Counting Computations with Formulae: Logical Characterisations of Counting Complexity Classes

Antonis Achilleos
Department of Computer Science, Reykjavik University, Iceland

Aggeliki Chalki
Department of Computer Science, Reykjavik University, Iceland

Abstract

We present quantitative logics with two-step semantics based on the framework of quantitative logics introduced by Arenas et al. (2020) and the two-step semantics defined in the context of weighted logics by Gastin & Monmege (2018). We show that some of the fragments of our logics augmented with a least fixed point operator capture interesting classes of counting problems. Specifically, we answer an open question in the area of descriptive complexity of counting problems by providing logical characterisations of two subclasses of #P, namely SpanL and TotP, that play a significant role in the study of approximable counting problems. Moreover, we define logics that capture FPSPACE and SpanPSPACE, which are counting versions of PSPACE.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Complexity classes

Keywords and phrases descriptive complexity, quantitative logics, counting problems, #P

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.7

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2304.10334

Funding This work has been funded by the projects "Open Problems in the Equational Logic of Processes (OPEL)" (grant no. 196050), "Mode(l)s of Verification and Monitorability" (MoVeMnt) (grant no 217987) of the Icelandic Research Fund, and the Basic Research Program PEVE 2020 of the National Technical University of Athens.

Acknowledgements The authors would like to thank Stathis Zachos and Aris Pagourtzis for fruitful discussions and Luca Aceto for sound advice. We also thank the anonymous reviewers for their suggestions and constructive comments.

1 Introduction

We examine counting problems from the viewpoint of descriptive complexity. We present a quantitative logic with a least fixed point operator and two-step semantics. In the first step, given a structure, a formula generates a set. In the second step, a quantitative interpretation results from the cardinality of that set. These semantics allow us to use a uniform approach to identify logical fragments that capture several counting complexity classes.

In 1979, Valiant introduced the complexity class #P in his seminal paper [32] and used it to characterise the complexity of computing the permanent function. #P is the class of functions that count accepting paths of non-deterministic poly-time Turing machines, or, equivalently, the number of solutions to problems in NP. For example, #Sat is the function that, on input a formula φ in CNF, returns the number of satisfying assignments of φ. Since then, counting complexity has played an important role in computational complexity theory.

Descriptive complexity provides characterisations of complexity classes in terms of the logic needed to express their problems. The Büchi–Elgot–Trakhtenbrot theorem [9, 15, 31] characterising regular languages in terms of Monadic Second-Order logic and Fagin's theorem [17], which states that Existential Second-Order logic captures NP, are two fundamental results in this area. Another prominent result was the introduction of the class MaxSNP [29],

which has played a central role in the study of the hardness of approximation for optimization problems [6]. Moreover, descriptive complexity is an interesting and active research field with more recent results in the logical characterisation of the class P [20], dynamic complexity [33], symmetric linear programs [7], and counting complexity [5, 12], among others.

As in the case of optimization problems, an interesting, long-standing question is the logical characterisation of approximable counting problems. This is also a meaningful line of research since very few counting problems can be computed exactly in polynomial time. In the case of counting problems, the appropriate notion of approximability is the existence of a fully polynomial randomized approximation scheme (fpras). We denote the class of approximable counting problems by FPRAS [13, 8].

A counting class is considered *robust* if it has either natural complete problems or nice closure properties. Two robust subclasses of #P defined in terms of Turing Machines (TMs), are of great significance in the quest for a characterisation of approximable counting problems. The first one is TotP, which contains all self-reducible counting problems whose decision version is in P. It is noteworthy that TotP is not contained in FPRAS, unless RP = NP [8], but almost all known approximable counting problems belong to TotP (see e.g. [23, 22, 27]). The second class, namely SpanL [2], is contained in TotP, and it consists of the functions that count different outputs of non-deterministic log-space *transducers*, i.e. TMs with output. To the best of our knowledge, SpanL is the only counting class so far defined in terms of TMs, that, despite containing #P-complete problems [2], contains only approximable problems [4].

Our contribution. Our main objective is to provide logical characterisations of the classes SpanL and TotP, which was posed as an open question in [5]. To this end, we introduce a variant of the quantitative logics from [5]. Our two-step semantic definition is the key difference between our approach and that in [5]. The first step is an *intermediate semantics*, where the meaning of a formula is given as a set of strings that, intuitively, represent computation paths. In the second step, a concrete semantics associates with each formula the size of the set resulting from the intermediate semantics. Gastin et al. follow an analogous approach for weighted logics in [18], to give a connection to weighted automata.

In Section 4, we introduce logics equipped with least fixed point formulae that capture "span-classes" of restricted space, namely SpanL and SpanPSPACE, in a natural way (Theorems 4.7 and 4.13). When we consider such classes, we are interested in counting the number of different outputs produced by a transducer. Semantics that map the set of quantitative formulae to $\mathbb{N}$ interpret every accepting path as a contributing unit. Then, by evaluating the sum of formulae as the sum of natural numbers, one can sum up the accepting paths of a TM. On the other hand, when a formula is evaluated as a set of output strings and the sum of formulae as the union of sets, they can count the number of different TM outputs.

We also consider two classes, namely #PSPACE and TotP, which contain functions that count the accepting or all paths of TMs with restricted resources, respectively. Using our alternative semantics, a computation path can be encoded as a sequence of configurations visited by the TM along that path – in other words, its computation history – so that different paths are mapped to different sequences. In Section 5, we provide a logical characterisation of the class of functions that count the number of accepting paths of poly-space TMs, namely #PSPACE [25] (Theorem 5.3), which coincides with FPSPACE, i.e. the class of poly-space computable functions. FPSPACE has already been characterised by a logic with a partial fixed point [5]. Interestingly, the logic we define here uses a least fixed point. In Section 6, we introduce a quantitative logic that captures TotP (Theorem 6.6). In Section 7, we discuss how to obtain two least fixed point logics that capture NL and PSPACE by specialising the semantics. We believe that the semantics we propose in this paper can contribute insight to the study of counting complexity classes.

