DESCRIPTIVE COMPLEXITY FOR COUNTING COMPLEXITY CLASSES

MARCELO ARENAS, MARTIN MUÑOZ, AND CRISTIAN RIVEROS

Pontificia Universidad Católica de Chile
e-mail address: marenas@ing.puc.cl

Pontificia Universidad Católica de Chile
e-mail address: mmunos@uc.cl

Pontificia Universidad Católica de Chile
e-mail address: 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.

1. Introduction

The goal of descriptive complexity is to measure the complexity of a problem in terms of the logical constructors needed to express it [26]. The starting point of this branch of complexity theory is Fagin's theorem [13], 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 [19] for a survey) and descriptive complexity has found a variety of applications in different areas [26,33]. For instance, Fagin's theorem was the key ingredient to define the class MaxSNP [37], which was later shown to be a fundamental class in the study of hardness of approximation [5]. 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 [37].

2012 ACM CCS: [**Theory of computation**]: Logic—Finite Model Theory [**Theory of computation**]: Computational complexity and cryptography—Complexity classes.

Key words and phrases: Function complexity clases, descriptive complexity.

Counting problems differ from decision problems in that the value of a function has to be computed. 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 [4, 16, 22]. 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, other function complexity classes have emerged from the need to understand the complexity of some computation problems for which little can be said if their decision counterparts are considered. This is the case of the class #P, a counting complexity class introduced in [40] 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 [40] are difficult, namely, #P-complete. Starting from #P, many more natural counting complexity classes have been defined, such as #L, SpanP and GapP [16, 22].

Although counting problems play a prominent role in computational complexity, descriptive complexity for this type of problems 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 [6, 39]. 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) [7], a general logical framework that combines Boolean formulae (e.g. in FO or SO) with operations over a fixed semiring (e.g. $\mathbb{N}$). Specifically, we propose a restriction of WL over natural numbers, called Quantitative Second Order Logics (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 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 Section 5). 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

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

inside #P, called ΣQSO(FO)-hierarchy, and we show that it is closely related to the FO-hierarchy introduced in [39]. Looking inside the ΣQSO(FO)-hierarchy, we propose the class $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ 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 [18], 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.

It is important to mention that this paper is an extension of the conference article [3]. In this version, we have included the complete proofs of all the results in the paper, paying special attention in showing the main techniques used to establish them. Besides, we have simplified some of the terminology used in [3], with the goal of presenting the main notions studied in the paper in a simple way.

Organisation. The main terminology used in the paper is given in Section 2. Then the logical framework is introduced in Section 3, and it is used to capture standard counting complexity classes in Section 4. The structure of #P is studied in Section 5. Section 6 is devoted to define recursion in QSO, and to show how to capture classes below FP. Finally, we give some concluding remarks in Section 7.

2. Preliminaries

2.1. **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 $\text{arity}(R_i)$. 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 $R_i^{\mathfrak{A}} \subseteq A^{\text{arity}(R_i)}$ for every $i \in \{1, \ldots, k\}$. Further, 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. We denote by $\text{OrdStruct}[\mathbf{R}]$ the class of all finite ordered $\mathbf{R}$-structures. In this paper we only consider finite ordered structures, so we will usually omit the word finite or ordered when referring to them.

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 $\text{arity}(X)$. 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 $\text{arity}(R)$, $\top$ 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 $\text{arity}(X)$, 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 $A^{\text{arity}(X)}$. 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 $B \subseteq A^{\text{arity}(X)}$, 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 this paper, we consider several fragments or extensions of SO like first-order logic (FO), least fixed point logic (LFP) and partial fixed point logic (PFP) [33]. 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 $\mathcal{L}_2 \nsubseteq \mathcal{L}_1$.

2.2. **Counting complexity classes.** We consider several counting complexity classes in this paper, some of the are recalled here (see [16,21]). 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 $\#\text{accept}_M(x)$ 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 $f(x) = \#\text{accept}_M(x)$ for every input x, while #L is the class of functions f for which there exists a logarithmic-space NTM M such that $f(x) = \#\text{accept}_M(x)$ 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 #P $\subseteq$ SpanP, and this inclusion is believed to be strict.

3. 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) [7] 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 3.1 for a more detailed discussion).

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

Table 1: The semantics of QSO formulae.

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:

$$\alpha \;:=\; \varphi \;\mid\; s \;\mid\; (\alpha + \alpha) \;\mid\; (\alpha \cdot \alpha) \;\mid \Sigma x.\, \alpha \;\mid\; \Pi x.\, \alpha \;\mid\; \Sigma X.\, \alpha \;\mid\; \Pi X.\, \alpha \tag{3.1}$$

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.

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 1. 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 $\mathfrak{A}$.

Example 3.1. Let $\mathbf{G} = \{E(\cdot,\cdot), <\}$ be the vocabulary for graphs and $\mathfrak{G}$ be an ordered $\mathbf{G}$-structure encoding a non-directed 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:

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

where $\mathrm{clique}(X) := \forall x.\, \forall y.\, ((X(x) \wedge X(y) \wedge x \neq y) \rightarrow E(x,y))$. 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 3.2. For an 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:

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

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 class #P [40].

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, namely, 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 or extensions 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 3.1, 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 sum 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 3.1 are formulae of ΣQSO, while α_3 in Example 3.2 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 QSO($\mathscr{L}$) to be the fragment of QSO obtained by restricting φ in (3.1) to be a formula in $\mathscr{L}$. Moreover, we also restrict other fragments of QSO by using the same idea. For example, we define QFO(FO) to be the fragment of QFO obtained by restricting φ in (3.1) to be an FO-formula, and likewise for ΣQSO(FO).

In the following section, we use different fragments or extensions 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.

3.1. **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) [7]. 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 [8] 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 6) which differ from previous approaches in WL.

Another approach that resembles QSO are logics with counting [11, 20, 27, 33], 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 [39] and [6] is of particular interest for our research. In [39], 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 [39] 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 [39], they proved several results about capturing counting complexity classes which are relevant for our work. We discuss and use these results in Sections 4

and 5. 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 [39] 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 6). Furthermore, we show in the next section how to capture some classes that cannot be captured by following the approach in [39].

4. 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 [26] from decision problems to counting 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 $\mathbf{R}$-structure with a domain $A = \{a_1, \ldots, a_n\}$ and $a_1 < a_2 < \ldots < a_n$. For every $i \in \{1, \ldots, k\}$, define the encoding of $R_i^{\mathfrak{A}}$, denoted by $\text{enc}(R_i^{\mathfrak{A}})$, as the following binary string. Assume that $\ell = \text{arity}(R_i)$ and consider an enumeration of the ℓ-tuples over A in the lexicographic order induced by $<$. Then let $\text{enc}(R_i^{\mathfrak{A}})$ be a binary string of length n^ℓ such that the i-th bit of $\text{enc}(R_i^{\mathfrak{A}})$ is 1 if the i-th tuple in the previous enumeration belongs to $R_i^{\mathfrak{A}}$, and 0 otherwise. Moreover, define the encoding of $\mathfrak{A}$, denoted by $\text{enc}(\mathfrak{A})$, as the string [33]:

$$0^n\, 1\, \text{enc}(R_1^{\mathfrak{A}}) \cdots \text{enc}(R_k^{\mathfrak{A}}).$$

We can now formalize the notion of capturing a counting complexity class.

Definition 4.1. 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 $[\![\alpha]\!](\mathfrak{A}) = f(\text{enc}(\mathfrak{A}))$ for every $\mathfrak{A} \in \text{OrdStruct}[\mathbf{R}]$.
(2) for every $f \in \mathscr{C}$, there exists $\alpha \in \mathscr{F}$ such that $f(\text{enc}(\mathfrak{A})) = [\![\alpha]\!](\mathfrak{A})$ for every $\mathfrak{A} \in \text{OrdStruct}[\mathbf{R}]$.

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 4.1, 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 [26, 33], and it has also been used in the previous work on capturing complexity classes of functions [6, 39].

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 [4]. Since #P has a strong similarity with NP, one could expect a "Fagin-like" Theorem [13] for this class. Actually, in [39] it was shown that the class #FO captures #P. In our setting, the class #FO is contained in ΣQSO(FO), which also captures #P as expected.

Proposition 4.2. ΣQSO(FO) *captures* #P *over ordered structures.*

Proof. We briefly explain that the two conditions of Definition 4.1 are satisfied. First, for condition (2) Saluja et al. proved that #P = #FO [39]. Hence, given that every function in

#FO can be trivially defined as a formula in ΣQSO(FO) (see Section 3.1) then condition (2) holds. For condition (1), let $\alpha \in \Sigma\text{QSO(FO)}$ over some signature $\mathbf{R}$. Given a FO formula φ, checking whether $\mathfrak{A} \models \varphi$ can be done in deterministic polynomial time on the size of $\mathfrak{A}$ and the constant function s can be trivially simulated in #P. These facts, together with the closures under exponential sum and polynomial product of #P [16], suffice to show that the function represented by α is in #P. □

By following the same approach as [39], Compton and Grädel [6] show that #(∃SO) captures SpanP, where ∃SO is the existential fragment of SO. As one could expect, if we parametrize ΣQSO with ∃SO, we can also capture SpanP.

Proposition 4.3. ΣQSO(∃SO) *captures* SpanP *over ordered structures.*

Proof. To prove the condition (2), we use the fact that SpanP = #(∃SO). The condition holds using the same argument as in Proposition 4.2. For condition (1), notice that given an ∃SO formula φ, checking whether $\mathfrak{A} \models \varphi$ can be done in non-deterministic polynomial time on the size of $\mathfrak{A}$ [12]. Therefore, a SpanP machine for φ will simulate the non-deterministic polynomial time machine and produce the same string as output in each accepting non-deterministic run. Furthermore, the constant function s for some $s \in \mathbb{N}$ can be trivially simulated in SpanP and, thus, condition (1) holds analogously to Proposition 4.2 since SpanP is also closed under exponential sum and polynomial product [34]. □

Can we capture FP by using $\#\mathscr{L}$ for some fragment $\mathscr{L}$ of SO? A first attempt could be based on the use of a fragment $\mathscr{L}$ of SO that capture either P or NL [18]. Such an approach fails as $\#\mathscr{L}$ can encode #P-complete problems in both cases; in the first case, one can encode the problem of counting the number of satisfying assignments of a Horn propositional formula, while in the second case one can encode the problem of counting the number of satisfying assignments of a 2-CNF propositional formula. A second attempt could be based then 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 [39]. One last attempt could be based on disallowing the use of second-order free variables in #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 [39]. On the other hand, if we consider our framework and move out from ΣQSO, we have other alternatives 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 4.4. QFO(LFP) *captures* FP *over ordered structures.*

