Descriptive Complexity for Counting Complexity Classes

Marcelo Arenas
Pontificia U. Católica de Chile
marenas@ing.puc.cl

Martin Muñoz
Pontificia U. Católica de Chile
mmunos@uc.cl

Cristian Riveros
Pontificia U. Católica de Chile
cristian.riveros@uc.cl

***Abstract*—Descriptive Complexity has been very successful in characterizing complexity classes of decision problems in terms of the properties definable in some logics. However, descriptive complexity for counting complexity classes, such as FP and #P, has not been systematically studied, and it is not as developed as its decision counterpart. In this paper, we propose a framework based on Weighted Logics to address this issue. Specifically, by focusing on the natural numbers we obtain a logic called Quantitative Second Order Logics (QSO), and show how some of its fragments can be used to capture fundamental counting complexity classes such as FP, #P and FPSPACE, among others. We also use QSO to define a hierarchy inside #P, identifying counting complexity classes with good closure and approximation properties, and which admit natural complete problems. Finally, we add recursion to QSO, and show how this extension naturally captures lower counting complexity classes such as #L.**

I. Introduction

The goal of descriptive complexity is to measure the complexity of a problem in terms of the logical constructors needed to express it [22]. The starting point of this branch of complexity theory is Fagin's theorem [10], which states that NP is equal to existential second-order logic. Since then, many more complexity classes have been characterized in terms of logics (see [16] for a survey) and descriptive complexity has found a variety of applications in different areas [22], [29]. For instance, Fagin's theorem was the key ingredient to define the class MaxSNP [33], which was later shown to be a fundamental class in the study of hardness of approximation [3]. It is important to mention here that the definition of MaxSNP would not have been possible without the machine-independent point of view of descriptive complexity, as pointed out in [33].

Counting problems differ from decision problems in that what has to be computed is the value of a function. More generally, a counting problem corresponds to computing a function f from a set of instances (e.g. graphs, formulae, etc.) to natural numbers.[1] The study of counting problems has given rise to a rich theory of counting complexity classes [2], [13], [18]. Some of these classes are natural counterparts of some classes of decision problems; for example, FP is the class of all functions that can be computed in polynomial time, the natural counterpart of P. However, the existence of computation problems for which little can be said by considering solely their decision counterparts has engendered other function complexity classes. This is the case of the class #P, a counting complexity class introduced in [36] to prove that natural problems like counting the number of satisfying assignments of a propositional formula or the number of perfect matchings of a bipartite graph [36] are difficult, namely, #P-complete. Starting from #P, many more natural counting complexity classes have been defined, such as #L, SpanP and GapP [13], [18].

Although counting problems play a prominent role in computational complexity, descriptive complexity for this type of problem has not been systematically studied and it is not as developed as for the case of decision problems. Insightful characterizations of #P and some of its extensions have been provided [4], [35]. However, these characterizations do not define function problems in terms of a logic, but instead in terms of some counting problems associated to a logic like FO. Thus, it is not clear how these characterizations can be used to provide a general descriptive complexity framework for counting complexity classes like FP and FPSPACE (the class of functions computable in polynomial space).

In this paper, we propose to study the descriptive complexity of counting complexity classes in terms of Weighted Logics (WL) [5], a general logical framework that combines Boolean formulae (e.g. in FO or SO) with operations over a fixed semi-ring (e.g. $\mathbb{N}$). Specifically, we propose a restriction of WL over natural numbers, called Quantitative Second Order Logic (QSO), and study its expressive power for defining counting complexity classes over ordered structures. As a proof of concept, we show that natural syntactical fragments and extensions of QSO captures counting complexity classes like #P, SpanP, FP and FPSPACE. Furthermore, by slightly extending the framework we can prove that QSO can also capture classes like GapP and OptP, showing the robustness of our approach.

The next step is to use the machine-independent point of view of QSO to search for subclasses of #P with some fundamental properties. The question here is, what properties are desirable for a subclass of #P? First, it is desirable to have a class of counting problems whose associated decision versions are tractable, in the sense that one can decide in polynomial time whether the value of the function is greater than 0. In fact, this requirement is crucial in order to find efficient approximation algorithms for a given function (see

[1] This value is usually associated to counting the number of solutions in a search problem, but here we consider a more general definition.

Section V). Second, we expect that the class is closed under basic arithmetical operations like sum, multiplication and subtraction by one. This is a common topic for counting complexity classes; for example, it is known that #P is not closed under subtraction by one (under some complexity-theoretical assumption). Finally, we want a class with natural complete problems, which characterize all problems in it.

In this paper, we give the first results towards defining subclasses of #P that are robust in terms of existence of efficient approximations, having good closure properties, and existence of natural complete problems. Specifically, we introduce a syntactic hierarchy inside #P, called $\Sigma\mathrm{QSO}(\mathrm{FO})$-hierarchy, and we show that it is closely related to the FO-hierarchy introduced in [35]. Looking inside the $\Sigma\mathrm{QSO}(\mathrm{FO})$-hierarchy, we propose the class $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ and show that every function in it has a tractable associated decision version, and it is closed under sum, multiplication, and subtraction by one. Unfortunately, it is not clear whether this class admits a natural complete problem. Thus, we also introduce a Horn-style syntactic class, inspired by [15], that has tractable associated decision versions and a natural complete problem.

After studying the structure of #P, we move beyond QSO by introducing new quantifiers. By adding variables for functions on top of QSO, we introduce a quantitative least fixed point operator to the logic. Adding finite recursion to a numerical setting is subtle since functions over natural numbers can easily diverge without finding any fixed point. By using the support of the functions, we give a natural halting condition that generalizes the least fixed point operator of Boolean logics. Then, with a quantitative recursion at hand we show how to capture FP from a different perspective and, moreover, how to restrict recursion to capture lower complexity classes such as #L, the counting version of NL.

Organisation. The main terminology used in the paper is given in Section II. Then the logical framework is introduced in Section III, and it is used to capture standard counting complexity classes in Section IV. The structure of #P is studied in Section V. Section VI is devoted to define recursion in QSO, and to show how to capture classes below FP. Finally, we give some concluding remarks in Section VII.

II. Preliminaries

A. Second-order logic, LFP and PFP

A relational signature $\mathbf{R}$ (or just signature) is a finite set $\{R_1, \ldots, R_k\}$, where each R_i ($1 \leqslant i \leqslant k$) is a relation name with an associated arity greater than 0, which is denoted by $\mathrm{arity}(R_i)$. A finite structure over $\mathbf{R}$ (or just finite $\mathbf{R}$-structure) is a tuple $\mathfrak{A} = \langle A, R_1^{\mathfrak{A}}, \ldots, R_k^{\mathfrak{A}} \rangle$ such that A is a finite set and $R_i^{\mathfrak{A}} \subseteq A^{\mathrm{arity}(R_i)}$ for every $i \in \{1, \ldots, k\}$. In this paper we only consider finite structures, so we omit the word finite when referring to them. An $\mathbf{R}$-structure $\mathfrak{A}$ is said to be ordered if $<$ is a binary predicate name in $\mathbf{R}$ and $<^{\mathfrak{A}}$ is a linear order on A. Let $\textsc{Struct}[\mathbf{R}]$ be the class of all $\mathbf{R}$-structures and $\textsc{OrdStruct}[\mathbf{R}]$ be the class of all ordered $\mathbf{R}$-structures.

From now on, assume given disjoint infinite sets $\mathbf{FV}$ and $\mathbf{SV}$ of first-order variables and second-order variables, respectively. Notice that every variable in $\mathbf{SV}$ has an associated arity, which is denoted by $\mathrm{arity}(X)$. Then given a signature $\mathbf{R}$, the set of second-order logic formulae (SO-formulae) over $\mathbf{R}$ is given by the following grammar:

$$\varphi \;:=\; x = y \;\mid\; R(\bar{u}) \;\mid\; \top \;\mid\; X(\bar{v}) \;\mid\; \neg\varphi \;\mid\; (\varphi \vee \varphi) \;\mid\; \exists x\, \varphi \;\mid\; \exists X\, \varphi$$

where $x, y \in \mathbf{FV}$, $R \in \mathbf{R}$, $\bar{u}$ is a tuple of (not necessarily distinct) variables from $\mathbf{FV}$ whose length is $\mathrm{arity}(R)$, $\top$ is a reserved symbol to represent a tautology, $X \in \mathbf{SV}$, $\bar{v}$ is a tuple of (not necessarily distinct) variables from $\mathbf{FV}$ whose length is $\mathrm{arity}(X)$, and $x \in \mathbf{FV}$.

We assume that the reader is familiar with the semantics of SO, so we only introduce here some notation that will be used in this paper. Given a signature $\mathbf{R}$ and an $\mathbf{R}$-structure $\mathfrak{A}$ with domain A, a first-order assignment v for $\mathfrak{A}$ is a total function from $\mathbf{FV}$ to A, while a second-order assignment V for $\mathfrak{A}$ is a total function with domain $\mathbf{SV}$ that maps each $X \in \mathbf{SV}$ to a subset of $A^{\mathrm{arity}(X)}$. Moreover, given a first-order assignment v for $\mathfrak{A}$, $x \in \mathbf{FV}$ and $a \in A$, we denote by $v[a/x]$ a first-order assignment such that $v[a/x](x) = a$ and $v[a/x](y) = v(y)$ for every $y \in \mathbf{FV}$ distinct from x. Similarly, given a second-order assignment V for $\mathfrak{A}$, $X \in \mathbf{SV}$ and $B \subseteq A^{\mathrm{arity}(X)}$, we denote by $V[B/X]$ a second-order assignment such that $V[B/X](X) = B$ and $V[B/X](Y) = V(Y)$ for every $Y \in \mathbf{SV}$ distinct from X. We use notation $(\mathfrak{A}, v, V) \models \varphi$ to indicate that structure $\mathfrak{A}$ satisfies φ under v and V. In particular, we have that $(\mathfrak{A}, v, V) \models \top$.

In this paper, we consider several fragments and extensions of SO, in particular first-order logic (FO), least fixed point logic (LFP) and partial fixed point logic (PFP) [29]. Moreover, for every $i \in \mathbb{N}$, we consider the fragment Σ_i (resp., Π_i) of FO, which is the set of FO-formulae of the form $\exists \bar{x}_1 \forall \bar{x}_2 \cdots \exists \bar{x}_{i-1} \forall \bar{x}_i\, \psi$ (resp., $\forall \bar{x}_1 \exists \bar{x}_2 \cdots \forall \bar{x}_{i-1} \exists \bar{x}_i\, \psi$) if i is even, and of the form $\exists \bar{x}_1 \forall \bar{x}_2 \cdots \forall \bar{x}_{i-1} \exists \bar{x}_i\, \psi$ (resp., $\forall \bar{x}_1 \exists \bar{x}_2 \cdots \exists \bar{x}_{i-1} \forall \bar{x}_i\, \psi$) if i is odd, where ψ is a quantifier-free formula. Finally, we say that a fragment $\mathcal{L}_1$ is contained in a fragment $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \subseteq \mathcal{L}_2$, if for every formula φ in $\mathcal{L}_1$, there exists a formula ψ in $\mathcal{L}_2$ such that φ is logically equivalent to ψ. Besides, we say that $\mathcal{L}_1$ is properly contained in $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \subsetneq \mathcal{L}_2$, if $\mathcal{L}_1 \subseteq \mathcal{L}_2$ and $\mathcal{L}_2 \nsubseteq \mathcal{L}_1$.

B. Counting complexity classes

We consider the following counting complexity classes in this paper. FP is the class of functions $f : \Sigma^* \to \mathbb{N}$ computable in polynomial time, while FPSPACE is the class of functions $f : \Sigma^* \to \mathbb{N}$ computable in polynomial space. Given a nondeterministic Turing Machine (NTM) M, let $\#\mathrm{accept}_M(x)$ be the number of accepting runs of M with input x. Then #P is the class of functions f for which there exists a polynomial-time NTM M such that $f(x) = \#\mathrm{accept}_M(x)$ for every input x, while #L is the class of functions f for which there exists a logarithmic-space NTM M such that $f(x) = \#\mathrm{accept}_M(x)$

for every input x. Given an NTM M with output tape, let $\#\text{output}_M(x)$ be the number of distinct outputs of M with input x (notice that M produces an output if it halts in an accepting state). Then SPANP is the class of functions f for which there exists a polynomial-time NTM M such that $f(x) = \#\text{output}_M(x)$ for every input x. Notice that $\#\text{P} \subseteq \text{SPANP}$, and this inclusion is believed to be strict.

III. A LOGIC FOR QUANTITATIVE FUNCTIONS

We introduce here the logical framework that we use for studying counting complexity classes. This framework is based on the framework of Weighted Logics (WL) [5] that has been used in the context of weighted automata for studying functions from words (or trees) to semirings. We propose here to use the framework of WL over any relational structure and to restrict the semiring to natural numbers. The extension to any relational structure will allow us to study general counting complexity classes and the restriction to the natural numbers will simplify the notation in this context (see Section III-A for a more detailed discussion).

Given a relational signature $\mathbf{R}$, the set of Quantitative Second-Order logic formulae (or just QSO-formulae) over $\mathbf{R}$ is given by the following grammar:

$$\alpha \;:=\; \varphi \;\mid\; s \;\mid\; (\alpha + \alpha) \;\mid\; (\alpha \cdot \alpha) \;\mid\; \Sigma x.\,\alpha \;\mid\; \Pi x.\,\alpha \;\mid\; \Sigma X.\,\alpha \;\mid\; \Pi X.\,\alpha \quad (1)$$

where φ is an SO-formula over $\mathbf{R}$, $s \in \mathbb{N}$, $x \in \mathbf{FV}$ and $X \in \mathbf{SV}$. Moreover, if $\mathbf{R}$ is not mentioned, then QSO refers to the set of QSO formulae over all possible relational signatures.

Note that the syntax of QSO formulae is divided in two levels. The first level is composed by SO-formulae over $\mathbf{R}$ (called Boolean formulae) and the second level is made by counting operators of addition and multiplication. For this reason, the quantifiers in SO (e.g. $\exists x$ or $\exists X$) are called Boolean quantifiers and the quantifiers that make use of addition and multiplication (e.g. Σx or ΠX) are called *quantitative quantifiers*. Furthermore, Σx and ΣX are called first- and second-order sum, and Πx and ΠX are called first- and second-order product, respectively. This division between Boolean and quantitative level is essential for understanding the difference between the logic and the quantitative part. Furthermore, this will allow us later to parametrize both levels of the logic in order to capture different counting complexity classes.

Let $\mathbf{R}$ be a signature, $\mathfrak{A}$ an $\mathbf{R}$-structure with domain A, v a first-order assignment for $\mathfrak{A}$ and V a second-order assignment for $\mathfrak{A}$. Then the *evaluation* of a QSO-formula α over $(\mathfrak{A}, v, V)$ is defined as a function $[\![\alpha]\!]$ that on input $(\mathfrak{A}, v, V)$ returns a number in $\mathbb{N}$. Formally, the function $[\![\alpha]\!]$ is recursively defined in Table I. A QSO-formula α is said to be a *sentence* if it does not have any free variable, that is, every variable in α is under the scope of a usual quantifier or a quantitative quantifier. It is important to notice that if α is a QSO-sentence over a signature $\mathbf{R}$, then for every $\mathbf{R}$-structure $\mathfrak{A}$, first-order assignments v_1, v_2 for $\mathfrak{A}$ and second-order assignments V_1, V_2 for $\mathfrak{A}$, it holds that $[\![\alpha]\!](\mathfrak{A}, v_1, V_1) = [\![\alpha]\!](\mathfrak{A}, v_2, V_2)$. Thus, in such a case we use the term $[\![\alpha]\!](\mathfrak{A})$ to denote $[\![\alpha]\!](\mathfrak{A}, v, V)$, for some arbitrary first-order assignment v for $\mathfrak{A}$ and some arbitrary second-order assignment V for $\mathfrak{A}$.

$$
\begin{aligned}
[\![\varphi]\!](\mathfrak{A}, v, V) &= \begin{cases} 1 & \text{if } (\mathfrak{A}, v, V) \models \varphi \\ 0 & \text{otherwise} \end{cases} \\
[\![s]\!](\mathfrak{A}, v, V) &= s \\
[\![\alpha_1 + \alpha_2]\!](\mathfrak{A}, v, V) &= [\![\alpha_1]\!](\mathfrak{A}, v, V) + [\![\alpha_2]\!](\mathfrak{A}, v, V) \\
[\![\alpha_1 \cdot \alpha_2]\!](\mathfrak{A}, v, V) &= [\![\alpha_1]\!](\mathfrak{A}, v, V) \cdot [\![\alpha_2]\!](\mathfrak{A}, v, V) \\
[\![\Sigma x.\,\alpha]\!](\mathfrak{A}, v, V) &= \sum_{a \in A} [\![\alpha]\!](\mathfrak{A}, v[a/x], V) \\
[\![\Pi x.\,\alpha]\!](\mathfrak{A}, v, V) &= \prod_{a \in A} [\![\alpha]\!](\mathfrak{A}, v[a/x], V) \\
[\![\Sigma X.\,\alpha]\!](\mathfrak{A}, v, V) &= \sum_{B \subseteq A^{\text{arity}(X)}} [\![\alpha]\!](\mathfrak{A}, v, V[B/X]) \\
[\![\Pi X.\,\alpha]\!](\mathfrak{A}, v, V) &= \prod_{B \subseteq A^{\text{arity}(X)}} [\![\alpha]\!](\mathfrak{A}, v, V[B/X])
\end{aligned}
$$

Table I
THE SEMANTICS OF QSO FORMULAE.

Example III.1. *Let $\mathbf{G} = \{E(\cdot,\cdot), <\}$ be the vocabulary for graphs and $\mathfrak{G}$ be an ordered $\mathbf{G}$-structure encoding an undirected graph. Suppose that we want to count the number of triangles in $\mathfrak{G}$. Then this can be defined as follows:*

$$\alpha_1 \;:=\; \Sigma x.\,\Sigma y.\,\Sigma z.\,(E(x,y) \wedge E(y,z) \wedge E(z,x) \wedge x < y \wedge y < z)$$

We encode a triangle in α_1 as an increasing sequence of nodes $\{x, y, z\}$, in order to count each triangle once. Then the Boolean subformula $E(x,y) \wedge E(y,z) \wedge E(z,x) \wedge x < y \wedge y < z$ is checking the triangle property, by returning 1 if $\{x, y, z\}$ forms a triangle in $\mathfrak{G}$ and 0 otherwise. Finally, the sum quantifiers in α_1 aggregates all the values, counting the number of triangles in $\mathfrak{G}$.

Suppose now that we want to count the number of cliques in $\mathfrak{G}$. We can define this function with the following formula:

$$\alpha_2 \;:=\; \Sigma X.\ \text{clique}(X),$$

where $\text{clique}(X) := \forall x \forall y((X(x) \wedge X(y) \wedge x \neq y) \rightarrow E(x,y))$. In the Boolean sub-formula of α_2 we check whether X is a clique, and with the sum quantifier we add one for each clique in $\mathfrak{G}$. But in contrast to α_1, in α_2 we need a second-order quantifier in the quantitative level. This is according to the complexity of evaluating each formula: α_1 defines an FP*-function while α_2 defines a* #P*-complete function.*

Example III.2. *For a more involved example that includes multiplication, let $\mathbf{M} = \{M(\cdot,\cdot), <\}$ be a vocabulary for storing 0-1 matrices; in particular, a structure $\mathfrak{M}$ over $\mathbf{M}$ encodes a 0-1 matrix A as follows: if $A[i,j] = 1$, then $M(i,j)$ is true, otherwise $M(i.j)$ is false. Suppose now that we want to compute the permanent of an n-by-n 0-1 matrix A, that is:*

$$\text{perm}(A) \;=\; \sum_{\sigma \in S_n} \prod_{i=1}^{n} A[i, \sigma(i)],$$

where S_n is the set of all permutations over $\{1, \ldots, n\}$. The permanent is a fundamental function on matrices that has found many applications; in fact, showing that this function is hard to compute was one of the main motivations behind the definition of the counting class #P *[36].*

To define the permanent of a 0-1 matrix in QSO*, assume that for a binary relation symbol S,* $\mathrm{permut}(S)$ *is an* FO*-formula that is true if and only if S is a permutation, that is, a total bijective function (the definition of* $\mathrm{permut}(S)$ *is straightforward). Then the following is a* QSO*-formula defining the permanent of a matrix:*

$$\alpha_3 := \Sigma S.\ \mathrm{permut}(S) \cdot \Pi x.\, (\exists y.\ S(x,y) \wedge M(x,y)).$$

Intuitively, the subformula $\beta(S) := \Pi x.\, (\exists y.\ S(x,y) \wedge M(x,y))$ calculates the value $\prod_{i=1}^{n} A[i, \sigma(i)]$ whenever S encodes a permutation σ. Moreover, the subformula $\mathrm{permut}(S) \cdot \beta(S)$ *returns $\beta(S)$ when S is a permutation, and returns* 0 *otherwise (i.e.* $\mathrm{permut}(S)$ *behaves like a filter). Finally, the second order sum aggregates these values iterating over all binary relations and calculating the permanent of the matrix. We would like to finish with this example by highlighting the similarity of α_3 with the permanent formula. Indeed, an advantage of* QSO*-formulae is that the first- and second-order quantifiers in the quantitative level naturally reflect the operations used to define mathematical formulae.*

We consider several fragments of QSO, which are obtained by restricting the syntax of the Boolean formulae or the use of the quantitative quantifiers. In this direction, we denote by QFO the fragment of QSO where second-order sum and product are not allowed. For instance, for the QSO-formulae defined in Example III.1, we have that α_1 is in QFO and α_2 is not. Another interesting fragment of QSO consists of the QSO-formulae where only sum operators and quantifiers are allowed. Formally, we denote by ΣQSO the fragment of QSO where first- and second-order products (i.e. $\Pi x.$ and $\Pi X.$) are not allowed. For example, α_1 and α_2 in Example III.1 are formulae of ΣQSO, while α_3 in Example III.2 is not. We also consider fragments of QSO by further restricting the Boolean part of the logic. If $\mathscr{L}$ is a fragment of SO, then we define the quantitative logic $\mathrm{QSO}(\mathscr{L})$ to be the fragment of QSO obtained by restricting φ in (1) to be a formula in $\mathscr{L}$. Moreover, we also consider other fragments of QSO by using the same idea. For example, we define $\mathrm{QFO}(\mathrm{FO})$ to be the fragment of QFO obtained by restricting φ in (1) to be an FO-formula, and likewise for $\Sigma\mathrm{QSO}(\mathrm{FO})$.

In the following section, we use different fragments of QSO to capture counting complexity classes. But before doing this, we show the connection of QSO with previous frameworks for defining functions over relational structures.

A. Previous frameworks for quantitative functions

In this section, we discuss some previous frameworks proposed in the literature and how they differ from our approach. We start by discussing the connection between QSO and weighted logics (WL) [5]. As it was previously discussed, QSO is a fragment of WL. The main difference is that we restrict the semiring used in WL to natural numbers in order to study counting complexity classes. Another difference of WL with our approach is that, to the best of our knowledge, this is the first paper to study weighted logics over general relational signatures, in order to do descriptive complexity for counting complexity classes. Previous works on WL usually restrict the signature of the logic to strings, trees, and other specific structures (see [6] for more examples), and they did not study the logic over general structures. Furthermore, in this paper we propose further extensions for QSO (see Section VI) which differ from previous approaches in WL.

Another approach that resembles QSO are logics with counting [9], [17], [23], [29], which include operators that extend FO with quantifiers that allow to count in how many ways a formula is satisfied (the result of this counting is a value of a second sort, in this case the natural numbers). In contrast to our approach, counting operators are usually used for checking Boolean properties over structures and not for producing values (i.e. they do not define a function). In particular, we are not aware of any paper that uses this approach for capturing counting complexity classes.

Finally, the work in [35] and [4] is of particular interest for our research. In [35], it was proposed to define a function over a structure by using free variables in an SO-formula; in particular, the function is defined by the number of instantiations of the free variables that are satisfied by the structure. Formally, Saluja et. al [35] define a family of counting classes $\#\mathscr{L}$ for a fragment $\mathscr{L}$ of FO. For a formula $\varphi(\bar{x}, \bar{X})$ over $\mathbf{R}$, the function $f_{\varphi(\bar{x},\bar{X})}$ is defined as $f_{\varphi(\bar{x},\bar{X})}(\mathfrak{A}) = |\{(\bar{a}, \bar{A}) \mid \mathfrak{A} \models \varphi(\bar{a}, \bar{A})\}|$, for every $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$. Then a function $g : \textsc{OrdStruct}[\mathbf{R}] \to \mathbb{N}$ is in $\#\mathscr{L}$ if there exists a formula $\varphi(\bar{x}, \bar{X})$ in $\mathscr{L}$ such that $g = f_{\varphi(\bar{x},\bar{X})}$. In [35], several results were proved about capturing counting complexity classes which are relevant for our work. We discuss and use these results in Sections IV and V. Notice that for every formula $\varphi(\bar{x}, \bar{X})$, it holds that $f_{\varphi(\bar{x},\bar{X})}$ is the same function as $[\![\Sigma \bar{X}.\, \Sigma \bar{x}.\, \varphi(\bar{x}, \bar{X})]\!]$, that is, the approach in [35] can be seen as a syntactical restriction of our approach based on QSO. Thus, the advantage of our approach relies on the flexibility to define functions by alternating sum with product operators and, moreover, by introducing new quantitative operators (see Section VI). Furthermore, we show in the following section how to capture some fundamental classes that cannot be captured by following the approach in [35].