Related work. Arenas et al. and Saluja et al. give logical characterisations of #P in [30, 5]. The authors of [30] substitute existential quantification over second-order variables of $\exists\mathsf{SO}$ with counting second-order variables. The work in [5] incorporated counting into the syntax of the logic by introducing Quantitative Second-Order logic ($\mathtt{QSO}$), a logic for quantitative functions, which is based on the framework of weighted logics [11, 18, 1]. There has been progress in characterising counting classes with respect to their approximability in the context of descriptive complexity. Saluja et al. defined the classes $\#\Sigma_1$ and $\#\mathsf{R}\Sigma_2$ in [30], and proved that they contain only problems that admit an fpras. A more recent variant of $\#\Sigma_1$ [12] is also a subclass of FPRAS. The class $\#\mathsf{R}\Pi\mathsf{H}_1$ [13] is conjectured to contain problems which are neither as hard to approximate as #Sat nor admit an fpras, and it was used to classify Boolean #CSP with respect to their approximability [14]. Since NP-complete problems cannot have approximable counting versions unless RP = NP [13], Arenas et al. suggested in [5] to examine robust classes of counting problems with an easy decision version. The papers [5, 8] defined such counting classes and examined them with respect to the approximability of their problems. There is also work on logics that capture superclasses of #P, namely SpanP [24] and FPSPACE [25]. Compton and Grädel were the first to characterise SpanP in [10], followed by Arenas et al. in [5], where they also introduced a logic that captures FPSPACE. Finally, in [12], Durand et al. introduced a framework for the descriptive complexity of arithmetic circuit classes.

2 Preliminaries

Turing machines. A (*two-tape non-deterministic*) *Turing machine* (TM) N is a quintuple $N = (\mathcal{Q}, \Sigma, \delta, q_0, q_F)$, where $\mathcal{Q}$ is a set of states, $\Sigma = \{0, 1\}$ is the alphabet, $\delta \subseteq (\mathcal{Q} \times (\Sigma \cup \{\llcorner\})^2) \times (\mathcal{Q} \times (\Sigma \cup \{\llcorner\}) \times \{L, R\}^2)$ is the transition relation, q_0 is the initial state, and q_F is the final accepting state. We assume the TM N has a read-only input tape and a work tape that it can read and write on. L and R in a transition designate that the respective tape head moves to the left or right. A *configuration* c of N encodes a snapshot of the computation of N and is defined in the usual way (see e.g. [28]). We can apply a compatible transition to a configuration to result in a new configuration in the expected way. W.l.o.g. we assume that every TM has a binary computation tree: any configuration is compatible with zero, one or two transitions. In the latter case, we call these transitions, the *left* and *right* non-deterministic transition. A *transducer* M is a TM with a write-only output tape, on which a string over Σ is written from left to right. The output of a computation is *valid* if M stops in the accepting state. A TM or transducer is called *deterministic* if at every configuration at most one transition can be applied. By restricting the time or space resources of a TM or transducer in the usual way, we can obtain an NPTM (non-deterministic poly-time TM), an NL-transducer (non-deterministic log-space transducer) etc.

We say that f is *computable in polynomial time* (resp. logarithmic/polynomial space), if there is a deterministic polynomial-time (resp. log-space/poly-space) transducer M, such that for every $x \in \Sigma^*$, $f(x)$ is the valid output of M on input x. We define the functions that count paths (resp. outputs) of a TM (resp. transducer) as follows.

▶ **Definition 2.1.** *Let M be a Turing machine and T a transducer. We define functions $acc_M, tot_M, span_T : \Sigma^* \to \mathbb{N} \cup \{+\infty\}$, such that for every $x \in \Sigma^*$:*

(a) $acc_M(x) = \#(\textit{accepting computation paths of } M \textit{ on input } x)$,

(b) $tot_M(x) = \#(\textit{computation paths of } M \textit{ on input } x) - 1$,

(c) $span_T(x) = \#(\textit{different valid outputs of } T \textit{ on input } x)$.

Classes of counting problems. The classes defined in Definition 2.2 are already known, except for SpanPSPACE, which is presently defined.

▶ **Definition 2.2** ([2, 27, 25]). **(a)** $\mathsf{SpanL} = \{span_M : \Sigma^* \to \mathbb{N} \mid M \text{ is an NL-transducer}\}$,
(b) $\mathsf{TotP} = \{tot_M : \Sigma^* \to \mathbb{N} \mid M \text{ is an NPTM}\}$,
(c) $\mathsf{FPSPACE} = \{f : \Sigma^* \to \mathbb{N} \mid f \text{ is computable in polynomial space}\}$,
(d) $\#\mathsf{PSPACE} = \{acc_M : \Sigma^* \to \mathbb{N} \mid M \text{ is a non-deterministic poly-space TM}\}$.
(e) $\mathsf{SpanPSPACE} = \{span_M : \Sigma^* \to \mathbb{N} \mid M \text{ is a non-deterministic poly-space transducer}\}$.

▶ Remark 2.3. Note that in the definition of TotP, one is subtracted from the total number of paths so that a function can take the zero value. Since a TotP function f can be associated with an NPTM M that has a binary computation tree, $f(x) = tot_M(x) = \#(\text{branchings of } M \text{ on input } x)$, where a branching is an occurrence of a configuration on the computation tree, where M makes a non-deterministic choice.

▶ Remark 2.4. For the class SpanL, note that, by the pigeonhole principle, an NL-transducer has infinitely many accepting paths if and only if the length of its accepting runs is not bounded by a polynomial. It then makes sense to attach a clock that imposes a polynomial-time bound to each NLTM, as suggested in [2]. In this way, every NLTM is also an NPTM with a finite number of computation paths. Similarly, we assume that a clock that imposes an exponential-time bound can be attached to a non-deterministic poly-space TM.

▶ **Proposition 2.5** ([2, 27, 25]). $\mathsf{SpanL} \subseteq \mathsf{TotP} \subseteq \#\mathsf{P} \subsetneq \mathsf{FPSPACE} = \#\mathsf{PSPACE} \subseteq \mathsf{SpanPSPACE}$. *The first two inclusions are proper unless* $\mathsf{P} = \mathsf{NP}$.

The *decision version* of a function $f : \Sigma^* \to \mathbb{N}$ is $\{x \mid f(x) > 0\}$. We say that a function $f : \Sigma^* \to \mathbb{N}$ is *self-reducible* if its value on an instance can be recursively computed by evaluating f on a polynomial number of smaller instances. A formal definition of self-reducibility can be found in [3]. TotP can be characterised as the closure under parsimonious reductions of the class of self-reducible #P functions whose decision version is in P [27].

▶ **Example 2.6.** Consider the problem of counting independent sets of all sizes in a graph G, denoted by #IS. Let M be the NPTM that makes the following computation: given G_{i-1} and $v_i, \ldots, v_n$, M non-deterministically chooses to add vertex v_i to the independent set or not, and defines G_i to be either G_{i-1} where v_i, all its neighbours, and all edges adjacent to them have been removed, or G_{i-1} where v_i and its adjacent edges have been removed, respectively. Then, M recursively continues on G_i and $v_{i+1}, \ldots, v_n$. Consider M' that on input $G = \langle V = \{v_1, \ldots, v_n\}, E\rangle$ simulates M on G and $v_1, \ldots, v_n$, and has also an additional dummy path. Then, $\#\mathrm{IS}(G) = \#(\text{paths of } M' \text{ on input } G) - 1$.