Proof. In this and the following proofs, we will reuse the symbol $<$ to denote the lexicographic order over same-sized tuples. Formally, for $\bar{x} = (x_1, \ldots, x_m)$ and $\bar{y} = (y_1, \ldots, y_m)$ we denote by $\bar{x} < \bar{y}$ the formula:

$$\bigvee_{i=1}^{m} \bigwedge_{j=1}^{i-1} (x_j = y_j \wedge x_i < y_i).$$

Similarly, we use $\bar{x} = \bar{y}$ to denote equality between tuples and $\bar{x} \leqslant \bar{y}$ to denote $\bar{x} < \bar{y} \vee \bar{x} = \bar{y}$. We will also use some syntactic sugar in QSO to simplify formulas. Specifically, we will use the *conditional count* symbol $(\varphi \mapsto \alpha)$ defined as $(\varphi \cdot \alpha) + \neg\varphi$ for any Boolean formula φ and

any quantitative formula α. Note that for each $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$, and each first-order (second-order) assignment v (V) over $\mathfrak{A}$:

$$[\![(\varphi \mapsto \alpha)]\!](\mathfrak{A}, v, V) = \begin{cases} [\![\alpha]\!](\mathfrak{A}, v, V) & \text{if } (\mathfrak{A}, v, V) \models \varphi, \\ 1 & \text{otherwise.} \end{cases}$$

Furthermore, we use $|\mathfrak{A}|$ to denote the size of an $\mathbf{R}$-structure $\mathfrak{A}$. Now we prove Theorem 4.4. For condition (1), recall that checking whether $\mathfrak{A} \models \varphi$ for any LFP formula φ can be done in deterministic polynomial time on the size of $\mathfrak{A}$ [23]. Furthermore, it is easy to check that FP is closed under polynomial sum and multiplication. We conclude then that any formula in QFO(LFP) can be computed in FP. For condition (2), let $\mathbf{R}$ be a signature, $f \in \text{FP}$ and $\ell \in \mathbb{N}$ such that $\log_2(f(\text{enc}(\mathfrak{A}))) \leqslant |\mathfrak{A}|^\ell$ for every $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$ (i.e. $|\mathfrak{A}|^\ell$ is an upper bound for the output size of f over $\mathfrak{A}$). Consider the language:

$$L = \{(\mathfrak{A}, \bar{a}) \mid \bar{a} \in A^l \text{ and the } \bar{a}\text{-th bit of } f(\text{enc}(\mathfrak{A})) \text{ is } 1\}.$$

where $\bar{a}$ encodes a number by following the lexicographic order over A^l. Clearly, the language L is in P and by [23] there exists a formula $\Phi(\bar{x})$ in LFP such that $\mathfrak{A} \models \Phi(\bar{a})$ if, and only if, $(\mathfrak{A}, \bar{a}) \in L$. We use then the following formula to encode f:

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

Note that the subformula $\Pi\bar{y}.\, (\bar{y} < \bar{x}) \mapsto 2$ takes the value 2^m if there exist m tuples in A^ℓ that are smaller than $\bar{x}$. Adding these values for each $\bar{a} \in A^\ell$ gives exactly $f(\text{enc}(\mathfrak{A}))$. In other words, $\Phi(\bar{x})$ simulates the behavior of the FP-machine and the formula α reconstructs the binary output bit by bit. Then α is in QFO(LFP) and $[\![\alpha]\!](\mathfrak{A}) = f(\text{enc}(\mathfrak{A}))$. □

At this point it is natural to ask whether one can extend the previous idea to capture FPSPACE [32], the class of functions computable in polynomial space. Of course, for capturing this class one needs 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 ΣQSO is not enough for the quantitative layer of a logic for FPSPACE. In fact, by considering second-order product we obtain the fragment QSO(PFP) that captures FPSPACE.

Theorem 4.5. QSO(PFP) *captures* FPSPACE *over ordered structures.*

Proof. For the first condition of Definition 4.1, notice that each PFP formula can be evaluated in deterministic polynomial space, the constant function s can be trivially simulated in FPSPACE, and FPSPACE is closed under exponential sum and multiplication. This suffices to show that the condition holds. For the second condition, the proof is similar than in Theorem 4.4. Let $f \in \text{FPSPACE}$ defined over some $\mathbf{R}$ and $\ell \in \mathbb{N}$ such that $\log_2(f(\text{enc}(\mathfrak{A}))) \leqslant 2^{|\mathfrak{A}|^\ell}$ for every $\mathfrak{A} \in \textsc{OrdStruct}[\mathbf{R}]$ (i.e. $2^{|\mathfrak{A}|^\ell}$ is an upper bound for the output size). Let X be a second-order variable of arity ℓ. Consider the linear order induced by $<$ over predicates of arity ℓ which can be defined by the following 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 $2^{|\mathfrak{A}|^\ell}$-bits where the empty relation represents 0 and the total-relation represents $2^{2^{|\mathfrak{A}|^\ell}} - 1$. Furthermore, we can use a relation X to index a position in the binary output of $f(\text{enc}(\mathfrak{A}))$ as follows. Define the language:

$$L = \{(\mathfrak{A}, B) \mid B \subseteq A^\ell \text{ and the } B\text{-th bit of } f(\text{enc}(\mathfrak{A})) \text{ is } 1\}.$$

Since L is in PSPACE, it can be specified in PFP [1] by a formula $\Phi(X)$ where the free variable X encodes relation B in L. Then, similar than the previous proof we define:

$$\alpha := \Sigma X.\, \Phi(X) \cdot \Pi Y.\, (\varphi_<(Y, X) \mapsto 2).$$

where $\Pi Y.\, (\varphi_<(Y, X) \mapsto 2)$ takes the value 2^m if there exist m predicates that are smaller than X and α reconstruct the output of $f(\text{enc}(\mathfrak{A}))$ by simulating f with $\Phi(X)$. Using an analogous argument, we conclude that $\alpha \in \text{QSO(PFP)}$ and $[\![\alpha]\!](\mathfrak{A}) = f(\text{enc}(\mathfrak{A}))$. □

From the proof of the previous theorem a natural question follows: what happens if we use first-order quantitative quantifiers and PFP? In [32], 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 4.6. QFO(PFP) *captures* FPSPACE(POLY) *over ordered structures.*

Proof. In this proof, both conditions are analogous to Theorem 4.4 and 4.5. For the first condition, each PFP formula φ can be evaluated in PSPACE and the class is closed under first sum and product. For the second condition, we use the same language L defined in the proof of Theorem 4.4, which in this case is in PSPACE. The same construction of α, which in turn is in QFO(PFP), is used to show that the condition holds. □

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

4.1. **Extending** QSO **to capture classes beyond counting.** There exist complexity classes that do not fit in our framework because either the output of a function in 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). 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 [34]. To overcome this limitation, GAPP was introduced in [15] 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 (3.1) and whose semantics is the same as in Table 1 except that the atomic formula s (i.e. a constant) comes from $\mathbb{Z}$. Similar than for QSO, we define the fragment $\Sigma\text{QSO}_\mathbb{Z}$ as the extension of ΣQSO with constants in $\mathbb{Z}$.

Example 4.7. Recall the setting of Example 3.1 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 4.8. $\Sigma\text{QSO}_{\mathbb{Z}}(\text{FO})$ *captures* GapP *over ordered structures.*

This is an interesting result that shows how robust and versatile is QSO for capturing different counting complexity classes even beyond $\mathbb{N}$.

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 [31] 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. Given that in [31] Krentel did not make the distinction between max and min, in [43] they defined the classes MaxP and MinP as the max and min version of the problems in OptP (i.e. OptP = MaxP $\cup$ 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 (3.1) 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 1) by maximizing or minimizing, respectively, instead of computing a sum or a product.

Example 4.9. Recall again the setting of Example 3.1 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 distinguished 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 4.9 is in MaxQSO. As one could expect, MaxQSO and MinQSO are the needed logics to capture MaxP and MinP.

Theorem 4.10. MaxQSO(FO) *and* MinQSO(FO) *capture* MaxP *and* MinP*, respectively, over ordered structures.*

Proof. It is straightforward to prove that MaxP can compute any FO-formula, is closed under first-order sum and product, and second-order maximization. Therefore, condition (1) in Definition 4.1 follows similar than in the previous characterizations. Furthermore, one can easily see that the same holds with MinQSO(FO). The proof for the other direction is similar than in [30] extended with the ideas of Theorem 4.4. Let $f \in$ MaxP be a

function defined over some signature $\mathbf{R}$ and $\ell \in \mathbb{N}$ such that $\lceil \log_2 f(\text{enc}(\mathfrak{A}))\rceil \leqslant |\mathfrak{A}|^\ell$ for each $\mathfrak{A} \in \text{OrdStruct}[\mathbf{R}]$. For $U \subseteq A^\ell$, we can interpret the encoding of U ($\text{enc}(U)$) as the binary encoding of a number with $|\mathfrak{A}|^\ell$-bits. We denote this value by $\text{val}(\text{enc}(U))$. Then, given $\mathfrak{A} \in \text{OrdStruct}[\mathbf{R}]$ and $U \subseteq A^\ell$, consider the problem of checking whether $f(\text{enc}(\mathfrak{A})) \geqslant \text{val}(\text{enc}(U))$. 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 and $\text{arity}(Y) = \ell$ such that $f(\text{enc}(\mathfrak{A})) \geqslant \text{val}(\text{enc}(U))$ if, and only if, $(\mathfrak{A}, v, V) \models \exists \bar{X}.\, \Phi(\bar{X}, Y)$ with $V(Y) = U$. Then we can describe f by the following MaxQSO formula:

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

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 $\text{enc}(U)$ 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 MaxQSO(FO) over $\mathbf{R}$ and $[\![\alpha]\!](\mathfrak{A}) = f(\text{enc}(\mathfrak{A}))$.

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

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

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. $2^{|\mathfrak{A}|^\ell}$). On the other hand, if $\Phi(\bar{X}, Y)$ holds, the subformula will produce $\text{val}(\text{enc}(U))$. Similar than for max, we conclude that α is in MinQSO(FO) and $[\![\alpha]\!](\mathfrak{A}) = f(\text{enc}(\mathfrak{A}))$. □

It is important to mention that a similar result, following the framework of [39], was proved in [30] for the class MaxPB (resp., MinPB) of problems in MaxP (resp., MinP) whose output value is polynomially bounded. Interestingly, Theorem 4.10 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 MaxQSO(FO) and MinQSO(FO), respectively.