IV. Counting under QSO

In this section, we show that by syntactically restricting QSO one can capture different counting complexity classes. In other words, by using QSO we can extend the theory of descriptive complexity [22] from decision problems to computation problems. For this, we first formalize the notion of *capturing* a complexity class of functions.

Fix a signature $\mathbf{R} = \{R_1, \ldots, R_k\}$ and assume that $\mathfrak{A}$ is an ordered (finite) $\mathbf{R}$-structure with a domain $A = \{a_1, \ldots, a_n\}$.

Recall that $<$ is a linear order on A, say $a_1 < a_2 < \ldots < a_n$. For every $i \in \{1, \ldots, k\}$, define the encoding of $R_i^{\mathfrak{A}}$, denoted by $\mathrm{enc}(R_i^{\mathfrak{A}})$, as the following binary string. Assume that $\ell = \mathrm{arity}(R_i)$ and consider an enumeration of the ℓ-tuples over A in the lexicographic order induced by $<$. Then let $\mathrm{enc}(R_i^{\mathfrak{A}})$ be a binary string of length n^ℓ such that the i-th bit of $\mathrm{enc}(R_i^{\mathfrak{A}})$ is 1 if the i-th tuple in the enumeration belongs to $R_i^{\mathfrak{A}}$, and 0 otherwise. Moreover, define the encoding of $\mathfrak{A}$, denoted by $\mathrm{enc}(\mathfrak{A})$, as the string $0^n\,1\,\mathrm{enc}(R_1^{\mathfrak{A}}) \cdots \mathrm{enc}(R_k^{\mathfrak{A}})$ [29]. We can now formalize the notion of capturing a counting complexity class.

Definition IV.1. *Let $\mathscr{F}$ be a fragment of* QSO *and $\mathscr{C}$ a counting complexity class. Then $\mathscr{F}$* captures *$\mathscr{C}$* over ordered $\mathbf{R}$-structures *if the following conditions hold:*

1) *for every $\alpha \in \mathscr{F}$, there exists $f \in \mathscr{C}$ such that $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$ for every $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$.*
2) *for every $f \in \mathscr{C}$, there exists $\alpha \in \mathscr{F}$ such that $f(\mathrm{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$ for every $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$.*

Moreover, $\mathscr{F}$ captures *$\mathscr{C}$* over ordered structures *if $\mathscr{F}$ captures $\mathscr{C}$ over ordered $\mathbf{R}$-structures for every signature $\mathbf{R}$.*

In Definition IV.1, function $f \in \mathscr{C}$ and formula $\alpha \in \mathscr{F}$ must coincide in all the strings that encode ordered $\mathbf{R}$-structures. Notice that this restriction is natural as we want to capture $\mathscr{C}$ over a fixed set of structures (e.g. graphs, matrices). Moreover, this restriction is fairly standard in descriptive complexity [22], [29], and it has also been used in the previous work on capturing complexity classes of functions [4], [35].

What counting complexity classes can be captured with fragments of QSO? For answering this question, it is reasonable to start with #P, a well-known and widely-studied counting complexity class [2]. Since #P has a strong similarity with NP, one could expect a "Fagin-like" Theorem [10] for this class. Actually, in [35] it was shown that the class $\#\mathrm{FO}$ captures #P. In our setting, the class $\#\mathrm{FO}$ is contained in $\Sigma\mathrm{QSO}(\mathrm{FO})$, which also captures #P as expected.

Proposition IV.2. $\Sigma\mathrm{QSO}(\mathrm{FO})$ *captures* #P *over ordered structures.*

Recall that every function class $\#\mathscr{L}$ is contained in $\Sigma\mathrm{QSO}(\mathscr{L})$, for some fragment $\mathscr{L}$ of FO. (see Section III-A). Thus, it directly follows from [35] that every #P-function can be defined in $\Sigma\mathrm{QSO}(\mathrm{FO})$. The other direction of Proposition IV.2 follows by the fact that #P is closed under first- and second-order sum.

By following the same approach as [35], Compton and Grädel [4] show that #($\exists$SO) captures SpanP, where $\exists$SO is the existential fragment of SO. As one could expect, if we parametrize ΣQSO with $\exists$SO, we can also capture SpanP.

Proposition IV.3. $\Sigma\mathrm{QSO}(\exists\mathrm{SO})$ *captures* SpanP *over ordered structures.*

Can we capture FP by using $\#\mathscr{L}$ for some fragment $\mathscr{L}$ of SO? A first attempt could be based on considering a fragment $\mathscr{L}$ of FO. But even if we consider the existential fragment Σ_1 of FO the approach fails, as $\#\Sigma_1$ can encode #P-complete problems like counting the number of satisfying assignments of a 3-DNF propositional formula [35]. A second attempt could be based on disallowing the use of second-order free variables in $\#\mathrm{FO}$. But in this case one cannot capture exponential functions definable in FP such as 2^n. Thus, it is not clear how to capture FP by following the approach proposed in [35]. On the other hand, if we consider our framework and move out from ΣQSO, we have other options for counting like first- and second-order products. In fact, the combination of QFO with LFP is exactly what we need to capture FP.

Theorem IV.4. $\mathrm{QFO}(\mathrm{LFP})$ *captures* FP *over ordered structures.*

To prove this theorem, one first shows that every formula in $\mathrm{QFO}(\mathrm{LFP})$ can be evaluated in polynomial time. Indeed, LFP is a polynomial-time logic [20], [38], and the sum and product quantifiers can also be computed in polynomial time. For the other direction, one has to use $\mathrm{QFO}(\mathrm{LFP})$ to simulate the run of a polynomial time TM M computing a function, in particular using the quantitative quantifiers to reconstruct the natural number returned by M in the output tape. It is important to notice that the alternation between sum and product quantifiers is used for this reconstruction.

At this point it is natural to ask whether one can extend the previous idea to capture FPSPACE [28], the class of functions computable in polynomial space. For capturing this class one can use a logical core powerful enough, like PFP, for simulating the run of a polynomial-space TM. Moreover, one also needs more powerful quantitative quantifiers as functions like 2^{2^n} can be computed in polynomial space, so second-order sum is not enough for the quantitative layer of a logic for FPSPACE. In fact, by considering second-order product we obtain the fragment $\mathrm{QSO}(\mathrm{PFP})$ that captures FPSPACE.

Theorem IV.5. $\mathrm{QSO}(\mathrm{PFP})$ *captures* FPSPACE *over ordered structures.*

The proof of the previous theorem follows the same line as for the logical characterization of FP: one shows that each function in $\mathrm{QSO}(\mathrm{PFP})$ can be computed in FPSPACE and, conversely, the output of a polynomial-space TM can be reconstructed by using PFP and quantitative quantifiers.

From the proof of the previous theorem a natural question follows: what happens if we use first-order quantitative quantifiers and PFP? In [28], Ladner also introduced the class FPSPACE(poly) of all functions computed by polynomial-space TMs with output length bounded by a polynomial. Interestingly, if we restrict to FO-quantitative quantifiers we can also capture this class.

Corollary IV.6. $\mathrm{QFO}(\mathrm{PFP})$ *captures* FPSPACE(poly) *over ordered structures.*

The results of this section validate QSO as an appropriate logical framework for extending the theory of descriptive complexity to counting complexity classes. In the following sections, we provide more arguments for this claim, by con-

sidering some fragments of ΣQSO and, moreover, by showing how to go beyond ΣQSO to capture other classes.

A. Extending QSO to capture non-counting classes

There exist complexity classes that do not fit in our framework because either the output of a function is not a natural number (e.g. a negative number) or the class is not defined purely in terms of arithmetical operations (e.g. min and max are used). To remedy this problem, we show here how QSO can be easily extended to capture such classes that go beyond sum and product over natural numbers.

It is well-known that, under some reasonable complexity-theoretical assumptions, #P is not closed under subtraction, not even under subtraction by one [30]. To overcome this limitation, GAPP was introduced in [12] as the class of functions f for which there exists a polynomial-time NTM M such that $f(x) = \#\text{accept}_M(x) - \#\text{reject}_M(x)$, where $\#\text{reject}_M(x)$ is the number of rejecting runs of M with input x. That is, GAPP is the closure of #P functions under subtraction, and its functions can obviously take negative values. Given that our logical framework was built on top of the natural numbers, we need to extend QSO in order to capture GAPP. The most elegant way to do this is by allowing constants coming from $\mathbb{Z}$ instead of just $\mathbb{N}$. Formally, we define the logic $\text{QSO}_{\mathbb{Z}}$ whose syntax is the same as in (1) and whose semantics is the same as in Table I except that the atomic formula s (i.e. a constant) comes from $\mathbb{Z}$. Just as we did for QSO, we define the fragment $\Sigma\text{QSO}_{\mathbb{Z}}$ as the extension of ΣQSO with constants in $\mathbb{Z}$.

Example IV.7. *Recall the setting of Example III.1 and suppose now that we want to compute the number of cliques in a graph that are not triangles. This can be easily done in* $\text{QSO}_{\mathbb{Z}}$ *with the formula:* $\alpha_5 := \alpha_2 + (-1) \cdot \alpha_1$.

Adding negative constants is a mild extension to allow subtraction in the logic. It follows from our characterization of #P that this is exactly what we need to capture GAPP.

Corollary IV.8. $\Sigma\text{QSO}_{\mathbb{Z}}(\text{FO})$ *captures* GAPP *over ordered structures.*

This result shows how robust QSO is when capturing different complexity classes.

A different class of functions comes from considering the optimization version of a decision problem. For example, one can define MAX-SAT as the problem of determining the maximum number of clauses, of a given CNF propositional formula, that can be made true by an assignment. Here, MAX-SAT is defined in terms of a maximization problem which in its essence differs from the functions in #P. To formalize this set of optimization problems, Krentel defined OPTP [27] as the class of functions computable by taking the maximum or minimum of the output values over all runs of a polynomial-time NTM machine with output tape (i.e. each run produces a binary string which is interpreted as a number). For instance, MAX-SAT is in OPTP as many other optimization versions of NP-problems are. Given that in [27] the author does not make the distinction between $\max$ and $\min$, in [39] the authors define the classes MAXP and MINP as the max and min version of the problems in OPTP (i.e. $\text{OPTP} = \text{MAXP} \cup \text{MINP}$).

In order to capture classes of optimization functions, we extend as follows QSO with $\max$ and $\min$ quantifiers (called OptQSO). Given a signature $\mathbf{R}$, the set of OptQSO-formulae over $\mathbf{R}$ is given by extending the syntax in (1) with the following operators:

$$\max\{\alpha, \alpha\} \mid \min\{\alpha, \alpha\} \mid \text{Max}\, x.\, \alpha \mid \text{Min}\, x.\, \alpha \mid \text{Max}\, X.\, \alpha \mid \text{Min}\, X.\, \alpha$$

where $x \in \mathbf{FV}$ and $X \in \mathbf{SV}$. The semantics of the QSO-operators in OptQSO are defined as usual. Furthermore, the semantics of the max and min quantifiers are defined as the natural extension of the sum quantifiers in QSO (see Table I) by maximizing or minimizing, respectively, instead of computing a sum.

Example IV.9. *Recall again the setting of Example III.1 and suppose now that we want to compute the size of the largest clique in a graph. This can be done in* OptQSO *as follows:*

$$\alpha_6 := \text{Max}\, X.\, (\, \text{clique}(X) \cdot \Sigma z.\, X(z)\,)$$

Notice that formula $\Sigma z.\, X(z)$ *is used to compute the number of nodes in a set* X.

Similar than for MAXP and MINP, we have to distinguish between the $\max$ and $\min$ fragments of OptQSO. For this, we define the fragment MaxQSO of all OptQSO formulae constructed from QFO operators and $\max$-formulae $\max\{\alpha, \alpha\}$, $\text{Max}\, x.\, \alpha$ and $\text{Max}\, X.\, \alpha$. The class MinQSO is defined analogously changing $\max$ with $\min$. Notice that in MaxQSO and MinQSO, second-order sum and product are not allowed. For instance, formula α_6 in Example IV.9 is in MaxQSO. As one could expect, MaxQSO and MinQSO are the logics needed to capture MAXP and MINP.

Theorem IV.10. $\text{MaxQSO}(\text{FO})$ *and* $\text{MinQSO}(\text{FO})$ *capture* MAXP *and* MINP*, respectively, over ordered structures.*

It is important to mention that a similar result was proved in [26] for the class MAXPB (resp., MINPB) of problems in MAXP (resp., MINP) whose output value is polynomially bounded. Interestingly, Theorem IV.10 is stronger since our logic has the freedom to use sum and product quantifiers, instead of using a max-and-count problem over Boolean formulae. Finally, it is easy to prove that our framework can also capture MAXPB and MINPB by disallowing the product Πx in $\text{MaxQSO}(\text{FO})$ and $\text{MinQSO}(\text{FO})$, respectively.

V. EXPLORING THE STRUCTURE OF #P THROUGH QSO

The class #P was introduced in [36] to prove that computing the permanent of a matrix, as defined in Example III.2, is a #P-complete problem. As a consequence of this result many counting problems have been proved to be #P-complete [2], [37]. Among them, problems having easy decision counterparts play a fundamental role, as a counting problem with a

hard decision version is expected to be hard. Formally, the decision problem associated to a function $f : \Sigma^* \to \mathbb{N}$ is defined as $L_f = \{x \in \Sigma^* \mid f(x) > 0\}$, and f is said to have an easy decision version if $L_f \in$ P. Many prominent examples satisfy this property, like computing the number of: perfect matchings of a bipartite graph (#PERFECTMATCHING) [36], satisfying assignments of a DNF (#DNF) [7], [25] or Horn (#HORNSAT) [37] propositional formula, among others.

Counting problems with easy decision versions play a fundamental role in the search of efficient approximation algorithms for functions in #P. A fully-polynomial randomized approximation scheme (FPRAS) for a function $f : \Sigma^* \to \mathbb{N}$ is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0,1) \to \mathbb{N}$ such that: (1) for every string $x \in \Sigma^*$ and real value $\varepsilon \in (0,1)$, the probability that $|f(x) - \mathcal{A}(x,\varepsilon)| \leqslant \varepsilon \cdot f(x)$ is at least $\frac{3}{4}$, and (2) the running time of $\mathcal{A}$ is polynomial in the size of x and $1/\varepsilon$ [25]. Notably, there exist #P-complete functions that can be efficiently approximated as they admit FPRAS; for instance, there exist FPRAS for #DNF [25] and #PERFECTMATCHING [24]. A key observation here is that if a function f admits an FPRAS, then L_f is in the randomized complexity class BPP [14]. Hence, under the widely believed assumption that NP $\not\subseteq$ BPP, we cannot hope for an FPRAS for a function in #P whose decision counterpart is NP-complete, and we have to concentrate on the class of counting problems with easy decision versions.

The importance of the class of counting problems with easy decision counterparts has motivated the search of robust classes of functions in #P with this property [31]. But the key question here is what should be considered a *robust* class. A first desirable condition has to do with the closure properties satisfied by the class, which is a common theme when studying function complexity classes [11], [30]. As in the cases of P and NP that are closed under intersection and union, we expect our class to be closed under multiplication and sum. For a more elaborated closure property, assume that *sat_one* is a function that returns one plus the number of satisfying assignments of a propositional formula. Clearly *sat_one* is a #P-complete function whose decision counterpart L_{sat_one} is trivial. But should *sat_one* be part of a robust class of counting functions with easy decision versions? The key insight here is that if a function in #P has an easy decision counterpart L, then as $L \in$ NP we expect to have a polynomial-time algorithm that verifies whether $x \in L$ by constructing witnesses for x. Moreover, if such an algorithm for constructing witnesses exists, then we also expect to be able to manipulate such witnesses and in some cases to remove them. In other words, we expect a robust class $\mathscr{C}$ of counting functions with easy decision versions to be closed under subtraction by one, that is, if $g \in \mathscr{C}$, then the function $g \dot{-} 1$ should also be in $\mathscr{C}$, where $(g \dot{-} 1)(x)$ is defined as $g(x) - 1$ if $g(x) \geqslant 1$, and as 0 otherwise. Notice that, unless P = NP, no such class can contain the function *sat_one* because *sat_one* $\dot{-} 1$ counts the number of satisfying assignments of a propositional formula.

A second desirable condition of robustness is the existence of natural complete problems [32]. Special attention has to be paid here to the notion of reduction used for completeness. Notice that under the notion of Cook reduction, originally used in [36], the problems #DNF and #SAT are #P-complete. However, #DNF has an easy decision counterpart and admits an FPRAS, while #SAT does not satisfy these conditions unless P = NP. Hence a more strict notion of reduction has to be considered; in particular, the notion of parsimonious reduction (to be defined later) satisfies that if a function f is parsimoniously reducible to a function g, then $L_g \in$ P implies that $L_f \in$ P and the existence of an FPRAS for g implies the existence of a FPRAS for f.

In this section, we use the framework developed in this paper to address the problem of defining a robust class of functions with easy decision versions. More specifically, we use the framework to introduce in Section V-A a syntactic hierarchy of counting complexity classes contained in #P. Then this hierarchy is used in Section V-B to define a class of functions with easy decision versions and good closure properties, and in Section V-C to define a class of functions with easy decision versions and natural complete problems.

A. The ΣQSO(FO)-hierarchy inside #P

Inspired by the connection between #P and $\#\mathrm{FO}$, a hierarchy of subclases of $\#\mathrm{FO}$ was introduced in [35] by restricting the alternation of quantifiers in Boolean formulae. Specifically, the $\#\mathrm{FO}$-*hierarchy* consists of the the classes $\#\Sigma_i$ and $\#\Pi_i$ for every $i \geqslant 0$, where $\#\Sigma_i$ (resp., $\#\Pi_i$) is defined as $\#\mathrm{FO}$ but restricting the formulae used to be in Σ_i (resp., Π_i). By definition, we have that $\#\Pi_0 = \#\Sigma_0$. Moreover, it is shown in [35] that:

$$\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2 = \#\mathrm{FO}$$

In light of the framework introduced in this paper, natural extensions of these classes are obtained by considering $\Sigma\mathrm{QSO}(\Sigma_i)$ and $\Sigma\mathrm{QSO}(\Pi_i)$ for every $i \geqslant 0$, which form the $\Sigma\mathrm{QSO}(\mathrm{FO})$-*hierarchy*. Clearly, we have that $\#\Sigma_i \subseteq \Sigma\mathrm{QSO}(\Sigma_i)$ and $\#\Pi_i \subseteq \Sigma\mathrm{QSO}(\Pi_i)$. Indeed, each formula $\varphi(\bar{X}, \bar{x})$ in $\#\Sigma_i$ is equivalent to the formula $\Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi(\bar{X}, \bar{x})$ in $\Sigma\mathrm{QSO}(\Sigma_i)$, and likewise for $\#\Pi_i$ and $\Sigma\mathrm{QSO}(\Pi_i)$. But what is the exact relationship between these two hierarchies? To answer this question, we first introduce two normal forms for $\Sigma\mathrm{QSO}(\mathscr{L})$ that helps us to characterize the expressive power of this quantitative logic. A formula α in $\Sigma\mathrm{QSO}(\mathscr{L})$ is in $\mathscr{L}$-*prenex normal form* ($\mathscr{L}$-*PNF*) if α is of the form $\Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi(\bar{X}, \bar{x})$, where $\bar{X}$ and $\bar{x}$ are sequences of zero or more second-order and first-order variables, respectively, and $\varphi(\bar{X}, \bar{x})$ is a formula in $\mathscr{L}$. Notice that a formula $\varphi(\bar{X}, \bar{x})$ in $\#\mathscr{L}$ is equivalent to the formula $\Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi(\bar{X}, \bar{x})$ in $\mathscr{L}$-PNF. Moreover, a formula α in $\Sigma\mathrm{QSO}(\mathscr{L})$ is in $\mathscr{L}$-*sum normal form* ($\mathscr{L}$-*SNF*) if α is of the form $\Sigma_{i=1}^{n} \alpha_i$ where each α_i is in $\mathscr{L}$-PNF.

Proposition V.1. *Every formula in $\Sigma\mathrm{QSO}(\mathscr{L})$ can be rewritten in $\mathscr{L}$-SNF.*

If a formula is in $\mathscr{L}$-PNF then clearly the formula is in $\mathscr{L}$-SNF. Unfortunately, for some $\mathscr{L}$ there exist formulae in

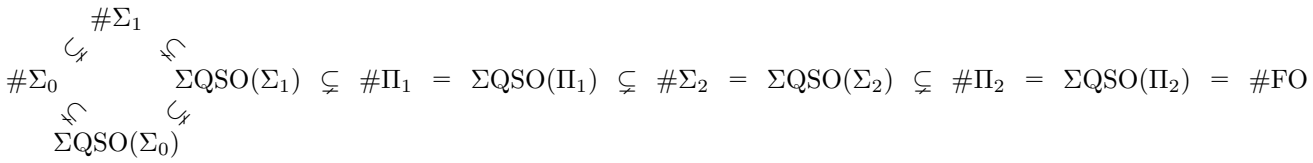

Figure 1. The relationship between the #FO-hierarchy and the $\Sigma\mathrm{QSO}(\mathrm{FO})$-hierarchy, where $\#\Sigma_1$ and $\Sigma\mathrm{QSO}(\Sigma_0)$ are incomparable.

$\Sigma\mathrm{QSO}(\mathscr{L})$ that cannot be rewritten in $\mathscr{L}$-PNF. Therefore, to unveil the relationship between the #FO-hierarchy and the $\Sigma\mathrm{QSO}(\mathrm{FO})$-hierarchy, we need to understand the boundary between PNF and SNF. We do this in the following theorem.

Theorem V.2. *For $i = 0, 1$, there exists a formula α_i in $\Sigma\mathrm{QSO}(\Sigma_i)$ that is not equivalent to any formula in Σ_i-PNF. On the other hand, if $\Pi_1 \subseteq \mathscr{L}$ and $\mathscr{L}$ is closed under conjunction and disjunction, then every formula in $\Sigma\mathrm{QSO}(\mathscr{L})$ can be rewritten in $\mathscr{L}$-PNF.*

As a consequence of Proposition V.1 and Theorem V.2, we obtain that $\#\Sigma_i \subsetneq \Sigma\mathrm{QSO}(\Sigma_i)$ for $i = 0, 1$, and that $\#\mathscr{L} = \Sigma\mathrm{QSO}(\mathscr{L})$ for $\mathscr{L}$ equal to Π_1, Σ_2 or Π_2. The following proposition completes our picture of the relationship between the #FO-hierarchy and the $\Sigma\mathrm{QSO}(\mathrm{FO})$-hierarchy.

Proposition V.3. *The following properties hold:*

- $\Sigma\mathrm{QSO}(\Sigma_0)$ *and* $\#\Sigma_1$ *are incomparable, that is,* $\#\Sigma_1 \nsubseteq \Sigma\mathrm{QSO}(\Sigma_0)$ *and* $\Sigma\mathrm{QSO}(\Sigma_0) \nsubseteq \#\Sigma_1$
- $\Sigma\mathrm{QSO}(\Sigma_1) \subsetneq \Sigma\mathrm{QSO}(\Pi_1)$

The relationship between the two hierarchies is summarized in Figure 1. Our hierarchy and the one proposed in [35] only differ in Σ_0 and Σ_1. Interestingly, we show next that this difference is crucial for finding classes of functions with easy decision versions and good closure properties.

B. Defining a class of functions with easy decision versions and good closure properties

We use the $\Sigma\mathrm{QSO}(\mathrm{FO})$-*hierarchy* to define syntactic classes of functions with good algorithmic and closure properties. But before doing this, we introduce a more strict notion of counting problem with easy decision version. Recall that a function $f : \Sigma^* \to \mathbb{N}$ has an easy decision counterpart if $L_f = \{x \in \Sigma^* \mid f(x) > 0\}$ is a language in P. As the goal of this section is to define a syntactic class of functions in #P with easy decision versions and good closure properties, we do not directly consider the semantic condition $L_f \in \mathrm{P}$, but instead we consider a more restricted syntactic condition. More precisely, a function $f : \Sigma^* \to \mathbb{N}$ is said to be in the complexity class TOTP [31] if there exists a polynomial-time NTM M such that $f(x) = \#\mathrm{total}_M(x) - 1$ for every $x \in \Sigma^*$, where $\#\mathrm{total}_M(x)$ is the total number of runs of M with input x. Notice that one is subtracted from $\#\mathrm{total}_M(x)$ to allow for $f(x) = 0$. Besides, notice that TOTP $\subseteq$ #P and that $f \in$ TOTP implies that $L_f \in \mathrm{P}$.

The complexity class TOTP contains many important counting problems with easy decision counterparts, such as #PERFECTMATCHING, #DNF, and #HORNSAT among others [31]. Besides, TOTP has good closure properties as it is closed under sum, multiplication and subtraction by one. However, some functions in TOTP do not admit FPRAS under standard complexity-theoretical assumptions,[2] and no natural complete problems are known for this class [31]. Hence, we use the $\Sigma\mathrm{QSO}(\mathrm{FO})$-hierarchy to find restrictions of TOTP with good approximation and closure properties.