Logics. A relational vocabulary $\sigma = \{\mathcal{R}_1^{k_1}, ..., \mathcal{R}_m^{k_m}\}$ is a finite set of relation symbols. Each relation symbol $\mathcal{R}_i$ has a positive integer k_i as its designated arity. A *finite structure* $\mathcal{A} = \langle A, R_1, ..., R_m\rangle$ over σ consists of a finite set A, which is called the *universe* of $\mathcal{A}$ and relations R_1,...,R_m of arities $k_1, .., k_m$ on A, which are interpretations of the corresponding relation symbols. We may write that $\mathsf{arity}(R_i) = k_i$ or that R_i is a k_i-ary relation. The *size of the structure*, denoted by $|\mathcal{A}|$ or $|A|$, is the size of its universe. A *finite ordered structure* is a finite structure with an extra relation $\leq$, which is interpreted as a total order on the elements of the universe. In sequel, $\mathcal{A}$ denotes a finite ordered structure unless otherwise specified. For convenience we use letters B, C, R, S, and so on, to denote both relation symbols and their interpretations. For example, the vocabulary of binary strings is $\sigma_{bs} = \{\leq^2, B^1\}$. Binary string $x = 00101$ corresponds to the structure $\mathcal{A} = \langle\{0, 1, ..., 4\}, \leq, B = \{2, 4\}\rangle$, where relation B represents the positions where x is one. Moreover, $|\mathcal{A}| = 5$.

First-order formulae over σ are defined in the usual way, using first-order variables that range over the universe of a structure, the relation symbols from σ, equality, the logical operators $\wedge, \vee, \neg, \rightarrow$, and first-order quantifiers $\forall x$ and $\exists x$. For convenience and clarity, we omit function and constant symbols from the syntax of FO, but we include $\top$, which is the logical constant for truth. A first-order formula with no free variable occurrences is called a first-order *sentence*, where an occurrence of x is free if it does not lie in the scope of either $\exists x$ or $\forall x$. In addition to the syntax of FO, SO includes and quantifies over second-order variables that range over relations, are denoted by uppercase letters, and each of them has an arity. SO includes formulae of the form $X(x_1, \ldots, x_k)$, where X is a second-order variable of arity k, and $x_1, \ldots, x_k$ are first-order variables. The fragment of SO consisting only of existential second-order formulae is called existential second-order logic and is abbreviated as $\exists$SO. We use the usual $\mathcal{A}, v, V \models \varphi$ interpretation of an SO-formula φ, given a structure $\mathcal{A}$ and first- and second-order assignments v and V, respectively. If φ has no free first- or second-order variables, v or V, respectively, can be omitted. We refer the reader to [16] for a more extensive presentation of FO and SO.

The logical symbols of Quantitative Second-Order logic, denoted by QSO, include all the logical symbols of SO and the quantitative quantifiers Σ and Π for sum and product quantification, respectively. The arity of a second-order variable X is denoted by $\mathsf{arity}(X)$. When we write logic Λ over σ, we mean the set of Λ formulae over σ. The set of QSO formulae over σ are defined by the following grammar:

$$\alpha ::= \varphi \mid s \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma x.\alpha \mid \Pi x.\alpha \mid \Sigma X.\alpha \mid \Pi X.\alpha \tag{1}$$

where φ is an SO formula over σ, $s \in \mathbb{N}$, x is a first-order and X a second-order variable. A formula α in QSO is a sentence if every variable occurrence in α is bound by a first-order, second-order, or quantitative quantifier. The evaluation of a QSO formula α is a function $[\![\alpha]\!]$ that on input $\mathcal{A}, v$, and V returns a number in $\mathbb{N}$. We refer the reader to [5, p. 5] for the definition of the semantics of QSO formulae. When α is a sentence, $[\![\alpha]\!](\mathcal{A})$ is used to denote $[\![\alpha]\!](\mathcal{A}, v, V)$ for any v, V. We say that $f \in \mathsf{QSO}$ if there exists $\alpha \in$ QSO such that $f(\mathrm{enc}(\mathcal{A})) = [\![\alpha]\!](\mathcal{A})$, for every $\mathcal{A}$. Note that QSO is a set of logical formulae, whereas QSO is a class of functions. For every logic Λ, we can define a corresponding class of functions as above, and we denote it by $\mathsf{\Lambda}$.

▶ **Definition 2.7.** *A logic Λ captures a complexity class C, and equivalently $\mathsf{C} = \mathsf{\Lambda}$, over finite ordered structures over σ, if the following two conditions hold:*

1. *For every $f \in \mathsf{C}$, there is a sentence $\alpha \in \Lambda$, such that $f(\mathrm{enc}(\mathcal{A})) = [\![\alpha]\!](\mathcal{A})$ for every finite ordered structure $\mathcal{A}$ over σ.*
2. *For every sentence $\alpha \in \Lambda$, there is a function $f \in \mathsf{C}$, such that $[\![\alpha]\!](\mathcal{A}) = f(\mathrm{enc}(\mathcal{A}))$ for every finite ordered structure $\mathcal{A}$ over σ.*

Moreover, Λ captures C over finite ordered structures if Λ captures C over finite ordered structures over σ, for every σ.

For example, ΣQSO(FO) = #P over finite ordered structures [5], where ΣQSO(FO) is the set of QSO formulae that Π is not allowed and φ in (1) is restricted to be an FO formula.

Triples $(\mathcal{A}, v, V)$ can be encoded in space polynomial in $|A|$ using a standard mapping from finite ordered structures to strings over $\{0, 1\}$ (see for example [26, Chapter 6]). We assume that a TM M takes as input the encoding of $\mathcal{A}$ (or $(\mathcal{A}, v, V)$), denoted by $\mathrm{enc}(\mathcal{A})$ (resp. $\mathrm{enc}(\mathcal{A}, v, V)$), even if we write $M(\mathcal{A})$ (resp. $M(\mathcal{A}, v, V)$) for the sake of brevity.

