Descriptive Complexity: Motivation & Fagin's Theorem*

Adithya Bhaskara

July 23, 2024

1 Introduction & Motivation

This work seeks to provide the reader with a brief introduction to the field of descriptive complexity theory. Assuming the background of an older undergraduate student or a beginning graduate student, we motivate descriptive complexity as a field, provide some background, and finally state and prove Fagin's Theorem, a seminal result. We conclude by providing a broad overview of the "descriptive complexity zoo."

Following the exposition of [5], the two standard measures of computational complexity are, of course, time and space complexity. For practical purposes, these measures do suffice; we have good reason to care how the time it takes for our algorithm to execute scales as a function of the input size, and we also often care about how the space the algorithm uses scales as well. That being said, it is difficult to see what the mathematical basis is for choosing time and space as our measures of complexity.

We have a zoo of complexity classes: P, NP, coNP, EXPTIME, AM, MA, PSPACE, NPSPACE, and many more. But, with the tools currently at our disposal, we cannot say too much about the inherent complexities of the languages in these classes. We can only tell how fast or how much space is required to decide membership in the languages.

Enter Ronald Fagin. In [3], Fagin showed that NP is precisely the class of languages that are able to be described by existential second-order logic. Fagin's result importantly connected the "mathematical" complexities of languages to their "practical" complexities. A statement about the number of steps a nondeterministic Turing machine could take to solve a decision problem immediately became a statement about how difficult it was to even express the decision problem.

*Originally a final project for CSCI 5444: Introduction to the Theory of Computation at the University of Colorado Boulder, taught by Ashutosh Trivedi, Ph. D.

Fagin's result wasn't the end. While this work will not explore other results in descriptive-complexity in great detail, we provide results relating to the classes coNP, AC^0, and PH here[1,2].

Theorem 1. *The class* coNP *is equivalent to the languages that can be expressed in universal second-order logic.*

Theorem 2. *The class* AC^0 *is equivalent to the languages that can be expressed in first-order logic.*

Theorem 3. *The polynomial time hierarchy* PH *is equivalent to the languages that can be expressed in second-order logic.*

Now that we have stated some motivation for descriptive complexity theory, we will provide some notations and preliminaries in the next section.

2 Notation & Preliminaries

Drawing from [4, 5], we first introduce vocabularies and structures.

Definition 3 (Vocabularies, [4])**.** *A vocabulary is a finite set*

$$\sigma = \{R_1, \ldots, R_m, c_1, \ldots, c_s\}$$

where $R_1, \ldots, R_m$ *are relation symbols with fixed arities and* $c_1, \ldots, c_s$ *are constant symbols.*

Definition 4 (Structures, [4])**.** *A* σ*-structure is the tuple*

$$\mathbf{A} = (A, R_1^{\mathbf{A}}, \ldots, R_m^{\mathbf{A}}, c_1^{\mathbf{A}}, \ldots, c_s^{\mathbf{A}}),$$

where A *is a nonempty finite set and each* $R_i^{\mathbf{A}}$ *is a relation on* A *with* $\text{arity}(R_i^{\mathbf{A}}) = \text{arity}(R_i)$ *for* $i \in \{1, \ldots, m\}$ *and each* $c_j^{\mathbf{A}}$ *is a "distinguished element" of* A *for* $j \in \{1, \ldots, s\}$*. We refer to* A *as the universe of* $\mathbf{A}$*. We denote* $||\mathbf{A}||$ *to mean* $|A|$*, the cardinality of the universe of* $\mathbf{A}$*.*

[1]For readers who have not seen the definitions of AC^0 and PH, we provide them here, using [1, 2].

Definition 1 (AC^0)**.** *A language* L *is in the complexity class* AC^0 *if it can be solved using polynomial-size, constant depth, unbounded fanin circuits using* AND*,* OR*, or* NOT *gates [1, 2].*

Definition 2 (PH)**.** *For* $i \geq 1$*, the language* L *is in* Σ_i^p *if there exists a polynomial-time Turing machine* M *and a polynomial* q *with*

$$x \in L \iff \exists u_1 \in X, \forall u_2 \in X, \cdots Q_i u_i \in X, M(x, u_1, \ldots, u_i) = 1$$

where $X = \{0,1\}^{q(|x|)}$ *and* Q_i *is either* $\forall$ *or* $\exists$ *depending on whether* i *is even or odd, respectively. We then define the polynomial hierarchy as* $\mathsf{PH} = \bigcup_i \Sigma_i^p$ *[2].*

[2]These results are explained thoroughly in [5].