It was proved in [35] that every function in $\#\Sigma_1$ admits an FPRAS. Besides, it can be proved that $\#\Sigma_1 \subseteq$ TOTP. However, this class is not closed under sum, and then it is not robust under basic closure properties.

Proposition V.4. *There exist functions $f, g \in \#\Sigma_1$ such that $(f + g) \notin \#\Sigma_1$.*

To overcome this limitation, one can consider the class $\Sigma\mathrm{QSO}(\Sigma_1)$, which is closed under sum by definition. In fact, the following proposition shows that the same good properties as for $\#\Sigma_1$ hold for $\Sigma\mathrm{QSO}(\Sigma_1)$, together with the fact that it is closed under sum and multiplication.

Proposition V.5. $\Sigma\mathrm{QSO}(\Sigma_1) \subseteq$ TOTP *and every function in $\Sigma\mathrm{QSO}(\Sigma_1)$ has an FPRAS. Moreover, $\Sigma\mathrm{QSO}(\Sigma_1)$ is closed under sum and multiplication.*

Hence, it only remains to prove that $\Sigma\mathrm{QSO}(\Sigma_1)$ is closed under subtraction by one. Unfortunately, it is not clear whether this property holds; in fact, we conjecture that it is not the case. Thus, we need to find an extension of $\Sigma\mathrm{QSO}(\Sigma_1)$ that keeps all the previous properties and is closed under subtraction by one. It is important to notice that #P is believed not to be closed under subtraction by one by some complexity-theoretical assumption.[3] So, the following proposition rules out any logic that extends Π_1 for a possible extension of $\Sigma\mathrm{QSO}(\Sigma_1)$ with the desired closure property.

Proposition V.6. *If $\Pi_1 \subseteq \mathscr{L} \subseteq$ FO and $\Sigma\mathrm{QSO}(\mathscr{L})$ is closed under subtraction by one, then* #P *is closed under subtraction by one.*

[2]As an example consider the problem of counting the number of independent sets in a graph, and the widely believed assumption that NP is not equal to the randomized complexity class RP (Randomized Polynomial-Time [14]). This counting problem is in TOTP, and it is known that NP = RP if there exists an FPRAS for it [8].

[3]A decision problem L is in the randomized complexity class SPP if there exists a polynomial-time NTM M such that for every $x \in L$ it holds that $\#\mathrm{accept}_M(x) - \#\mathrm{reject}_M(x) = 2$, and for every $x \notin L$ it holds that $\#\mathrm{accept}_M(x) = \#\mathrm{reject}_M(x)$ [12], [30]. It is believed that NP $\nsubseteq$ SPP. However, if #P is closed under subtraction by one, then it holds that NP $\subseteq$ SPP [30].

Therefore, the desired extension has to be achieved by allowing some local extensions to Σ_1. More precisely, we define $\Sigma_1[\mathrm{FO}]$ as Σ_1 but allowing atomic formulae over a signature $\mathbf{R}$ to be of the form either (1) $u = v$, or (2) $X(\bar{u})$, where X is a second-order variable, or (3) $\varphi(\bar{u})$, where $\varphi(\bar{u})$ is a first-order formula over $\mathbf{R}$ (in particular, (3) does not mention any second-order variable). With this extension we obtain a class with the desired properties.

Theorem V.7. *The class $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ is closed under sum, multiplication and subtraction by one. Moreover, $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}]) \subseteq$ TOTP and every function in $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ has an FPRAS.*

The proof that $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ is closed under subtraction by one is the most involved of the paper. We think the main technique used in this proof, which is based on considering some witnesses of logarithmic size, is of independent interest.

C. Defining a class of functions with easy decision versions and natural complete problems

The goal of this section is to define a class of functions in #P with easy decision counterparts and natural complete problems. To this end, we consider the notion of parsimonious reduction. Formally, a function $f : \Sigma^* \to \mathbb{N}$ is parsimoniously reducible to a function $g : \Sigma^* \to \mathbb{N}$ if there exists a function $h : \Sigma^* \to \Sigma^*$ such that h is computable in polynomial time and $f(x) = g(h(x))$ for every $x \in \Sigma^*$. As mentioned at the beginning of this section, if f can be parsimoniously reduced to g, then $L_g \in \mathrm{P}$ implies that $L_f \in \mathrm{P}$ and the existence of an FPRAS for g implies the existence of an FPRAS for f.

In the previous section, we show that the class $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ has good closure and approximation properties. Unfortunately, it is not clear whether it admits a *natural* complete problem under parsimonious reductions, where *natural* means any of the counting problems defined in this section or any other well-known counting problem (not one specifically designed to be complete for the class). Hence, in this section we follow a different approach to find a class of functions in #P with easy decision counterparts and natural complete problems, which is inspired by the approach followed in [15] that uses a restriction of second-order logic to Horn clauses for capturing P (over ordered structures). The following example shows how our approach works.

Example V.8. *Let $\mathbf{R} = \{P(\cdot,\cdot), N(\cdot,\cdot), V(\cdot), NC(\cdot), <\}$. This vocabulary is used as follows to encode a Horn formula. A fact $P(c,x)$ indicates that propositional variable x is a disjunct in a clause c, while $N(c,x)$ indicates that $\neg x$ is a disjunct in c. Furthermore, $V(x)$ holds if x is a propositional variable, and $NC(c)$ holds if c is a clause containing only negative literals, that is, c is of the form $(\neg x_1 \vee \cdots \vee \neg x_n)$.*

To define #HORNSAT*, we consider an* SO*-formula $\varphi(T)$ over $\mathbf{R}$, where T is a unary predicate, such that for every Horn formula θ encoded by an $\mathbf{R}$-structure $\mathfrak{A}$, the number of satisfying assignments of θ is equal to $[\![\Sigma T.\, \varphi(T)]\!](\mathfrak{A})$. In particular, $T(x)$ holds if and only if x is a propositional variable that is assigned value true. More specifically,*

$$\begin{aligned}\varphi(T) \;:=\; & \forall x\,(T(x) \to V(x)) \;\wedge \\ & \forall c\,(NC(c) \to \exists x\,(N(c,x) \wedge \neg T(x))) \;\wedge \\ & \forall c \forall x\,([P(c,x) \wedge \forall y\,(N(c,y) \to T(y))] \to T(x)).\end{aligned}$$

We can rewrite $\varphi(T)$ in the following way:

$$\begin{aligned}& \forall x\,(\neg T(x) \vee V(x)) \;\wedge \\ & \forall c\,(\neg NC(c) \vee \exists x\,(N(c,x) \wedge \neg T(x))) \;\wedge \\ & \forall c \forall x\,(\neg P(c,x) \vee \exists y\,(N(c,y) \wedge \neg T(y)) \vee T(x)).\end{aligned}$$

Moreover, by introducing an auxiliary predicate A defined as:

$$\forall c \forall x\,(\neg A(c,x) \leftrightarrow [N(c,x) \wedge \neg T(x)]),$$

we can translate $\varphi(T)$ into the following equivalent formula:

$$\begin{aligned}\psi(T,A) \;:=\; & \forall x\,(\neg T(x) \vee V(x)) \;\wedge \\ & \forall c\,(\neg NC(c) \vee \exists x\,\neg A(c,x)) \;\wedge \\ & \forall c \forall x\,(\neg P(c,x) \vee \exists y\,\neg A(c,y) \vee T(x)) \;\wedge \\ & \forall c \forall x\,(\neg N(c,x) \vee T(x) \vee \neg A(c,x)) \;\wedge \\ & \forall c \forall x\,(A(c,x) \vee N(c,x)) \;\wedge \\ & \forall c \forall x\,(A(c,x) \vee \neg T(x)).\end{aligned}$$

More precisely, we have that:

$$[\![\Sigma T.\, \varphi(T)]\!](\mathfrak{A}) \quad = \quad [\![\Sigma T.\, \Sigma A.\, \psi(T,A)]\!](\mathfrak{A}),$$

for every $\mathbf{R}$-structure $\mathfrak{A}$ encoding a Horn formula. Therefore, the formula $\psi(T,A)$ also defines #HORNSAT*. More importantly, $\psi(T,A)$ resembles a conjunction of Horn clauses except for the use of negative literals of the form $\exists v\,\neg A(u,v)$.*

The previous example suggests that to define #HORNSAT, we can use Horn formulae defined as follows. A positive literal is a formula of the form $X(\bar{x})$, where X is a second-order variable and $\bar{x}$ is a tuple of first-order variables, and a negative literal is a formula of the form $\exists \bar{v}\,\neg X(\bar{u},\bar{v})$, where $\bar{u}$ and $\bar{v}$ are tuples of first-order variables. Given a signature $\mathbf{R}$, a clause over $\mathbf{R}$ is a formula of the form $\forall \bar{x}\,(\varphi_1 \vee \cdots \vee \varphi_n)$, where each φ_i $(1 \leqslant i \leqslant n)$ is either a positive literal, a negative literal or an FO-formula over $\mathbf{R}$. A clause is said to be Horn if it contains at most one positive literal, and a formula is said to be Horn if it is a conjunction of Horn clauses. With this terminology, we define Π_1-HORN as the set of Horn formulae over a signature $\mathbf{R}$.

We have that #HORNSAT $\in \Sigma\mathrm{QSO}(\Pi_1\text{-HORN})$. Moreover, one can show that $\Sigma\mathrm{QSO}(\Pi_1\text{-HORN})$ forms a class of functions with easy decision counterparts, namely, $\Sigma\mathrm{QSO}(\Pi_1\text{-HORN}) \subseteq$ TOTP. Thus, $\Sigma\mathrm{QSO}(\Pi_1\text{-HORN})$ is a new alternative in our search for a class of functions in #P with easy decision counterparts and natural complete problems. Moreover, an even larger class for our search can be generated by extending the definition of Π_1-HORN with outermost existential quantification. Formally, a formula φ is in Σ_2-HORN if φ is of the form $\exists \bar{x}\,\psi$ with ψ a Horn formula.

Proposition V.9. $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN}) \subseteq \mathrm{TOTP}$.

Interestingly, we have that both #HORNSAT and #DNF belong to $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$. An imperative question at this point is whether in the definitions of Π_1-HORN and Σ_2-HORN, it is necessary to allow negative literals of the form $\exists\bar{v}\,\neg X(\bar{u},\bar{v})$. Actually, this forces our Horn classes to be included in $\Sigma\mathrm{QSO}(\Pi_2)$ and not necessarily in $\Sigma\mathrm{QSO}(\Sigma_2)$. The following result shows that this is indeed the case.

Proposition V.10. $\#\mathrm{HORNSAT} \notin \Sigma\mathrm{QSO}(\Sigma_2)$.

We conclude this section by showing that $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$ is the class we were looking for, as not only every function in $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$ has an easy decision counterpart, but also $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$ admits a natural complete problem under parsimonious reductions. More precisely, define #DISJHORNSAT as the problem of counting the satisfying assignments of a formula Φ that is a disjunction of Horn formulae. Then we have that:

Theorem V.11. #DISJHORNSAT *is* $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$*-complete under parsimonious reductions.*

VI. ADDING RECURSION TO QSO

We have used weighted logics to give a framework for descriptive complexity of counting complexity classes. Here, we go beyond weighted logics and give the first steps on defining recursion at the quantitative level. This goal is not trivial not only because we want to add recursion over functions, but also because it is not clear what could be the right notion of "fixed point". To this end, we show first how to extend QSO with function symbols that are later used to define a natural generalization for functions of the notion of least fixed point of LFP. As a proof of concept, we show how this notion can be used to capture FP. Moreover, we use this concept to define an operator for counting paths in a graph, a natural generalization of the transitive closure operator [22], and show that this gives rise to a logic that captures #L.

We start by defining an extension of QSO with function symbols. Assume that $\mathbf{FS}$ is an infinite set of function symbols, where each $h \in \mathbf{FS}$ has an associated arity denoted by $\mathrm{arity}(h)$. Then the set of FQSO formulae over a signature $\mathbf{R}$ is defined by the following grammar:

$$\alpha := \varphi \mid s \mid h(x_1,\ldots,x_\ell) \mid (\alpha+\alpha) \mid (\alpha\cdot\alpha) \mid \Sigma x.\,\alpha \mid \Pi x.\,\alpha \mid \Sigma X.\,\alpha \mid \Pi X.\,\alpha, \quad (2)$$

where $h \in \mathbf{FS}$, $\mathrm{arity}(h) = \ell$ and $x_1,\ldots,x_\ell$ is a sequence of (not necessarily distinct) first-order variables. Given an $\mathbf{R}$-structure $\mathfrak{A}$ with domain A, we say that F is a *function assignment* for $\mathfrak{A}$ if for every $h \in \mathbf{FS}$ with $\mathrm{arity}(h) = \ell$, we have that $F(h) : A^\ell \to \mathbb{N}$. The notion of function assignment is used to extend the semantics of QSO to the case of a quantitative formula of the form $h(x_1,\ldots,x_\ell)$. More precisely, given first-order and second-order assignments v and V for $\mathfrak{A}$, respectively, we have that:

$$[\![h(x_1,\ldots,x_\ell)]\!](\mathfrak{A},v,V,F) \quad = \quad F(h)(v(x_1),\ldots,v(x_\ell)).$$

As for the case of QFO, we define FQFO disallowing quantifiers ΣX and ΠX in (2).

It is worth noting that function symbols in FQSO represent functions from tuples to natural numbers, so they are different from the classical notion of function symbol in FO [29]. Furthermore, a function symbol can be seen as an "oracle" that is instantiated by the function assignment. To the best of our knowledge, this is the first paper to propose this extension of weighted logics, which should be further investigated.

We define an extension of LFP [20], [38] to allow counting. More precisely, the set of RQFO(FO) formulae over a signature $\mathbf{R}$, where RQFO stands for recursive QFO, is defined as an extension of QFO(FO) that includes the formula $[\mathbf{lsfp}\,\beta(\bar{x},h)]$, where (1) $\bar{x} = (x_1,\ldots,x_\ell)$ is a sequence of ℓ distinct first-order variables, (2) $\beta(\bar{x},h)$ is an FQFO(FO)-formula over $\mathbf{R}$ whose only function symbol is h, and (3) $\mathrm{arity}(h) = \ell$. The free variables of the formula $[\mathbf{lsfp}\,\beta(\bar{x},h)]$ are $x_1,\ldots,x_\ell$; in particular, h is not considered to be free.

Fix an $\mathbf{R}$-structure with domain A and a quantitative formula $[\mathbf{lsfp}\,\beta(\bar{x},h)]$ with $\mathrm{arity}(h) = \ell$, and assume that $\mathcal{F}$ is the set of functions $f : A^\ell \to \mathbb{N}$. To define the semantics of $[\mathbf{lsfp}\,\beta(\bar{x},h)]$, we first show how $\beta(\bar{x},h)$ can be interpreted as an operator T_β on $\mathcal{F}$. More precisely, for every $f \in \mathcal{F}$ and tuple $\bar{a} = (a_1,\ldots,a_\ell) \in A^\ell$, the function $T_\beta(f)$ satisfies that:

$$T_\beta(f)(\bar{a}) \quad = \quad [\![\beta(\bar{x},h)]\!](\mathfrak{A},v,F),$$

where v is a first-order assignment for $\mathfrak{A}$ such that $v(x_i) = a_i$ for every $i \in \{1,\ldots,\ell\}$, and F is a function assignment for $\mathfrak{A}$ such that $F(h) = f$.

As for the case of LFP, it would be natural to consider the point-wise partial order $\leqslant$ on $\mathcal{F}$ defined as $f \leqslant g$ if, and only if, $f(i) \leqslant g(i)$ for every $i \in \{1,\ldots,\ell\}$, and let the semantics of $[\mathbf{lsfp}\,\beta(\bar{x},h)]$ be the least fixed point of the operator T_β. However, $(\mathcal{F},\leqslant)$ is not a complete lattice, so we do not have a Knaster-Tarski Theorem ensuring that such a fixed point exists. Instead, we generalize the semantics of LFP as follows. In the definition of the semantics of LFP, an operator T on relations is considered, and the semantics is defined in terms of the least fixed point of T, that is, a relation R such that [20], [38]: (a) $T(R) = R$, and (b) $R \subseteq S$ for every S such that $T(S) = S$. We can view T as an operator on functions if we consider the characteristic function of a relation. Given a relation $R \subseteq A^\ell$, let χ_R be its characteristic function, that is $\chi_R(\bar{a}) = 1$ if $\bar{a} \in R$, and $\chi_R(\bar{a}) = 0$ otherwise. Then define an operator $T^\star$ on characteristic functions as $T^\star(\chi_R) = \chi_{T(R)}$. Moreover, we can rewrite the conditions defining a least fixed point of T in terms of the operator $T^\star$ if we consider the notion of support of a function. Given a function $f \in \mathcal{F}$, define the support of f, denoted by $\mathrm{supp}(f)$, as $\{\bar{a} \in A^\ell \mid f(\bar{a}) > 0\}$. Then given that $\mathrm{supp}(\chi_R) = R$, we have that the conditions (a) and (b) are equivalent to the following conditions on $T^\star$: (a) $\mathrm{supp}(T^\star(\chi_R)) = \mathrm{supp}(\chi_R)$, and (b) $\mathrm{supp}(\chi_R) \subseteq \mathrm{supp}(\chi_S)$ for every S such that $\mathrm{supp}(T^\star(\chi_S)) = \mathrm{supp}(\chi_S)$. To define a notion of fixed point for T_β we simply generalized these conditions. More precisely, a function $f \in \mathcal{F}$ is a *s-fixed point* of T_β if $\mathrm{supp}(T_\beta(f)) = \mathrm{supp}(f)$, and f is a *least s-fixed point*

of T_β if f is a s-fixed point of T_β and for every s-fixed point g of T_β it holds that $\text{supp}(f) \subseteq \text{supp}(g)$. The existence of such fixed point is ensured by the following lemma:

Lemma VI.1. *If $f, g \in \mathcal{F}$ and $\text{supp}(f) \subseteq \text{supp}(g)$, then $\text{supp}(T_\beta(f)) \subseteq \text{supp}(T_\beta(g))$.*

In fact, as for the case of LFP, this lemma gives us a simple way to compute a least s-fixed point of T_β. Let $f_0 \in \mathcal{F}$ be a function such that $f_0(\bar{a}) = 0$ for every $\bar{a} \in A^\ell$ (i.e. f_0 is the only function with empty support), and let function f_{i+1} be defined as $T_\beta(f_i)$ for every $i \in \mathbb{N}$. Then there exists $j \geqslant 0$ such that $\text{supp}(f_j) = \text{supp}(T_\beta(f_j))$. Let k be the smallest natural number such that $\text{supp}(f_k) = \text{supp}(T_\beta(f_k))$. We have that f_k is a least s-fixed point of T_β, which is used to defined the semantics of $[\mathbf{lsfp}\, \beta(\bar{x}, h)]$. More specifically, for an arbitrary first-order assignment v for $\mathfrak{A}$:

$$[\![[\mathbf{lsfp}\, \beta(\bar{x}, h)]]\!](\mathfrak{A}, v) \quad = \quad f_k(v(\bar{x}))$$

Example VI.2. *We would like to define an* RQFO(FO)*-formula that, given a directed acyclic graph G with n nodes and a pair of nodes b, c in G, counts the number of paths of length at most n from b to c in G. To this end, assume that graphs are encoded using the signature $\mathbf{R} = \{E(\cdot,\cdot), <\}$, and then define formula $\alpha(x, y, f)$ as follows:*

$$E(x, y) + \Sigma z.\, f(x, z) \cdot E(z, y).$$

We have that $[\mathbf{lsfp}\, \alpha(x, y, f)]$ defines our counting function. In fact, assume that $\mathfrak{A}$ is an $\mathbf{R}$-structure with n elements in its domain encoding an acyclic directed graph. Moreover, assume that b, c are elements of $\mathfrak{A}$ and v is a first-order assignment over $\mathfrak{A}$ such that $v(x) = b$ and $v(y) = c$. Then we have that $[\![[\mathbf{lsfp}\, \alpha(x, y, f)]]\!](\mathfrak{A}, v)$ is equal to the number of paths in $\mathfrak{A}$ from b to c of length at most n.

Assume now that we need to extend our previous counting function to the case of arbitrary directed graphs. To this end, suppose that $\varphi_{\text{first}}(x)$ and $\varphi_{succ}(x, y)$ are the FO*-formulae for defining the first and successor predicates, respectively, of $<$. Moreover, define formula $\beta(x, y, t, g)$ as follows:*

$$\begin{aligned}(E(x, y) + \Sigma z.\, g(x, z, t) \cdot E(z, y)) \cdot \varphi_{\text{first}}(t) \; + \\ \Sigma t'.\, \varphi_{succ}(t', t) \cdot \big(\Sigma x'.\, \Sigma y'.\, g(x', y', t')\big)\end{aligned}$$

Then our extended counting function is defined by:

$$\Sigma t.\, (\varphi_{\text{first}}(t) \cdot [\mathbf{lsfp}\, \beta(x, y, t, g)]).$$

In fact, the number of paths of length at most n from a node x to a node y is recursively computed by using the formula $(E(x, y) + \Sigma z.\, g(x, z, t) \cdot E(z, y)) \cdot \varphi_{\text{first}}(t)$, which stores this value in $g(x, y, t)$ with t the first element in the domain. The other formula $\Sigma t'.\, \varphi_{succ}(t', t) \cdot (\Sigma x'.\, \Sigma y'.\, g(x', y', t'))$ is just an auxiliary artifact that is used as a counter to allow reaching a fixed point in the support of g in n steps. Notice that the use of the filter $\varphi_{succ}(t', t)$ prevents this formula for incrementing the value of $g(x, y, t)$ when t is the first element in the domain.

In contrast with LFP, to reach a fixed point we do not need to impose any positive restriction on the formula $\beta(\bar{x}, h)$. Indeed, since β is constructed from monotone operations (i.e. sum and product) over the natural numbers, the resulting operator T_β is monotone as well.

Now that a least fixed point operator over functions is defined, the next step is to understand its expressive power. In the following theorem, we show that this operator can be used to capture FP.

Theorem VI.3. RQFO(FO) *captures* FP *over ordered structures.*

Our last goal in this section is to use the new characterization of FP to explore classes below it. It was shown in [20], [21] that FO extended with a transitive closure operator captures NL. Inspired by this work, we show that a restricted version of RQFO can be used to capture #L, the counting version of NL. Specifically, we use RQFO to define an operator for counting the number of paths in a directed graph, which is what is needed to capture #L.

Given a relational signature $\mathbf{R}$, the set of transitive QFO formulae (TQFO-formulae) is defined as an extension of QFO with the formula $[\mathbf{path}\ \psi(\bar{x}, \bar{y})]$, where $\psi(\bar{x}, \bar{y})$ is an FO-formula over $\mathbf{R}$, and $\bar{x} = (x_1, \ldots, x_k)$, $\bar{y} = (y_1, \ldots, y_k)$ are tuples of pairwise distinct first-order variables. The semantics of $[\mathbf{path}\ \psi(\bar{x}, \bar{y})]$ can easily be defined in terms of RQFO(FO) as follows. Given an $\mathbf{R}$-structure $\mathfrak{A}$ with domain A, define a (directed) graph $\mathcal{G}_\psi(\mathfrak{A}) = (N, E)$ such that $N = A^k$ and for every pair $\bar{b}, \bar{c} \in N$, it holds that $(\bar{b}, \bar{c}) \in E$ if, and only if, $\mathfrak{A} \models \psi(\bar{b}, \bar{c})$. Just as we did for Example VI.2, we can count the paths of length at most $|A^k|$ in $\mathcal{G}_\psi(\mathfrak{A})$ with the formula $\beta_{\psi(\bar{x},\bar{y})}(\bar{x}, \bar{y}, \bar{t}, g)$:

$$\begin{aligned}(\psi(\bar{x}, \bar{y}) + \Sigma \bar{z}.\, g(\bar{x}, \bar{z}, \bar{t}) \cdot \psi(\bar{z}, \bar{y})) \cdot \varphi_{\text{first-lex}}(\bar{t}) \; + \\ \Sigma \bar{t}'.\, \varphi_{\text{succ-lex}}(\bar{t}', \bar{t}) \cdot \big(\Sigma \bar{x}'.\, \Sigma \bar{y}'.\, g(\bar{x}', \bar{y}', \bar{t}')\big)\,,\end{aligned}$$

where $\varphi_{\text{first-lex}}$ and $\varphi_{\text{succ-lex}}$ are FO-formulae defining the first and successor predicates over tuples in A^k, following the lexicographic order induced by $<$. Then the semantics of the path operator can be defined by using the following definition of $[\mathbf{path}\ \psi(\bar{x}, \bar{y})]$ in RQFO:

$$[\mathbf{path}\ \psi(\bar{x}, \bar{y})] \;:=\; \Sigma \bar{t}.\, (\varphi_{\text{first}}(\bar{t}) \cdot [\mathbf{lsfp}\, \beta_{\psi(\bar{x},\bar{y})}(\bar{x}, \bar{y}, \bar{t}, g)]).$$

In other words, $[\![[\mathbf{path}\ \psi(\bar{x}, \bar{y})]]\!](\mathfrak{A}, v)$ counts the number of paths from $v(\bar{x})$ to $v(\bar{y})$ in the graph $\mathcal{G}_\psi(\mathfrak{A})$ whose length is at most $|A^k|$. As it was previously said, the operator for counting paths is exactly what we need to capture #L.