In all cases that we consider in this paper, the initial configuration of a TM is FO definable [21] and therefore, to prove that Λ captures C, it suffices to verify conditions 1 and 2 in Definition 2.7 for $f(\mathrm{enc}(\mathcal{A}, v, V)) = [\![\alpha]\!](\mathcal{A}, v, V)$, where v, V encode the initial

$$
\begin{aligned}
\mathrm{Expl}[x](\mathcal{A}, v, V) &= \{v(x)\} \\
\mathrm{Expl}[X](\mathcal{A}, v, V) &= \{V(X)\} \\
\mathrm{Expl}[\varphi](\mathcal{A}, v, V) &= \begin{cases} \{\varepsilon\}, & \text{if } \mathcal{A}, v, V \models \varphi \\ \emptyset, & \text{otherwise} \end{cases} \\
\mathrm{Expl}[\alpha_1 + \alpha_2](\mathcal{A}, v, V) &= \mathrm{Expl}[\alpha_1](\mathcal{A}, v, V) \cup \mathrm{Expl}[\alpha_2](\mathcal{A}, v, V) \\
\mathrm{Expl}[\alpha_1 \cdot \alpha_2](\mathcal{A}, v, V) &= \mathrm{Expl}[\alpha_1](\mathcal{A}, v, V) \circ \mathrm{Expl}[\alpha_2](\mathcal{A}, v, V) \\
\mathrm{Expl}[\Sigma y.\alpha](\mathcal{A}, v, V) &= \bigcup_{a \in A} \mathrm{Expl}[\alpha](\mathcal{A}, v[a/y], V) \\
\mathrm{Expl}[\Sigma Y.\alpha](\mathcal{A}, v, V) &= \bigcup_{B \subseteq A^k} \mathrm{Expl}[\alpha](\mathcal{A}, v, V[B/Y])
\end{aligned}
$$

Table 1 Intermediate semantics of $\Sigma\mathtt{SO}(\underline{\Lambda})$ formulae.

configuration of a TM that corresponds to f. Finally, we often use that (a) $\mathcal{A}, v, V \models \varphi$ can be decided in deterministic logarithmic space, if φ is an $\mathtt{FO}$ formula, and in deterministic polynomial space, if $\varphi \in \mathtt{SO}$, for every finite structure $\mathcal{A}$ [21], and (b) given $\mathcal{A}$, the lexicographic order on k-tuples over A induced by $\leq$ is $\mathtt{FO}$ expressible and is also denoted by $\leq$.

3 The quantitative logic $\Sigma\mathtt{SO}(\underline{\Lambda})$

The logic $\Sigma\mathtt{SO}(\underline{\Lambda})$ over σ, where $\Lambda \in \{\mathtt{FO}, \mathtt{SO}\}$, is defined by the following grammar:

$$\alpha ::= x \mid X \mid \varphi \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma y.\alpha \mid \Sigma Y.\alpha \tag{2}$$

where φ is in Λ, x, y are first-order variables, and X, Y are second-order variables. The syntax of logic $\Sigma\mathtt{SO}(\underline{\Lambda})$ is the same as that of $\Sigma\mathtt{QSO}(\Lambda)$, where a formula can also be a first- and second-order variable, but not a number $s \in \mathbb{N}$. $\Sigma\mathtt{FO}(\underline{\Lambda})$ is the fragment of $\Sigma\mathtt{SO}(\underline{\Lambda})$ in which Σ is not allowed over second-order variables. We say that a $\Sigma\mathtt{SO}(\underline{\Lambda})$ formula is *x-free* (resp. *X-free*) if it is given by grammar (2) without x (resp. X).

▶ Notation Remark 3.1. We denote $X \cdot \varphi(X)$ (or $\varphi(X) \cdot X$) by $\varphi(\underline{X})$.

We define the semantics of the logic $\Sigma\mathtt{SO}(\underline{\Lambda})$ in two phases: a formula α is mapped to a set of strings. Then, the semantic interpretation of formula α is defined to be the size of this set. Formally, $[\![\alpha]\!](\mathcal{A}, v, V) = |\mathrm{Expl}[\alpha](\mathcal{A}, v, V)|$, where $\mathrm{Expl}[\alpha](\mathcal{A}, v, V)$ is recursively defined in Table 1. Expl stands for Explicit and we call $\mathrm{Expl}[\alpha](\mathcal{A}, v, V)$ the *intermediate semantic interpretation* of formula α. Note that $\cup$ and $\circ$ between sets of strings have replaced sum and multiplication of natural numbers, respectively, in the semantics of $\mathtt{QSO}$. $S_1 \cup S_2$ is the union of S_1 and S_2, whereas $S_1 \circ S_2$ is *concatenation* of sets of strings lifted from the concatenation operation on strings, that is $S_1 \circ S_2 = \{x \circ y \mid x \in S_1, y \in S_2\}$. For example, $\{\varepsilon, a_1, a_2a_3\} \circ \{\varepsilon, a_2a_3\} = \{\varepsilon, a_2a_3, a_1, a_1a_2a_3, a_2a_3a_2a_3\}$, where ε denotes the empty string. In specific, if one of S_1, S_2 is $\emptyset$, then $S_1 \circ S_2 = \emptyset$.

▶ Notation Remark 3.2. For a finite set K, $K^* := \bigcup_{n \in \mathbb{N}} K^n$ denotes the set of strings over K, $\mathcal{P}(K^*)$ the powerset of K^*, and ε the empty string. For an $\mathcal{A}$ over σ, $\mathcal{R}_k := \mathcal{P}(A^k)$ denotes the set of relations on A of arity k.

▶ Remark 3.3. Note that for a formula $\alpha \in \Sigma\mathtt{SO}(\underline{\Lambda})$ and $s \in \mathrm{Expl}[\alpha](\mathcal{A}, v, V)$, we have that $s \in (A \cup \bigcup_{i\in\mathbb{N}} \mathcal{R}_i)^*$. In this paper, we consider logics that are either X-free or x-free, and so for a formula α in some of these logics and $s \in \mathrm{Expl}[\alpha](\mathcal{A}, v, V)$, either $s \in A^*$ or $s \in (\bigcup_{i\in\mathbb{N}} \mathcal{R}_i)^*$, respectively.

The length of α, denoted by $|\alpha|$, is defined as the length of α as a string of symbols, boolean formulae and sum operators are treated as one symbol. The length of $s \in A^* \cup (\bigcup_{i\in\mathbb{N}} \mathcal{R}_i)^*$, denoted by $|s|$, is the standard length of strings. It is not hard to define an *encoding* $\mathrm{enc}(s)$ of s, such that $|\mathrm{enc}(s)| \leq |s| \cdot \log|A|$, if $s \in A^*$, and $|\mathrm{enc}(s)| \leq |s| \cdot |A|^k$, if $s \in (\bigcup_{1\leq i\leq k} \mathcal{R}_i)^*$.