5. Exploring the structure of #P through QSO

The class #P was introduced in [40] to prove that computing the permanent of a matrix, as defined in Example 3.2, is a #P-complete problem. As a consequence of this result many counting problems have been proved to be #P-complete [4, 41]. 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 $L_f \in \text{P}$. Many prominent examples satisfy this property, like computing the number of: perfect matchings of a bipartite graph (#PerfectMatching) [40], satisfying assignments of a DNF (#DNF) [9, 29] or Horn (#HornSAT) [41] 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 $1/\varepsilon$ [29]. Notably, there exist #P-complete functions that can be efficiently approximated as they admit FPRAS; for instance, there exist FPRAS for #DNF [29] and #PerfectMatching [28]. A key observation here is that if a function f admits an FPRAS, then L_f is in the randomized complexity class BPP [17]. 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 [35]. 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 [14, 34]. 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 [36]. 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 [40], 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 5.1 a syntactic hierarchy of counting complexity classes contained in #P. Then this hierarchy is used in Section 5.2 to define a class of functions with easy decision versions and good closure properties, and in Section 5.3 to define a class of functions with easy decision versions and natural complete problems.

5.1. **The ΣQSO(FO)-hierarchy inside #P.** Inspired by the connection between #P and #FO, a hierarchy of subclases of #FO was introduced in [39] by restricting the alternation of quantifiers in Boolean formulae. Specifically, the #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 #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 [39] 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, (as expected, $\Sigma\bar{X}.$ is the respective nesting of $\Sigma X.$'s) 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 $\sum_{i=1}^{n} \alpha_i$ where each α_i is in $\mathscr{L}$-PNF.

Proposition 5.1. *Every formula in $\Sigma\mathrm{QSO}(\mathscr{L})$ can be rewritten in $\mathscr{L}$-SNF.*

Proof. Recall that a formula in $\Sigma\mathrm{QSO}(\mathscr{L})$ is defined by 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 $s \in \mathbb{N}$. To find an equivalent formula in $\mathscr{L}$-SNF for every $\alpha \in \Sigma\mathrm{QSO}(\mathscr{L})$, we give a recursive function τ such that $\tau(\alpha)$ is in $\mathscr{L}$-SNF and $\tau(\alpha) \equiv \alpha$. Specifically, if $\alpha = \varphi$, define $\tau(\alpha) = \alpha$; if $\alpha = s$, define $\tau(\alpha) = (\top + \overset{s \text{ times}}{\cdots} + \top)$; if $\alpha = (\alpha_1 + \alpha_2)$, define $\tau(\alpha) = (\tau(\alpha_1) + \tau(\alpha_2))$; if $\alpha = \Sigma x.\, \beta$, assume $\tau(\beta) = \sum_{i=1}^{k} \beta_i$ such that each β_i is in $\mathscr{L}$-PNF, and define $\tau(\alpha) = \sum_{i=1}^{k} \Sigma x.\, \beta_i$; and 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. □

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 $\Sigma\mathrm{QSO}(\mathscr{L})$ 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 5.2. *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.*

Proof. From now on, for every first-order tuple $\bar{x}$ or second-order tuple $\bar{X}$ we write $|\bar{x}|$ or $|\bar{X}|$ as the number of variables in $\bar{x}$ or $\bar{X}$ respectively. 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 $2^{|\mathfrak{A}|} + 1$) 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 α_0. In [39], it was proved that if $|\bar{X}| > 0$, the function defined by α is always even for big enough structures, 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 leads to a contradiction.

Second, we prove that the formula $\alpha_1 = 2$ (i.e. $[\![\alpha_1]\!]$ is the constant function 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 α_1. 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 some 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 taking the disjoint union of $\mathfrak{A}$ twice. Indeed, 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 assignment $(\bar{B}'_1, \bar{b}'_1, \bar{a}'_1)$ that is isomorphic to $(\bar{B}_1, \bar{b}_1, \bar{a}_1)$, in the other half of the structure, such that $\mathfrak{A}' \models \varphi(\bar{B}'_1, \bar{b}'_1, \bar{a}'_1)$. As a result, we have that $[\![\alpha]\!](\mathfrak{A}') \geqslant 3$ which leads to a contradiction.

For the last part of the proof, we 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 exists an equivalent formula in $\mathscr{L}$-PNF. Similarly to Theorem 5.1, we show a recursive function τ that produces such a formula. Assume that $\alpha = \sum_{i=1}^{n} \alpha_i$ is in $\mathscr{L}$-SNF where each α_i is in $\mathscr{L}$-PNF. Without loss 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 that is not the case, we can replace each α_i by the equivalent formula

$$\Sigma\bar{X}.\,\Sigma Y.\,\Sigma\bar{x}.\,\Sigma y.\,(\varphi_i(\bar{X},\bar{x}) \wedge \forall z.\,Y(z) \wedge \forall z.\,z \leqslant y).$$

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 $\tau(\alpha_2) = \Sigma\bar{Y}.\,\Sigma\bar{y}.\,\psi(\bar{Y},\bar{y})$. Our construction 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 $(\bar{Y},\bar{y})$ using 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 can define the formulas $\gamma_{\text{first}}(\bar{Y},\bar{y})$ and $\gamma_{\text{last}}(\bar{Y},\bar{y})$. 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 the assignments that are not first and the ones that are not last. We do this by negating the two formulas above and grouping together the first-order variables:

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

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

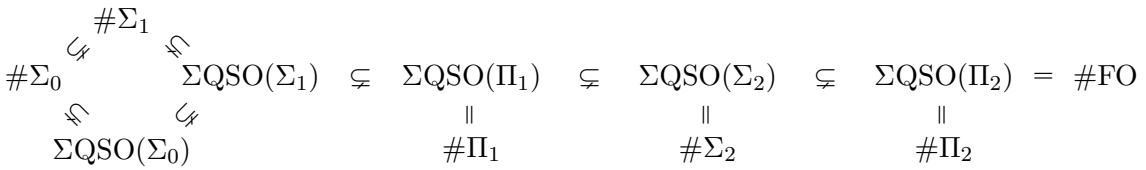

Figure 1: The relationship between the #FO-hierarchy and the ΣQSO(FO)-hierarchy, where $\#\Sigma_1$ and $\Sigma\mathrm{QSO}(\Sigma_0)$ are incomparable.

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

$$\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{Y}.\,\Sigma\bar{y}.\,[(\varphi(\bar{X},\bar{x}) \wedge \gamma_{\text{not-first}}(\bar{X},\bar{x}) \wedge \gamma_{\text{first}}(\bar{Y},\bar{y}))\vee \tag{5.1}$$

$$(\varphi(\bar{X},\bar{x}) \wedge \gamma_{\text{first}}(\bar{X},\bar{x}) \wedge \gamma_{\text{last}}(\bar{Y},\bar{y}))\vee \tag{5.2}$$

$$(\psi(\bar{Y},\bar{y}) \wedge \gamma_{\text{first}}(\bar{X},\bar{x}) \wedge \gamma_{\text{not-last}}(\bar{Y},\bar{y}))\vee \tag{5.3}$$

$$(\psi(\bar{Y},\bar{y}) \wedge \gamma_{\text{last}}(\bar{X},\bar{x}) \wedge \gamma_{\text{last}}(\bar{Y},\bar{y}))]. \tag{5.4}$$

To show that the formula is indeed equivalent to α, note that the formulas in lines (5.1) and (5.2) 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 (5.3) and (5.4) 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 sub-formulas $\gamma_{\text{not-first}}$ and $\gamma_{\text{not-last}}$. To solve this, we can replace each existential quantifier by a first order sum that counts just the first assignment that satisfies the inner formula and this can be defined in Π_1. A similar construction was used in [39].

Finally, consider a $\Sigma\mathrm{QSO}(\mathscr{L})$ formula α in $\mathscr{L}$-SNF. If $\alpha = \sum_{i=1}^{n} \alpha_i$, then by induction we 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 hypothesis. □

As a consequence of Proposition 5.1 and Theorem 5.2, 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 ΣQSO(FO)-hierarchy.

Proposition 5.3. *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* $\Sigma\mathrm{QSO}(\Sigma_0) \nsubseteq \#\Sigma_1$,
- $\Sigma\mathrm{QSO}(\Sigma_1) \subsetneq \Sigma\mathrm{QSO}(\Pi_1)$.

Proof. We give this proof in three parts. First, we show that $\#\Sigma_1 \nsubseteq \Sigma\mathrm{QSO}(\Sigma_0)$. Towards a contradiction, let $\mathbf{R} = \{<\}$ and suppose that there is a $\Sigma\mathrm{QSO}(\Sigma_0)$ formula α over $\mathbf{R}$ which is equivalent to the $\#\Sigma_1$ formula $\Sigma x.\,\exists y.\,(x < y)$. This is, for every finite $\mathbf{R}$-structure $\mathfrak{A}$, $[\![\alpha]\!](\mathfrak{A}) = |\mathfrak{A}| - 1$.

Suppose that α is in SNF, namely, $\alpha = \sum_{i=1}^{k} \alpha_i$ for some fixed k. 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 $[\![\alpha]\!]$ is in $\Omega(2^{|\mathfrak{A}|})$, as it was proven

in [39]. Therefore, we have that $\alpha_i = \Sigma\bar{x}.\,\varphi(\bar{x})$. We conclude our proof with the following claim.

Claim 5.4. Let $\alpha = \Sigma\bar{x}.\,\varphi(\bar{x})$ where φ is quantifier free. Then the function $[\![\alpha]\!]$ is either null, greater or equal to n, or is in $\Omega(n^2)$, where n is the size of the input structure.

Proof. Note that each atomic sub-formula in $\varphi(\bar{x})$ is either $(x = y)$, $(x < y)$, $\top$ or a negation thereof, where $x, y \in \bar{x}$. Suppose $[\![\alpha]\!]$ is not null and consider some $\mathbf{R}$-structure $\mathfrak{A}$ such that $[\![\alpha]\!](\mathfrak{A}) > 0$. Let $\bar{a}$ be an assignment to $\bar{x}$ such that $\mathfrak{A} \models \varphi(\bar{a})$. It can be seen that each assignment $\bar{a}'$ that has the same order over its variables[2] as $\bar{a}$ also satisfies $\mathfrak{A} \models \varphi(\bar{a}')$. If this order has k partitions then $\binom{|\mathfrak{A}|}{k}$ assignments for $\bar{x}$ satisfy this order, and therefore, $[\![\alpha]\!] \geqslant \binom{|\mathfrak{A}|}{k}$. If $k = 1$, then $\binom{|\mathfrak{A}|}{k} = |\mathfrak{A}|$, and if $k \geqslant 2$, then $\binom{|\mathfrak{A}|}{k} \in \Omega(|\mathfrak{A}|^2)$, which proves the claim. □

Now we show that $\Sigma\text{QSO}(\Sigma_0) \nsubseteq \#\Sigma_1$. In Theorem 5.2 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 $2 \in \Sigma\text{QSO}(\Sigma_0)$ and $2 \notin \#\Sigma_1$.

Finally, we prove that $\Sigma\text{QSO}(\Sigma_1) \subsetneq \Sigma\text{QSO}(\Pi_1)$. For inclusion, let α be a formula in $\Sigma\text{QSO}(\Sigma_1)$. Suppose that it is in Σ_1-SNF, namely, $\alpha = c + \sum_{i=1}^{n} \alpha_i$. 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 [39], 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 $(\bar{X}, \bar{x})$. Doing this replacement for each α_i renders an equivalent formula in $\Sigma\text{QSO}(\Pi_1)$.