Theorem VI.4. TQFO(FO) *captures* #L *over ordered structures.*

This last result perfectly illustrates the benefits of our logical framework for the development of descriptive complexity for counting complexity classes. The distinction in the language between the Boolean and the quantitative level allows us to define operators at the latter level that cannot be defined at the former. As a example showing how fundamental this separation is, consider the issue of extending QFO(FO) at the Boolean level in order to capture #L. The natural alternative to

do this is to use FO extended with a transitive closure operator, which is denoted by TC. But then the problem is that for every language $L \in \text{NL}$, it holds that its characteristic function χ_L is in $\text{QFO}(\text{TC})$, where $\chi_L(x) = 1$ if $x \in L$, and $\chi_L(x) = 0$ otherwise. Thus, if we assume that $\text{QFO}(\text{TC})$ captures #L (over ordered structures), then we have that $\chi_L \in \#\text{L}$ for every $L \in \text{NL}$. This would imply that $\text{NL} = \text{UL}$,[4] solving an outstanding open problem [34].

VII. Concluding remarks and future work

We proposed a framework based on Weighted Logics to develop a descriptive complexity theory for complexity classes of functions. We consider the results of this paper as a first step in this direction. In this sense, there are several directions for future research, some of which are mentioned here. TotP is an interesting counting complexity class as it naturally defines a class of functions in #P with easy decision counterparts. However, we do not have a logical characterization of this class. In the same direction, we are missing characterizations of complexity classes such as SpanL, or characterizations of quantitative logics such as $\text{QSO}(\text{FO})$. We would also like to define a larger syntactic subclass of #P where each function admits an FPRAS; notice that #PerfectMatching is an important problem admitting an FPRAS [24] that is not included in the classes defined in Section V-B. Moreover, by following the approach proposed in [19], we would like to include second-order free variables in the operator for counting paths introduced in Section VI, so to have alternative ways to capture FPSPACE and even #P. Finally, the least fixed point operator introduced in Section VI clearly deserves further investigation.

VIII. Acknowledgements

The authors are grateful to Luis Alberto Croquevielle for providing the proof of Proposition V.10. This research has been supported by the Fondecyt grant 1161473 and the Millennium Nucleus Center for Semantic Web Research under grant NC120004.

References

[1] S. Abiteboul and V. Vianu, "Fixpoint extensions of first-order logic and datalog-like languages," in *Proceedings of LICS'89*, 1989, pp. 71–79.
[2] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.
[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and the hardness of approximation problems," *J. ACM*, vol. 45, no. 3, pp. 501–555, 1998.
[4] K. J. Compton and E. Grädel, "Logical definability of counting functions," *J. Comput. Syst. Sci.*, vol. 53, no. 2, pp. 283–297, 1996.
[5] M. Droste and P. Gastin, "Weighted automata and weighted logics," *Theor. Comput. Sci.*, vol. 380, no. 1-2, pp. 69–86, 2007.
[6] M. Droste, W. Kuich, and H. Vogler, *Handbook of weighted automata*. Springer Science & Business Media, 2009.
[7] A. Durand, M. Hermann, and P. G. Kolaitis, "Subtractive reductions and complete problems for counting complexity classes," *Theor. Comput. Sci.*, vol. 340, no. 3, pp. 496–513, 2005.
[8] M. E. Dyer, A. M. Frieze, and M. Jerrum, "On counting independent sets in sparse graphs," *SIAM J. Comput.*, vol. 31, no. 5, pp. 1527–1541, 2002.
[9] K. Etessami, "Counting quantifiers, successor relations, and logarithmic space," *J. Comput. Syst. Sci.*, vol. 54, no. 3, pp. 400–411, 1997.
[10] R. Fagin, "Monadic generalized spectra," *Math. Log. Q.*, vol. 21, no. 1, pp. 89–96, 1975.
[11] P. Faliszewski and L. A. Hemaspaandra, "The consequences of eliminating NP solutions," *Comp. Sci. Review*, vol. 2, no. 1, pp. 40–54, 2008.
[12] S. A. Fenner, L. Fortnow, and S. A. Kurtz, "Gap-definable counting classes," *J. Comput. Syst. Sci.*, vol. 48, no. 1, pp. 116–148, 1994.
[13] L. Fortnow, "Counting complexity," in *Complexity Theory Retrospective II*. Springer, 1997, pp. 81–107.
[14] J. Gill, "Computational complexity of probabilistic turing machines," *SIAM J. Comput.*, vol. 6, no. 4, pp. 675–695, 1977.
[15] E. Grädel, "Capturing complexity classes by fragments of second-order logic," *Theor. Comput. Sci.*, vol. 101, no. 1, pp. 35–57, 1992.
[16] ——, "Finite model theory and descriptive complexity," in *Finite Model Theory and Its Applications*. Springer, 2007, pp. 125–230.
[17] E. Grädel and Y. Gurevich, "Metafinite model theory," *Inf. Comput.*, vol. 140, no. 1, pp. 26–81, 1998.
[18] L. A. Hemaspaandra and H. Vollmer, "The satanic notations: counting classes beyond #P and other definitional adventures," *SIGACT News*, vol. 26, no. 1, pp. 2–13, 1995.
[19] N. Immerman, "Languages which capture complexity classes (preliminary report)," in *Proceedings of STOC'83*, 1983, pp. 347–354.
[20] ——, "Relational queries computable in polynomial time," *Information and Control*, vol. 68, no. 1-3, pp. 86–104, 1986.
[21] ——, "Nondeterministic space is closed under complementation," *SIAM J. Comput.*, vol. 17, no. 5, pp. 935–938, 1988.
[22] ——, *Descriptive complexity*, ser. Graduate texts in computer science. Springer, 1999.
[23] N. Immerman and E. Lander, "Describing graphs: a first order approach to graph canonization," in *Complexity Theory Retrospective*, A. L. Selman, Ed. Springer-Verlag, 1990, pp. 59–81.
[24] M. Jerrum, A. Sinclair, and E. Vigoda, "A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries," *J. ACM*, vol. 51, no. 4, pp. 671–697, 2004.
[25] R. M. Karp and M. Luby, "Monte-carlo algorithms for enumeration and reliability problems," in *Proceedings of FOCS'83*, 1983, pp. 56–64.
[26] P. G. Kolaitis and M. N. Thakur, "Logical definability of NP optimization problems," *Information and Computation*, vol. 115, no. 2, pp. 321–353, 1994.
[27] M. W. Krentel, "The complexity of optimization problems," *Journal of computer and system sciences*, vol. 36, no. 3, pp. 490–509, 1988.
[28] R. E. Ladner, "Polynomial space counting problems," *SIAM J. Comput.*, vol. 18, no. 6, pp. 1087–1097, 1989.
[29] L. Libkin, *Elements of Finite Model Theory*. Springer, 2004.
[30] M. Ogiwara and L. A. Hemachandra, "A complexity theory for feasible closure properties," *J. Comput. Syst. Sci.*, vol. 46, no. 3, pp. 295–325, 1993.
[31] A. Pagourtzis and S. Zachos, "The complexity of counting functions with easy decision version," in *Proceedings of MFCS'06*, 2006, pp. 741–752.
[32] C. H. Papadimitriou, *Computational complexity*. Addison-Wesley, 1994.
[33] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *J. Comput. Syst. Sci.*, vol. 43, no. 3, pp. 425–440, 1991.
[34] K. Reinhardt and E. Allender, "Making nondeterminism unambiguous," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 4, no. 14, 1997.
[35] S. Saluja, K. V. Subrahmanyam, and M. N. Thakur, "Descriptive Complexity of #P Functions," *J. Comput. Syst. Sci.*, vol. 50, no. 3, pp. 493–505, 1995.
[36] L. G. Valiant, "The complexity of computing the permanent," *Theor. Comput. Sci.*, vol. 8, pp. 189–201, 1979.
[37] ——, "The complexity of enumeration and reliability problems," *SIAM J. Comput.*, vol. 8, no. 3, pp. 410–421, 1979.
[38] M. Vardi, "The complexity of relational query languages," in *Proceedings of STOC'82*, 1982, pp. 137–146.
[39] H. Vollmer and K. W. Wagner, "Complexity classes of optimization functions," *Inf. and Comp.*, vol. 120, no. 2, pp. 198–219, 1995.

[4] A decision language L is in UL is there exists a logarithmic-space NTM M accepting L and satisfying that $\#\text{accept}_M(x) = 1$ for every $x \in L$.

APPENDIX

A. Notation for the appendix

For a given signature $\mathbf{R}$, we define $\text{ORDSTRUCT}[\mathbf{R}]^*$ as

$$\text{ORDSTRUCT}[\mathbf{R}]^* = \{(\mathfrak{A}, v, V) \mid \mathfrak{A} \in \text{ORDSTRUCT}[\mathbf{R}],\ v\ (V) \text{ is a first-order (second-order) assignment for } \mathfrak{A}\}.$$

The *conditional count* symbol $(\varphi \mapsto \alpha)$ is defined as $(\neg\varphi + (\varphi \cdot \alpha))$ for given SO formula φ and QSO formula α. Note that for each $(\mathfrak{A}, v, V) \in \text{ORDSTRUCT}[\mathbf{R}]^*$,

$$[\![(\varphi \mapsto \alpha)]\!](\mathfrak{A}, v, V) = \begin{cases} [\![\alpha]\!](\mathfrak{A}, v, V) & \text{if } (\mathfrak{A}, v, V) \models \varphi, \\ 0 & \text{otherwise.} \end{cases}$$

We will use the symbol $<$ also to denote the lexicographic order over same-sized tuples. If $\bar{x} = (x_1, \ldots, x_m)$ and $\bar{y} = (y_1, \ldots, y_m)$ are tuples of first-order variables, we denote $\bar{x} < \bar{y}$ for the formula $\bigvee_{i=1}^{m}[\bigwedge_{j=1}^{i-1} x_j = y_j \wedge x_i < y_i]$. Similarly, we use $=$ to denote equality between tuples, as $\bar{x} = \bar{y}$ denotes $\bigwedge_{i=1}^{m}(x_i = y_i)$, and also $\bar{x} \leqslant \bar{y}$ denotes $\bar{x} < \bar{y} \vee \bar{x} = \bar{y}$. We also denote $\min(\bar{x}) := \forall\bar{y}(\bar{x} \leqslant \bar{y})$.

If $\bar{x} = (x_1, \ldots, x_m)$ ($\bar{X} = (X_1, \ldots, X_m)$) is a tuple of first-order (second-order) variables, we denote $\Sigma\bar{x}.\,\alpha$ for $\Sigma x_1.\ \cdots \Sigma x_m.\,\alpha$ and $\Sigma\bar{X}.\,\alpha$ for $\Sigma X_1.\ \cdots \Sigma X_m.\,\alpha$ for each QSO formula α. We also denote $|\bar{x}|$ as the size of $\bar{x}$ ($|\bar{X}|$ as the size of $\bar{X}$). In this case, $|\bar{x}| = m$ ($|\bar{X}| = m$).

B. Proofs from Section IV

Proof of Proposition IV.2

We construct recursively a #P-machine M_α for each $\Sigma\text{QSO}(\text{FO})$ formula α over a signature $\mathbf{R}$. This machine, on input $(\mathfrak{A}, v, V)$ accepts in $[\![\alpha]\!](\mathfrak{A}, v, V)$ of its non-deterministic paths for each $(\mathfrak{A}, v, V) \in \text{ORDSTRUCT}[\mathbf{R}]^*$. Suppose $\mathfrak{A}$ has domain A. If α is a FO-formula φ, then the machine checks if $(\mathfrak{A}, v, V) \models \varphi$ deterministically in polynomial time, and accepts if and only if it holds true. If α is a constant s, it produces s branches and accepts in all of them. If $\alpha = (\beta + \gamma)$, then it chooses between 0 or 1, if it is 0 (1), it simulates M_β (M_γ) on input $(\mathfrak{A}, v, V)$. If $\alpha = \Sigma x.\,\beta$, it chooses $a \in A$ non-deterministically and simulates M_β on input $(\mathfrak{A}, v[a/x], V)$. If $\alpha = \Sigma X.\,\beta$, it chooses $B \in A^{arity(X)}$ and simulates M_β on input $(\mathfrak{A}, v, V[B/X])$. This covers all possible cases for α. Let α be a formula in $\Sigma\text{QSO}(\text{FO})$ over a signature $\mathbf{R}$ and let f be a function over $\mathbf{R}$ such that $f(\text{enc}(\mathfrak{A}))$ is equal to the accepting paths of M_α on input $(\mathfrak{A}, v, V)$ for some $(\mathfrak{A}, v, V) \in \text{ORDSTRUCT}[\mathbf{R}]^*$. We have that f is a #P-function over $\mathbf{R}$ and $f(\text{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$ for every $\mathfrak{A} \in \text{ORDSTRUCT}[\mathbf{R}]$.

For the other direction, note that Saluja et al. [35] proved that #P = #FO. Since a function in #FO can also be defined $\Sigma\text{QSO}(\text{FO})$ (see Section III-A), the condition holds. □

Proof of Proposition IV.3

Similar than the previous proof, we construct recursively a SPANP machine M_α for each $\Sigma\text{QSO}(\exists\text{SO})$ formula α over a signature $\mathbf{R}$. This machine, on input $(\mathfrak{A}, v, V)$, non-deterministically produces $[\![\alpha]\!](\mathfrak{A}, v, V)$ distinct accepting outputs for each $(\mathfrak{A}, v, V) \in \text{ORDSTRUCT}[\mathbf{R}]^*$. Suppose $\mathfrak{A}$ has domain A. If α is a $\exists$SO-formula φ it checks if $(\mathfrak{A}, v, V) \models \varphi$ non-deterministically in polynomial time [10], and accepts if and only if the condition holds true. If α is a constant s, then the machine produces s branches and accepts in all of them. If $\alpha = (\beta + \gamma)$, then it chooses between 0 or 1, if it is 0 (1), it simulates M_β (M_γ) on input $(\mathfrak{A}, v, V)$. If $\alpha = \Sigma x.\,\beta$, it chooses $a \in A$ non-deterministically and simulates M_β on input $(\mathfrak{A}, v[a/x], V)$. If $\alpha = \Sigma X.\,\beta$, it chooses $B \in A^{\text{arity}(X)}$ and simulates M_β on input $(\mathfrak{A}, v, V[B/X])$. This covers all possible cases for α. Additionally, the machine produces a different output on each path. This can be done by printing the trace of all the non-deterministic choices. However, when the machine starts checking whether $(\mathfrak{A}, v, V) \models \varphi$ for some $\exists$SO formula φ, it stops printing in the output tape. This way the machine produces exactly one output from that point onwards. Let α be a formula in $\Sigma\text{QSO}(\exists\text{SO})$ over a signature $\mathbf{R}$ and let f be a function over $\mathbf{R}$ such that $f(\text{enc}(\mathfrak{A}))$ is equal to the number of accepting outputs of M_α on input $(\mathfrak{A}, v, V)$ for some $(\mathfrak{A}, v, V) \in \text{ORDSTRUCT}[\mathbf{R}]^*$. We have that f is a SPANP function over $\mathbf{R}$ and that $f(\text{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$ for every $\mathfrak{A} \in \text{ORDSTRUCT}[\mathbf{R}]$.

For the other direction, Compton et al. [4] proved that SPANP = #$\exists$SO. Since a function in #$\exists$SO can also be defined in $\Sigma\text{QSO}(\exists\text{SO})$, then $\Sigma\text{QSO}(\exists\text{SO})$ captures SPANP over ordered structures.

Proof of Theorem IV.4

For the first condition, let $\alpha \in \mathrm{QFO(LFP)}$ over some signature $\mathbf{R}$. Let f be a function over $\mathbf{R}$ defined by the following procedure. Let $\mathrm{enc}(\mathfrak{A})$ be an input, where $\mathfrak{A}$ is an ordered structure over $\mathbf{R}$ with domain $A = \{1, \ldots, n\}$. In the procedure we slightly extend the grammar of QFO(LFP) to include constants. We replace each first order sum and first order product in α by an expansion using the elements in A. This is, $\Sigma x.\, \beta(x)$ is replaced by $(\beta(1) + \cdots + \beta(n))$ and $\Pi x.\, \beta(x)$ is replaced by $(\beta(1) \cdot \cdots \cdot \beta(n))$. Then each sub-formula $\varphi \in \mathrm{LFP}$ in α is evaluated in polynomial time and replaced by 1 if $\mathfrak{A} \models \varphi$ and by 0 otherwise. The resulting formula is an arithmetic expression of polynomial size (recall that α is fixed) which is evaluated and lastly given as output. Note that $f \in \mathrm{FP}$ and $f(\mathrm{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$.

For the second condition, let $f \in \mathrm{FP}$ defined over some signature $\mathbf{R}$. Let $\ell \in \mathbb{N}$ be such that for each $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$, $\lceil \log_2 f(\mathrm{enc}(\mathfrak{A})) \rceil \leqslant n^\ell$ (i.e. n^ℓ is an upper bound for the output size), where $\mathfrak{A}$ has a domain of size n. Let $\bar{x} = (x_1, \ldots, x_\ell)$. Consider a procedure that receives $\mathrm{enc}(\mathfrak{A})$ and an assignment $\bar{a}$ to $\bar{x}$. Let m be the position of $\bar{a}$ in the lexicographic order of the tuples in A^ℓ. The procedure then computes the m-th bit of $f(\mathrm{enc}(\mathfrak{A}))$, from least to most significant. Since this procedure works in polynomial time, it can be described by an LFP formula $\Phi(\bar{x})$. Then we use

$$\alpha = \Sigma \bar{x}.\, \Phi(\bar{x}) \cdot \varphi(\bar{x}),$$

where $\varphi(\bar{x}) := \Pi \bar{y}.\, (\bar{y} < \bar{x} \mapsto 2)$. Note that if $\bar{a} \in A^\ell$ is the m-th tuple in the given order (starting from 0), then $[\![\varphi(\bar{a})]\!](\mathfrak{A}) = 2^m$. Adding these values for each $\bar{a} \in A^\ell$ gives exactly $f(\mathrm{enc}(\mathfrak{A}))$. In other words, $\Phi(\bar{x})$ simulates the behavior of the FP-machine and the formula α reconstruct the binary output. Then, α is in QFO(LFP) over $\mathbf{R}$ and $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$.

Proof of Theorem IV.5

To show how to evaluate a QSO(PFP)-formula, we construct recursively a #PSPACE-machine M_α for each QSO(PFP) formula α over a signature $\mathbf{R}$. This machine runs in non-deterministic polynomial space and, on input $(\mathfrak{A}, v, V)$, accepts in $[\![\alpha]\!](\mathfrak{A}, v, V)$ of its non-deterministic paths for each $(\mathfrak{A}, v, V) \in \textsc{OrdStruct}[\mathbf{R}]^*$. Suppose $\mathfrak{A}$ has domain A. If α is a PFP-formula φ, then the machine checks if $(\mathfrak{A}, v, V) \models \varphi$ deterministically in polynomial space [29], and accepts if and only if it holds true. If α is a constant s, it produces s branches and accepts in all of them. If $\alpha = (\beta + \gamma)$, then it chooses between 0 or 1, if it is 0 (1), it simulates M_β (M_γ) on input $(\mathfrak{A}, v, V)$. If $\alpha = (\beta \cdot \gamma)$, it simulates M_β on input $(\mathfrak{A}, v, V)$ and on each accepting path, it continues simulating M_γ on input $(\mathfrak{A}, v, V)$. If $\alpha = \Sigma x.\, \beta$, it chooses $a \in A$ non-deterministically and simulates M_β on input $(\mathfrak{A}, v[a/x], V)$. If $\alpha = \Pi x.\, \beta$, it simulates M_β on input $(\mathfrak{A}, v[a/x], V)$ consecutively for each $a \in A$, continuing to the next A-value if the run of M_β accepts. If $\alpha = \Sigma X.\, \beta$, it chooses $B \in A^{arity(X)}$ and simulates M_β on input $(\mathfrak{A}, v, V[B/X])$. If $\alpha = \Pi X.\, \beta$, it simulates M_β on input $(\mathfrak{A}, v, V[B/X])$ consecutively for each $B \in A^{arity(X)}$, again continuing to the next A-value if the run of M_β accepts. This covers all possible cases for α, and each of these steps can be computed in polynomial space. Let α be a formula in QSO(PFP) over a signature $\mathbf{R}$ and let f be a function over $\mathbf{R}$ such that $f(\mathrm{enc}(\mathfrak{A}))$ is equal to the accepting paths of M_α on input $(\mathfrak{A}, v, V)$ for some $(\mathfrak{A}, v, V) \in \textsc{OrdStruct}[\mathbf{R}]^*$. We have that f is a #PSPACE function over $\mathbf{R}$, which implies that f is also a FPSPACE function over $\mathbf{R}$, by the fact that #PSPACE = FPSPACE [28], and that $f(\mathrm{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$ for every $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$.

For the second condition, let $f \in \mathrm{FPSPACE}$ defined over some $\mathbf{R}$. Let $\ell \in \mathbb{N}$ be such that for each $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$, $\lceil \log_2 f(\mathrm{enc}(\mathfrak{A})) \rceil \leqslant 2^{n^\ell}$ (i.e. 2^{n^ℓ} is an upper bound for the output size), where $\mathfrak{A}$ has a domain of size n. Let X be a second-order variable of arity ℓ. Consider a linear order over predicates of arity ℓ given by the formula

$$\varphi_<(X, Y) = \exists \bar{u}\big[\neg X(\bar{u}) \wedge Y(\bar{u}) \wedge \forall \bar{v}\big(\bar{u} < \bar{v} \rightarrow (X(\bar{u}) \leftrightarrow Y(\bar{v}))\big)\big].$$

Namely, we use relations to encode numbers with at most 2^{n^ℓ}-bits where the empty relation represents 0 and the total-relation represents $2^{2^{n^\ell}} - 1$. Furthermore, each relation X indexes a position in the binary output of $f(\mathrm{enc}(\mathfrak{A}))$ as follows. Consider a polynomial space machine over the $\mathbf{R}$ that receives as input an $\mathbf{R}$-structure $\mathfrak{A}$ and a number p encoded by a relation X. Then the machine accepts if, and only if, the p-th bit of $f(\mathrm{enc}(\mathfrak{A}))$ is 1. Since this procedure works in polynomial space, it can be described in PFP [1] by a formula $\Phi(X)$ where the free variable X encodes the number p. Then, similar than the previous proof we define:

$$\alpha := \Sigma X.\, (\Phi(X) \cdot \varphi(X)),$$

where $\varphi(X) = \Pi Y.\, (\varphi_<(Y, X) \mapsto 2)$. Note that for each $B \subseteq A^\ell$ such that B is the m-th element in the order $\varphi_<(X, Y)$, it holds that $[\![\varphi(B)]\!](\mathfrak{A}) = 2^m$. Therefore, $\alpha \in \mathrm{QSO(PFP)}$ and $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$.

Proof of Corollary IV.6

For the first condition, let $\alpha \in \mathrm{QFO(PFP)}$ over some signature $\mathbf{R}$. Let f be a function over $\mathbf{R}$ defined by the following procedure. Let $\mathrm{enc}(\mathfrak{A})$ be an input, where $\mathfrak{A}$ is an ordered structure over $\mathbf{R}$ with domain $A = \{1, \ldots, n\}$. In the procedure we slightly extend the grammar of QFO(PFP) to include constants. We replace each first order sum and first order product in α

by an expansion using the elements in A. This is, $\Sigma x.\,\beta(x)$ is replaced by $(\beta(1) + \cdots + \beta(n))$ and $\Pi x.\,\beta(x)$ is replaced by $(\beta(1) \cdot \cdots \cdot \beta(n))$. Then each sub-formula $\varphi \in \mathrm{PFP}$ in α is evaluated in polynomial space and replaced by 1 if $\mathfrak{A} \models \varphi$ and by 0 otherwise. The resulting formula is an arithmetic expression of polynomial size (recall that α is fixed) which is evaluated and lastly given as output. Note that $f \in \text{FPSPACE(POLY)}$ and $f(\mathrm{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$.

For the second condition, let $f \in \text{FPSPACE}$ defined over some signature $\mathbf{R}$. Let $\ell \in \mathbb{N}$ be such that for each $\mathfrak{A} \in \text{ORDSTRUCT}[\mathbf{R}]$, $\lceil \log_2 f(\mathrm{enc}(\mathfrak{A})) \rceil \leqslant n^\ell$, where $\mathfrak{A}$ has a domain of size n. Let $\bar{x} = (x_1, \ldots, x_\ell)$. Consider a procedure that receives $\mathrm{enc}(\mathfrak{A})$ and an assignment $\bar{a}$ to $\bar{x}$. Let m be the position of $\bar{a}$ in the lexicographic order of the tuples in A^ℓ. The procedure then computes the m-th bit of $f(\mathrm{enc}(\mathfrak{A}))$, from least to most significant. Since this procedure works in polynomial space, it can be described by an PFP formula $\Phi(\bar{x})$. Then we use

$$\alpha = \Sigma \bar{x}.\, \Phi(\bar{x}) \cdot \varphi(\bar{x}),$$

where $\varphi(\bar{x}) := \Pi \bar{y}.\, (\bar{y} < \bar{x} \mapsto 2)$. Note that if $\bar{a} \in A^\ell$ is the m-th tuple in the given order (starting from 0), then $[\![\varphi(\bar{a})]\!](\mathfrak{A}) = 2^m$. Adding these values for each $\bar{a} \in A^\ell$ gives exactly $f(\mathrm{enc}(\mathfrak{A}))$. Then, α is in $\mathrm{QFO(PFP)}$ over $\mathbf{R}$ and $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$.