The above definitions are dense, so we will proceed with a few examples.

Example 1 ([4])**.** *An undirected graph can be represented as the structure* $\mathbf{G} = (V, E)$*. The universe of* $\mathbf{G}$ *is the vertex set* V*, and* $E \subseteq V \times V$ *is a relation on* V *with arity* 2*; we can write* $u \sim_E v$*, or equivalently,* $(u, v) \in E$*, to denote that* $u, v \in V$ *are related. That is,* u *and* v *share an edge in* $\mathbf{G}$*.*

Example 2. *An flow network can be represented as the structure* $\mathbf{N} = (V, E, C, s, t)$*. The universe of* $\mathbf{N}$ *is the vertex set* V*, and* $E \subseteq V \times V$ *is a relation on* V *with arity* 2*. The capacity of each edge is given by* $C \subseteq V \times V \times [0, \infty)$*, and* s *and* t *are constant symbols representing the source and sink vertices, respectively.*

We now define what it means for two structures to be isomorphic, i.e., what it means for two structures to "be the same."

Definition 5 (Isomorphism of Structures, [4])**.** *Let*

$$\mathbf{A} = (A, R_1^{\mathbf{A}}, \ldots, R_m^{\mathbf{A}}, c_1^{\mathbf{A}}, \ldots, c_s^{\mathbf{A}}), \quad \mathbf{B} = (B, R_1^{\mathbf{B}}, \ldots, R_m^{\mathbf{B}}, c_1^{\mathbf{B}}, \ldots, c_s^{\mathbf{B}})$$

be σ*-structures. The map* $f : A \to B$ *is an isomorphism if*

1. f *is bijective,*
2. *for all* c_j *with* $j \in \{1, \ldots, s\}$*,* $f(c_j^{\mathbf{A}}) = c_j^{\mathbf{B}}$*, and*
3. *for all* R_i *with* $i \in \{1, \ldots, m\}$ *with arity* t_i*, and for all* $a_1, \ldots, a_{t_i} \in A$*, it is the case that* $R_i^{\mathbf{A}}(a_1, \ldots, a_{t_i})$ *if and only if* $R_i^{\mathbf{B}}(f(a_1), \ldots, f(a_{t_i}))$*.*

We say that $\mathbf{A}$ *is isomorphic to* $\mathbf{B}$ *if there exists an isomorphism from* A *to* B*, and we write* $\mathbf{A} \cong \mathbf{B}$*.*

The key idea behind the isomorphism of structures is that two structures are the same if there exists a bijective map between them such that the constant symbols and the relations map to each other "nicely." Isomorphism is a technical condition, but it helps us define boolean queries[3].

Definition 6 (Boolean Queries, [4])**.** *A class of* σ*-structures is a set* $\mathcal{C}$ *of* σ*-structures closed under isomorphism. That is, if* $\mathbf{A} \in \mathcal{C}$ *and* $\mathbf{A} \cong \mathbf{B}$*, then* $\mathbf{B} \in \mathcal{C}$*. Let* $k \in \mathbb{Z}^+$*. Then, a boolean query on* $\mathcal{C}$ *is* $Q : \mathcal{C} \to \{0, 1\}$ *where* $Q(\mathbf{A})$ *is a* k*-ary relation on* $\mathbf{A}$ *with* $\mathbf{A} \cong \mathbf{B}$ *implies* $Q(\mathbf{A}) = Q(\mathbf{B})$*.*

In other words, $\mathcal{C}$ is a set of σ-structures that are all isomorphic to each other, and $Q(\mathbf{A})$ elicits some property of $\mathbf{A}$. Two isomorphic structures have the same image under Q. We provide some examples of queries for clarity.

Example 3 (Hamiltonian Path)**.**

$$\textsc{HamiltonianPath}(\mathbf{G}) = \begin{cases} 0 & \mathbf{G} \textit{ does not have a Hamiltonian path.} \\ 1 & \mathbf{G} \textit{ has a hamiltonian path.} \end{cases}$$

[3]Here, we sacrifice some generality for the sake of clarity.

Example 4 (Clique)**.**

$$\textsc{Clique}(\mathbf{G}) = \begin{cases} 0 & \mathbf{G} \textit{ does not have a clique of size } k. \\ 1 & \mathbf{G} \textit{ has a clique of size } k. \end{cases}$$

We can now define what it means for some property, or boolean query, to be definable in some system of logic L.