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

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

Suppose that there exists an equivalent formula α in $\Sigma\text{QSO}(\Sigma_1)$. Also, suppose that it is in $\mathcal{L}$-PNF, so $\alpha = \sum_{i=1}^{n} \Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}.\,\varphi_i(\bar{X}, \bar{x}, \bar{y})$. Consider a structure $\mathfrak{A}'$ 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 $\mathfrak{A}' \models \varphi_i(\bar{B}, \bar{b}, \bar{a})$. Consider now the structure $\mathfrak{A}''$ that is obtained by duplicating $\mathfrak{A}'$, as we did for Theorem 5.2. Note that $\mathfrak{A}'' \models \varphi_i(\bar{B}, \bar{b}, \bar{a})$, which implies that $[\![\alpha]\!](\mathfrak{A}' \uplus \mathfrak{A}'') > 1$, which leads to a contradiction. □

The relationship between the two hierarchies is summarized in Figure 1. Our hierarchy and the one proposed in [39] 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.

5.2. **Defining a class of functions with easy decision versions and good closure properties.** We use the $\Sigma\text{QSO}(\text{FO})$-*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 $L_f \in \text{P}$,

[2]For example, if $\bar{a} = (a_1, a_2, a_3, a_4)$, such order may be $a_1 > a_3 = a_4 > a_2$, which has tree partitions.

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 [35] if there exists a polynomial-time NTM M such that $f(x) = \#\text{total}_M(x) - 1$ for every $x \in \Sigma^*$, where $\#\text{total}_M(x)$ is the total number of runs of M with input x. Notice that one is subtracted from $\#\text{total}_M(x)$ to allow for $f(x) = 0$. Besides, notice that TotP $\subseteq$ #P and that $f \in$ TotP implies that $L_f \in$ P.

The complexity class TotP contains many important counting problems with easy decision counterparts, such as #PerfectMatching, #DNF, and #HornSAT among others [35]. 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[3], and no natural complete problems are known for this class [35]. Hence, we use the ΣQSO(FO)-hierarchy to find restrictions of TotP with good approximation and closure properties.

It was proved in [39] 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 5.5. *There exist functions $f, g \in \#\Sigma_1$ such that $(f + g) \notin \#\Sigma_1$.*

Proof. Towards a contradiction, assume that $\#\Sigma_1$ is closed under binary sum. Consider the formula $\alpha = \Sigma x.(x = x) \in \#\Sigma_1$ over some signature $\mathbf{R}$. This defines the function $[\![\alpha]\!](\mathfrak{A}) = |\mathfrak{A}|$. From our assumption, there exists some formula in $\#\Sigma_1$ equivalent to the formula $\alpha + \alpha$, which describes the function $2|\mathfrak{A}|$. Let $\Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{X}, \bar{x}, \bar{y})$ be this formula, where φ is in first-order and quantifier-free. For each $\mathbf{R}$-structure $\mathfrak{A}$, we have the following inequality:

$$[\![\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{y}.\ \varphi(\bar{X}, \bar{x}, \bar{y})]\!](\mathfrak{A}) \leqslant [\![\Sigma\bar{X}.\,\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{X}, \bar{x}, \bar{y})]\!](\mathfrak{A}) \cdot |\mathfrak{A}|^{|\bar{y}|} \leqslant 2|\mathfrak{A}|^{|\bar{y}|+1}$$

Note that the formula $\Sigma\bar{X}.\,\Sigma\bar{x}.\,\Sigma\bar{y}.\ \varphi(\bar{X}, \bar{x}, \bar{y})$ defines a function in $\#\Sigma_0$. Therefore, as it was proven in [39], if $|\bar{X}| > 0$ then the function is in $\Omega(2^{|\mathfrak{A}|})$, which violates the inequality.

We now have that $|\bar{X}| = 0$. Consider a structure $\mathfrak{1}$ with only one element. We have that $[\![\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{x}, \bar{y})]\!](\mathfrak{1}) = 2$, but since the structure has only one element, there is only one possible assignment to $\bar{x}$. And so, $[\![\Sigma\bar{x}.\,\exists\bar{y}\,\varphi(\bar{x}, \bar{y})]\!](\mathfrak{1}) \leqslant 1$, which leads to a contradiction. □

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

Proposition 5.6. $\Sigma\text{QSO}(\Sigma_1) \subseteq$ TotP *and every function in* $\Sigma\text{QSO}(\Sigma_1)$ *has an FPRAS. Moreover,* $\Sigma\text{QSO}(\Sigma_1)$ *is closed under sum and multiplication.*

Proof. The authors in [39] 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 $g :$ OrdStruct$[\mathbf{R}] \to$ OrdStruct$[\mathbf{R}_{\text{DNF}}]$ and $h : \mathbb{N} \to \mathbb{N}$ such that for every $\mathbf{R}$-structure $\mathfrak{A}$, it holds that $[\![\alpha]\!](\mathfrak{A}) = \#\text{DNF}(\text{enc}(g(\mathfrak{A}))) \cdot h(|\mathfrak{A}|)$. We use this fact in the following arguments.

[3]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 [17]). This counting problem is in TotP, and it is known that NP = RP if there exists an FPRAS for it [10].

To show that $\Sigma\text{QSO}(\Sigma_1)$ 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 [39] and we obtain $g(\text{enc}(\mathfrak{A}))$, which is an instance to #DNF. Since #DNF is also in TotP [35], we simulate the corresponding nondeterministic procedure that generates exactly $\#\text{DNF}(\text{enc}(g(\mathfrak{A})))$ branches. Since, FP $\subseteq$ TotP [35], each polynomially computable function is also in TotP, and then on each of these branches we simulate the corresponding nondeterministic procedure to generate $h(|\mathfrak{A}|)$ more. The number of branches for each α_i is $[\![\alpha_i]\!](\mathfrak{A}) = \#\text{DNF}(\text{enc}(g(\mathfrak{A}))) \cdot h(|\mathfrak{A}|)$, and the total number of branches is equal to $[\![\alpha]\!](\mathfrak{A})$. We conclude that $\alpha \in$ TotP.

To show that every function in $\Sigma\text{QSO}(\Sigma_1)$ 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 $\alpha_i \in \#\Sigma_1$ [39]. 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.

Finally, we show that $\Sigma\text{QSO}(\Sigma_1)$ is closed under sum and multiplication. Since $\Sigma\text{QSO}(\Sigma_1)$ is closed under sum by definition, we focus only in proving that it 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 5.7. *If $\mathscr{L}$ is a fragment closed under conjunction, then $\Sigma\text{QSO}(\mathscr{L})$ is closed under binary multiplication.*

Proof. Given two formulas α, β in $\Sigma\text{QSO}(\mathscr{L})$ we will show a formula in the grammar which is equivalent to $(\alpha \cdot \beta)$. From what was proven in Proposition 5.1, we may assume that α and β are in $\mathscr{L}$-SNF. Let $\alpha = \sum_{i=1}^{n} \Sigma\bar{X}_i.\, \Sigma\bar{x}_i.\, \varphi_i(\bar{X}_i, \bar{x}_i)$, and $\beta = \sum_{i=1}^{m} \Sigma\bar{Y}_i.\, \Sigma\bar{y}_i.\, \psi_i(\bar{Y}_i, \bar{y}_i)$. Expanding the product in $(\alpha \cdot \beta)$ and reorganizing results in the equivalent formula

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \Sigma\bar{X}_i.\, \Sigma\bar{Y}_j.\, \Sigma\bar{x}_i.\, \Sigma\bar{y}_j.\, (\varphi_i(\bar{X}_i, \bar{x}_i) \wedge \psi_i(\bar{Y}_j, \bar{y}_j)),$$

which is in $\mathscr{L}$-SNF, and therefore, in $\Sigma\text{QSO}(\mathscr{L})$. □

Since Σ_1 is closed under conjunction, then Lemma 5.7 holds for $\Sigma\text{QSO}(\Sigma_1)$. This concludes the proof. □

Hence, it only remains to prove that $\Sigma\text{QSO}(\Sigma_1)$ 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 $\Sigma\text{QSO}(\Sigma_1)$ 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[4]. So, the following proposition rules out any logic that extends Π_1 for a possible extension of $\Sigma\text{QSO}(\Sigma_1)$ with the desired closure property.

Proposition 5.8. *If $\Pi_1 \subseteq \mathscr{L} \subseteq \text{FO}$ and $\Sigma\text{QSO}(\mathscr{L})$ is closed under subtraction by one, then #P is closed under subtraction by one.*

Proof. 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 $\Sigma\text{QSO}(\mathscr{L})$. Recall also that $\Sigma\text{QSO}(\mathscr{L}) \subseteq \Sigma\text{QSO}(\text{FO}) = \#\text{P}$ and that #3-CNF is #P-complete under parsimonious reductions[5]. Let f be a function in #P, and consider the nondeterministic polynomial-time procedure that on input $\text{enc}(\mathfrak{A})$ computes the corresponding reduction $g(\text{enc}(\mathfrak{A}))$ into #3-CNF and simulates the #P procedure for #3-CNF $- 1$ on input $g(\text{enc}(\mathfrak{A}))$. This is a #P procedure that computes $f - 1$, from which we conclude that #P is closed under subtraction by one. □

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

Theorem 5.9. *The class $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ is closed under sum, multiplication and subtraction by one. Moreover, $\Sigma\text{QSO}(\Sigma_1[\text{FO}]) \subseteq \text{TotP}$ and every function in $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ has an FPRAS.*

Proof. For the sake of readability, we divide the proof in three parts. The last part, i.e. subtraction by one, is probably the most technical proof of the paper.

Closed under sum and multiplication. By the previous results, it is straightforward to prove that $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ is closed under sum and multiplication. Indeed, $\Sigma\text{QSO}(\mathscr{L})$ is closed under sum by definition for every fragment $\mathscr{L}$, and since $\Sigma_1[\text{FO}]$ is closed under conjunction, from Lemma 5.7 it follows that $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ is closed under multiplication.

Easy decision version and FPRAS. We show here that $\Sigma\text{QSO}(\Sigma_1[\text{FO}]) \subseteq \text{TotP}$ and every function in $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ has an FPRAS. We do this by showing a parsimonious reduction from a function in $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ to some function in $\Sigma\text{QSO}(\Sigma_1)$, and using the result of Proposition 5.6. First, we define a function that converts a formula α in $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ over a signature $\mathbf{R}$ into a formula $\lambda(\alpha)$ in $\Sigma\text{QSO}(\Sigma_1)$ 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 $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$. The signature $\mathbf{R}_\alpha$ is obtained by adding the symbol R_ψ to $\mathbf{R}$ for every FO formula $\psi(\bar{z})$ in α. Each symbol R_ψ represents a predicate with arity

[4]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 $\#\text{accept}_M(x) - \#\text{reject}_M(x) = 2$, and for every $x \notin L$ it holds that $\#\text{accept}_M(x) = \#\text{reject}_M(x)$ [15,34]. It is believed that NP $\nsubseteq$ SPP. However, if #P is closed under subtraction by one, then it holds that NP $\subseteq$ SPP [34].