Proof of Theorem IV.10

Similar than the previous proofs, we construct recursively a non-deterministic polynomial time Turing machine M_α with output tape for each $\mathrm{MaxQSO(FO)}$ formula α over a signature $\mathbf{R}$. This machine, on input $(\mathfrak{A}, v, V)$, non-deterministically produces $[\![\alpha]\!](\mathfrak{A}, v, V)$ (in binary) over the output tape of some run, and this value is the maximum value over all runs. Suppose $\mathfrak{A}$ has domain A. If α is a FO-formula φ, then the machine checks if $(\mathfrak{A}, v, V) \models \varphi$ in deterministic polynomial time, and output 1 if, and only if, $(\mathfrak{A}, v, V)$ satisfies φ. If α is a constant s, it outputs s in binary over the output tape. If $\alpha = (\beta + \gamma)$, then it simulates M_β and M_γ on input $(\mathfrak{A}, v, V)$, adds the output of both machines and prints this value over the output tape. If $\alpha = (\beta \cdot \gamma)$ or $\alpha = \max\{\beta, \gamma\}$, then it does the same than the previous case but it multiplies or maximizes, respectively, the outputs of both machines instead of adding. If $\alpha = \Sigma x.\, \beta$, it iterates over all elements $a \in A$ simulating and adding the output of M_β on input $(\mathfrak{A}, v[a/x], V)$. M_α finally outputs the aggregated value in the output tape. If $\alpha = \Pi x.\, \beta$ or $\alpha = \mathrm{Max}\, x.\, \beta$, then M_α does the same idea than the previous case with the difference that the output of M_β on input $(\mathfrak{A}, v[a/x], V)$ is multiplied or maximized, respectively. If $\alpha = \mathrm{Max}\, X.\, \beta$, it chooses $B \in A^{arity(X)}$ and simulates M_β on input $(\mathfrak{A}, v, V[B/X])$. This covers all possible cases for α. Furthermore, it is straightforward to prove that each of these steps are correct and can be computed with a non-deterministic polynomial time Turing machine with output tape. Let α be a formula in MaxQSO over a signature $\mathbf{R}$ and let f be a function over $\mathbf{R}$ such that $f(\mathrm{enc}(\mathfrak{A}))$ is equal to the maximum run (with respect to the output value) of M_α on input $(\mathfrak{A}, v, V)$ for some $(\mathfrak{A}, v, V) \in \text{ORDSTRUCT}[\mathbf{R}]^*$. We have that f is a MAXP function over $\mathbf{R}$ and that $f(\mathrm{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$ for every $\mathfrak{A} \in \text{ORDSTRUCT}[\mathbf{R}]$. Finally, one can easily see that the same construction holds with $\mathrm{MinQSO(FO)}$ by constructing a Turing machine that take the min over all runs instead of max.

The proof for the other direction is similar than in [26] extended with the ideas of Theorem IV.4. Let $f \in \text{MAXP}$ be a function defined over some signature $\mathbf{R}$ and $\ell \in \mathbb{N}$ such that $\lceil \log_2 f(\mathrm{enc}(\mathfrak{A})) \rceil \leqslant n^\ell$ for each $\mathfrak{A} \in \text{ORDSTRUCT}[\mathbf{R}]$ of size n. For $U \subseteq A^\ell$, we can interpret the encoding of U ($\mathrm{enc}(U)$) as the binary encoding of a number with n^l-bits. We denote this value by $\mathrm{val}(\mathrm{enc}(U))$. Then, given $\mathfrak{A} \in \text{ORDSTRUCT}[\mathbf{R}]$ and $U \subseteq A^\ell$, consider the problem of checking whether $f(\mathrm{enc}(\mathfrak{A})) \geqslant \mathrm{val}(\mathrm{enc}(U))$. Clearly, this is an NP-problem and, by Fagin's theorem, there exists a formula of the form $\exists \bar{X}.\, \Phi(\bar{X}, Y)$ with $\Phi(\bar{X}, Y)$ in FO-logic and $\mathrm{arity}(Y) = \ell$ such that $f(\mathrm{enc}(\mathfrak{A})) \geqslant \mathrm{val}(\mathrm{enc}(U))$ if, and only if, $(\mathfrak{A}, v, V) \models \exists \bar{X}.\, \Phi(\bar{X}, Y)$ with $V(Y) = U$. Then we can describe the function f by the following MaxQSO formula:

$$\alpha = \mathrm{Max}\, \bar{X}.\ \mathrm{Max}\, Y.\ \ \Phi(\bar{X}, Y) \cdot \big(\Sigma \bar{x}.\, Y(\bar{x}) \cdot \Pi \bar{y}.\, (\bar{x} < \bar{y} \mapsto 2)\big).$$

Note that, in contrast with previous proofs, we use $\bar{x} < \bar{y}$ instead of $\bar{y} < \bar{x}$ because the most significant bit in $\mathrm{enc}(U)$ correspond to the smallest tuple in U. It is easy to check that $\Phi(\bar{X}, Y)$ simulates the NP-machine and, if $\Phi(\bar{X}, Y)$ holds, the formula to the right reconstructs the binary output from the relation in Y. Then, α is in $\mathrm{MaxQSO(FO)}$ over $\mathbf{R}$ and $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$.

For the case of $\mathrm{MinQSO(FO)}$ and a function $f \in \text{MINP}$, one has to follow the same approach but considering the NP-problem of checking whether $f(\mathrm{enc}(\mathfrak{A})) \leqslant \mathrm{val}(\mathrm{enc}(U))$. Then, the formula for describing f is the following:

$$\alpha = \mathrm{Min}\, \bar{X}.\ \mathrm{Min}\, Y.\ \ \Sigma \bar{x}.\, \big((\Phi(\bar{X}, Y) \to Y(\bar{x})) \cdot \Pi \bar{y}.\, (\bar{x} < \bar{y} \mapsto 2)\big).$$

In this case, if the formula $\Phi(\bar{X}, Y)$ is false, then the output produced by the subformula inside the $\min$-quantifiers will be the biggest possible value (i.e. 2^{n^ℓ}). On the other hand, if $\Phi(\bar{X}, Y)$ holds, the subformula will produce $\mathrm{val}(\mathrm{enc}(U))$. Similar than for $\max$, we conclude that α is in $\mathrm{MinQSO(FO)}$ and $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$.

C. Proofs from Section V

Proof of Theorem V.1

Recall that a formula in $\Sigma\mathrm{QSO}(\mathscr{L})$ is on the following grammar:

$$\alpha = \varphi \mid s \mid (\alpha + \alpha) \mid \Sigma x.\, \alpha \mid \Sigma X.\, \alpha,$$

where φ is a formula in $\mathscr{L}$ and $s \in \mathbb{N}$. We will construct a recursive function τ such that for every $\Sigma\mathrm{QSO}(\mathscr{L})$ formula α, it outputs an equivalent formula $\tau(\alpha)$ which is in $\mathscr{L}$-SNF. If $\alpha = \varphi$, then we define $\tau(\alpha) = \alpha$ which is clearly equivalent and already in $\mathscr{L}$-SNF. If $\alpha = s$, then we define $(\top + \cdots + \top)$ (s times), which also satisfies the condition. We assume that for every sub-formula β of α, $\tau(\beta)$ is an equivalent formula in $\mathscr{L}$-SNF. If $\alpha = (\alpha_1 + \alpha_2)$, then we define $\tau(\alpha) = (\tau(\alpha_1) + \tau(\alpha_2))$. If $\alpha = \Sigma x.\, \beta$, then let $\tau(\beta) = \sum_{i=1}^{k} \beta_i$ for some k where each β_i is in $\mathscr{L}$-PNF. By grouping together the terms in the sum, we define $\tau(\alpha) = \sum_{i=1}^{m} \Sigma x.\, \beta_i$ which satisfies the condition and is equivalent to α. If $\alpha = \Sigma X.\, \beta$, then we proceed analogously as in the previous case. This covers all possible cases for α and we conclude the proof by taking $\tau(\alpha)$ as the desired rewrite of α.

Proof of theorem V.2

We divide the proof in three parts.

First, we prove that the formula $\alpha_0 = (\Sigma X.\, 1) + 1$ with $\mathrm{arity}(X) = 1$ (i.e. the function $2^n + 1$, where n is the size of the input structure) is not equivalent to any formula in Σ_0-PNF. Suppose that there exists some formula $\alpha = \Sigma \bar{X}.\, \Sigma \bar{x}.\, \varphi(\bar{X}, \bar{x})$ in Σ_0-PNF that is equivalent to the $\Sigma\mathrm{QSO}(\Sigma_0)$ formula α_0. In [35], it was proved that if $|\bar{X}| > 0$, then the function defined by α, for big enough structures, is always even which is not possible in our case. On the other hand, if α is of the form $\Sigma \bar{x}.\, \varphi(\bar{x})$, then α defines a polynomially bounded function which also leads to a contradiction.

Second, we prove that the formula $\alpha_1 = 2$ (i.e. the constant 2) is not equivalent to any formula in Σ_1-PNF. Suppose that there exists some formula $\alpha = \Sigma \bar{X}.\, \Sigma \bar{x}.\, \exists \bar{y}\, \varphi(\bar{X}, \bar{x}, \bar{y})$ in Σ_1-PNF that is equivalent to the $\Sigma\mathrm{QSO}(\Sigma_1)$ formula 2. First, if $|\bar{X}| = |\bar{x}| = 0$, then the function defined by α is never greater than 1. Therefore, suppose that $|\bar{X}| > 0$ or $|\bar{x}| > 0$, and consider any ordered structure $\mathfrak{A}$. Since $[\![\alpha]\!](\mathfrak{A}) = 2$, there exist at least two assignments $(\bar{B}_1, \bar{b}_1, \bar{a}_1)$, $(\bar{B}_2, \bar{b}_2, \bar{a}_2)$ to $(\bar{X}, \bar{x}, \bar{y})$ such that for both, $\mathfrak{A} \models \varphi(\bar{B}_i, \bar{b}_i, \bar{a}_i)$. Now consider the ordered structure $\mathfrak{A}'$ that is obtained by duplicating $\mathfrak{A}$. This is, each half of $\mathfrak{A}'$ is isomorphic to $\mathfrak{A}$. Note that $\mathfrak{A}' \models \varphi(\bar{B}_i, \bar{b}_i, \bar{a}_i)$ for $i = 1, 2$ and there exists a third assignments $(\bar{B}'_1, \bar{b}'_1, \bar{a}'_1)$ that is isomorphic to $(\bar{B}_1, \bar{b}_1, \bar{a}_1)$ but in the other half to the structure such that $\mathfrak{A}' \models \varphi(\bar{B}'_1, \bar{b}'_1, \bar{a}'_1)$. We have that $[\![\alpha]\!](\mathfrak{A}) \geqslant 3$ which is a contradiction.

We now show that if $\mathscr{L}$ contains Π_1 and is closed under conjunction and disjunction, then for every formula α in $\Sigma\mathrm{QSO}(\mathscr{L})$ there is an equivalent formula β in $\mathscr{L}$-PNF. As in Theorem V.1, we show a recursive function τ that produces such formula. As we showed, there exists an equivalent formula in $\mathscr{L}$-SNF, so we assume that α is in that form. Let $\alpha = \sum_{i=1}^{n} \alpha_i$ where each α_i is in $\mathscr{L}$-SNF. Without lost of generality, we assume that each $\alpha_i = \Sigma \bar{X}.\, \Sigma \bar{x}.\, \varphi_i(\bar{X}, \bar{x})$ with $|\bar{X}| > 0$ and $|\bar{x}| > 0$. If not, we replace each α_i for the equivalent formula $\Sigma \bar{X}.\, \Sigma Y.\, \Sigma \bar{x}.\, \Sigma y.\, (\varphi_i(\bar{X}, \bar{x}) \wedge \forall z\, Y(z) \wedge \forall z (y \leqslant z))$.

Now we begin describing the function τ. If $\alpha = \Sigma \bar{X}.\, \Sigma \bar{x}.\, \varphi(\bar{X}, \bar{x})$, then the formula is already in $\mathscr{L}$-PNF so we define $\tau(\alpha) = \alpha$. If $\alpha = \alpha_1 + \alpha_2$, then we assume that $\tau(\alpha_1) = \Sigma \bar{X}.\, \Sigma \bar{x}.\, \varphi(\bar{X}, \bar{x})$ and $\tau(\alpha_2) = \Sigma \bar{Y}.\, \Sigma \bar{y}.\, \psi(\bar{Y}, \bar{y})$. The construction that we will provide for this function works by identifying a "first" assignment for both $(\bar{X}, \bar{x})$ and $(\bar{Y}, \bar{y})$ and a "last" assignment for both $(\bar{X}, \bar{x})$ and $(\bar{Y}, \bar{y})$. These are identified by the following formulas:

$$\gamma_{\text{first}}(\bar{X}, \bar{x}) = \bigwedge_{i=1}^{|\bar{X}|} \forall \bar{z} \neg X_i(\bar{z}) \wedge \forall \bar{z} (\bar{x} \leqslant \bar{z}),$$

$$\gamma_{\text{last}}(\bar{X}, \bar{x}) = \bigwedge_{i=1}^{|\bar{X}|} \forall \bar{z} X_i(\bar{z}) \wedge \forall \bar{z} (\bar{z} \leqslant \bar{x}).$$

Similarly, we define the formulas $\gamma_{\text{first}}(\bar{Y}, \bar{y})$ and $\gamma_{\text{last}}(\bar{Y}, \bar{y})$ (for the sake of simplicity we reuse the names γ_{first} and γ_{last}). In other words, the "first" assignment is the one where every second-order predicate is empty and the first-order assignment is the lexicographically smallest, and the "last" assignment is the one where every second-order predicate is full and the first-order assignment is the lexicographically greatest. We also need to identify assignments that are not first and are not last. We do

this by negating the two formulas above and grouping together the first-order variables:

$$\gamma_{\text{not first}}(\bar{X},\bar{x}) = \exists \bar{z}(\bar{z}_0 < \bar{x} \vee \bigvee_{i=1}^{|\bar{X}|} X(\bar{z}_i)),$$

$$\gamma_{\text{not last}}(\bar{X},\bar{x}) = \exists \bar{z}(\bar{x} < \bar{z}_0 \vee \bigvee_{i=1}^{|\bar{X}|} \neg X(\bar{z}_i)),$$

where $\bar{z} = (\bar{z}_0, \bar{z}_1, \ldots, \bar{z}_{|\bar{X}|})$. The following formula is equivalent to α:

$$\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{Y}.\,\Sigma\bar{y}.\,[(\varphi(\bar{X},\bar{x}) \wedge \gamma_{\text{not first}}(\bar{X},\bar{x}) \wedge \gamma_{\text{first}}(\bar{Y},\bar{y}))\vee \tag{3}$$

$$(\varphi(\bar{X},\bar{x}) \wedge \gamma_{\text{first}}(\bar{X},\bar{x}) \wedge \gamma_{\text{last}}(\bar{Y},\bar{y}))\vee \tag{4}$$

$$(\psi(\bar{Y},\bar{y}) \wedge \gamma_{\text{first}}(\bar{X},\bar{x}) \wedge \gamma_{\text{not last}}(\bar{Y},\bar{y}))\vee \tag{5}$$

$$(\psi(\bar{Y},\bar{y}) \wedge \gamma_{\text{last}}(\bar{X},\bar{x}) \wedge \gamma_{\text{last}}(\bar{Y},\bar{y}))]. \tag{6}$$

To show that the formula is indeed equivalent to α, note that the formulas in lines (3) and (4) form a partition over the assignments of $(\bar{X},\bar{x})$, while fixing an assignment for $(\bar{Y},\bar{y})$, and the formulas in lines (5) and (6) form a partition over the assignments of $(\bar{Y},\bar{y})$, while fixing an assignment for $(\bar{X},\bar{x})$. Altogether, the four lines define pairwise disjoint assignments for $(\bar{X},\bar{x}),(\bar{Y},\bar{y})$. With this, it is straightforward to show that the above formula is equivalent to α. However, the formula is not yet in the correct form since it has existential quantifiers in the subformulas $\gamma_{\text{not first}}$ and $\gamma_{\text{not last}}$. To solve this, first let take a close look to the complete formula:

$$\begin{aligned}\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{Y}.\,\Sigma\bar{y}.\,[&(\varphi(\bar{X},\bar{x}) \wedge \exists\bar{v}(\bar{v}_0 < \bar{x} \vee \bigvee_{i=1}^{|\bar{X}|} X(\bar{v}_i)) \wedge \bigwedge_{i=1}^{|\bar{Y}|} \forall\bar{z}\neg Y_i(\bar{z}) \wedge \forall\bar{z}(\bar{y} \leqslant \bar{z})\vee\\
&(\varphi(\bar{X},\bar{x}) \wedge \bigwedge_{i=1}^{|\bar{X}|} \forall\bar{z}\neg X_i(\bar{z}) \wedge \forall\bar{z}(\bar{x} \leqslant \bar{z}) \wedge \bigwedge_{i=1}^{|\bar{Y}|} \forall\bar{z} Y_i(\bar{z}) \wedge \forall\bar{z}(\bar{z} \leqslant \bar{y}))\vee\\
&(\psi(\bar{Y},\bar{y}) \wedge \bigwedge_{i=1}^{|\bar{X}|} \forall\bar{z}\neg X_i(\bar{z}) \wedge \forall\bar{z}(\bar{x} \leqslant \bar{z}) \wedge \exists\bar{w}(\bar{y} < \bar{w}_0 \vee \bigvee_{i=1}^{|\bar{Y}|} \neg Y(\bar{w}_i))\vee\\
&(\psi(\bar{Y},\bar{y}) \wedge \bigwedge_{i=1}^{|\bar{X}|} \forall\bar{z} X_i(\bar{z}) \wedge \forall\bar{z}(\bar{z} \leqslant \bar{x}) \wedge \bigwedge_{i=1}^{|\bar{Y}|} \forall\bar{z} Y_i(\bar{z}) \wedge \forall\bar{z}(\bar{z} \leqslant \bar{y}))].\end{aligned}$$

To construct an equivalent formula that is in the correct form, we define $\bar{u} = (\bar{v},\bar{w})$ and we replace the first-order quantifiers by a first-sum and count the first assignment to $\bar{v}$ and $\bar{w}$ that satisfies the formula. A similar construction was used in [35]. Then the final formula equivalent to α is the following:

$$\begin{aligned}\Sigma\bar{X}.\,\Sigma\bar{Y}.\,\Sigma\bar{x}.\,\Sigma\bar{y}.\,\Sigma\bar{u}.\,[&(\varphi(\bar{X},\bar{x}) \wedge (\bar{v}_0 < \bar{x} \vee \bigvee_{i=1}^{|\bar{X}|} X(\bar{v}_i)) \wedge \forall\bar{u}'((\bar{v}'_0 < \bar{x} \vee \bigvee_{i=1}^{|\bar{X}|} X(\bar{v}'_i)) \rightarrow \bar{u} \leqslant \bar{u}')\wedge\\
&\qquad \bigwedge_{i=1}^{|\bar{Y}|} \forall\bar{z}\neg Y_i(\bar{z}) \wedge \forall\bar{z}(\bar{y} \leqslant \bar{z})\vee\\
&(\varphi(\bar{X},\bar{x}) \wedge \bigwedge_{i=1}^{|\bar{X}|} \forall\bar{z}\neg X_i(\bar{z}) \wedge \forall\bar{z}(\bar{x} \leqslant \bar{z}) \wedge \bigwedge_{i=1}^{|\bar{Y}|} \forall\bar{z} Y_i(\bar{z}) \wedge \forall\bar{z}(\bar{z} \leqslant \bar{y}) \wedge \forall\bar{u}'(\bar{u} \leqslant \bar{u}'))\vee\\
&(\psi(\bar{Y},\bar{y}) \wedge \bigwedge_{i=1}^{|\bar{X}|} \forall\bar{z}\neg X_i(\bar{z}) \wedge \forall\bar{z}(\bar{x} \leqslant \bar{z})\wedge\\
&\qquad (\bar{y} < \bar{w}_0 \vee \bigvee_{i=1}^{|\bar{Y}|} \neg Y(\bar{w}_i)) \wedge \forall\bar{u}'(\bar{y} < \bar{w}'_0 \vee \bigvee_{i=1}^{|\bar{Y}|} \neg Y(\bar{w}'_i)) \rightarrow \bar{u} \leqslant \bar{u}')\vee\\
&(\psi(\bar{Y},\bar{y}) \wedge \bigwedge_{i=1}^{|\bar{X}|} \forall\bar{z} X_i(\bar{z}) \wedge \forall\bar{z}(\bar{z} \leqslant \bar{x}) \wedge \bigwedge_{i=1}^{|\bar{Y}|} \forall\bar{z} Y_i(\bar{z}) \wedge \forall\bar{z}(\bar{z} \leqslant \bar{y}) \wedge \forall\bar{u}'(\bar{u} \leqslant \bar{u}'))].\end{aligned}$$

Finally, consider a $\Sigma\text{QSO}(\mathscr{L})$ formula α in $\mathscr{L}$-SNF. If $\alpha = \sum_{i=1}^{n} \alpha_i$, then by induction we can consider $\alpha = \alpha_1 + (\sum_{i=2}^{n} \alpha_i)$ and use $\tau(\alpha_1 + \tau(\sum_{i=2}^{n} \alpha_i))$ as the rewrite of α, which satisfies the condition in the hypothesis.

Proof of Theorem V.3

We give this proof in three parts.

First, we show that $\Sigma\mathrm{QSO}(\Sigma_0) \not\subseteq \#\Sigma_1$. By contradiction, suppose that there is a $\Sigma\mathrm{QSO}(\Sigma_0)$ formula α over some signature $\mathbf{R}$ such that defines the following function. For every finite $\mathbf{R}$-structure with n elements, and where every predicate in $\mathbf{R}$ is empty, $\alpha(\mathrm{enc}(\mathfrak{A})) = n - 1$. We use the following claim.

Claim A.1. *Let $\alpha = \Sigma\bar{x}.\,\varphi(\bar{x})$ where φ is quantifier free. Then the function defined by α is either null, greater or equal to n, or is in $\Omega(n^2)$.*

Proof. Suppose that the function defined by α is not 0 and that φ is in DNF. Furthermore, suppose $\bar{x} = (x_1, \ldots, x_{|\bar{x}|})$. Then $\alpha = \Sigma\bar{x}.\,\varphi_1(\bar{x}) \vee \cdots \vee \varphi_n(\bar{x})$. Since α is not null, then some φ_i must be satisfiable. This is, the function defined by $\Sigma\bar{x}.\,\varphi(\bar{x})$ is not null. We will prove by induction on $|\bar{x}|$ that the function defined by $\Sigma\bar{x}.\,\varphi(\bar{x})$ is either greater or equal to n, or in $\Omega(n^2)$. We address the case $|\bar{x}| = 1$, then $\alpha = \Sigma x.\,\bigwedge \psi(x)$. If any $\psi(x) = (x = x)$ or $\neg(x < x)$, then we can eliminate it and we obtain the same function. If any $\psi = (x < x)$ or $\neg(x = x)$, then the function becomes null. If $\psi(x) = R(x, \ldots, x)$ for some $R \in \mathbf{R}$ the function becomes null for the structures we are considering. If $\psi(x) = \neg R(x, \ldots, x)$, we can eliminate it and for the structures we are considering we obtain the same function. The only possible α left is $\alpha = \Sigma x.\,\top$ which is equal to the function n. This covers all possible cases for $|\bar{x}| = 1$. Now suppose that it holds for $|\bar{x}| = k$ and suppose $\alpha = \Sigma\bar{x}.\,\bigwedge \psi(\bar{x})$ for $|\bar{x}| = k + 1$. If any $\psi(\bar{x}) = (x_i = x_j)$ where $i \neq j$, then α describes the same function as α where x_j has been replaced by x_i. In this formula the tuple of first-order variables has k elements so the function it describes if one of the mentioned in the hypothesis. If $i = j$, then we can eliminate it and obtain the same function. If any $\psi(\bar{x}) = R(\bar{v})$ or $\neg R(\bar{v})$ where $\bar{v}$ is a sub-tuple of $\bar{x}$ then we can either eliminate it or the function becomes null, following the same argument as in the case $|\bar{x}| = 1$. If any $\psi(\bar{x}) = \neg(x_i = x_j)$ or $(x_i < x_j)$ where $i = j$, then the function becomes null. If any $\psi(\bar{x}) = \neg(x_i < x_j)$ where $i = j$, we can eliminate it. The remaining formulas in $\bigwedge \psi(\bar{x})$ are either $\neg(x_i = x_j)$, $(x_i < x_j)$ or $\neg(x_i < x_j)$. If the formula violates transitivity in $<$ (for example, $x < y \wedge y < z \wedge z < x$), then the function α describes is null. Therefore, there is some order over $\bar{x}$ that satisfies $\bigwedge \psi(\bar{x})$. Consider the formula that describes this order (like $x_1 < x_3 \wedge x_3 < x_4 \wedge x_4 < x_2$). The function α describes is greater or equal to the one this formula describes, which is exactly $\binom{n}{|\bar{x}|}$ which is in $\Omega(n^{|\bar{x}|}) \subseteq \Omega(n^2)$ if $|\bar{x}| > 1$. This concludes the proof of the claim. □

We suppose that α is in SNF, this is, $\alpha = \sum_{i=1}^{n} \alpha_i$. Since α is not null, consider some α_i that describes a non-null function. Let $\alpha_i = \Sigma\bar{X}.\,\Sigma\bar{x}.\,\varphi(\bar{X}, \bar{x})$, where φ is quantifier-free. Note that if $|\bar{X}| > 0$, then the function α describes is in $\Omega(2^n)$, as it was proven by the authors in [35]. We have that $\alpha_i = \Sigma\bar{x}.\,\varphi(\bar{x})$, as we proved in the claim, describes either some function greater or equal to n, or in $\Omega(n^2)$, which leads to a contradiction. Lastly, note that the formula $\Sigma x.\,\exists y(x < y)$ is in $\#\Sigma_0$ and describes the function $n - 1$, which concludes the proof.