▶ **Lemma 3.4.** *Let α be a $\Sigma\mathtt{SO}(\underline{\Lambda})$ formula over σ. For every finite ordered structure $\mathcal{A}$ over σ, v, and V, and every $s \in \mathrm{Expl}[\alpha](\mathcal{A}, v, V)$, $|s| \leq |\alpha|$. Moreover, (a) if α is an X-free formula, then $|\mathrm{enc}(s)| \leq |\alpha| \cdot \log|A|$, and (b) if α is an x-free formula, then $|\mathrm{enc}(s)| \leq |\alpha| \cdot \mathrm{poly}(|A|)$.*

3.1 The logic $\Sigma\mathtt{SO}(\underline{\Lambda})$ with recursion

By adding a function symbol f to the syntax of $\Sigma\mathtt{SO}(\underline{\Lambda})$, we obtain formulae defined below:

$$\beta ::= x \mid X \mid \varphi \mid f(x_1, \ldots, x_k) \mid (\beta + \beta) \mid (\beta \cdot \beta) \mid \Sigma y.\beta \mid \Sigma Y.\beta \tag{3}$$

where f is a *first-order function symbol* with $\mathsf{arity}(f) = k$, and $x_1, \ldots, x_k$ are first-order variables, also denoted by $\vec{x}$. In like manner, we can add a *second-order function symbol* to $\Sigma\mathtt{SO}(\underline{\Lambda})$. In particular, we consider only second-order function symbols of arity 1, i.e. of the form $f(X)$, where X is a second-order variable. A $\Sigma\mathtt{SO}(\underline{\Lambda})$ formula $\beta(X, f)$ with a second-order function symbol $f(Y)$ is called *arity-consistent* when it has at most one free second-order variable X, where X has the same arity as Y. We fix an arity k for the first-order function symbol, or the argument of the second-order function symbol.

To extend the semantics of $\Sigma\mathtt{SO}(\underline{\Lambda})$ to the case of formula $f(x_1, \ldots, x_k)$, we say that F is a *first-order function assignment* for $\mathcal{A}$, if $F(f) : A^k \to \mathcal{P}(A^*)$. In the case of formula $f(X)$, we say that F is a *second-order function assignment* for $\mathcal{A}$, if $F(f) : \mathcal{R}_k \to \mathcal{P}(K^*)$, where K can be either A or $\bigcup_{i\in\mathbb{N}} \mathcal{R}_i$. We define $\mathcal{FOF}$ to be the set of functions $h : A^k \to \mathcal{P}(A^*)$, $\mathcal{SOF}$ the set of functions $h : \mathcal{R}_k \to \mathcal{P}(A^*)$, and $\mathcal{RSOF}$ the set of functions $h : \mathcal{R}_k \to \mathcal{P}((\bigcup_{i\in\mathbb{N}} \mathcal{R}_i)^*)$.

Given v and V, we define $\mathrm{Expl}[f(\vec{x})](\mathcal{A}, v, V, F) := F(f)(v(\vec{x}))$ and $[\![f(\vec{x})]\!](\mathcal{A}, v, V, F) := |F(f)(v(\vec{x}))|$. The semantics of $f(X)$ are defined in an analogous way. Now we can add to the syntax of $\Sigma\mathtt{SO}(\underline{\Lambda})$, formulae of the form $[\mathrm{lfp}_f\beta](\vec{x})$ (resp. $[\mathrm{lfp}_f\beta](X)$), where β is a (resp. arity-consistent) $\Sigma\mathtt{SO}(\underline{\Lambda})$ formula equipped with a first-order (resp. second-order) function symbol f. To define the semantics of $[\mathrm{lfp}_f\,\beta](\vec{x})$, we first define two lattices. The first lattice is $(\mathcal{P}(A^*), \subseteq)$, i.e. it contains all sets of strings over A. The bottom element is $\emptyset$ and the top element is the set A^*. The second lattice is $(\mathcal{FOF}, \leq_F)$: for $g, h \in \mathcal{FOF}$, $g \leq_F h$ iff $g(\vec{x}) \subseteq h(\vec{x})$, for every $\vec{x}$. The bottom element is g_0 which takes the value $\emptyset$ for every $\vec{x}$, and the top element is g_{max}, which is equal to A^* for every $\vec{x}$. For an infinite increasing sequence of functions $h_1 \leq_F h_2 \leq_F h_3 \leq_F \cdots$ from $\mathcal{FOF}$, we define $\lim_{n\to+\infty} h_n := h$, where for every $x \in A^k$, $h(x) = \bigcup_{n\in\mathbb{N}} h_n(x)$.

We interpret $\beta(\vec{x}, f)$ as an operator T_β on $\mathcal{FOF}$. For every $h \in \mathcal{FOF}$ and $\vec{a} \in A^k$, $T_\beta(h)(\vec{a}) = \mathrm{Expl}[\beta(\vec{x}, f)](\mathcal{A}, v, V, F)$, where v is a first-order assignment for $\mathcal{A}$ such that $v(\vec{x}) = \vec{a}$ and F is a first-order function assignment for $\mathcal{A}$ such that $F(f) = h$. The following propositions state that T_β is monotone on $(\mathcal{FOF}, \leq_F)$.

▶ **Proposition 3.5.** *Let f be a first-order function symbol with $\mathsf{arity}(f) = k$ and β be a formula over σ defined by grammar (3), such that if β contains a function symbol, then this function symbol is f. Let also $\mathcal{A}$ be a finite ordered structure over σ, $h, g : A^k \to \mathcal{P}(A^*)$ and*

H, G be function assignments such that $H(f) = h$ and $G(f) = g$. If $h \leq_F g$, then for every first- and second-order assignments v and V, respectively:

$$\mathrm{Expl}[\beta](\mathcal{A}, v, V, H) \subseteq \mathrm{Expl}[\beta](\mathcal{A}, v, V, G).$$

▶ **Proposition 3.6.** *For every formula $[\mathrm{lfp}_f\, \beta](\vec{x})$, where β is in $\Sigma\mathtt{SO}(\underline{\Lambda})$ equipped with a first-order function symbol, operator T_β is monotone on the complete lattice $(\mathcal{FOF}, \leq_F)$. In other words, for every $h, g \in \mathcal{FOF}$, if $h \leq_F g$, then $T_\beta(h) \leq_F T_\beta(g)$.*

Thus, by the Knaster–Tarski theorem, T_β has a least fixed point. To compute the least fixed point of T_β, let us consider the sequence of functions $\{h_i\}_{i\in\mathbb{N}}$, $h_i : A^k \to \mathcal{P}(A^*)$, where $h_0(\vec{a}) = \emptyset$ for every $\vec{a} \in A^k$, and $h_{i+1} := T_\beta(h_i)$, for every $i \in \mathbb{N}$. We define $\mathrm{lfp}(T_\beta) := \lim_{n\to+\infty} h_n$. Finally, $\mathrm{Expl}[\,[\mathrm{lfp}_f\beta](\vec{x})\,](\mathcal{A}, v, V) := \mathrm{lfp}(T_\beta)(v(\vec{x})) = \lim_{n\to+\infty} h_n(v(\vec{x}))$ and $[\![\,[\mathrm{lfp}_f\beta](\vec{x})\,]\!](\mathcal{A}, v, V) = |\lim_{n\to+\infty} h_n(v(\vec{x}))|$. The semantics of $[\mathrm{lfp}_f\beta](X)$ are defined in a completely analogous way. Examples 4.3, 4.8, and 4.9 make clear how formulae of the form $[\mathrm{lfp}_f\beta](\vec{x})$ are interpreted.

