicity. Thanks to monotonicity, the Knaster-Tarski theorem ensured that we can find the least fixed-point in polynomial time, but this is (likely) too weak to capture **PSPACE**. We now present *iterative definitions*, an extension of inductive definitions in which we drop the monotonicity requirement.

Definition 5.3 (Partial fixed-points, iterative definitions and $\mathbf{FO}(\mathsf{PFP})$)**.** The *partial fixed-point* operator, pfix, is an extension of the least fixed-point operator fix from section 2.1. Given a vocabulary τ and a formula $\varphi(R, x_1, \dots, x_k)$, where R is a new relation symbol of arity k and φ is not necessarily monotone, for every structure $\mathcal{A} \in \mathsf{STRUC}[\tau]$, we say that $\mathsf{pfix}_{R(x_1,\dots,x_k)}\,\varphi$ is an *iterative definition*, where

$$\mathsf{pfix}_{R(x_1,\dots,x_k)}\,\varphi = \begin{cases} \varphi^r_{\mathcal{A}}(\emptyset) & \text{if there is an } r \text{ such that } \varphi^r_{\mathcal{A}}(\emptyset) = \varphi^{r+1}_{\mathcal{A}}(\emptyset) \\ \emptyset & \text{otherwise} \end{cases}$$

We denote by $\mathbf{FO}(\mathsf{PFP})$ the set of all Boolean queries definable using first-order logic with the pfix operator.

Example 5.1 ($\mathsf{TMREACH} \in \mathbf{FO}(\mathsf{PFP})$). Consider an input structure $\mathcal{M}$ implicitly encoding a machine $M = (\Gamma, Q, \delta)$, an input $x \in \{0,1\}^*$ to that machine, and a number $k \in \mathbb{N}$, so that the task is to decide whether $M(x) = 1$ in space $|x|^k$.

With some work, one can write an iterative definition for counting up to 2^{n^k} in binary. Then we could combine the counter with a formula that starts in the accepting state of the configuration graph and tries to reach the starting state, while the counter does not overflow. This can be recursively defined using pfix, but the encoding is not at all obvious. □

From the example above we see that iterative definitions are strong enough to capture $\mathbf{PSPACE}$-complete problems. It turns out that $\mathbf{FO}(\mathsf{PFP})$ is precisely $\mathbf{PSPACE}$.

Theorem 5.4. $\mathbf{FO}(\mathsf{PFP}) = \mathbf{PSPACE}$.

Proof. The proof goes very much in the same lines as the proof of Theorem 2.3 ($\mathbf{FO}(\mathsf{LFP}) = \mathbf{P}$). For the forward inclusion, the only difference is that we need to make sure that the partial fixed-point can be computed in polynomial space. Since we have dropped monotonicity, at every step tuples cannot only be added, but also removed. Hence, there are at most 2^{n^k} possible interpretations for the partial fixed-point. Although this takes exponential time, in each iteration we only check which of polynomially many tuples enter or leave the relation. So we only need $O(n^k)$ space to keep track of this. Hence $\mathbf{FO}(\mathsf{PFP}) \subseteq \mathbf{PSPACE}$.

For the backwards inclusion, observe that TQBF is $\mathbf{PSPACE}$-complete (Theorem 5.1), so every every $Q \in \mathbf{PSPACE}$, $Q \leq_{\text{fo}} \mathsf{TQBF}$. At the same time, by Example 5.1, $\mathsf{TQBF} \in \mathbf{FO}(\mathsf{PFP})$, and it is easy to check that $\mathbf{FO}(\mathsf{PFP})$ is closed under first-order reductions, so $Q \in \mathbf{FO}(\mathsf{PFP})$. We have that $\mathbf{PSPACE} \subseteq \mathbf{FO}(\mathsf{PFP})$. This completes the proof. □

Analogously to the idea of $\mathbf{FO}[f(n)]$ developed in section 2.3, the fact that pfix iterates exponentially many times lets us characterize **PSPACE** as $\mathbf{FO}[2^{n^{O(1)}}]$. Note that for such a super-polynomial function, Theorem 2.5 breaks. We could, however, state a more general version of it, denoting by $\mathbf{ITER}[f(n)]$ the set of queries definable using pfix up to exponential depth, having $\mathbf{ITER}[2^{n^{O(1)}}] = \mathbf{FO}[2^{n^{O(1)}}]$.

5.3 Capturing PSPACE with second-order logic

Interestingly, **PSPACE** can be captured with both extensions of first-order and second-order logic. As with first-order logic, the second-order characterization of **PSPACE** can be done either with a new operator (the transitive closure operator) or through quantifier block iterations. We have not properly introduced the former, so we simply develop the intuition behind the quantifier block iterations.

As we just saw, $\mathbf{PSPACE} = \mathbf{FO}[2^{n^{O(1)}}]$. If we define $\mathbf{SO}[f(n)]$ as $\mathbf{FO}[f(n)]$ but for second-order quantifier blocks, then it turns out that $\mathbf{PSPACE} = \mathbf{SO}[n^{O(1)}]$. In other words, polynomial space corresponds to exponentially many first-order quantifier block iterations, while for second-order logic, we only need polynomially many iterations.

Bibliography

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

[2] Paul Beame. *A switching lemma primer*. (Manuscript). 1994.

[3] Neil Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. New York: Springer, 1999.

[4] Ryan O'Donnell. *Random Restrictions and* $\mathbf{AC}^0$ *Circuit Lower Bounds* (*Lecture 10 from Graduate Complexity Theory at CMU*). Available at `https://youtu.be/lC7m0u2b40c`. 2017.

[5] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

These notes have been compiled in the context of a reading project at the University of Amsterdam during the academic year 2020/21.