Now we show that $\#\Sigma_1 \not\subseteq \Sigma\mathrm{QSO}(\Sigma_0)$. In Theorem V.2 we proved that there is no formula in Σ_1-PNF equivalent to the formula $\alpha = 2$. Every formula in $\#\Sigma_1$ can be expressed in Σ_1-PNF, which implies that $2 \in \Sigma\mathrm{QSO}(\Sigma_0)$ and $2 \notin \#\Sigma_1$.

Lastly, we prove that $\Sigma\mathrm{QSO}(\Sigma_1) \subsetneq \Sigma\mathrm{QSO}(\Pi_1)$. For inclusion, let α be a formula in $\Sigma\mathrm{QSO}(\Sigma_1)$. Suppose that it is in Σ_1-SNF. This is, $\alpha = c + \sum_{i=1}^{n} \alpha_i$. Let $\alpha_i = \Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\,\varphi_i(\bar{X}, \bar{x}, \bar{y})$, where φ_i is quantifier-free, for each α_i. We use the same construction used in [35], and we obtain that the formula $\exists\bar{y}\,\varphi_i(\bar{X}, \bar{x}, \bar{y})$ is equivalent to $\Sigma\bar{y}.\,\varphi_i(\bar{X}, \bar{x}, \bar{y}) \wedge \forall\bar{y}'(\varphi_i(\bar{X}, \bar{x}, \bar{y}') \rightarrow \bar{y} \leqslant \bar{y}')$ for every assignment to $(\bar{X}, \bar{x})$. We do this replacement for each α_i and we obtain an equivalent formula in $\Sigma\mathrm{QSO}(\Pi_1)$.

To prove that the inclusion is proper, consider the $\Sigma\mathrm{QSO}(\Pi_1)$ formula $\Sigma x.\,\forall y(y = x)$. This formula defines the following function that takes an ordered structure $\mathfrak{A}$ as input:

$$[\![\alpha]\!](\mathfrak{A}) = \begin{cases} 1 & \mathfrak{A} \text{ has one element} \\ 0 & \text{otherwise.} \end{cases}$$

Suppose that there exists an equivalent formula α in $\Sigma\mathrm{QSO}(\Sigma_1)$. Also, suppose that it is in $\mathcal{L}$-PNF, so $\alpha = c + \sum_{i=1}^{n} \Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\varphi_i(\bar{X}, \bar{x}, \bar{y})$. Since α takes the value 0 for some structures, c must be 0. Consider a structure $\mathfrak{1}$ with one element. We have that for some i, there exists an assignment $(\bar{B}, \bar{b}, \bar{a})$ for $(\bar{X}, \bar{x}, \bar{y})$ such that $\mathfrak{1} \models \varphi_i(\bar{B}, \bar{b}, \bar{a})$. Consider now the structure $\mathfrak{2}$ that is obtained by duplicating $\mathfrak{1}$, as we did for Theorem V.2. Note that $\mathfrak{2} \models \varphi_i(\bar{B}, \bar{b}, \bar{a})$, which implies that $[\![\alpha]\!](\mathfrak{2}) \geqslant 1$, which leads to a contradiction.

Proof of Proposition V.4

Towards a contradiction, assume that the statement is false. This is, that $\#\Sigma_1$ is closed under binary sum. Consider the formula $\Sigma x.\,(x = x)$ which is in $\#\Sigma_1$ over some signature $\mathbf{R}$. For every finite $\mathbf{R}$-structure $\mathfrak{A}$ with n elements, and where every predicate in $\mathbf{R}$ is empty, $\alpha(\mathrm{enc}(\mathfrak{A})) = n$. From our assumption, there exists some formula in $\#\Sigma_1$ equivalent to the

formula $\alpha + \alpha$, which describes the function $2n$. Let $\Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{X},\bar{x},\bar{y})$ be this formula, where φ is quantifier-free. Note that the function defined by this formula is equal or greater than the one defined by $\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{y}.\ \varphi(\bar{X},\bar{x},\bar{y})$ divided by a polynomial factor. More specifically, for each ordered structure $\mathfrak{A}$ with domain A, we have the following inequality:

$$[\![\Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{X},\bar{x},\bar{y})]\!](\mathfrak{A})\cdot|A|^{|\bar{y}|} \geqslant [\![\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{y}.\ \varphi(\bar{X},\bar{x},\bar{y})]\!](\mathfrak{A})$$

Note that the formula $\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{y}.\ \varphi(\bar{X},\bar{x},\bar{y})$ defines a function in $\#\Sigma_0$. It was shown by the authors in [35] that every function in $\#\Sigma_0$ grows exponentially over the size of the structure for large enough structures, when $|\bar{X}| > 0$. This function divided by a polynomial factor still grows exponentially. Therefore, for $\Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{X},\bar{x},\bar{y})$ we have that $|\bar{X}| = 0$.

Now, for the formula $\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{x},\bar{y})$ consider a structure $\mathfrak{1}$ with only one element a. We have that $[\![\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{x},\bar{y})]\!](\mathfrak{1}) = 2$, but the only possible assignment to $\bar{x}$ is the tuple $(a,\ldots,a)$ so $[\![\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{x},\bar{y})]\!](\mathfrak{1}) \leqslant 1$, which follows to a contradiction.

Proof of Proposition V.5

The authors in [35] proved that there exists a *product reduction* from every function in $\#\Sigma_1$ to a restricted version of #DNF. This is, if $\alpha \in \#\Sigma_1$ over some signature $\mathbf{R}$, there exist polynomially computable functions $g : \text{ORDSTRUCT}[\mathbf{R}] \to \text{ORDSTRUCT}[\mathbf{R}_{\text{DNF}}]$ and $h : \mathbb{N} \to \mathbb{N}$ such that for every finite $\mathbf{R}$-structure $\mathfrak{A}$ with domain A, it holds that $[\![\alpha]\!](\mathfrak{A}) = \#\text{DNF}(\text{enc}(g(\mathfrak{A})))\cdot h(|A|)$. We base our proof on this fact.

ΣQSO(Σ_1) is contained in TOTP. Let α be a $\Sigma\text{QSO}(\Sigma_1)$ formula and assume that it is in Σ_1-SNF. This is, $\alpha = \sum_{i=1}^{n}\alpha_i$ where each α_i is in Σ_1-PNF. Consider the following nondeterministic procedure that on input $\text{enc}(\mathfrak{A})$ generates $[\![\alpha]\!](\mathfrak{A})$ branches. For each $\alpha_i = \varphi$, where φ is a Σ_1 formula, it checks if $\mathfrak{A} \models \varphi$ in polynomial time and generates a new branch if that is the case. For each $\alpha_i = \Sigma\bar{X}.\,\Sigma\bar{x}.\,\varphi$, this formula is also in $\#\Sigma_1$. We use the reduction to #DNF provided in [35] and we obtain $g(\text{enc}(\mathfrak{A}))$, which is an instance to #DNF. Since #DNF is also in TOTP [31], we simulate the corresponding nondeterministic procedure that generates exactly $\#\text{DNF}(\text{enc}(g(\mathfrak{A})))$ branches. Since, FP $\subseteq$ TOTP [31], each polynomially computable function is also in TOTP, and then on each of these branches we simulate the corresponding nondeterministic procedure to generate $h(|A|)$ more. The number of branches for each α_i is $[\![\alpha_i]\!](\mathfrak{A}) = \#\text{DNF}(\text{enc}(g(\mathfrak{A})))\cdot h(|A|)$, and the total number of branches in the procedure amounts to $[\![\alpha]\!](\mathfrak{A})$. We conclude that $\alpha \in$ TOTP.

Every function in ΣQSO(Σ_1) has an FPRAS. let α be a $\Sigma\text{QSO}(\Sigma_1)$ formula and assume that it is in Σ_1-SNF. This is, $\alpha = \sum_{i=1}^{n}\alpha_i$ where each α_i is in Σ_1-PNF. Note that each α_i that is equal to some Σ_1 formula φ has an FPRAS given by the procedure that simply checks if $\mathfrak{A} \models \varphi$ and returns 1 if it does and 0 otherwise. Also, each remaining α_i has an FPRAS since $\alpha_i \in \#\Sigma_1$ [35]. If two functions have an FPRAS, then their sum also has one given by the procedure that simulates them both and sums the results. We conclude that α has an FPRAS.

ΣQSO(Σ_1) is closed under sum and multiplication. Since $\Sigma\text{QSO}(\Sigma_1)$ is closed under sum by definition, we focus only in proving that the class is closed under multiplication. We prove this for a more general case for $\Sigma\text{QSO}(\mathscr{L})$ where $\mathscr{L}$ is a fragment of SO.

Lemma A.2. *If $\mathscr{L}$ is a fragment closed under conjunction, then $\Sigma\text{QSO}(\mathscr{L})$ is closed under binary multiplication.*

Proof. We will define a recursive function τ that receives a formula α over the grammar of $\Sigma\text{QSO}(\mathscr{L})$ extended by binary product, and outputs an equivalent formula $\tau(\alpha)$ over the unextended grammar of $\Sigma\text{QSO}(\mathscr{L})$. In fact, the formula $\tau(\alpha)$ is in $\mathscr{L}$-SNF. First we replace each constant s in α for $(\top + \cdots + \top)$ (s times). If $\alpha = \varphi$, then we define $\tau(\alpha) = \alpha$. We assume that for every β that has less algebraic operators than α, $\tau(\beta)$ is in $\mathscr{L}$-SNF. If $\alpha = (\alpha_1 + \alpha_2)$ then we define $\tau(\alpha) = \tau(\alpha_1) + \tau(\alpha_2)$. If $\alpha = \Sigma x.\,\beta$ or $\alpha = \Sigma X.\,\beta$, then we define $\tau(\alpha)$ as the formula in $\mathscr{L}$-SNF that is equivalent to $\Sigma x.\,\tau(\beta)$ and to $\tau(\alpha) = \Sigma X.\,\tau(\beta)$, respectively. If $\alpha = (\alpha_1\cdot\alpha_2)$, we assume that each α_i is in $\mathscr{L}$-SNF. We identify three cases. (1) Some α_i is equal to $\sum_{j=1}^{n}\beta_j$ for $n > 1$. Suppose wlog. that it is α_1. We then define $\tau(\alpha) = \sum_{j=1}^{n}\tau(\beta_j\cdot\alpha_2)$. In the following cases, α_1 and α_2 are in $\mathscr{L}$-SNF. (2) If some α_i is equal to $\Sigma X.\,\beta$ or $\Sigma x.\,\beta$, we define $\tau(\alpha)$ as the $\mathscr{L}$-SNF formula that is equivalent to $\Sigma x.\,\tau(\beta\cdot\alpha_2)$ and $\Sigma X.\,\tau(\beta\cdot\alpha_2)$, respectively. The remaining case is (3) $\alpha_1 = \varphi_1$ and $\alpha_2 = \varphi_2$ where each φ is an $\mathscr{L}$ formula. Then we define $\tau(\alpha) = \varphi_1 \wedge \varphi_2$. This covers all possible cases for α. For every pair of formulas α,β in $\Sigma\text{QSO}(\mathscr{L})$, we have that their multiplication $(\alpha\cdot\beta)$ is a formula in the grammar $\Sigma\text{QSO}(\mathscr{L})$ extended by binary product, and so, there exists an equivalent formula $\tau(\alpha\cdot\beta)$ which is in unextended $\Sigma\text{QSO}(\mathscr{L})$. □

Since Σ_1 is closed under conjunction, this also holds for $\Sigma\text{QSO}(\Sigma_1)$. This concludes the proof.

Proof of Proposition V.6

Let $\mathscr{L}$ be a fragment of FO that contains Π_1. Then we have that every function in $\#\Pi_1$ is expressible in $\Sigma\text{QSO}(\mathscr{L})$. In particular, #3-CNF $\in \Sigma\text{QSO}(\mathscr{L})$. Suppose that $\Sigma\text{QSO}(\mathscr{L})$ is closed under subtraction by one. Then, the function #3-CNF $-$ 1, which counts the number of satisfying assignments of a 3-CNF formula minus one, is also in $\Sigma\text{QSO}(\mathscr{L})$. Note that

$\Sigma\mathrm{QSO}(\mathscr{L}) \subseteq \Sigma\mathrm{QSO}(\mathrm{FO}) = \#\mathrm{P}$. We have that #3-CNF is #P-complete under parsimonious reductions[5]. Now, let f be a function in #P, and consider the nondeterministic polynomial-time procedure that on input $\mathrm{enc}(\mathfrak{A})$ computes the corresponding reduction to #3-CNF, name it $g(\mathrm{enc}(\mathfrak{A}))$, and simulates the #P procedure for #3-CNF -1 on input $g(\mathrm{enc}(\mathfrak{A}))$. We have that this is a #P procedure that computes $f-1$, from which we conclude that #P is closed under subtraction by one.

Proof of Theorem V.7

Closed under sum and multiplication. We show here that $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ is closed under sum and multiplication. This can be seen because $\Sigma\mathrm{QSO}(\mathscr{L})$ is closed under sum for every fragment $\mathscr{L}$ by definition, and since $\Sigma_1[\mathrm{FO}]$ is closed under conjunction, from Lemma A.2 it follows that $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ is closed under multiplication.

Easy decision version and FRPRAS. We show here that $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}]) \subseteq \mathrm{TOTP}$ and every function in $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ has an FPRAS. We do this by showing a parsimonious reduction from a function in $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ to some function in $\Sigma\mathrm{QSO}(\Sigma_1)$, and using the result of Theorem V.5. First, we define a function that converts a formula α in $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ over a signature $\mathbf{R}$ into a formula $\lambda(\alpha)$ in $\Sigma\mathrm{QSO}(\Sigma_1)$ over a signature $\mathbf{R}_\alpha$. Afterwards, we define a function g_α that receives an $\mathbf{R}$-structure $\mathfrak{A}$ and outputs an $\mathbf{R}_\alpha$-structure $g_\alpha(\mathfrak{A})$.

Let α be in $\Sigma\mathrm{QSO}(\Sigma_0[\mathrm{FO}])$. The signature $\mathbf{R}_\alpha$ is obtained by adding the symbol R_ψ, for every FO formula $\psi(\bar{z})$ in α, to $\mathbf{R}$. Each symbol R_ψ represents a predicate with arity $|\bar{z}|$. Then, $\lambda(\alpha)$ is defined as α where each FO formula $\psi(\bar{z})$ has been replaced by $R_\psi(\bar{z})$. We now define the function g_α procedurally. Let $\mathfrak{A}$ be a $\mathbf{R}$-structure with domain A. Let $\mathfrak{A}'$ be an $\mathbf{R}_\alpha$-structure obtained by copying $\mathfrak{A}$ and leaving each $R_\psi^{\mathfrak{A}}$ empty. For each FO-formula $\psi(\bar{z})$ with $|\bar{z}|$ open first-order variables, we iterate for every tuple $\bar{a} \in A^{|\bar{z}|}$. If $\mathfrak{A} \models \psi(\bar{a})$, then the tuple $\bar{a}$ is added to $R_\psi^{\mathfrak{A}'}$ (this can be done in P). This concludes the construction of $\mathfrak{A}'$. Note that the number of FO subformulas, arity and tuple size is fixed in α, so computing this function takes polynomial time over the size of the structure. Moreover, the encoding of $\mathfrak{A}'$ has polynomial size over the size of $\mathrm{enc}(\mathfrak{A})$. We define $g_\alpha(\mathfrak{A}) = \mathfrak{A}'$ and we have that for each $\mathbf{R}$-structure $\mathfrak{A}$: $[\![\alpha]\!](\mathfrak{A}) = [\![\lambda(\alpha)]\!](g_\alpha(\mathfrak{A}))$. Therefore, we have a parsimonious reduction from α to the $\Sigma\mathrm{QSO}(\Sigma_1)$ formula $\lambda(\alpha)$.

To show that α is in TOTP, we can convert α and $\mathrm{enc}(\mathfrak{A})$ into $\lambda(\alpha)$ and $\mathrm{enc}(g_\alpha(\mathfrak{A}))$, respectively, and run the procedure in Proposition V.5. Similarly, to show that α has an FPRAS, we do the same as before and simulates the FPRAS for $\lambda(\alpha)$ in Proposition V.5. These procedures also takes polynomial time and satisfies the condition.

Closed under subtraction by one. We prove here that $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ is closed under subtraction by one. For this, given $\alpha \in \Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ over a signature $\mathbf{R}$, we will define a $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ formula $\kappa(\alpha)$ such that for each finite structure A over $\mathbf{R}$: $[\![\kappa(\alpha)]\!](\mathfrak{A}) = [\![\alpha]\!](\mathfrak{A}) \dot{-} 1$. Without lost of generality, we assume that α is in $\Sigma_1[\mathrm{FO}]$-SNF and $\alpha = \sum_{i=1}^{n} \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi_i$ where each φ_i is in $\Sigma_1[\mathrm{FO}]$. Moreover, we assume that $|\bar{x}| > 0$ since, if this is not the case, we can replace φ_i for the equivalent formula $\Sigma\bar{X}.\, \Sigma y.\, \varphi_i \wedge \mathrm{first}(y)$.

The proof will be separated in two parts. In the first part, we suppose that α is in $\Sigma_1[\mathrm{FO}]$-PNF, namely, $\alpha = \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi$ for some φ in $\Sigma_1[\mathrm{FO}]$. Then we will show how to define a formula φ' that satisfies the following condition: for each $\mathfrak{A}$, if $(\mathfrak{A}, V, v) \models \varphi(\bar{X}, \bar{x})$ for some V and v over $\mathfrak{A}$, then there exists exactly one assignment to $(\bar{X}, \bar{x})$ that satisfies φ and not φ'. From this, we clearly have that $\kappa(\alpha) = \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi'$ is the formula. In the second part, we suppose that α is of the form $\beta + \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi$ with β the sum of one or more formulas in $\Sigma_1[\mathrm{FO}]$-PNF. We define a formula φ' that satisfies the following condition: if $(\mathfrak{A}, V, v) \models \varphi(\bar{X}, \bar{x})$ and $[\![\beta]\!](\mathfrak{A}) = 0$, then there exists exactly one assignment to $(\bar{X}, \bar{x})$ that satisfies φ and not φ'. From here, we can define $\kappa(\alpha)$ recursively as $\kappa(\alpha) = \kappa(\beta) + \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi'$ and the property of subtraction by one will be proven.

Part (1). Let $\alpha = \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi(\bar{X}, \bar{x})$ where φ is an FO-formula. Note that, if α is of the form $\alpha = \Sigma\bar{x}.\, \varphi(\bar{x})$ (i.e. $|\bar{X}| = 0$), we can define $\kappa(\alpha) = \Sigma\bar{x}.\, [\varphi(\bar{x}) \wedge \exists\bar{z}(\varphi(\bar{z}) \wedge \bar{z} < \bar{x})]$, which is in $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ and fulfills the desired condition. Therefore, for the rest of the proof we can assume that $|\bar{X}| > 0$ and $|\bar{x}| > 0$.

To simplify the analysis of φ, the first step is to rewrite φ in a DNF formula. More precisely, we rewrite φ into an equivalent formula of the form $\bigvee_{i=1}^{m} \varphi_i$ for some $m \in \mathbb{N}$ where each $\varphi_i(\bar{X}, \bar{x}) = \exists\bar{y}\, \varphi_i'(\bar{X}, \bar{x}, \bar{y})$ and $\varphi_i'(\bar{X}, \bar{x}, \bar{y})$ is a conjunction of atomic formulas or negation of atomic formulas. Furthermore, we suppose that each conjunct $\varphi_i'(\bar{X}, \bar{x}, \bar{y})$ has the form:

$$\varphi_i'(\bar{X}, \bar{x}, \bar{y}) = \underbrace{\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})}_{\text{an FO formula}} \wedge \underbrace{\varphi_i^{+}(\bar{X}, \bar{x}, \bar{y})}_{\text{conjunction of } X_j\text{'s}} \wedge \underbrace{\varphi_i^{-}(\bar{X}, \bar{x}, \bar{y})}_{\text{conjunction of } \neg X_j\text{'s}} .$$

Note that atomic formulas of the form $R(\bar{z})$ where $R \in \mathbf{R}$ will appear in the subformula $\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$.

Now, we define a series of rewritings to φ that will make each formula φ_i satisfy the following three conditions: (a) no variable from $\bar{x}$ are mentioned in $\varphi_i^{-}(\bar{X}, \bar{x}, \bar{y}) \wedge \varphi_i^{+}(\bar{X}, \bar{x}, \bar{y})$, (b) $\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ defines an ordered partition over the variables

[5]It can be easily verified that the standard reduction from SAT to 3-CNF (or 3-SAT) preserves the number of satisfying assignments

in $(\bar{x}, \bar{y})$ (see below for the definition of ordered partition) and (c) if $X_j(\bar{z})$ and $\neg X_j(\bar{w})$ are mentioned, then the ordered partition should not satisfy $\bar{z} = \bar{w}$. We explain below how to rewrite φ_i in order to satisfy each condition.

(a) In order to satisfy the first condition, consider some instance of a $X_j(\bar{w})$ in φ_i, where $\bar{w}$ is a subtuple of $(\bar{x}, \bar{y})$. We add $|\bar{w}|$ new variables $z_1, \ldots, z_{|\bar{w}|}$ to the formula and let $\bar{z} = (z_1, \ldots, z_{|\bar{w}|})$. We redefine $\varphi_i^+(\bar{X}, \bar{x}, \bar{y})$ by replacing $X_j(\bar{w})$ with $X_j(\bar{z})$. The formula φ_i is now defined as

$$\varphi_i(\bar{X}, \bar{x}) = \exists \bar{y} \exists \bar{z}\ (\bar{z} = \bar{w} \wedge \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})) \wedge \varphi_i^-(\bar{X}, \bar{x}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{x}, \bar{y})).$$

We repeat this process for each instance of a $X_j(\bar{w})$ in φ_i, and we obtain a formula where none of the X_j's acts over any variable in $\bar{x}$. We add the new first-order variables to $\bar{y}$ and we redefine φ_i as

$$\varphi_i(\bar{X}, \bar{x}) = \exists \bar{y}\ \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y}) \wedge \varphi_i^-(\bar{X}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{y}).$$

For example, if $\bar{x} = x$, $\bar{y} = y$ and $\varphi_i = \exists \bar{y}\ X(x, y) \wedge \neg X(x, x) \wedge x < y$, then we redefine $\bar{y} = (y, v_1, v_2, v_3, v_4)$ and $\varphi_i := \exists \bar{y}\ X(v_1, v_2) \wedge \neg X(v_3, v_4) \wedge x < y \wedge v_1 = x \wedge v_2 = y \wedge v_3 = x \wedge v_4 = x$.

(b) An ordered partition on a set S is defined by an equivalence relation $\sim$ over S, and a linear order over $S/\sim$. For example, let $\bar{x} = (x_1, x_2, x_3, x_4)$. A possible ordered partition would be defined by the formula $\theta(\bar{x}) = x_2 < x_1 \wedge x_1 = x_4 \wedge x_4 < x_3$. On the other hand, the formula $\theta'(\bar{x}) = x_1 < x_2 \wedge x_1 < x_4 \wedge x_2 = x_3$ does not define an ordered partition since both $\{x_1\} < \{x_2, x_3\} < \{x_4\}$ and $\{x_1\} < \{x_2, x_3, x_4\}$ satisfy θ'. For a given k, let $\mathcal{B}_k$ be the number of possible ordered partitions for a set of size k. For $1 \leqslant j \leqslant \mathcal{B}_{|(\bar{x}, \bar{y})|}$ let $\theta^j(\bar{x}, \bar{y})$ be a formula that defines an ordered partition over $(\bar{x}, \bar{y})$. Thus, the formula $\varphi_i(\bar{X}, \bar{x})$ is then redefined as:

$$\bigvee_{i=1}^{m} \bigvee_{j=1}^{\mathcal{B}_{|(\bar{x}, \bar{y})|}} \exists \bar{y}\ [\theta^j(\bar{x}, \bar{y}) \wedge \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y}) \wedge \varphi_i^-(\bar{X}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{y})],$$

Note that each $\theta^j(\bar{x}, \bar{y})$ is an FO-formula. Then, by redefining $\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ as $\theta^j(\bar{x}, \bar{y}) \wedge \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$, we can suppose that each $\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ defines an ordered partition over the variables in $(\bar{x}, \bar{y})$.