The logics we define below are fragments of $\Sigma\mathtt{SO}(\underline{\mathtt{SO}})$ with recursion. Given a formula $[\mathrm{lfp}_f\beta](\vec{x})$ or $[\mathrm{lfp}_f\beta](X)$ in any of them, operator T_β is monotone on the complete lattice $(\mathcal{F}, \leq_F)$, where $\mathcal{F}$ can be $\mathcal{FOF}$, $\mathcal{SOF}$, or $\mathcal{RSOF}$.

▶ Remark 3.7. The name of a logic with recursion will be of the form $\mathtt{R}_{\mathtt{L}_1}\Sigma_{\mathtt{L}_2}(\mathtt{L}_3)$, where $\mathtt{L}_1 \in \{\mathtt{fo}, \mathtt{so}\}$ indicates that function symbol f is over first- or second-order variables, respectively, $\mathtt{L}_2 \in \{\mathtt{fo}, \mathtt{so}\}$ means that quantifier Σ is over first- or second-order variables, respectively, and $\mathtt{L}_3 \in \{\mathtt{FO}, \mathtt{SO}\}$ means that φ in (2) is in $\mathtt{L}_3$.

4 Logics that capture SpanL and SpanPSPACE

▶ **Definition 4.1.** $\mathtt{R}_{\mathtt{fo}}\Sigma_{\mathtt{fo}}(\mathtt{FO})$ *over σ is the set of formulae $[\mathrm{lfp}_f\beta](\vec{x})$, where β is defined by:*

$$\beta ::= \ \alpha \ \mid \ f(x_1, \ldots, x_k) \ \mid \ (\beta + \beta) \ \mid \ (\alpha \cdot \beta) \ \mid \ \Sigma y.\beta \tag{4}$$

where α is an X-free $\Sigma\mathtt{FO}(\underline{\mathtt{FO}})$ formula over σ, $x_1, \ldots, x_k, y$ are first-order variables, and f is a first-order function symbol.

▶ Remark 4.2. Notice that for a formula $[\mathrm{lfp}_f\beta](\vec{x}) \in \mathtt{R}_{\mathtt{fo}}\Sigma_{\mathtt{fo}}(\mathtt{FO})$, it may be the case that $[\![\,[\mathrm{lfp}_f\beta](\vec{x})\,]\!](\mathcal{A}, v, V) = +\infty$ analogously to the fact that the computation of an NLTM may contain cycles. For the sake of simplicity, we assume that an NL-transducer M can have infinitely many accepting paths and SpanL contains functions from Σ^* to $\mathbb{N} \cup \{+\infty\}$. To be in accordance with the literature, we can adjust the syntax of $\mathtt{R}_{\mathtt{fo}}\Sigma_{\mathtt{fo}}(\mathtt{FO})$ formulae to express the operation of the clock attached to NLTMs as discussed in Remark 2.4.

Let N be an NL-transducer and $\mathcal{A}$ be over σ with $|A| = n$. The number of different configurations of N is at most $n^k - 1$ for some $k \in \mathbb{N}$. To encode them, we use k-tuples over A. To encode the output symbol, if any, that is produced at some configuration, it suffices to use two distinct elements of A, since the output alphabet is $\Sigma = \{0, 1\}$; we use the minimum element and the successor of the minimum element, which are both FO expressible. Below, we informally write $\varphi(c)$ to denote $\varphi(x)$ interpreted in $\mathcal{A}$ where first-order variable x is assigned $c \in A$. Formula $[\mathrm{lfp}_f\mathsf{span}_{\mathtt{L}}](\vec{x})$ counts the different valid outputs of N, where $\mathsf{span}_{\mathtt{L}}(\vec{x}, f)$ is:

$$\mathsf{acc}(\vec{x}) \ + \Sigma\vec{y}.\Sigma z.\big(\mathsf{output}_0(\vec{x}, \vec{y}, \underline{z}) + \mathsf{output}_1(\vec{x}, \vec{y}, \underline{z}) + \mathsf{next}_0(\vec{x}, \vec{y}) + \mathsf{next}_1(\vec{x}, \vec{y})\big) \cdot f(\vec{y}).$$

Algorithm 1 NLTM MSp^{sub}_{β}.

Input: $\gamma, \mathcal{A}, v, V$, where γ is a subformula of β
if $\gamma == \alpha$ *has no function symbol* **then** simulate the transducer from Proposition 4.5
if $\gamma == f(\vec{y})$ **then** simulate $MSp^{sub}_{\beta}(\beta, \mathcal{A}, v[v(\vec{y})/\vec{x}], V)$
if $\gamma == \gamma_1 + \gamma_2$ **then**
  non-deterministically choose $\gamma' \in \{\gamma_1, \gamma_2\}$
  simulate $MSp^{sub}_{\beta}(\gamma', \mathcal{A}, v, V)$
if $\gamma == \alpha \cdot \gamma'$ **then**
  for $s \in A^*$ *where* $|s| \leq |\alpha|$ **do**
    if $s \in \mathrm{Expl}[\alpha](\mathcal{A}, v, V)$ **then** simulate $MSp^{sub}_{\beta}(\gamma', \mathcal{A}, v, V)$
if $\gamma == \sum y.\gamma'$ **then**
  non-deterministically choose $a \in A$
  simulate $MSp^{sub}_{\beta}(\gamma', \mathcal{A}, v[a/y], V)$

Interpretations of z and $\vec{x}, \vec{y}$ encode a bit of the output, and configurations of N, respectively. Formulae $\mathsf{next}_i(\vec{c}, \vec{c'})$, $i = 0, 1$, say that if N is in configuration $\vec{c}$ and makes non-deterministic choice i, then it is in $\vec{c'}$, and no output symbol is produced. Formulae $\mathsf{output}_i(\vec{c}, \vec{c'}, b)$, $i = 0, 1$, state that N makes choice i and so it transitions from configuration $\vec{c}$ to $\vec{c'}$ and writes the bit encoded by b on the next output cell. When N is in some $\vec{c}$ that only a deterministic transition can be made, then exactly one of $\mathsf{next}_i(\vec{c}, \vec{c'})$, $\mathsf{output}_i(\vec{c}, \vec{c'}, b)$, $i = 0, 1$, is satisfied in $\mathcal{A}$ for a $\vec{c'} \in A^k$ (and a $b \in A$). Formula $\mathsf{acc}(\vec{c})$ states that $\vec{c}$ is the accepting configuration. All aforementioned formulae can be expressed in FO. Note that for any $\mathcal{A}$, v, and V, $\mathrm{Expl}[\,[\mathrm{lfp}_f \mathsf{span_L}](\vec{x})\,](\mathcal{A}, v, V)$ is a set of strings in A^* that encode the outputs of N.