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

$|\bar{z}|$. 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 with a polynomial time procedure. 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 $\mathfrak{A} \models \psi(\bar{a})$ (this can be done in P), then the tuple $\bar{a}$ is added to $R_\psi^{\mathfrak{A}'}$. 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 $\text{enc}(\mathfrak{A})$. 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 $\Sigma\text{QSO}(\Sigma_1)$ formula $\lambda(\alpha)$.

To show that α is in TotP, we can convert α and $\text{enc}(\mathfrak{A})$ into $\lambda(\alpha)$ and $\text{enc}(g_\alpha(\mathfrak{A}))$, respectively, and run the procedure in Proposition 5.6. Similarly, to show that α has an FPRAS, we do the same as before and simulates the FPRAS for $\lambda(\alpha)$ in Proposition 5.6. These procedures also takes polynomial time and satisfies the required conditions.

Closed under subtraction by one. We prove here that $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ is closed under subtraction by one. For this, given $\alpha \in \Sigma\text{QSO}(\Sigma_1[\text{FO}])$ over a signature $\mathbf{R}$, we will define a $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$-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 $\Sigma_1[\text{FO}]$-SNF, that is, $\alpha = \sum_{i=1}^{n} \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi_i$ where each φ_i is in $\Sigma_1[\text{FO}]$. Moreover, we assume that $|\bar{x}| > 0$ since, if this is not the case, we can replace $\Sigma\bar{X}.\, \varphi_i$ for the equivalent formula $\Sigma\bar{X}.\, \Sigma y.\, \varphi_i \wedge \text{first}(y)$.

The proof will be separated in two parts. In the first part, we will assume that α is in $\Sigma_1[\text{FO}]$-PNF, namely, $\alpha = \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi$ for some φ in $\Sigma_1[\text{FO}]$. 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 $\kappa(\alpha) = \Sigma\bar{X}.\, \Sigma\bar{x}.\, \varphi'$ will be the desired 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 $\Sigma_1[\text{FO}]$-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 $\Sigma\text{QSO}(\Sigma_1[\text{FO}])$ 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 assume that each $\varphi_i'(\bar{X}, \bar{x}, \bar{y})$ has the form:

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

Note that atomic formulas, like $R(\bar{z})$ for $R \in \mathbf{R}$, will appear in the subformula $\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$.

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) $\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$ defines an ordered partition over the variables in $(\bar{x}, \bar{y})$ (see below for the formal 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 $X_j(\bar{z})$ (denoted by $\varphi_i^+(\bar{X}, \bar{x}, \bar{y})[X_j(\bar{w}) \leftarrow X_j(\bar{z})]$ and then the formula φ_i is equivalently defined as:

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

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

$$\varphi_i \ := \ \exists \bar{y}.\, v_1 = x \wedge v_2 = y \wedge v_3 = x \wedge v_4 = x \wedge x < y \wedge X(v_1, v_2) \wedge \neg X(v_3, v_4).$$

(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 the formula that defines the j-th ordered partition over $(\bar{x}, \bar{y})$. Thus, the formula $\varphi(\bar{X}, \bar{x})$ is then redefined as:

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

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 $\varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ forces an ordered partition over the variables in $(\bar{x}, \bar{y})$.

(c) Finally, to rewrite the formula such 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 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 5.10. For an ordered structure $\mathfrak{A}$ and a FO assignment v for $\mathfrak{A}$, $(\mathfrak{A}, v) \models \varphi_i^{\mathrm{FO}}(\bar{x}, \bar{y})$ if, and only if, there exists a SO 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 $(\mathfrak{A}, v) \models \varphi_i^{\text{FO}}(\bar{x}, \bar{y})$. 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 $(\mathfrak{A}, V, v) \not\models \varphi_i^{\text{FO}}(\bar{x}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{y}) \wedge \varphi_i^-(\bar{X}, \bar{y})$. By the choice of v, and construction of V it is clear that $(\mathfrak{A}, V, v) \models \varphi_i^{\text{FO}}(\bar{x}, \bar{y}) \wedge \varphi_i^+(\bar{X}, \bar{y})$, 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 $(\mathfrak{A}, v) \models \varphi_i^{\text{FO}}(\bar{x}, \bar{y})$ and $v(\bar{z}) = v(\bar{w})$, then the ordered partition in φ_i^{FO} satisfies $\bar{z} = \bar{w}$. This violates condition (c) above 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 $(\mathfrak{A}, v) \models \varphi_i^{\text{FO}}(\bar{x}, \bar{y})$ since $\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$ 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. This motivates the main idea of Part (1): by choosing one particular v the plan is to remove $\bar{B}^v$ as an assignment over $\bar{X}$ in φ_i. For this, we choose the minimal v that satisfies $\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$ which can be defined with the following formula:

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

If φ_i^{FO} is satisfiable, let v be the only assignment such that $(\mathfrak{A}, v) \models \mathit{min}\text{-}\varphi_i^{\text{FO}}(\bar{x}, \bar{y})$. Furthermore, let V^* be 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, $(\mathfrak{A}, V^*, v^*) \models \varphi_i(\bar{X}, \bar{x})$.

Now, we have all the ingredients 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, if $\varphi_i(\bar{X}, \bar{x})$ is satisfiable, then $V \neq V^*$ or $v \neq v^*$. This property can be defined as follows:

$$\psi_i(\bar{X}, \bar{x}) \;:=\; \big(\, \exists \bar{x}.\, \exists \bar{y}.\, \varphi_i^{\text{FO}}(\bar{x}, \bar{y}) \,\big) \rightarrow \tag{5.5}$$

$$\Big(\, \exists \bar{y}.\, \mathit{min}\text{-}\varphi_i^{\text{FO}}(\bar{x}, \bar{y}) \wedge \big(\, \varphi_i'(\bar{X}, \bar{x}, \bar{y}) \rightarrow \bigvee_{X \in \bar{X}} \exists \bar{z}.\, \big[\, X(\bar{z}) \wedge \bigwedge_{X(\bar{w}) \in \varphi_i^+(\bar{X}, \bar{v})} \bar{w} \neq \bar{z} \,\big] \,\big) \vee \tag{5.6}$$

$$\big(\, \exists \bar{x}'.\, \exists \bar{y}.\, \varphi_i'(\bar{X}, \bar{x}', \bar{y}) \wedge \bar{x}' < \bar{x} \,\big) \,\Big) \tag{5.7}$$

To understand the formula, first notice that the premise of the implication at (5.5) is true if, and only if, $\varphi_i(\bar{X}, \bar{x})$ is satisfiable. Indeed, by Claim 5.10 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 (5.6) and (5.7)), take care that $V(\bar{X}) \neq V^*(\bar{X})$ or $v(\bar{x}) \neq v^*(\bar{x})$. Here, the first disjunct (5.6) 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 (5.7) 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 $\chi_i = \neg\exists\bar{x}.\,\exists\bar{y}.\,\varphi_i^{\mathrm{FO}}(\bar{x},\bar{y})$. This formula basically checks if φ_i is not satisfiable (recall Claim 5.10). 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:

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

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 $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ formula β. We define $\kappa(\alpha)$ as follows. First, rewrite $\varphi(\bar{X},\bar{x})$ as in Part (1). Let $\varphi = \bigvee_{i=1}^{m}\varphi_i(\bar{X},\bar{x})$ where each φ_i satisfies conditions (a), (b) and (c) defined above. Also, consider the previously defined formulas χ_i and ψ_i, for each $i \leqslant m$. 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 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, following the same ideas as in Part (1) we define a formula $\varphi_i^*(\bar{X},\bar{x})$ that removes the minimal (V^*,v^*) of φ_i whenever β cannot be satisfied (i.e. $\lambda(\beta)$ is true). Formally, we define φ_i^* as follows:

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

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 $\Sigma\mathrm{QSO}(\Sigma_1[\mathrm{FO}])$ and satisfies the desired conditions. This concludes the proof. □

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.

5.3. **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 [18] 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 5.11. 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:

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

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 formulae ψ such that ψ is a Horn formula over a signature $\mathbf{R}$.

As we have seen, we have that #HornSAT $\in$ ΣQSO(Π_1-Horn). Moreover, one can show that ΣQSO(Π_1-Horn) forms a class of functions with easy decision counterparts, namely, ΣQSO(Π_1-Horn) $\subseteq$ TotP. Thus, ΣQSO(Π_1-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 5.12. ΣQSO(Σ_2-Horn) $\subseteq$ TotP.

For the sake of simplification, we postpone the proof the previous proposition after the proof of Theorem 5.14.

Interestingly, we have that both #HornSAT and #DNF belong to ΣQSO(Σ_2-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 ΣQSO(Π_2) and not necessarily in ΣQSO(Σ_2). The following result shows that this is indeed the case.

Proposition 5.13. #HornSAT $\notin$ ΣQSO(Σ_2).

Proof. Suppose that the statement is false, this is, #HornSAT $\in$ ΣQSO(Σ_2). Consider the signature $\mathbf{R}$ from Example 5.11 and let $\alpha \in \Sigma$QSO(Σ_2) be a formula over $\mathbf{R}$ that defines #HornSAT. By Proposition 5.1 we know that every formula in ΣQSO(Σ_2) can be rewritten in Σ_2-PNF, so we can assume that α is of the form $\Sigma \bar{X}.\,\Sigma \bar{x}.\,\exists \bar{y}.\,\forall \bar{z}.\,\varphi(\bar{X},\bar{x},\bar{y},\bar{z})$.