(c) For showing that no $X_j(\bar{z})$ and $\neg X_j(\bar{w})$ are mentioned in φ_i with $\bar{z}$ and $\bar{w}$ equivalent in the ordered partition (i.e. φ_i is inconsistent), we do the following. If there exists an instance of $X_j(\bar{z})$ in φ_i^+, an instance of $\neg X_j(\bar{w})$ in φ_i^- and the ordered partition in φ_i^{FO} satisfies $\bar{z} = \bar{w}$, then the entire formula φ_i is removed from φ.

It is important to notice that the resulting φ is equivalent to the initial one, and it is still a formula in $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$. From now on, we assume that each $\varphi_i(\bar{X}, \bar{x}) = \exists \bar{y}\ \varphi_i'(\bar{X}, \bar{x}, \bar{y})$ satisfies conditions (a), (b) and (c), and $\varphi_i'(\bar{X}, \bar{x}, \bar{y})$ has the following form:

$$\varphi_i'(\bar{X}, \bar{x}, \bar{y}) \ = \ \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{y}) \wedge \varphi_i^-(\bar{X}, \bar{y})$$

where φ^+ and φ_i^- do not depend on $\bar{x}$.

Claim A.3. *For a given ordered structure $\mathfrak{A}$ and a first-order assignment v for $\mathfrak{A}$, $(\mathfrak{A}, v) \models \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ if, and only if, there exists a second-order assignment V for $\mathfrak{A}$ such that $(\mathfrak{A}, V, v) \models \varphi_i'(\bar{X}, \bar{x}, \bar{y})$.*

Proof. Let $\mathfrak{A}$ be an ordered structure with domain A and let v be a first-order assignment for $\mathfrak{A}$, such that $(\mathfrak{A}, v) \models \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$. Define $\bar{B} = (B_1, \ldots, B_{|\bar{X}|})$ as $B_j = \{v(\bar{w}) \mid \ X_j(\bar{w})$ is mentioned in $\varphi_i^+(\bar{X}, \bar{y})\}$, and let V be a second-order assignment for which $V(\bar{X}) = \bar{B}$. Towards a contradiction, suppose that $(\mathfrak{A}, V, v) \not\models \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{y}) \wedge \varphi_i^-(\bar{X}, \bar{y})$. By the choice of v, and construction of V it is clear that $(\mathfrak{A}, V, v) \models \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{y})$, so we necessarily have that $(\mathfrak{A}, V, v) \not\models \varphi_i^-(\bar{X}, \bar{y})$. Let X_t be such that $\neg X_t(\bar{w})$ is mentioned in $\varphi_i^-(\bar{X}, \bar{y})$ and $(\mathfrak{A}, V, v) \not\models \neg X_t(\bar{w})$, namely, $v(\bar{w}) \in B_t$. However, by the construction of B_t, there exists a subtuple $\bar{z}$ of $\bar{y}$ such that $X_t(\bar{z})$ appears in $\varphi_i^+(\bar{X}, \bar{y})$ and $v(\bar{z}) = v(\bar{w})$. Since $(\mathfrak{A}, v) \models \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ and $v(\bar{z}) = v(\bar{w})$, then the ordered partition in φ_i^{FO} satisfies $\bar{z} = \bar{w}$. This violates condition (c) since $\neg X_t(\bar{w})$ appears in φ_i^- and $X_t(\bar{z})$ appears in φ_i^+, which leads to a contradiction.

For the other direction, if $(\mathfrak{A}, V, v) \models \varphi_i'(\bar{X}, \bar{x}, \bar{y})$ for a second order assignment V for $\mathfrak{A}$, then it is easy to check $(\mathfrak{A}, v) \models \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ since $\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ is a subformula of φ_i'. □

The previous claim and proof motivates the following definitions. For a structure $\mathfrak{A}$ and a first-order assignment v for $\mathfrak{A}$, define $\bar{B}^v = (B_1^v, \ldots, B_{|\bar{X}|}^v)$ where each $B_j^v = \{v(\bar{w}) \mid X_j(\bar{w})$ is mentioned in $\varphi_i^+(\bar{X}, \bar{y})\}$. One can easily check that for every assignments (V, v) such that $(\mathfrak{A}, V, v) \models \varphi_i'(\bar{X}, \bar{x}, \bar{y})$, it holds that $(\mathfrak{A}, \bar{B}^v, v) \models \varphi_i'(\bar{X}, \bar{x}, \bar{y})$ and $\bar{B}^v \subseteq V(\bar{X})$, namely, $\bar{B}^v$ is a valid candidate for $\bar{X}$ and, furthermore, it is contained in all assignments of $\bar{X}$ when v is fixed. Therefore, by choosing one particular v the plan is to remove $\bar{B}^v$ as an assignment over $\bar{X}$ in φ_i. For this, the idea is to choose the minimal v that satisfies $\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ which can be defined with the following formula:

$$\mathit{min}\text{-}\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y}) \ = \ \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y}) \wedge \forall \bar{x}' \forall \bar{y}' (\varphi_i^{\mathrm{FO}}(\bar{x}', \bar{y}') \rightarrow (\bar{x} \leqslant \bar{x}' \wedge \bar{y} \leqslant \bar{y}')).$$

If φ_i^{FO} is satisfiable, let v be the only assignment such that $(\mathfrak{A}, v) \models min\text{-}\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$. Furthermore, let V^* the second order assignment and v^* the first order assignment that satisfy $V^*(\bar{X}) = \bar{B}^v$ and $v^*(\bar{x}) = v(\bar{x})$. By the previous discussion, we have that $(\mathfrak{A}, V^*, v^*) \models \varphi_i(\bar{X}, \bar{x})$.

Now, we have all the tools to introduce one of the main formulas in order to define $\kappa(\alpha)$. Intuitively, we want to exclude the assignment (V^*, v^*) from the satisfying assignments of $\varphi_i(\bar{X}, \bar{x})$. Towards this goal, we can define a formula $\psi_i(\bar{X}, \bar{x})$ such that $(\mathfrak{A}, V, v) \models \psi_i(\bar{X}, \bar{x})$ if, and only if, $V \neq V^*$ or $v \neq v^*$ whenever $\varphi_i(\bar{X}, \bar{x})$ is satisfiable. This property can be defined as follows:

$$\psi_i(\bar{X}, \bar{x}) = \big(\, \exists\bar{x}\exists\bar{y}\, \varphi_i^{\text{FO}}(\bar{x}, \bar{y})\,\big) \rightarrow \tag{7}$$

$$\Big(\, \exists\bar{y}\; min\text{-}\varphi_i^{\text{FO}}(\bar{x}, \bar{y}) \wedge \big(\varphi_i'(\bar{X}, \bar{x}, \bar{y}) \rightarrow \bigvee_{X \in \bar{X}} \exists\bar{z}\,\big(\, X(\bar{z}) \wedge \bigwedge_{X(\bar{w}) \,\in\, \varphi_i^+(\bar{X}, \bar{v})} \bar{w} \neq \bar{z}\,\big)\big) \quad \vee \tag{8}$$

$$\big(\; \exists\bar{x}'\exists\bar{y}\; \varphi_i'(\bar{X}, \bar{x}', \bar{y}) \wedge \bar{x}' < \bar{x}\;\big)\;\Big) \tag{9}$$

To understand the formula, first notice that the premise of the implication at (7) is true if, and only if, $\varphi_i(\bar{X}, \bar{x})$ is satisfiable. Indeed, by Claim A.3 we know that if $\exists\bar{x}\exists\bar{y}\,\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$ is true, then there exists assignments V and v such that $(\mathfrak{A}, V, v) \models \varphi_i'(\bar{X}, \bar{x}, \bar{y})$. Then, the conclusion of the implication (divided into (8) and (9)), take care that $V(\bar{X}) \neq V^*(\bar{X})$ or $v(\bar{x}) \neq v^*(\bar{x})$. Here, the first disjunct (8) checks that $V(\bar{X}) \neq V^*(\bar{X})$ by defining that if $\varphi_i'(\bar{X}, \bar{x}, \bar{y})$ is satisfied then $V^*(\bar{X}) \subsetneq V(\bar{X})$. The second disjunct (9) is satisfied when $v(\bar{x})$ is not the lexicographically smallest tuple that satisfies φ_i (i.e. $v(\bar{x}) \neq v^*(\bar{x})$). Finally, from the previous discussion one can easily check that $\psi_i(\bar{X}, \bar{x})$ satisfies the desire property.

We are ready to define the formula $\kappa(\alpha)$ as $\Sigma\bar{X}.\,\Sigma\bar{x}.\,\bigvee_{i=1}^{m} \varphi_i^*(\bar{X}, \bar{x})$ where each modified disjunct $\varphi_i^*(\bar{X}, \bar{x})$ is constructed as follows. For the sake of simplification, define the auxiliary formula $\chi_i = \neg\exists\bar{x}\exists\bar{y}\,\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$. This formula basically checks if φ_i is not satisfiable (recall Claim A.3). Define the first formula φ_1^* as:

$$\varphi_1^*(\bar{X}, \bar{x}) = \varphi_1(\bar{X}, \bar{x}) \wedge \psi_1(\bar{X}, \bar{x}).$$

This formula accepts all the assignments that satisfy φ_1, except for the assignment (V^*, v^*) of φ_1. The second formula φ_2^* is defined as:

$$\varphi_2^*(\bar{X}, \bar{x}) = \varphi_2(\bar{X}, \bar{x}) \wedge \psi_1(\bar{X}, \bar{x}) \wedge (\chi_1 \rightarrow \psi_2(\bar{X}, \bar{x})).$$

This models all the assignments that satisfy φ_2, except for the assignment (V^*, v^*) of φ_1. Moreover, if φ_1 is not satisfiable, then $\psi_1(\bar{X}, \bar{x})$ and χ_1 will hold, and the formula $\psi_2(\bar{X}, \bar{x})$ will forbid the assignment (V^*, v^*) of φ_2. One can easily generalize this construction for each φ_i as follows:

$$\varphi_i^*(\bar{X}, \bar{x}) = \varphi_i(\bar{X}, \bar{x}) \wedge \psi_1(\bar{X}, \bar{x}) \wedge (\chi_1 \rightarrow \psi_2(\bar{X}, \bar{x})) \wedge ((\chi_1 \wedge \chi_2) \rightarrow \psi_3(\bar{X}, \bar{x})) \wedge \cdots \wedge (\bigwedge_{j=1}^{j=i-1} \chi_j \rightarrow \psi_i(\bar{X}, \bar{x})),$$

From the construction of $\kappa(\alpha)$, one can easily check that $[\![\kappa(\alpha)]\!](\mathfrak{A}) = [\![\alpha]\!](\mathfrak{A}) - 1$ for each $\mathfrak{A}$.

Part (2). Let $\alpha = (\beta + \Sigma\bar{X}.\,\Sigma\bar{x}.\,\varphi(\bar{X}, \bar{x})$ for some $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ formula β. We define $\kappa(\alpha)$ as follows. First, rewrite $\varphi(\bar{X}, \bar{x})$ as in the previous section. Let $\varphi = \bigvee_{i=1}^{m} \varphi_i(\bar{X}, \bar{x})$ where each φ_i satisfies conditions (a), (b) and (c). Also, consider the previously defined formulas χ_i and ψ_i, for each i.

We construct a function λ that receives a quantitative formula β and produces a logic formula $\lambda(\beta)$ that satisfies $\mathfrak{A} \models \lambda(\beta)$ if, and only if, $[\![\beta]\!](\mathfrak{A}) = 0$. If $\beta = \Sigma\bar{x}.\,\varphi(\bar{x})$, then $\lambda(\beta) = \neg\exists\bar{x}'\varphi(\bar{x}')$. If $\beta = \Sigma\bar{X}.\,\Sigma\bar{x}.\,\varphi(\bar{X}, \bar{x})$, then let $\varphi = \bigvee_{i=1}^{m} \varphi_i$ where each φ_i satisfies conditions (a), (b) and (c) of the previous section, and define $\lambda(\beta) = \chi_1 \wedge \cdots \wedge \chi_m$. If $\beta = (\beta_1 + \beta_2)$, then $\lambda(\beta) = \lambda(\beta_1) \wedge \lambda(\beta_2)$. Now, for each φ_i we define:

$$\varphi_i^*(\bar{X}, \bar{x}) = \varphi_i(\bar{X}, \bar{x}) \wedge \Big(\lambda(\beta) \rightarrow \Big(\psi_1(\bar{X}, \bar{x}) \wedge (\chi_1 \rightarrow \psi_2(\bar{X}, \bar{x})) \wedge ((\chi_1 \wedge \chi_2) \rightarrow \psi_3(\bar{X}, \bar{x})) \wedge \cdots \wedge (\bigwedge_{j=1}^{j=i-1} \chi_j \rightarrow \psi_i(\bar{X}, \bar{x}))\Big)\Big).$$

Finally, $\kappa(\alpha)$ is defined as $\kappa(\alpha) = \kappa(\beta) + \Sigma\bar{X}.\,\Sigma\bar{x}.\,\bigvee_{i=1}^{m} \varphi_i^*(\bar{X}, \bar{x})$, which is in $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ and satisfies the desired condition.

Proof of Proposition V.9

Pagourtzis and Zachos mention a TOTP procedure that computes the number of satisfying assignments of a DNF formula [31]. This procedure can be easily extended to receive Horn formulas, and furthermore, a disjunction of Horn formulas. Hence #DISJHORNSAT is in TOTP.

As we show in Proposition V.11, #DISJHORNSAT is complete for $\Sigma\text{QSO}(\Sigma_2\text{-HORN})$ under parsimonious reductions. Let α be a formula in $\Sigma\text{QSO}(\Sigma_2\text{-HORN})$ and let g_α be the reduction to #DISJHORNSAT. The TOTP procedure we construct, for

each input $\mathrm{enc}(\mathfrak{A})$, is simply to compute $g_\alpha(\mathrm{enc}(\mathfrak{A}))$, and then simulate the TOTP procedure for #DISJHORNSAT on input $g_\alpha(\mathrm{enc}(\mathfrak{A}))$. We conclude that α is in TOTP.

Proof of Proposition V.10

We use a similar proof to the one provided by the authors in [35] to separate the classes $\#\Sigma_2$ and $\#\Pi_2$. Suppose that the statement is false, this is, #HORNSAT $\in \Sigma\mathrm{QSO}(\Sigma_2)$. We consider the signature $\mathbf{R}$ that we used as the encoding for a Horn formula (Example V.8) and that the formula $\alpha \in \Sigma\mathrm{QSO}(\Sigma_2)$ follows the encoding in the same way. From what we proved in Theorem V.1, we have that every formula in $\Sigma\mathrm{QSO}(\Sigma_2)$ can be rewritten in Σ_2-PNF, so we assume that α is in this form. Let $\alpha = \Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\,\forall\bar{z}\,\varphi(\bar{X},\bar{x},\bar{y},\bar{z})$. Consider the following Horn formula Φ:

$$\Phi = p \wedge \bigwedge_{i=1}^{n}(t_i \wedge p \rightarrow q) \wedge \neg q,$$

where $n = |\bar{x}|+|\bar{y}|+1$. Let $\mathfrak{A}$ be the encoding of this formula. In the encoding, each variable appears as an element in the domain of $\mathfrak{A}$. This formula is satisfiable, so $[\![\alpha]\!](\mathfrak{A}) \geqslant 1$. Let $(\bar{B},\bar{b},\bar{a})$ be an assignment to $(\bar{X},\bar{x},\bar{y})$ such that $\mathfrak{A} \models \forall\bar{z}\,\varphi(\bar{B},\bar{b},\bar{a},\bar{z})$. Let t_ℓ be such that it does not appear in $\bar{b}$ or $\bar{a}$. Consider the induced substructure $\mathfrak{A}'$ that is obtained by removing t_ℓ from $\mathfrak{A}$ and $\bar{B}'$ as the subset of $\bar{B}$ obtained by deleting each appearance of t_ℓ in $\bar{B}$. We have that $\mathfrak{A}' \models \forall\bar{z}\,\varphi(\bar{B},\bar{b},\bar{a},\bar{z})$. This is because each subformula of the form $\exists y \neg B_i$ is still true, and universal formulas are monotone over induced substructures. It follows that $[\![\alpha]\!](\mathfrak{A}') \geqslant 1$ which is not possible since $\mathfrak{A}'$ encodes the formula

$$\Phi' = p \wedge \bigwedge_{i=1}^{\ell-1}(t_i \wedge p \rightarrow q) \wedge (p \rightarrow q) \wedge \bigwedge_{i=\ell+1}^{n}(t_i \wedge p \rightarrow q) \wedge \neg q,$$

which is unsatisfiable. We arrive to a contradiction and we conclude that #HORNSAT is not in $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$.

Proof of Theorem V.11

First we prove that #DISJHORNSAT is in $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$. Recall that each instance of #DISJHORNSAT is a disjunction of Horn formulas. Let $\mathbf{R} = \{P(\cdot,\cdot), N(\cdot,\cdot), V(\cdot), NC(\cdot), D(\cdot,\cdot)\}$. Each symbol in this vocabulary is used to indicate the same as in Example V.8, with the addition of $D(d,c)$ which indicates that c is a clause in the formula d. Recall that the formula

$$\begin{aligned}
&\forall x\,(\neg T(x) \vee V(x)) \wedge \\
&\forall c\,(\neg NC(c) \vee \exists x\,\neg A(c,x)) \wedge \\
&\forall c \forall x\,(\neg P(c,x) \vee \exists y\,\neg A(c,y) \vee T(x)) \wedge \\
&\forall c \forall x\,(\neg N(c,x) \vee T(x) \vee \neg A(c,x)) \wedge \\
&\forall c \forall x\,(A(c,x) \vee N(c,x)) \wedge \\
&\forall c \forall x\,(A(c,x) \vee \neg T(x)).
\end{aligned}$$

defines #HORNSAT. We obtain the following formula $\psi(T,A)$ in Σ_2-HORN:

$$\begin{aligned}
\exists d[&\forall x\,(\neg T(x) \vee V(x)) \wedge \\
&\forall c\,(\neg D(c,d) \vee \neg NC(c) \vee \exists x\,\neg A(c,x)) \wedge \\
&\forall c \forall x\,(\neg D(c,d) \vee \neg P(c,x) \vee \exists y\,\neg A(c,y) \vee T(x)) \wedge \\
&\forall c \forall x\,(\neg D(c,d) \vee \neg N(c,x) \vee T(x) \vee \neg A(c,x)) \wedge \\
&\forall c \forall x\,(\neg D(c,d) \vee A(c,x) \vee N(c,x)) \wedge \\
&\forall c \forall x\,(\neg D(c,d) \vee A(c,x) \vee \neg T(x))]
\end{aligned}$$

effectively defines #HORNSAT as for every disjunction of Horn formulas $\theta = \theta_1 \vee \cdots \vee \theta_m$ encoded by an $\mathbf{R}$-structure $\mathfrak{A}$, the number of satisfying assignments of θ is equal to $[\![\Sigma T.\,\Sigma A.\,\psi(T,A)]\!](\mathfrak{A})$. Therefore, we conclude that #DISJHORNSAT $\in \Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$.

We will now prove that #DISJHORNSAT is hard for $\Sigma\mathrm{QSO}$ over a signature $\mathbf{R}$ under parsimonious reductions. For each $\Sigma\mathrm{QSO}(\Sigma_2\text{-HORN})$ formula α over $\mathbf{R}$, we will define a polynomial-time procedure that computes a function g_α. This function receives a finite $\mathbf{R}$-structure $\mathfrak{A}$ and outputs an instance of #DISJHORNSAT such that $[\![\alpha]\!](\mathfrak{A}) = \#\mathrm{DISJHORNSAT}(g_\alpha(\mathfrak{A}))$. We suppose that α is in sum normal form and:

$$\alpha = \sum_{i=1}^{\#\text{clauses}} \Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\bigwedge_{j=1}^{n}\forall\bar{z}\,\varphi_j^i(\bar{X},\bar{x},\bar{y},\bar{z}),$$

where each φ_j^i is a Horn clause.

Consider a finite $\mathbf{R}$-structure $\mathfrak{A}$ with domain A. To simplify the proof, we extend our grammar to allow first-order constants. Consider each tuple $\bar{a} \in A^{|\bar{x}|}$, each $\bar{b} \in A^{|\bar{y}|}$ and each $\bar{c} \in A^{|\bar{z}|}$ as a tuple of first-order constants. The following formula defines the same function as α:

$$\sum_{i=1}^{\#clauses} \sum_{\bar{a} \in A^{|\bar{x}|}} \Sigma \bar{X}. \bigvee_{\bar{b} \in A^{|\bar{y}|}} \bigwedge_{j=1}^{n} \bigwedge_{\bar{c} \in A^{|\bar{z}|}} \varphi_j^i(\bar{X}, \bar{a}, \bar{b}, \bar{c}).$$

Note that each FO formula over $(\bar{x}, \bar{y}, \bar{z})$ in each φ_j^i has no free variables. Therefore, we can evaluate each of these in polynomial time and replace them by $\bot$ and $\top$ where it corresponds. Each φ_j^i will be of the form $\bot \vee \chi_j^i(\bar{X})$ or $\top \vee \chi_j^i(\bar{X})$ where χ_j^i is a disjunction of $\neg X_\ell$'s and at most one X_ℓ. The formulas of the form $\top \vee \chi_j^i(\bar{X})$ can be removed entirely, and the formulas of the form $\bot \vee \chi_j^i(\bar{X})$ can be replaced by $\chi_j^i(\bar{X})$. We obtain the formula

$$\sum_{i=1}^{m} \Sigma \bar{X}. \bigvee_{j=1}^{\#d} \bigwedge_{k=1}^{\#c} \psi_{j,k}^i(\bar{X})$$

where every $\psi_{j,k}^i(\bar{X})$ is a disjunction of $\neg X_\ell$'s and zero or one X_ℓ.

Our idea for the rest of the proof is to define g_α for each $\alpha = \Sigma \bar{X}. \bigvee_{j=1}^{\#d} \bigwedge_{k=1}^{\#c} \psi_{j,k}^i(\bar{X})$, for $\alpha = c$ and for $\alpha = \beta_1 + \cdots + \beta_m$ where each β_i is in one of the two previous cases.

If α is equal to $\Sigma \bar{X}. \bigvee_{j=1}^{\#d} \bigwedge_{k=1}^{\#c} \psi_{j,k}(\bar{X})$ where $\psi_{j,k}(\bar{X})$ is a disjunction of $\neg X_\ell$'s and zero or one X_ℓ, then we obtain the **propositional formula** $g_\alpha(\mathfrak{A}) = \bigvee_{j=1}^{\#d} \bigwedge_{k=1}^{\#c} \psi_{j,k}(\bar{X})$ over the propositional alphabet $\{X(\bar{e}) \mid X \in \bar{X} \text{ and } \bar{e} \in A^{\text{arity}(X)}\}$ which has exactly $[\![\alpha]\!](\mathfrak{A})$ satisfying assignments and is precisely a disjunction of Horn formulas.

If α is equal to a constant c, then we define $g_\alpha(\mathfrak{A})$ as the following formula that has exactly c satisfying assignments:

$$g_\alpha(\mathfrak{A}) = \bigvee_{i=1}^{c} \neg t_1 \wedge \cdots \wedge \neg t_{i-1} \wedge t_i \wedge \neg t_{i+1} \wedge \cdots \wedge \neg p_c.$$

If $\alpha = \beta_1 + \cdots + \beta_m$, let $g_{\beta_i}(\mathfrak{A}) = \bigvee_{j=1}^{\#d} \bigwedge_{k=1}^{\#c} \theta_{j,k}^i$ for each β_i where each $\theta_{j,k}^i$ is a Horn clause. Let $\Theta_i = g_{\beta_i}(\mathfrak{A})$. We rename the variables in each Θ_i so none of them are mentioned in any other Θ_j. We add m new variables $t_1, \ldots, t_m$ and we define:

$$\begin{aligned} g_\alpha(\mathfrak{A}) = &\bigvee_{i=1}^{\#d} \Big(\bigwedge_{j=1}^{\#c} \theta_{i,j}^1 \wedge \Big(\bigwedge_{\substack{\text{each } t \\ \text{mentioned in} \\ \Theta_2, \ldots, \Theta_m}} t \Big) \wedge \Big(t_1 \wedge \bigwedge_{\ell=2}^{m} \neg t_\ell \Big) \Big) \vee \\ &\bigvee_{i=1}^{\#d} \Big(\bigwedge_{j=1}^{\#c} \theta_{i,j}^2 \wedge \Big(\bigwedge_{\substack{\text{each } t \\ \text{mentioned in} \\ \Theta_1, \Theta_3, \ldots, \Theta_m}} t \Big) \wedge \Big(t_2 \wedge \bigwedge_{\substack{\ell=1 \\ \ell \neq 2}}^{m} \neg t_\ell \Big) \Big) \vee \cdots \vee \\ &\bigvee_{i=1}^{\#d} \Big(\bigwedge_{j=1}^{\#c} \theta_{i,j}^m \wedge \Big(\bigwedge_{\substack{\text{each } t \\ \text{mentioned in} \\ \Theta_2, \ldots, \Theta_{m-1}}} t \Big) \wedge \Big(t_m \wedge \bigwedge_{\ell=1}^{m-1} \neg t_\ell \Big) \Big). \end{aligned}$$

The formula is a disjunction of Horn formulas, and the number of satisfying assignments for this formula is exactly the sum of satisfying assignments for each $g_{\beta_i}(\mathfrak{A})$. This, at the same time, is equal to $[\![\alpha]\!](\mathfrak{A})$. This covers all possible cases for α, and the entire procedure takes polynomial time.