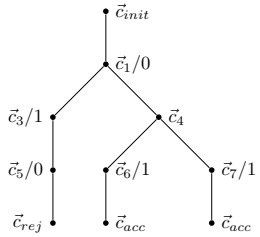

Figure 1 The computation tree of a transducer N on some input $\mathrm{enc}(\mathcal{A})$. $\vec{c}/b$ represents that N enters configuration encoded by $\vec{c}$ and writes bit b on the output tape.

▶ **Example 4.3.** Consider the computation tree shown in Figure 1 which corresponds to a transducer N that on input $\mathrm{enc}(\mathcal{A})$ has three outputs, and $span_N(\mathrm{enc}(\mathcal{A})) = 1$. Let $\mathbf{0}, \mathbf{1}$ denote the minimum and the successor of the minimum element of A, respectively. Then,

- $\mathrm{Expl}[\,[\mathrm{lfp}_f \mathsf{span_L}](\vec{x})\,](\mathcal{A}, v[\vec{c}_{acc}/\vec{x}]) = \{\varepsilon\}$, and
  $\mathrm{Expl}[\,[\mathrm{lfp}_f \mathsf{span_L}](\vec{x})\,](\mathcal{A}, v[\vec{c}_{rej}/\vec{x}]) = \emptyset$,
- $\mathrm{Expl}[\,[\mathrm{lfp}_f \mathsf{span_L}](\vec{x})\,](\mathcal{A}, v[\vec{c}_1/\vec{x}]) = \emptyset \cup \{\mathbf{1}\} \circ f(\vec{c}_3) \cup f(\vec{c}_4) = \{\mathbf{1}\} \circ (\mathbf{0} \circ \emptyset) \cup \{\mathbf{1}\} \circ \{\varepsilon\} = \{\mathbf{1}\}$,
- $\mathrm{Expl}[\,[\mathrm{lfp}_f \mathsf{span_L}](\vec{x})\,](\mathcal{A}, v[\vec{c}_{init}/\vec{x}]) = \emptyset \cup \{\mathbf{0}\} \circ f(\vec{c}_1) = \{\mathbf{01}\}$.

Intuitively, the intermediate interpretation of $[\mathrm{lfp}_f \mathsf{span_L}](\vec{c})$ is the set of the different valid outputs N produces during its computation starting from the configuration encoded by $\vec{c}$.

▶ **Proposition 4.4.** *Given an NL-transducer N, $span_N(\mathrm{enc}(\mathcal{A})) = [\![\, [\mathrm{lfp}_f \mathsf{spanL}](\vec{x}) \,]\!](\mathcal{A}, v, V)$, for every $\mathcal{A}$, v, and V, such that $v(\vec{x})$ encodes the starting configuration of N.*

To prove that $\mathsf{R_{fo}\Sigma_{fo}(FO)} \subseteq \mathsf{SpanL}$, first note that X-free $\Sigma\mathtt{FO}(\underline{\mathtt{FO}})$ formulae can be easily evaluated by NLTMs as Proposition 4.5 states.

▶ **Proposition 4.5.** *For every X-free $\Sigma\mathtt{FO}(\underline{\mathtt{FO}})$ formula α over σ, there is an NL-transducer M, that on input $\mathrm{enc}(\mathcal{A}, v, V)$ has exactly one accepting run for each $s \in \mathrm{Expl}[\alpha](\mathcal{A}, v, V)$, on which it outputs $enc(s)$, and no other accepting runs.*

▶ **Proposition 4.6.** *Let $[\mathrm{lfp}_f \beta](\vec{x})$ be an $\mathtt{R_{fo}\Sigma_{fo}(FO)}$ formula over σ. There is an NL-transducer M_β, such that $span_{M_\beta}(\mathrm{enc}(\mathcal{A}, v, V)) = [\![\, [\mathrm{lfp}_f \beta](\vec{x}) \,]\!](\mathcal{A}, v, V)$, for every $\mathcal{A}, v$ and V.*

Proof. Let $[\mathrm{lfp}_f \beta](\vec{x}) \in \mathtt{R_{fo}\Sigma_{fo}(FO)}$. The NL-transducer $M_\beta(\mathcal{A}, v, V)$ calls $MSp^{sub}_\beta(\beta, \mathcal{A}, v, V)$ from Algorithm 1. If β does not contain a function symbol, then $[\![\, [\mathrm{lfp}_f \beta](\vec{x}) \,]\!](\mathcal{A}, v, V) = [\![\beta]\!](\mathcal{A}, v, V)$. By Proposition 4.5, there is an NL-transducer M, such that $span_M(\mathrm{enc}(\mathcal{A}, v, V)) = [\![\beta]\!](\mathcal{A}, v, V)$. In this case, let M_β be M. Similarly, for any subformula α of β without function symbols, M_α is the NL-transducer associated with α from Proposition 4.5. ◀

▶ **Theorem 4.7.** $\mathsf{R_{fo}\Sigma_{fo}(FO) = SpanL}$ *over finite ordered structures.*

The following are examples of specific SpanL problems expressed in $\mathtt{R_{fo}\Sigma_{fo}(FO)}$.

▶ **Example 4.8.** Let $\mathcal{G} = \langle V, E, \leq \rangle$ represent a directed graph with a source. Then, $[\![\, [\mathrm{lfp}_f \beta](x) \,]\!](\mathcal{G}, v, V)$ is the number of sinks in the graph, where $\beta(x, f) := \forall y \neg E(x, y) \cdot x + \Sigma y.E(x, y) \cdot f(y)$, and $v(x)$ is the source of the graph.