Now, consider the following Horn formula:

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

such that $n = |\bar{x}| + |\bar{y}| + 1$ and let $\mathfrak{A}_\Phi$ be the encoding of this formula over $\mathbf{R}$. One can easily check that Φ is satisfiable, so $[\![\alpha]\!](\mathfrak{A}_\Phi) \geqslant 1$. Let $(\bar{B}, \bar{b}, \bar{a})$ be an assignment to $(\bar{X}, \bar{x}, \bar{y})$ such that $\mathfrak{A}_\Phi \models \forall \bar{z}.\, \varphi(\bar{B}, \bar{b}, \bar{a}, \bar{z})$ and let t_ℓ be such that it does not appear in $\bar{b}$ or $\bar{a}$ (recall that $n > |\bar{x}| + |\bar{y}|$). Consider the induced substructure $\mathfrak{A}'_\Phi$ that is obtained by removing t_ℓ from $\mathfrak{A}_\Phi$ and $\bar{B}'$ as the subset of $\bar{B}$ obtained by deleting each appearance of t_ℓ in $\bar{B}$. We have that $\mathfrak{A}'_\Phi \models \forall \bar{z}.\, \varphi(\bar{B}', \bar{b}, \bar{a}, \bar{z})$ since universal formulas are monotone over induced substructures. Then it follows that $[\![\alpha]\!](\mathfrak{A}'_\Phi) \geqslant 1$ which is not possible since $\mathfrak{A}'_\Phi$ 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. This leads to a contradiction and we conclude that #HornSAT is not in $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$. □

We conclude this section by showing that $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$ is the class we were looking for, as not only every function in $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$ has an easy decision counterpart, but also $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$ 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 5.14. #DisjHornSAT *is* $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$*-complete under parsimonious reductions.*

Proof. First we prove that #DisjHornSAT is in $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$. Recall that each instance of #DisjHornSAT is a disjunction of Horn formulas. Let $\mathbf{R}$ be a relational signature such that $\mathbf{R} = \{P(\cdot,\cdot), N(\cdot,\cdot), V(\cdot), NC(\cdot), D(\cdot,\cdot)\}$. Each symbol in this vocabulary is used to indicate the same as in Example 5.11, with the addition of $D(d, c)$ which indicates that c is a clause in the formula d. Define ψ as in Example 5.11 such that $\Sigma T.\, \Sigma A.\, \psi(T, A)$ defines #HornSAT. In order to encode #DisjHornSAT, we extend $\psi(T, A)$ by adding the information of $D(d, c)$ as follows:

$$\begin{aligned}
\psi'(T, A) \;:=\; \exists d.\, \big[\; & \forall x.\, (\neg T(x) \vee V(x)) \wedge \\
& \forall c.\, (\neg D(c, d) \vee \neg NC(c) \vee \exists x.\, \neg A(c, x)) \wedge \\
& \forall c.\, \forall x.\, (\neg D(c, d) \vee \neg P(c, x) \vee \exists y.\, \neg A(c, y) \vee T(x)) \wedge \\
& \forall c.\, \forall x.\, (\neg D(c, d) \vee \neg N(c, x) \vee T(x) \vee \neg A(c, x)) \wedge \\
& \forall c.\, \forall x.\, (\neg D(c, d) \vee A(c, x) \vee N(c, x)) \wedge \\
& \forall c.\, \forall x.\, (\neg D(c, d) \vee A(c, x) \vee \neg T(x)) \;\big].
\end{aligned}$$

One can check that $\psi'(T, A)$ effectively defines #DisjHornSAT 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 $\in \Sigma\text{QSO}(\Sigma_2\text{-Horn})$.

Next, we prove that #DisjHornSAT is hard for $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$ over a signature $\mathbf{R}$ under parsimonious reductions. For each $\Sigma\text{QSO}(\Sigma_2\text{-Horn})$ formula α over $\mathbf{R}$, we will define a polynomial-time function g_α that receives an $\mathbf{R}$-structure $\mathfrak{A}$ and outputs an instance of

#DisjHornSAT such that $[\![\alpha]\!](\mathfrak{A}) = \#\textsc{DisjHornSAT}(g_\alpha(\mathfrak{A}))$. By Proposition 5.1, we can assume that α is of the form:

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

where each φ_j^i is a Horn-clause, and each $\bar{X}_i$ is a sequence of second-order variables. Consider $\bar{X}$ as the union of all $\bar{X}_i$. We replace each of the m addends in α for

$$\Sigma \bar{X}.\, \Sigma \bar{x}.\, \exists \bar{y}.\, \bigwedge_{j=1}^{n} \forall \bar{z}.\, \varphi_j^i(\bar{X}_i, \bar{x}, \bar{y}, \bar{z}) \wedge \bigwedge_{X \notin \bar{X}_i} \forall \bar{u}.\, X(\bar{u}),$$

whose sum is equivalent to α. Now, consider a finite $\mathbf{R}$-structure $\mathfrak{A}$ with domain A. The next transformation of α and $\mathfrak{A}$ towards a disjunction of Horn-formulas is to expand each first-order quantifier (i.e. $\Sigma \bar{x}$, $\exists \bar{y}$, and $\forall \bar{z}$) as we replace their variables with first-order constants. Specifically, we obtain the following formula which defines the same function as α and is of polynomial size with respect to $\mathfrak{A}$:

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

Note that each first-order sub-formula in $\varphi_j^i(\bar{X}, \bar{a}, \bar{b}, \bar{c})$ has no free variables and, therefore, we can evaluate each of them in polynomial time and replace by $\bot$ and $\top$ wherever it corresponds. In other words, in polynomial time we can replace φ_j^i by a disjunction of $\neg X_h$ and at most one X_h, evaluated on constants. After simplifying, grouping and reordering the previous formula, we can obtain an equivalent formula $\alpha'_{\mathfrak{A}}$ of the form:

$$\alpha'_{\mathfrak{A}} \;:=\; \sum_{i=1}^{m'} \Sigma \bar{X}.\, \bigvee_{j=1}^{n'_1} \bigwedge_{k=1}^{n'_2} \psi_{j,k}^i(\bar{X})$$

where every $\psi_{j,k}^i(\bar{X})$ is a disjunction of $\neg X_h$ and at most one X_h. The reader can easily check that $[\![\alpha]\!](\mathfrak{A}) = [\![\alpha'_{\mathfrak{A}}]\!](\mathfrak{A})$.

The idea for the rest of the proof is to show how to obtain $g_\alpha(\mathfrak{A})$, i.e. an instance of #DisjHornSAT, from $\alpha'_{\mathfrak{A}}$. First, if $\alpha'_{\mathfrak{A}}$ is equal to $\Sigma \bar{X}.\, \bigvee_{j=1}^{n'_1} \bigwedge_{k=1}^{n'_2} \psi_{j,k}(\bar{X})$, then we can define $g_\alpha(\mathfrak{A})$ equal to the propositional formula $\bigvee_{j=1}^{n'_1} \bigwedge_{k=1}^{n'_2} \psi_{j,k}(\bar{X})$ 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. Otherwise, if $m' > 1$ we can use m' new fresh variables $t_1, \ldots, t_{m'}$ and define:

$$g_\alpha(\mathfrak{A}) \;:=\; \bigvee_{i=1}^{m'} \bigvee_{j=1}^{n'_1} \bigwedge_{k=1}^{n'_2} \psi_{j,k}^i(\bar{X}) \wedge t_i \wedge \bigwedge_{\ell \neq i} \neg t_\ell$$

over the propositional alphabet $\{X(\bar{e}) \mid X \in \bar{X} \text{ and } \bar{e} \in A^{\text{arity}(X)}\} \cup \{t_1, \ldots, t_{m'}\}$. Intuitively, variables $t_1, \ldots, t_{m'}$ are used to count from 1 to m' each subformula $\Sigma \bar{X}.\, \bigvee_{j=1}^{n'_1} \bigwedge_{k=1}^{n'_2} \psi_{j,k}^i(\bar{X})$ in $\alpha'_{\mathfrak{A}}$. One can easily check that $g_\alpha(\mathfrak{A})$ is a disjunction of Horn formulas, and the number of satisfying assignments is exactly $[\![\alpha]\!](\mathfrak{A})$. This covers all possible cases for α, and the entire procedure takes polynomial time. □

Proof of Proposition 5.12. In [35], Pagourtzis and Zachos gave a TotP procedure that computes the number of satisfying assignments of a DNF formula. This procedure can be easily extended to receive Horn formulas, and furthermore, a disjunction of Horn formulas. Hence #DisjHornSAT is in TotP. As we saw in Theorem 5.14, #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. Then the TotP procedure consist in computing $g_\alpha(\text{enc}(\mathfrak{A}))$ for each input $\text{enc}(\mathfrak{A})$ and then simulate the TotP procedure for #DisjHornSAT on input $g_\alpha(\text{enc}(\mathfrak{A}))$. Therefore, we conclude that α defines a function in TotP. □

6. 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 that this notion can be used to captures FP. Moreover, we use this concept to define an operator for counting paths in a graph, a natural generalization of the transitive closure operator [26], 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 **FS** is an infinite set of function symbols, where each $h \in \mathbf{FS}$ has an associated arity denoted by $\text{arity}(h)$. Then the set of FQSO formulae over a signature **R** is defined by the following grammar:

$$\alpha \;:=\; \varphi \;\mid\; s \;\mid\; h(x_1,\ldots,x_\ell) \;\mid\; (\alpha+\alpha) \;\mid\; (\alpha\cdot\alpha) \;\mid\; \Sigma x.\,\alpha \;\mid\; \Pi x.\,\alpha \;\mid\; \Sigma X.\,\alpha \;\mid\; \Pi X.\,\alpha, \quad (6.1)$$

where $h \in \mathbf{FS}$, $\text{arity}(h) = \ell$ and $x_1,\ldots,x_\ell$ is a sequence of (not necessarily distinct) first-order variables. Given an **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 $\text{arity}(h) = \ell$, 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:

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

As for the case of QFO, we define FQFO disallowing quantifiers ΣX and ΠX in (6.1).

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 [33]. 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 article to propose this extension on weighted logics, which we think should be further investigated.

We define an extension of LFP [24, 42] to allow counting. More precisely, the set of RQFO(FO) formulae over a signature **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) $\text{arity}(h) = \ell$. 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 $\text{arity}(h) = \ell$, 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:

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

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 [24, 42]: (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 $\text{supp}(f)$, as $\{\bar{a} \in A^\ell \mid f(\bar{a}) > 0\}$. Then given that $\text{supp}(\chi_R) = R$, we have that the conditions (a) and (b) are equivalent to the following conditions on $T^\star$: (a) $\text{supp}(T^\star(\chi_R)) = \text{supp}(\chi_R)$, and (b) $\text{supp}(\chi_R) \subseteq \text{supp}(\chi_S)$ for every S such that $\text{supp}(T^\star(\chi_S)) = \text{supp}(\chi_S)$. 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 $\text{supp}(T_\beta(f)) = \text{supp}(f)$, 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 6.1. *Let $h \in \mathbf{FS}$ such that $\text{arity}(h) = \ell$, and β be an FQFO(FO)-formula over a signature $\mathbf{R}$ such that if a function symbol occurs in β, then this function symbol is h. Moreover, let $\mathfrak{A}$ be an $\mathbf{R}$-structure with domain A, $f, g : A^\ell \to \mathbb{N}$ and F, G be function assignments such that $F(h) = f$ and $F(h) = g$. If $\text{supp}(f) \subseteq \text{supp}(g)$, then for every first-order and second-order assignments v and V, respectively, it holds that:*

$$\text{if } [\![\beta]\!](\mathfrak{A}, v, V, F) > 0, \text{ then } [\![\beta]\!](\mathfrak{A}, v, V, G) > 0.$$

Proof. We prove the lemma by induction on the structure of β. First we need to consider the base cases.