D. Proofs from Section VI

Proof of Theorem VI.1

For FQFO(FO). Let $\mathbf{R}$ be a signature. We prove the statement for FQFO(FO) over $\mathbf{R}$.

We prove inductively that for each formula $\beta(\bar{x}, h)$ in FQFO(FO) over $\mathbf{R}$, for a given pair of functions f, g such that $\text{supp}(f) \subseteq \text{supp}(g)$, it holds that $\text{supp}(T_\beta(f)) \subseteq \text{supp}(T_\beta(g))$. Let $|\bar{x}| = \ell$.

We separate the proof in each case determined by the FQFO grammar. For each of the following cases.

1. β is either equal to a constant s or an FO formula φ. Then h does not appear. Then, for each structure $\mathfrak{A}$, each first-order assignment v and functional assignments F, G over $\mathfrak{A}$, we have that $[\![\beta(\bar{x}, h)]\!](\mathfrak{A}, v, F) = [\![\beta(\bar{x}, h)]\!](\mathfrak{A}, v, G)$. As a result, $\text{supp}(T_\beta(f)) = \text{supp}(T_\beta(g))$ for every pair of functions f, g.
2. β is equal to $h(\bar{y})$ for some subtuple $\bar{y}$ of $\bar{x}$. Then $T_\beta(f) = f$ and $T_\beta(g) = g$ and the condition holds trivially.

Suppose that the statement holds for each formula smaller than β.

3. $\beta = (\beta_1 + \beta_2)$. It is easy to see that for each $\bar{a} \in A^\ell$ and function $f : A^\ell \to \mathbb{N}$: $T_\beta(f)(\bar{a}) = T_{\beta_1}(f)(\bar{a}) + T_{\beta_2}(f)(\bar{a})$. Suppose $\text{supp}(f) \subseteq \text{supp}(g)$ and let $\bar{a} \in \text{supp}(T_\beta(f))$, or in other words, $T_\beta(f)(\bar{a}) > 0$. Then, for some β_i it holds that $T_{\beta_i}(f)(\bar{a}) > 0$. From the supposition we have that $T_{\beta_i}(g)(\bar{a}) > 0$ from which the statement follows.
4. $\beta = (\beta_1 \cdot \beta_2)$. It is easy to see that for each $\bar{a}$ in A^ℓ and function $f : A^\ell \to \mathbb{N}$: $T_\beta(f)(\bar{a}) = T_{\beta_1}(f)(\bar{a}) \cdot T_{\beta_2}(f)(\bar{a})$. Suppose $\text{supp}(f) \subseteq \text{supp}(g)$ and let $\bar{a}$ be such that $T_\beta(f)(\bar{a}) > 0$. Then $T_{\beta_i}(f)(\bar{a}) > 0$ for both β_i. From the supposition we have that $T_{\beta_i}(g)(\bar{a}) > 0$ for both β_i and the statement holds.
5. $\beta = \Sigma y.\, \delta(y, \bar{x}, h)$. Here we extend the grammar slightly to allow constants, and we use the notation $\delta[a/y]$ to denote the formula obtained by replacing each instance of y by the constant a. It can be seen that $T_\beta(f)(\bar{a}) = \sum_{a \in A} T_{\delta[a/y]}(f)(\bar{a})$. Suppose $\text{supp}(f) \subseteq \text{supp}(g)$ and let $\bar{a}$ be such that $T_\beta(f)(\bar{a}) > 0$. Then for some $a \in A$ we have $T_{\delta[y/a]}(f)(\bar{a}) > 0$. The statement now follows as in the case 3.
6. $\beta = \Pi y.\, \delta(y, \bar{x}, h)$. It can be seen that $T_\beta(f)(\bar{a}) = \prod_{a \in A} T_{\delta[a/y]}(f)(\bar{a})$. The statement follows using the same argument from cases 4 and 5.

This covers all possible cases for β and we finish the proof of the statement for FQFO(FO).

For RQFO(FO). The only additional case is where $\beta = [\mathbf{lsfp}\, \delta(\bar{y}, h')]$ for some subtuple $\bar{y}$ of $\bar{x}$. We have that β does not mention h, and so, the statement follows directly as we showed in the previous part of the proof.

Proof of Theorem VI.3

Given the definition of the semantics of RQFO(FO), it is clear that a fixed formula $[\mathbf{lsfp}\, \beta(\bar{x}, h)]$ can be evaluated in polynomial time, from which it is possible to conclude that each fixed formula in RQFO(FO) can be evaluated in polynomial time. Thus, to prove that RQFO(FO) captures FP, we only need to prove the second condition in Definition IV.1.

Let f be a function in FP. We address the case when f is defined for the encodings of the structures of a relational signature $\mathbf{R} = \{E(\cdot,\cdot)\}$, as the proof for an arbitrary signature is analogous. Let M be a deterministic polynomial-time TM with a working tape and an output tape, such that the output of M on input $\text{enc}(\mathfrak{A})$ is $f(\text{enc}(\mathfrak{A}))$ for each $\mathbf{R}$-structure $\mathfrak{A}$. We assume that $M = (Q, \{0,1\}, q_0, \delta)$, where $Q = \{q_0, \ldots, q_\ell\}$, and $\delta : Q \times \{0, 1, \mathtt{B}, \vdash\} \to Q \times \{0, 1, \mathtt{B}, \vdash\} \times \{\leftarrow, \rightarrow\} \times \{0, 1, \varnothing\}$ is a partial function. In particular, the tapes of M are infinite to the right so the symbol $\vdash$ is used to indicate the first position in each tape, and M does not have any final states, as it produces an output for each input. Moreover, the only allowed operations in the output tape are: (1) writing 0 and moving the head one cell to the right, (2) writing 1 and moving the head one cell to the right, or (3) doing nothing. These operations are represented by the set $\{0, 1, \varnothing\}$. Finally, assume that M, on input $\text{enc}(\mathfrak{A})$ with domain $A = \{1, \ldots, n\}$, executes exactly n^k steps for some $k \geqslant 1$.

We construct a formula α in an extension of the grammar of RQFO(FO) such that $[\![\alpha]\!](\mathfrak{A}) = f(\text{enc}(\mathfrak{A}))$ for each $\mathbf{R}$-structure $\mathfrak{A}$. This extension allows defining the operator **lsfp** for multiple functions, analogously to the notion of simultaneous LFP [29]. Let $\bar{x} = (x_1, \ldots, x_k)$ and $\bar{t} = (t_1, \ldots, t_k)$. Then α is defined as:

$$
\begin{aligned}
\alpha = \Sigma \bar{t}.\, [\mathbf{lsfp}\, \mathit{out}(\bar{t}) :\, & \alpha_{T_0}(\bar{t}, \bar{x}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \alpha_{T_1}(\bar{t}, \bar{x}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \alpha_{T_\mathtt{B}}(\bar{t}, \bar{x}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \alpha_{T_\vdash}(\bar{t}, \bar{x}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \alpha_{h}(\bar{t}, \bar{x}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \alpha_{\hat{h}}(\bar{t}, \bar{x}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \alpha_{s_{q_0}}(\bar{t}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \vdots\\
& \alpha_{s_{q_\ell}}(\bar{t}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}),\\
& \alpha_{\mathit{out}}(\bar{t}, T_0, T_1, T_\mathtt{B}, T_\vdash, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out})] \cdot \text{last}(\bar{t}).
\end{aligned}
$$

Function T_0 is used to indicate whether the content of a cell of the working tape is 0 at some point of time, that is, $T_0(\bar{t}, \bar{x}) > 0$ if the cell at position $\bar{x}$ of the working tape contains the symbol 0 at step $\bar{t}$, and $T_0(\bar{t}, \bar{x}) = 0$ otherwise. Functions T_1, $T_\mathtt{B}$ and $T_\vdash$ are defined analogously. Function h is used to indicate whether the head of the working tape is in some position at some point of time, that is, $h(\bar{t}, \bar{x}) > 0$ if the head of the working tape is at position $\bar{x}$ at step $\bar{t}$, and $h(\bar{t}, \bar{x}) = 0$ otherwise. Function $\hat{h}$ is used to indicate whether the head of the working tape is **not** in some position at some point of time, that is, $\hat{h}(\bar{t}, \bar{x}) > 0$ if the head of the working tape is **not** at position $\bar{x}$ at step $\bar{t}$, and $h(\bar{t}, \bar{x}) = 0$ otherwise. For each $i \in \{0, \ldots, \ell\}$, function s_{q_i} is used to indicate whether the TM M is in state q_i at some point of time, that is, $s_{q_i}(\bar{t}) > 0$ if the TM M is in

state q_i at step $\bar{t}$, and $s_{q_i}(\bar{t},\bar{x}) = 0$ otherwise. Finally, function out stores the output of the TM M; in particular, $out(\bar{t})$ is the value returned by M when $\bar{t}$ is the last step (that is, when $\mathrm{last}(\bar{t})$ holds).

Formulas α_{T_0}, α_{T_1}, $\alpha_{T_\mathtt{B}}$ and $\alpha_{T_\vdash}$ are defined as follows, assuming that $\bar{y} = (y_1, \ldots, y_k)$:

$$\begin{aligned}
\alpha_{T_0}(\bar{t},\bar{x},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = {} & (\mathrm{first}(\bar{t}) \wedge \exists\bar{y}(\mathrm{first}(y_1,\ldots,y_{k-2}) \wedge \neg E(y_{k-1},y_k) \wedge \mathrm{succ}(\bar{y},\bar{x})))+ \\
& \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot \hat{h}(\bar{t}',\bar{x}) \cdot T_0(\bar{t}',\bar{x}))+ \\
& \bigdotplus_{\delta(q,a)=(q',0,op,v)} \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot h(\bar{t}',\bar{x}) \cdot T_a(\bar{t}',\bar{x}) \cdot s_q(\bar{t}')), \\
\alpha_{T_1}(\bar{t},\bar{x},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = {} & (\mathrm{first}(\bar{t}) \wedge \exists\bar{y}(\mathrm{first}(y_1,\ldots,y_{k-2}) \wedge E(y_{k-1},y_k) \wedge \mathrm{succ}(\bar{y},\bar{x})))+ \\
& \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot \hat{h}(\bar{t}',\bar{x}) \cdot T_1(\bar{t}',\bar{x}))+ \\
& \bigdotplus_{\delta(q,a)=(q',1,op,v)} \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot h(\bar{t}',\bar{x}) \cdot T_a(\bar{t}',\bar{x}) \cdot s_q(\bar{t}')), \\
\alpha_{T_\mathtt{B}}(\bar{t},\bar{x},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = {} & (\mathrm{first}(\bar{t}) \wedge \exists\bar{y}\exists\bar{y}'(\mathrm{first}(y_1,\ldots,y_{k-2}) \wedge \mathrm{last}(y_{k-1},y_k) \wedge \mathrm{succ}(\bar{y},\bar{y}') \wedge \bar{y}' < \bar{x}))+ \\
& \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot \hat{h}(\bar{t}',\bar{x}) \cdot T_\mathtt{B}(\bar{t}',\bar{x}))+ \\
& \bigdotplus_{\delta(q,a)=(q',\mathtt{B},op,v)} \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot h(\bar{t}',\bar{x}) \cdot T_a(\bar{t}',\bar{x}) \cdot s_q(\bar{t}')), \\
\alpha_{T_\vdash}(\bar{t},\bar{x},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = {} & (\mathrm{first}(\bar{t}) \wedge \mathrm{first}(\bar{x}))+ \\
& \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot \hat{h}(\bar{t}',\bar{x}) \cdot T_\vdash(\bar{t}',\bar{x}))+ \\
& \bigdotplus_{\delta(q,a)=(q',\vdash,op,v)} \Sigma\bar{t}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot h(\bar{t}',\bar{x}) \cdot T_a(\bar{t}',\bar{x}) \cdot s_q(\bar{t}')).
\end{aligned}$$

Formulas α_h and $\alpha_{\hat{h}}$ are defined as:

$$\begin{aligned}
\alpha_h(\bar{t},\bar{x},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = {} & \\
& (\mathrm{first}(\bar{t}) \wedge \mathrm{succ}(\bar{t},\bar{x}))+ \\
& \bigdotplus_{\delta(q,a)=(q',b,\leftarrow,v)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot \mathrm{succ}(\bar{x},\bar{x}') \cdot T_a(\bar{t}',\bar{x}') \cdot h(\bar{t}',\bar{x}') \cdot s_q(\bar{t}'))+ \\
& \bigdotplus_{\delta(q,a)=(q',b,\rightarrow,v)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot \mathrm{succ}(\bar{x}',\bar{x}) \cdot T_a(\bar{t}',\bar{x}') \cdot h(\bar{t}',\bar{x}') \cdot s_q(\bar{t}')), \\
\alpha_{\hat{h}}(\bar{t},\bar{x},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = {} & \\
& (\mathrm{first}(\bar{t}) \wedge \neg\mathrm{succ}(\bar{t},\bar{x}))+ \\
& \bigdotplus_{\delta(q,a)=(q',b,\leftarrow,v)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,\Sigma\bar{x}''.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot T_a(\bar{t}',\bar{x}') \cdot h(\bar{t}',\bar{x}') \cdot s_q(\bar{t}') \cdot \mathrm{succ}(\bar{x}'',\bar{x}') \cdot (\bar{x} \neq \bar{x}''))+ \\
& \bigdotplus_{\delta(q,a)=(q',b,\rightarrow,v)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,\Sigma\bar{x}''.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot T_a(\bar{t}',\bar{x}') \cdot h(\bar{t}',\bar{x}') \cdot s_q(\bar{t}') \cdot \mathrm{succ}(\bar{x}',\bar{x}'') \cdot (\bar{x} \neq \bar{x}'')).
\end{aligned}$$

Formula α_{q_0} is defined as:

$$\begin{aligned}
\alpha_{q_0}(\bar{t},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = {} & \mathrm{first}(\bar{t})+ \\
& \bigdotplus_{\delta(q,a)=(q_0,b,op,v)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot T_a(\bar{t}',\bar{x}') \cdot h(\bar{t}',\bar{x}') \cdot s_q(\bar{t}')).
\end{aligned}$$

Moreover, for every $i \in \{1,\ldots,\ell\}$, formula α_{q_i} is defined as:

$$\alpha_{q_i}(\bar{t},T_0,T_1,T_\mathtt{B},T_\vdash,h,\hat{h},s_{q_0},\ldots,s_{q_\ell},out) = \bigdotplus_{\delta(q,a)=(q_i,b,op,v)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot T_a(\bar{t}',\bar{x}') \cdot h(\bar{t}',\bar{x}') \cdot s_q(\bar{t}')).$$

Finally, formula α_{out} is defined as:

$$\begin{aligned}\alpha_{out}(\bar{t}, T_0, T_1, T_{\mathtt{B}}, T_{\vdash}, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, out) = \\ \bigdotplus_{\delta(q,a)=(q',b,op,0)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}', \bar{t}) \cdot h(\bar{t}', \bar{x}') \cdot T_a(\bar{t}', \bar{x}') \cdot s_q(\bar{t}') \cdot 2 \cdot out(\bar{t}')) + \\ \bigdotplus_{\delta(q,a)=(q',b,op,1)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}', \bar{t}) \cdot h(\bar{t}', \bar{x}') \cdot T_a(\bar{t}', \bar{x}') \cdot s_q(\bar{t}') \cdot (2 \cdot out(\bar{t}') + 1)) + \\ \bigdotplus_{\delta(q,a)=(q',b,op,\varnothing)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}', \bar{t}) \cdot h(\bar{t}', \bar{x}') \cdot T_a(\bar{t}', \bar{x}') \cdot s_q(\bar{t}') \cdot out(\bar{t}')).\end{aligned}$$

Clearly, at each iteration of the LSFP operator, the tuple $\bar{t}$ represents the step the machine is currently in. From the construction of the formula, and since the machine is deterministic, it can be seen that in each function $g \in \{T_0, T_1, T_{\mathtt{B}}, T_{\vdash}, h, \hat{h}\}$, at the $\bar{a}$-th iteration of the LSFP operator, it holds that $g(\bar{a}, \bar{b}) \leqslant 1$ for each $\bar{b} \in A^k$, that $g(\bar{a}+1, \bar{b}) = 0$ for each $\bar{b} \in A^k$. Also, at the $\bar{a}$-th iteration, $g(\bar{a}) \leqslant 1$ and $g(\bar{a}+1) = 0$ for each $g \in \{s_{q_1}, \ldots, s_{q_\ell}\}$. From this, we have that at each iteration $\bar{a}$ of the operator, $out(\bar{a})$ is equal to either $2 \cdot out(\bar{a}-1)$, $2 \cdot out(\bar{a}-1)+1$, or $out(\bar{a}-1)$, which represents precisely the value in the output tape at each step of M running on input $\mathrm{enc}(\mathfrak{A})$. From this argument, it can be seen that $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$.

To conclude the proof, we show that for each formula α in the previously defined extension of RQFO(FO), there exists an equivalent formula confirming to the grammar of RQFO(FO) defined in Section VI. It suffices to consider a formula α of the form

$$\alpha(\bar{x}_1) = [\mathbf{lsfp}\, f_1(\bar{x}_1) : \alpha_1(\bar{x}_1, f_1, \ldots, f_n), \alpha_2(\bar{x}_2, f_1, \ldots, f_n), \ldots, \alpha_n(\bar{x}_n, f_1, \ldots, f_n)],$$

and show an equivalent formula defined by a LSFP operator which uses one formula less in its definition.

We construct the equivalent formula as follows. We use a new function symbol f with arity $|\bar{x}_1| + |\bar{x}_2|$. For every $i \in \{1, \ldots, n\}$, let α_i' be the formula obtained by performing the following replacements in α_i:

$$\begin{aligned} f_1(\bar{y}_1) \text{ is replaced by } \Sigma\bar{y}_2.\, f(\bar{y}_1, \bar{y}_2) \cdot [\mathrm{first}(\bar{y}_1) \cdot \mathrm{last}(\bar{y}_2) + (\neg\, \mathrm{first}(\bar{y}_1)) \cdot \mathrm{first}(\bar{y}_2)], \\ f_2(\bar{y}_2) \text{ is replaced by } \Sigma\bar{y}_1.\, f(\bar{y}_1, \bar{y}_2) \cdot [\mathrm{first}(\bar{y}_1) \cdot \mathrm{first}(\bar{y}_2) + \mathrm{last}(\bar{y}_1) \cdot (\neg\, \mathrm{first}(\bar{y}_2))]. \end{aligned}$$

Moreover, let β be a formula defined as:

$$\begin{aligned}\beta(\bar{x}_1, \bar{x}_2) = \alpha_1'(\bar{x}_1) \cdot (\mathrm{first}(\bar{x}_1) \cdot \mathrm{last}(\bar{x}_2) + (\neg\, \mathrm{first}(\bar{x}_1)) \cdot \mathrm{first}(\bar{x}_2)) + \\ \alpha_2'(\bar{x}_2) \cdot (\mathrm{first}(\bar{x}_1) \cdot \mathrm{first}(\bar{x}_2) + \mathrm{last}(\bar{x}_1) \cdot (\neg\, \mathrm{first}(\bar{x}_2))).\end{aligned}$$

It can be seen that all values of f_1, besides the first one, are stored in the first assignment of $\bar{x}_2$, while the first value of f_1 is stored in the last assignment of $\bar{x}_2$. Moreover, all values of f_2, besides the first one, are stored in the last assignment of $\bar{x}_1$, while the first value of f_2 is stored in the first assignment of $\bar{x}_1$. We use formula β to define the following formula:

$$\begin{aligned}\alpha'(\bar{x}_1) = \Sigma\bar{x}_2.\, [\mathbf{lsfp}\, f(\bar{x}_1, \bar{x}_2) : \beta(\bar{x}_1, \bar{x}_2, f, f_3, \ldots, f_n), \\ \alpha_3'(\bar{x}_3, f, f_3, \ldots, f_n), \ldots, \alpha_n'(\bar{x}_n, f, f_3, \ldots, f_n)] \cdot (\mathrm{first}(\bar{x}_1) \cdot \mathrm{last}(\bar{x}_2) + (\neg\, \mathrm{first}(\bar{x}_1)) \cdot \mathrm{first}(\bar{x}_2))\end{aligned}$$

It is not difficult to see that $\alpha'(\bar{x}_1)$ is equivalent to $\alpha(\bar{x}_1)$, which concludes the proof.

Proof of Theorem VI.4

TQFO(FO) can be computed in #L. Let $\mathbf{R}$ be some relational signature. Let α be a formula in TQFO(FO). We will construct a nondeterministic logspace algorithm M_α that on input $\mathrm{enc}(\mathfrak{A})$, where a first-order assignment v is being stored in memory, accepts in $[\![\alpha]\!](\mathfrak{A}, v)$ paths. Suppose the domain of $\mathfrak{A}$ is $A = \{1, \ldots, n\}$. The algorithm needs $c \cdot \log_2(n)$ bits of memory to store v, where c is the total number of first-order variables in α. If $\alpha = \varphi$, we check if $(\mathfrak{A}, v) \models \varphi$ in deterministic logarithmic space, and accept if and only if it does. If $\alpha = s$, we generate s branches and accept in all of them. If $\alpha = (\alpha_1 + \alpha_2)$, we simulate M_{α_1} and M_{α_2} on separate branches. If $\alpha = (\alpha_1 \cdot \alpha_2)$, we simulate α_1 and if it accepts, instead of doing so, we simulate α_2. If $\alpha = \Sigma x.\, \beta$, for each $a \in A$ we generate a different branch where we simulate M_β while storing $v[a/x]$. If $\alpha = \Pi x.\, \beta$, we simulate M_β while storing $v[1/n]$, and on each accepting branch, instead of accepting we replace the assignment on x to 2, to simulate M_β while storing $v[2/x]$, and so on. If $\alpha = [\mathbf{path}\, \varphi(\bar{x}, \bar{y})]$ where φ is an FO formula, we simulate the #L procedure that counts the number of paths for a graph of a given size. This procedure starts by setting $\bar{a} = v(\bar{x})$. On each iteration, nondeterministically chooses an assignment $\bar{a}$ for $\bar{x}$, continues if $(\mathfrak{A}, v) \models \varphi(\bar{a}', \bar{a})$ where $\bar{a}'$ is the previously chosen value for $\bar{a}$, and rejects otherwise. If at any point we obtain that $\bar{a} = v(\bar{y})$, we generate an accepting branch, and continue simulating the procedure in the current branch. We simulate $n^{|\bar{x}|}$ iterations of the procedure, and this generates exactly $[\![[\mathbf{path}\, \varphi(\bar{x}, \bar{y})]]\!](\mathfrak{A}, v)$ accepting branches. This ends the construction of the algorithm. Consider f as the #L function associated to this procedure and we have that for each finite $\mathbf{R}$-structure $\mathfrak{A}$: $f(\mathrm{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$.

#L **can be modelled in TQFO(FO).** Let f be a function in #L and let M be a nondeterministic logspace machine such that $\mathbf{acc}_M(\mathrm{enc}(\mathfrak{A})) = f(\mathrm{enc}(\mathfrak{A}))$. We assume that M has only one accepting state and upon accepting it immediately stops. Moreover, we assume that there exists only one accepting configuration altogether. We make use of transitive closure logic (TC) to simplify our proof [16]. We have that TC captures NL [19], so there exists a formula such that $\mathfrak{A} \models \varphi$ if and only if M accepts $\mathrm{enc}(\mathfrak{A})$. This formula can be expressed as:

$$\varphi = \exists \bar{u} \exists \bar{z} (\mathrm{first}(\bar{u}) \wedge \psi_{\mathbf{acc}}(\bar{z}) \wedge [\mathbf{tc}_{\bar{x},\bar{y}}\, \psi_{\mathbf{next}}(\bar{x},\bar{y})](\bar{u},\bar{z})),$$

where $\psi_{\mathbf{acc}}(\bar{z})$ is an FO formula that expresses that $\bar{z}$ is an accepting configuration, and $\psi_{\mathbf{next}}(\bar{x},\bar{y})$ is an FO formula that expresses that $\bar{y}$ is the next configuration from $\bar{x}$ [16]. Here, there is a 1-1 correspondence between configurations of M and assignments to $\bar{z}$. As a consequence, given a structure $\mathfrak{A}$, and a first-order assignment v to $\mathfrak{A}$ where $v(\bar{x})$ is the starting configuration and $v(\bar{y})$ is the sole accepting configuration, the value of $[\![[\mathbf{path}\ \psi_{\mathbf{next}}(\bar{x},\bar{y})]]\!](\mathfrak{A}, v)$ is equal to $\mathbf{acc}_M(\mathrm{enc}(\mathfrak{A}))$. Finally, we define the TQFO(FO) formula

$$\alpha = \Sigma \bar{u}.\, \Sigma \bar{z}.\, (\mathrm{first}(\bar{u}) \cdot \psi_{\mathbf{acc}}(\bar{z}) \cdot [\mathbf{path}\ \psi_{\mathbf{next}}(\bar{u},\bar{z})]),$$

which satisfies $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$ for each structure $\mathfrak{A}$. This concludes the proof.