Definition 7 (L-Definability, [4])**.** *Let $Q : \mathcal{C} \to \{0,1\}$ be a boolean query on a class of σ-structures $\mathcal{C}$. Then, if L is a logic, Q is L-definable if there is an L-sentence ψ such that for all $\mathbf{A} \in \mathcal{C}$, $Q(\mathbf{A}) = 1$ if and only if $\mathbf{A}$ satisfies ψ.*

For the purposes of this work, first-order and second-order logic are defined as follows. We adapt the first-order definition from [5] and define second-order logic in a similar manner.

Definition 8 (The First-Order Language)**.** *The first-order language with respect to $\sigma = (R_1, \ldots, R_m, c_1, \ldots, c_s)$, denoted L^1_σ, is the set of formulas built from $R_1, \ldots, R_m$ and $c_1, \ldots, c_s$, the equality relation $=$, the boolean connectives $\wedge$ and $\neg$, primitive variables $\text{VAR} = \{x_1, \ldots\}$, and the existential quantifier $\exists$.*

We remark that the above definition captures the essentials of first-order language, we often use "syntactic sugar" in the form of $\neq$, $\neg$, $\vee$, $\implies$, $\iff$, and $\forall$. These are defined in the obvious manner.

Definition 9 (The Second-Order Language)**.** *The second-order language with respect to $\sigma = (R_1, \ldots, R_m, c_1, \ldots, c_s)$, denoted L^2_σ, can be described as L^1_σ equipped with relational variables $\text{RELVAR} = \{X_1, \ldots\}$ that we may quantify over. We use $\exists^k X \varphi$ to mean that there exists a k-ary relations X such that φ is satisfied.*

We now provide an example of using second-order logic to specify a decision problem.

Example 5 (Bipartite Graph)**.** *Let*

$$\Phi_{\textsc{Bipartite}} = (\exists X^1)(\forall u)\Big(\big(X(u) \vee \neg X(u)\big) \wedge (\forall v)\big((u,v) \in E \implies X(u) \wedge \neg X(v)\big)\Big).$$

Let $\textsc{Bipartite}(\mathbf{G})$ *be a boolean query defined as*

$$\textsc{Bipartite}(\mathbf{G}) = \begin{cases} 1 & \mathbf{G} \textit{ is a bipartite graph.} \\ 0 & \mathbf{G} \textit{ is not a bipartite graph.} \end{cases}.$$

Since $\textsc{Bipartite}(\mathbf{G}) = 1$ *if and only if* $\mathbf{G}$ *satisfies* $\Phi_{\textsc{Bipartite}}$*, and* $\Phi_{\textsc{Bipartite}}$ *is an* L^2_σ *sentence, we say that* $\textsc{Bipartite}(\cdot)$ *is* L^2_σ *definable*[4].

[4]Since we can decide if a graph is bipartite in polynomial time, we can actually make the stronger claim that Bipartite is definable in a first-order system with a least fixed point operator. For the sake of brevity, we will not discuss least fixed point operators further but instead refer readers to Chapter 4 of [5] and Chapter 2 of [4]. These operators allow us to define relations inductively.

Before formalizing how to capture languages in descriptive complexity classes, we informally describe existential second-order logic (ESO) and universal second-order logic (USO) in a similar manner to [4]. We have that ESO is the set of formulas in L^2_σ that can be written in the form

$$\exists X_1, \ldots, \exists X_m, \varphi(\bar{x}, X_1, \ldots, X_m).$$

where $\varphi(\bar{x}, X_1, \ldots, X_m) \in L^1_\sigma$ and $X_1, \ldots, X_m$ are relational variables in second-order logic. In a similar manner, USO is the set of formulas in L^2_σ that can be written in the form

$$\forall X_1, \ldots, \forall X_m, \varphi(\bar{x}, X_1, \ldots, X_m).$$

We finish this section by defining what it means for us to capture a complexity class K.

Definition 10 (Capturing Complexity Classes, [4])**.** *Let $\mathcal{C}$ be a class of structures, L be a logic, and K be a complexity class. We say that L captures K if for every query Q over $\mathcal{C}$, we have that Q is L-definable if and only if $Q \in \mathsf{K}$.*

As a notational convenience, we will write L to refer to the complexity class that logic L captures over all finite structures. So, ESO is precisely the complexity class that ESO captures, in other words, the complexity class that is described by existential second-order logic. Then, USO is defined analogously.

3 Fagin's Theorem

Ronald Fagin, in his Ph. D. thesis, [3], stated and proved what is today known as Fagin's Theorem. It is the oldest and most fundamental theorem behind descriptive complexity theory. We state and prove it below, following [4, 5].