(1) Assume that β is either a constant $s \in \mathbb{N}$ or an FO-formula φ. In both cases, function symbol h is not mentioned, so $[\![\beta]\!](\mathfrak{A}, v, V, F) = [\![\beta]\!](\mathfrak{A}, v, V, G)$ and it trivially holds that if $[\![\beta]\!](\mathfrak{A}, v, V, F) > 0$, then $[\![\beta]\!](\mathfrak{A}, v, V, G) > 0$.
(2) Assume that β is equal to $h(y_1, \ldots, y_\ell)$, where $y_1, \ldots, y_\ell$ is a sequence of (non-necessarily pairwise distinct) variables. Let $\bar{a} = (v(y_1), \ldots, v(y_\ell))$. Then we have that $[\![\beta]\!](\mathfrak{A}, v, V, F) = F(h)(v(y_1), \ldots, v(y_\ell)) = f(\bar{a})$ and $[\![\beta]\!](\mathfrak{A}, v, V, G) = g(\bar{a})$. Given that $\text{supp}(f) \subseteq$

$\mathrm{supp}(g)$, if $f(\bar{a}) > 0$, then $g(\bar{a}) > 0$. Hence, we conclude that if $[\![\beta]\!](\mathfrak{A}, v, V, F) > 0$, then $[\![\beta]\!](\mathfrak{A}, v, V, G) > 0$.

We now consider the inductive steps. Assume that the property holds for FQFO(FO)-formulae β_1, β_2 and δ

(3) Assume that $\beta = (\beta_1 + \beta_2)$. If $[\![\beta]\!](\mathfrak{A}, v, V, F) > 0$, then $[\![\beta_1]\!](\mathfrak{A}, v, V, F) > 0$ or $[\![\beta_2]\!](\mathfrak{A}, v, V, F) > 0$. Thus, by induction hypothesis we conclude that $[\![\beta_1]\!](\mathfrak{A}, v, V, G) > 0$ or $[\![\beta_2]\!](\mathfrak{A}, v, V, G) > 0$. Hence, we have that $[\![\beta]\!](\mathfrak{A}, v, V, G) > 0$.

(4) Assume that $\beta = (\beta_1 \cdot \beta_2)$. If $[\![\beta]\!](\mathfrak{A}, v, V, F) > 0$, then $[\![\beta_1]\!](\mathfrak{A}, v, V, F) > 0$ and $[\![\beta_2]\!](\mathfrak{A}, v, V, F) > 0$. Thus, by induction hypothesis we conclude that $[\![\beta_1]\!](\mathfrak{A}, v, V, G) > 0$ and $[\![\beta_2]\!](\mathfrak{A}, v, V, G) > 0$. Hence, we have that $[\![\beta]\!](\mathfrak{A}, v, V, G) > 0$.

(5) Suppose that $\beta = \Sigma x.\, \delta$. Then we have that $[\![\beta]\!](\mathfrak{A}, v, V, F) = \sum_{a \in A} [\![\delta]\!](\mathfrak{A}, v[a/x], V, F)$ and $[\![\beta]\!](\mathfrak{A}, v, V, G) = \sum_{a \in A} [\![\delta]\!](\mathfrak{A}, v[a/x], V, G)$. Thus, if we assume that $[\![\beta]\!](\mathfrak{A}, v, V, F) > 0$, then there exists $a \in A$ such that $[\![\delta]\!](\mathfrak{A}, v[a/x], V, F) > 0$. Hence, by induction hypothesis we have that $[\![\delta]\!](\mathfrak{A}, v[a/x], V, G) > 0$ and, therefore, we conclude that $[\![\beta]\!](\mathfrak{A}, v, V, G) > 0$.

(6) Suppose that $\beta = \Pi x.\, \delta$. Then we have that $[\![\beta]\!](\mathfrak{A}, v, V, F) = \prod_{a \in A} [\![\delta]\!](\mathfrak{A}, v[a/x], V, F)$ and $[\![\beta]\!](\mathfrak{A}, v, V, G) = \prod_{a \in A} [\![\delta]\!](\mathfrak{A}, v[a/x], V, G)$. Thus, if we assume that $[\![\beta]\!](\mathfrak{A}, v, V, F) > 0$, then $[\![\delta]\!](\mathfrak{A}, v[a/x], V, F) > 0$ for every $a \in A$. Hence, by induction hypothesis we have that $[\![\delta]\!](\mathfrak{A}, v[a/x], V, G) > 0$ for every $a \in A$, and, therefore, we conclude that $[\![\beta]\!](\mathfrak{A}, v, V, G) > 0$.

□

In the particular case of an RQFO(FO)-formula $[\mathbf{lsfp}\, \beta(\bar{x}, h)]$, Lemma 6.1 tell us that if $f, g \in \mathcal{F}$ and $\mathrm{supp}(f) \subseteq \mathrm{supp}(g)$, then $\mathrm{supp}(T_\beta(f)) \subseteq \mathrm{supp}(T_\beta(g))$. Hence, 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 $\mathrm{supp}(f_j) = \mathrm{supp}(T_\beta(f_j))$. Let k be the smallest natural number such that $\mathrm{supp}(f_k) = \mathrm{supp}(T_\beta(f_k))$. 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 $\mathfrak{A}$:

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

Example 6.2. 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 $\varphi_{\mathrm{first}}(x)$ and $\varphi_{\mathrm{succ}}(x, y)$ are FO-formulae

defining the first element of $<$ and the successor relation associated to $<$, respectively.[6] Moreover, define formula $\beta(x, y, t, g)$ as follows:

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

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 $\Sigma t'.\, \varphi_{\text{succ}}(t',t) \cdot (\Sigma x'.\, \Sigma y'.\, g(x',y',t'))$ 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 $\varphi_{\text{succ}}(t',t)$ 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 $\beta(\bar{x}, h)$. In fact, since β is constructed from monotones operations (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 6.3. RQFO(FO) *captures* FP *over ordered structures.*

Proof. 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 4.1.

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 $\{0,1,\varnothing\}$, respectively. Finally, assume that M, on input $\text{enc}(\mathfrak{A})$ with domain $A = \{1, \ldots, n\}$, executes exactly n^k steps for a fixed $k \geqslant 1$.

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 [33]. Let $\bar{x} =$

[6] Recall that in this paper we consider ordered $\mathbf{R}$-structures.

$(x_1, \ldots, x_k)$ and $\bar{t} = (t_1, \ldots, t_k)$. Then α is defined as:

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

In this formula, $T_0, T_1, T_{\mathtt{B}}, T_{\vdash}, h, \hat{h}$ are functions symbols of arity $2 \cdot k$, while $s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}$ are function symbols of arity k. For each one of these function symbols f, there is a formula α_f defining how the values of f are updated when computing the fixed point. For example, α_{T_0} is used to define the values of function T_0. Notice that the values of each function f in α depend on the values of the other functions, that is why we talk about a simultaneous computation. Besides, notice that notation $[\textbf{lsfp}\ \mathit{out}(\bar{t}) : \ldots]$ is used to indicate that (1) the free variables of the formula are $\bar{t} = (t_1, \ldots, t_k)$, and (2) once the least fixed point has been computed, the value of $[\textbf{lsfp}\ \mathit{out}(\bar{t}) : \ldots]$ for an assignment $\bar{a}$ to $\bar{t}$ is given by the value (in the fixed point) of function out on $\bar{a}$. Finally, it is important to notice that α can be transformed into a proper RQFO(FO) formula by using the same techniques used to prove that simultaneous LFP has the same expressiveness as LFP [33]; in particular, functions symbols $T_0, T_1, T_{\mathtt{B}}, T_{\vdash}, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}$ are replaced by a single function f of arity $6 \cdot 2 \cdot k + (\ell + 1) \cdot k + k = (\ell + 14) \cdot k$.

In the formula α, 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 $\hat{h}(\bar{t}, \bar{x}) = 0$ 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 $s_{q_i}(\bar{t}) = 0$ otherwise. Function out stores the output of the TM M; in particular, $\mathit{out}(\bar{t})$ is the value returned by M when $\bar{t}$ is the last step. Finally, assuming that $\varphi_{\mathrm{last}}(x)$ is an FO-formula defining the last element of $<$, we have that $\mathrm{last}(\bar{t}) = \bigwedge_{i=1}^{k} \varphi_{\mathrm{last}}(t_i)$, that is, $\mathrm{last}(\bar{t})$ holds if $\bar{t}$ is the last step. Therefore, the use of $\mathrm{last}(\bar{t})$ in α allows us to return the output of the TM M.

As in Example 6.2, assume that $\varphi_{\mathrm{first}}(x)$ is a FO-formula defining the first element of $<$. Moreover, assume that $\mathrm{first}(\bar{t})$ and $\mathrm{succ}(\bar{t}, \bar{t}')$ are FO-formulae such that $\mathrm{first}(\bar{t})$ holds if

$\bar{t}$ is the first step and $\mathrm{succ}(\bar{t},\bar{t}')$ holds if $\bar{t}'$ is the successor step of $\bar{t}$ (that is, $\mathrm{succ}(\cdot,\cdot)$ is the successor relation associated to the lexicographical order induced by $<$ on the tuples with k elements). Then formula $\alpha_{T_\vdash}$ is defined as follows:

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

Notation $\bigdotplus_{(q,a)\,:\,\delta(q,a)=(\_,\vdash,\_,\_)}$ in this formula is used to indicate that we are considering a sum over all pairs $(q,a)\in Q\times\{0,1,\mathtt{B},\vdash\}$ such that $\delta(q,a)=(q',\vdash,u,v)$ for some $q'\in Q$, $u\in\{\leftarrow,\rightarrow\}$ and $v\in\{0,1,\varnothing\}$. Formulas α_{T_0}, α_{T_1} and $\alpha_{T_{\mathtt{B}}}$ are defined analogously; for the sake of presentation, we only show here how α_{T_0} is defined, assuming that $\bar{y}=(y_1,\ldots,y_k)$:

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

Notice that in the initial step, the encoding of the structure $\mathfrak{A}$ has to be placed on the working tape of M from the second position since the symbol $\vdash$ is placed in the first position. This is the reason why we use tuple $\bar{y}$ and define $\bar{x}$ as the successor of $\bar{y}$. Formulas α_h is defined as follows:

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

Similarly, formula $\alpha_{\hat{h}}$ is defined as follows:

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

Notice that in the definition of $\alpha_{\hat{h}}$, FO-formula $\bar{x}\neq\bar{x}''$ is used to indicate that tuples $\bar{x}$ and $\bar{x}''$ are distinct (that is, there exists $i\in\{1,\ldots,k\}$ such that the i-th components of $\bar{x}$ and

$\bar{x}''$ are distinct). Formula α_{q_0} is defined as:

$$\alpha_{q_0}(\bar{t}, T_0, T_1, T_{\mathtt{B}}, T_{\vdash}, h, \hat{h}, s_{q_0}, \ldots, s_{q_\ell}, \mathit{out}) \;=\; \mathrm{first}(\bar{t}) \;+ \bigdotplus_{(q,a)\,:\,\delta(q,a)=(q_0,\_,\_,\_)} \Sigma\bar{t}'.\,\Sigma\bar{x}'.\,(\mathrm{succ}(\bar{t}',\bar{t}) \cdot h(\bar{t}',\bar{x}') \cdot T_a(\bar{t}',\bar{x}') \cdot s_q(\bar{t}')).$$

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

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

Clearly, at each iteration of the fixed point operator, the tuple $\bar{t}$ represents the step the machine is currently in. Assume that $\bar{a}$, $\bar{a}'$ are tuples with k elements such that $\bar{a}'$ is the successor of $\bar{a}$. From the construction of the formula, and since the machine is deterministic, it can be seen that for each function $f \in \{T_0, T_1, T_{\mathtt{B}}, T_{\vdash}, h, \hat{h}\}$, at the $\bar{a}$-th iteration of the fixed point operator, it holds that $f(\bar{a}, \bar{b}) \leqslant 1$ and $f(\bar{a}', \bar{b}) = 0$ for every $\bar{b} \in A^k$. In the same way, it can be seen that for each function $g \in \{s_{q_1}, \ldots, s_{q_\ell}\}$, at the $\bar{a}$-th iteration of the fixed point operator, it holds that $g(\bar{a}) \leqslant 1$ and $g(\bar{a}') = 0$. From this, we have that at the iteration $\bar{a}'$ of the fixed point operator, $\mathit{out}(\bar{a}')$ is equal to either $2 \cdot \mathit{out}(\bar{a})$, $2 \cdot \mathit{out}(\bar{a}) + 1$, or $\mathit{out}(\bar{a})$, which represents precisely the value in the output tape at each step of M running on the input $\mathrm{enc}(\mathfrak{A})$. From this argument, it can be seen that $[\![\alpha]\!](\mathfrak{A}) = f(\mathrm{enc}(\mathfrak{A}))$, which concludes the proof of the theorem. □

Our last goal in this section is to use the new characterization of FP to explore classes below it. It was shown in [24, 25] 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 relation 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, $\mathfrak{A} \models \psi(\bar{b}, \bar{c})$. Similar than for Example 6.2, we can count the paths of length at most $|A^k|$ in $\mathcal{G}_\psi(\mathfrak{A})$ with the formula $\beta_{\psi(\bar{x},\bar{y})}(\bar{x}, \bar{y}, \bar{t}, g)$:

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

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:

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

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 mentioned before, the operator for counting paths is exactly what we need to capture #L.

Theorem 6.4. TQFO(FO) *captures* #L *over ordered structures.*

Proof. First, we show that every formula in TQFO(FO) defines a function that is in #L. Let $\mathbf{R}$ be a relational signature and α a formula over $\mathbf{R}$ in TQFO(FO). Next we construct a logarithmic-space nondeterministic Turing Machine M_α that on input $(\text{enc}(\mathfrak{A}),v)$, where $\mathfrak{A}$ is an $\mathbf{R}$-structure and v is a first-order assignment for $\mathfrak{A}$, has $[\![\alpha]\!](\mathfrak{A},v)$ accepting runs (so that we can conclude that the function defined by α is in #L). Suppose that the domain of $\mathfrak{A}$ is $A=\{1,\ldots,n\}$. The TM M_α needs $\ell\cdot\log_2(n)$ bits of memory to store the first-order variables occurring in α, where ℓ is the number of variables occurring in this formula (which is the same as the number of variables in the domain of v). If $\alpha=\varphi$, where φ is an FO-formula, then we check if $(\mathfrak{A},v)\models\varphi$ in deterministic logarithmic space, and accept if and only if this condition holds. If $\alpha=s$, where s is a fixed natural number, then we generate s possible runs 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 M_{α_1} and if it accepts, then instead of accepting we simulate M_{α_2}. If $\alpha=\Sigma x.\,\beta$, for each $a\in A$ we generate a different run where we simulate M_β with input $v[a/x]$. If $\alpha=\Pi x.\,\beta$, we simulate M_β with input $v[1/n]$, and on each accepting run, instead of accepting we replace the assignment of x to 2, to simulate M_β with input $v[2/x]$, and so on. If $\alpha=[\mathbf{path}\ \varphi(\bar{x},\bar{y})]$, where φ is an FO-formula, then we simulate the #L procedure that counts the number of paths of a given length from a source to a target node in an input graph (where the length is at most the number of nodes in the graph).

Second, we show that every function in #L can be encoded by a formula in TQFO(FO). Let f be a function in #L and M a logarithmic-space nondeterministic Turing Machine such that $\#\text{accept}_M(\text{enc}(\mathfrak{A}))=f(\text{enc}(\mathfrak{A}))$. We assume that M has only one accepting state, and that no transition is defined for this state. Moreover, we assume that there exists only one accepting configuration. We make use of transitive closure logic (TC) to simplify our proof [19]. We have that TC captures NL [23], so that there exists a formula φ in TC such that $\mathfrak{A}\models\varphi$ if and only if M accepts $\text{enc}(\mathfrak{A})$. This formula can be expressed as:

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

where $\psi_{\text{initial}}(\bar{u})$ is an FO-formula that indicates that $\bar{u}$ is the initial configuration, $\psi_{\text{acc}}(\bar{z})$ is an FO-formula that indicates that $\bar{z}$ is an accepting configuration, and $\psi_{\text{next}}(\bar{x},\bar{y})$ is an FO-formula that indicates that $\bar{y}$ is a successor configuration of $\bar{x}$ [19]. Here, there is a one-to-one correspondence between configurations of M and assignments to $\bar{z}$. As a consequence, given a structure $\mathfrak{A}$ and a first-order assignment v for $\mathfrak{A}$, where $v(\bar{x})$ is the starting configuration and $v(\bar{y})$ is the sole accepting configuration, the value of $[\![[\mathbf{path}\ \psi_{\text{next}}(\bar{x},\bar{y})]]\!](\mathfrak{A},v)$ is equal to $\#\text{accept}_M(\text{enc}(\mathfrak{A}))$. Therefore, the TQFO(FO)-formula $\alpha=\Sigma\bar{u}.\,\Sigma\bar{z}.\,(\psi_{\text{initial}}(\bar{u})\cdot\psi_{\text{acc}}(\bar{z})\cdot[\mathbf{path}\ \psi_{\text{next}}(\bar{u},\bar{z})])$ satisfies that $[\![\alpha]\!](\mathfrak{A})=f(\text{enc}(\mathfrak{A}))$. This concludes the proof of the theorem. □

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 $L \in \mathrm{NL}$, it holds that its characteristic function χ_L is in QFO(TC), where $\chi_L(x) = 1$ if $x \in L$, and $\chi_L(x) = 0$ otherwise. Thus, if we assume that QFO(TC) captures #L (over ordered structures), then we have that $\chi_L \in \#\mathrm{L}$ for every $L \in \mathrm{NL}$. This would imply that $\mathrm{NL} = \mathrm{UL}$,[7] solving an outstanding open problem [38].

7. 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 logical characterizations of other fundamental complexity classes such as SpanL [2]. 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 [28] that is not included in the classes defined in Section 5.2. Moreover, by following the approach proposed in [23], we would like to include second-order free variables in the operator for counting paths introduced in Section 6, so to have alternative ways to capture FPSPACE and even #P. Finally, the least fixed point operator introduced in Section 6 clearly deserves further investigation.

References

[1] Serge Abiteboul and Victor Vianu. Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of LICS'89*, pages 71–79, 1989.

[2] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993.

[3] Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

[4] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[5] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[6] Kevin J. Compton and Erich Grädel. Logical definability of counting functions. *J. Comput. Syst. Sci.*, 53(2):283–297, 1996.

[7] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.

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

[8] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.
[9] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theor. Comput. Sci.*, 340(3):496–513, 2005.
[10] Martin E. Dyer, Alan M. Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM J. Comput.*, 31(5):1527–1541, 2002.
[11] Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *J. Comput. Syst. Sci.*, 54(3):400–411, 1997.
[12] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation, SIAM-AMS Proceedings*, (7):43–73, 1974.
[13] Ronald Fagin. Monadic generalized spectra. *Math. Log. Q.*, 21(1):89–96, 1975.
[14] Piotr Faliszewski and Lane A. Hemaspaandra. The consequences of eliminating NP solutions. *Comp. Sci. Review*, 2(1):40–54, 2008.
[15] Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *J. Comput. Syst. Sci.*, 48(1):116–148, 1994.
[16] Lance Fortnow. Counting complexity. In *Complexity Theory Retrospective II*, pages 81–107. Springer, 1997.
[17] John Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6(4):675–695, 1977.
[18] Erich Grädel. Capturing complexity classes by fragments of second-order logic. *Theor. Comput. Sci.*, 101(1):35–57, 1992.
[19] Erich Grädel. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications*, pages 125–230. Springer, 2007.
[20] Erich Grädel and Yuri Gurevich. Metafinite model theory. *Inf. Comput.*, 140(1):26–81, 1998.
[21] Lane Hemaspaandra and Mitsunori Ogihara. *The complexity theory companion*. Springer Science & Business Media, 2013.
[22] Lane A. Hemaspaandra and Heribert Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.
[23] Neil Immerman. Languages which capture complexity classes (preliminary report). In *Proceedings of STOC'83*, pages 347–354, 1983.
[24] Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1-3):86–104, 1986.
[25] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.
[26] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
[27] Neil Immerman and Eric Lander. Describing graphs: a first order approach to graph canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective*, pages 59–81. Springer-Verlag, 1990.
[28] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.
[29] Richard M. Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *Proceedings of FOCS'83*, pages 56–64, 1983.
[30] Phokion G Kolaitis and Madhukar N Thakur. Logical definability of NP optimization problems. *Information and Computation*, 115(2):321–353, 1994.
[31] Mark W Krentel. The complexity of optimization problems. *Journal of computer and system sciences*, 36(3):490–509, 1988.
[32] Richard E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.
[33] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.
[34] Mitsunori Ogiwara and Lane A. Hemachandra. A complexity theory for feasible closure properties. *J. Comput. Syst. Sci.*, 46(3):295–325, 1993.
[35] Aris Pagourtzis and Stathis Zachos. The complexity of counting functions with easy decision version. In *Proceedings of MFCS'06*, pages 741–752, 2006.
[36] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
[37] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

[38] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(14), 1997.
[39] Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. Descriptive Complexity of #P Functions. *J. Comput. Syst. Sci.*, 50(3):493–505, 1995.
[40] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.
[41] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
[42] Moshe Vardi. The complexity of relational query languages. In *Proceedings of STOC'82*, pages 137–146, 1982.
[43] Heribert Vollmer and Klaus W Wagner. Complexity classes of optimization functions. *Inf. and Comp.*, 120(2):198–219, 1995.