Lemma 1 (Fagin, $\mathsf{ESO} \subseteq \mathsf{NP}$)**.** *Every boolean query expressible in existential-second order logic is computable in NP.*

[5]. Let $\Phi = (\exists R_1^{r_1} \cdots \exists R_k^{r_k})\varphi$ be a sentence in ESO. We wish to find some nondeterministic Turing machine M such that for all finite σ-structures $\mathbf{A}$, $\mathbf{A}$ satisfies Φ if and only if M accepts $\mathbf{A}$ in polynomial time.

On input $\mathbf{A}$, M functions by nondeterministically choosing the bits of a binary string of length $||\mathbf{A}||^{r_i}$ for each $R_1, \ldots, R_k$. Each binary string of length $||\mathbf{A}||^{r_i}$ represents R_i. This process takes a polynomial number of steps. After this is done, we have a structure $\mathbf{A}' = (\mathbf{A}, R_1, \ldots, R_k)$ which can be viewed as an extension of $\mathbf{A}$. Then, M should accept $\mathbf{A}$ if and only if $\mathbf{A}'$ satisfies the first order sentence φ. We can determine if $\mathbf{A}'$ satisfies φ in logspace[5], so we can of course do so in polynomial time. By our construction of M, M accepts $\mathbf{A}$

[5] For the sake of brevity, we do not explain why this is true. See Theorem 3.1 in [4], stating that the set of boolean queries definable in first-order logic (FO) is contained by the set of boolean queries computable in deterministic logspace, i.e. $\mathsf{FO} \subseteq \mathsf{L}$.

	Space 0	1	p	$n-1$	n		n^k-1	Δ
Time 0	$\langle q_0, w_0\rangle$	w_1	$\cdots$	w_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	δ_0
1	w_0	$\langle q_1, w_1\rangle$	$\cdots$	w_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	δ_1
	$\vdots$	$\vdots$	$\vdots$			$\vdots$		$\vdots$
t			$a_{-1}a_0a_1$					δ_t
$t+1$			b					δ_{t+1}
	$\vdots$	$\vdots$	$\vdots$			$\vdots$		$\vdots$
n^k-1	$\langle q_f, 1\rangle$	$\cdots$	$\cdots$			$\cdots$		

Figure 1: M's computation tableau on input $w = w_0 \cdots w_{n-1}$, where $\sqcup$ is the blank symbol. Figure 7.9 from [5].

if and only if there exists some choice of $R_1, \ldots, R_k$ such that $\mathbf{A}'$ satisfies φ, which is equivalent to $\mathbf{A}$ satisfying $\Phi = (\exists R_1^{r_1} \cdots \exists R_k^{r_k})\varphi$. The result follows immediately. ■

We now provide a proof sketch of the other direction. We defer the reader to [5] for a full proof. The full proof very closely resembles the canonical proof of the Cook-Levin Theorem, of which a thorough treatment is given in [6].

Theorem 4 (Fagin, $\mathsf{ESO} = \mathsf{NP}$)**.** *The complexity class* NP *is equal to the set of boolean queries expressible in existential second-order logic.*

Sketch. [5]] Due to Lemma 3, we need only show $\mathsf{NP} \subseteq \mathsf{ESO}$. That is, given some nondeterministic Turing machine M taking $||\mathbf{A}||^k - 1$ steps on $\mathbf{A}$, we wish to construct an ESO sentence that is satisfiable if and only if there exists an accepting computation of M. This ESO sentence is of the form

$$\Phi = (\exists C_1^{2k} \cdots \exists C_g^{2k} \Delta^k)\varphi$$

where φ, a first-order sentence, is satisfiable if $\bar{C}, \bar{\Delta}$ is an accepting computation for M on $\mathbf{A}$. We have that φ takes the form

$$\varphi = \phi_{\textsc{Encode}} \wedge \phi_{\textsc{Consistency}} \wedge \phi_\delta \wedge \phi_{\textsc{Acceptance}},$$

where $\phi_{\textsc{Encode}}$ ensures that row 0 of the computation tableau correctly encodes $\mathbf{A}$ in binary, $\phi_{\textsc{Consistency}}$ ensures that each computation cell only contains one Turing machine configuration, ϕ_δ ensures that each row follows from the previous row according to the transition function of M, and $\phi_{\textsc{Acceptance}}$ ensures that the last row has an accepting configuration.

For clarity, we provide the following figure, showing the computation tableau of M on $\mathbf{A}$.

With this construction, we have that $\Phi = (\exists C_1^{2k} \cdots \exists C_g^{2k} \Delta^k)\varphi$ is true if and only if there exists some accepting computation, $\bar{C}, \bar{\Delta}$, of M on $\mathbf{A}$. As we have now shown $\mathsf{NP} \subseteq \mathsf{ESO}$, we can conclude $\mathsf{NP} = \mathsf{ESO}$, as desired. ■

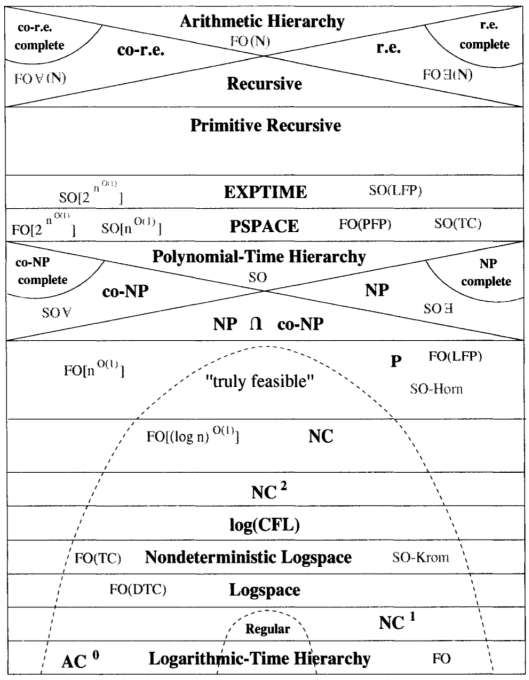

Figure 2: The descriptive complexity zoo. Figure 2.38 from [5].

.

4 The Descriptive Complexity Zoo & Applications

The "descriptive complexity zoo" can be characterized best by Figure 2[6]. With the assistance of descriptive complexity theory, we can characterize known results from classical complexity as a statement about the expressibility of the corresponding languages in different logics. We can also characterize open problems in complexity in a similar manner.

For example, the open problem of $\mathsf{NP} \stackrel{?}{=} \mathsf{coNP}$ can be restated as: "Is the set of queries definable in ESO equal to the set of queries definable in USO?" The famous $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question is similarly "Is the set of queries definable in first-order logic with a least fixed point operator equal to the set of queries definable in ESO?" We can also say that first-order queries with transitive closure express the same set of queries that second-order Krom queries do.

Descriptive complexity theory is applicable both in and outside of theoretical

[6]We state some notational differences here: $\mathsf{ESO} := \mathrm{SO}\exists$, $\mathsf{USO} := \mathrm{SO}\forall$, $\mathsf{FO} := \mathrm{FO}$, $\mathsf{SO} := \mathrm{SO}$. Notable acronyms include TC for transitive closure, DTC for transitive closure, -Krom for second-order Krom formulas, and -Horn for second-order Horn formulas. We do not explain these acronyms, as they are not the focus of this work. We direct interested readers to [5].

computer science. According to Neil Immerman, descriptive complexity was a source of inspiration for a proof of the Immerman-Szelepcsényi Theorem, which stated that nondeterministic space complexity classes were closed under complementation ($\mathsf{NL} = \mathsf{coNL}$). Furthermore, descriptive complexity theory also provides extensive mathematical structure to reason through measures of time and space: two resources that are fundamentally important in engineering. Outside of complexity theory, descriptive complexity provides insights into database querying. Relational databases can be thought of as structures, and it is indeed the case that many query languages can be captured using small extensions to first-order logic. All in all, descriptive complexity theory is a beautiful, and practical, field.

References

[1] Complexity zoo, https://complexityzoo.net/Complexity_Zoo, accessed July 23, 2024

[2] Arora, S., Barak, B.: Computational Complexity: A Modern Approach (2007), https://theory.cs.princeton.edu/complexity/book.pdf

[3] Fagin, R.: Generalized first-order spectra, and polynomial. time recognizable sets. SIAM-AMS Proceedings **7**, 43–73 (1974)

[4] Grädel, E., Kolaitis, P., Libkin, L., Marx, M., Spencer, J., Yardi, M., Venema, Y., Weinstein, S.: Finite Model Theory and Its Applications. Texts in Theoretical Computer Science. An EATCS Series, Springer New York, 1 edn. (2007)

[5] Immerman, N.: Descriptive Complexity. Texts in Computer Science, Springer New York (2012)

[6] Sipser, M.: Introduction to the Theory of Computation. Cengage Learning (2012), https://books.google.com/books?id=H94JzgEACAAJ