▶ **Example 4.9.** Let $\mathcal{N} = \langle Q = \{q_0, \ldots, q_{n-1}, \ell_0, \ldots, \ell_m\}, L, E_0, E_1, \leq \rangle$ represent an NFA N over the input alphabet $\{0, 1\}$, together with 1^m; Q is the universe, $L = \{\ell_0, \ldots, \ell_m\}$ is a relation that distinguishes states of N from the encoding of 1^m, and E_i, $i = 0, 1$, is the set of i-transitions of N. Let $\beta(x, y, f) := \mathsf{acc}(x) + (y < \max) \cdot \Sigma x'.\Sigma y'.(y' = y + 1) \cdot \big(E_0(x, x') \cdot \min_0 + E_1(x, x') \cdot \min_1\big) \cdot f(x', y')$, where $\min_0$ and $\min_1$, and max express the minimum, the successor of the minimum, and the maximum element of Q, respectively, $\mathsf{acc}(x)$ expresses that x is an accepting state of N, and $y' = y + 1$ is defined so that y' is the successor of y. Then, $[\![\, [\mathrm{lfp}_f \beta](x, y) \,]\!](\mathcal{N}, v, V)$ is the number of strings of length at most m accepted by N, where $v(x)$ encodes the starting state of N, and $v(y)$ encodes the minimum element of L. This problem is SpanL-complete and was defined in [2] as the *census function* of an NFA.

We now introduce the logic $\mathtt{R_{so}\Sigma_{so}(SO)}$, which captures SpanPSPACE.

▶ **Definition 4.10.** $\mathtt{R_{so}\Sigma_{so}(SO)}$ *over σ is the set of formulae $[\mathrm{lfp}_f \beta](X)$, where β is defined by:*

$$\beta ::= \alpha \mid f(X) \mid (\beta + \beta) \mid (\alpha \cdot \beta) \mid \Sigma y.\beta \mid \Sigma Y.\beta \tag{5}$$

where α is an X-free $\Sigma\mathtt{SO}(\underline{\mathtt{SO}})$ formula over σ, y is a first-order variable, X, Y are second-order variables, and f is a second-order function symbol.

▶ Remark 4.11. Relations $R_1, \ldots, R_m$ on A with $\mathsf{arity}(R_j) = k$, $1 \leq j \leq m$, can be encoded by one relation R on A of arity $k + \lceil \log m \rceil$, by defining $R(\vec{i}, \vec{a})$ iff $R_i(\vec{a})$, for every $\vec{a} \in A^k$, where $\vec{i}$ is the i-th smallest $\lceil \log m \rceil$-tuple over A. We use this observation to show that a second-order function symbol f with $\mathsf{arity}(f) = 1$, suffices to capture SpanPSPACE.

▶ Remark 4.12. To avoid formulae $[\mathrm{lfp}_f \beta](X) \in \mathtt{R_{so}\Sigma_{so}(SO)}$ with $[\![[\mathrm{lfp}_f \beta](X)]\!](\mathcal{A}, v, V) = +\infty$, we can adjust the syntax of $\mathtt{R_{so}\Sigma_{so}(SO)}$ similarly to Remark 4.2. The only difference is that now the clock imposes an exponential-time bound.

Let $\mathcal{A}$ over σ with $|A| = n$ and $M = (\mathcal{Q}, \Sigma, \delta, q_0, q_F)$ be a non-deterministic poly-space transducer that uses $n^d - 1$ space. Let also $k = \max\{d, \lceil \log |\mathcal{Q}| \rceil\}$. We can use k-tuples over A, to encode $n^d - 1$ tape cells and $|\mathcal{Q}|$ states. W.l.o.g. assume that M has a single tape. A configuration of M can be encoded by the tuple of k-ary relations $\vec{C} = (T, E, P, Q)$: $T(\vec{c})$ iff cell c encoded by $\vec{c}$ contains symbol 1 (tape contents), $E(\vec{c})$ denotes that all cells greater than c contain the symbol $\llcorner$ (end of zeros and ones on the tape), $P(\vec{c})$ indicates that the head is on cell c (head's position), and $Q(\vec{c})$ means that N is in state q that is encoded by $\vec{c}$. As in the case of SpanL, a bit that M outputs at some time step is encoded using two elements of A. Formulae $\mathsf{Next}_i(\vec{X}, \vec{Y})$, $\mathsf{Output}_i(\vec{X}, \vec{Y}, x)$, $i = 0, 1$, and $\mathsf{Acc}(\vec{X})$ express similar facts for the computation of M as the respective formulae defined for SpanL. They can be expressed in FO as the formulae that describe the computation of an NPTM in the proof of Fagin's theorem [21]. By Remark 4.11, the aforementioned formulae can be replaced by first-order formulae such that a unique relation is used to encode the configuration of M. Therefore, we abuse notation and write $\mathsf{Next}_i(X, Y)$, $\mathsf{Output}_i(X, Y, x)$, and $\mathsf{Acc}(X)$.

▶ **Theorem 4.13.** $\mathsf{R_{so}\Sigma_{so}(SO)} = \mathsf{SpanPSPACE}$ *over finite ordered structures.*

Proof. The proof of $\mathsf{R_{so}\Sigma_{so}(SO)} \subseteq \mathsf{SpanPSPACE}$ is analogous to that of Proposition 4.6. For the inclusion $\mathsf{SpanPSPACE} \subseteq \mathsf{R_{so}\Sigma_{so}(SO)}$, given a non-deterministic poly-space transducer M, consider the formula $\mathsf{span}_{\mathsf{pspace}}(X, f) := \mathsf{Acc}(X) + \Sigma Y.\Sigma x.\big(\mathsf{Output}_0(X, Y, \underline{x}) + \mathsf{Output}_1(X, Y, \underline{x}) + \mathsf{Next}_0(X, Y) + \mathsf{Next}_1(X, Y)\big) \cdot f(Y)$. Then, $[\![[\mathrm{lfp}_f \mathsf{span}_{\mathsf{pspace}}](X)]\!](\mathcal{A}, v, V) = span_M(\mathrm{enc}(\mathcal{A}))$, for every $\mathcal{A}$, v, V, such that $V(X)$ encodes the initial configuration of M. ◀

5 $\mathtt{R^r_{so}\Sigma_{so}(SO)}$ captures #PSPACE

In this section, we prove that the logic $\mathtt{\Sigma SO(\underline{SO})}$ equipped with a second-order function symbol and a restricted form of recursion captures #PSPACE over finite ordered structures. Superscript $\mathtt{r}$ in the name of the logic stands for the fact that recursion is restricted.

▶ **Definition 5.1.** $\mathtt{R^r_{so}\Sigma_{so}(SO)}$ *over σ is the set of formulae $[\mathrm{lfp}_f \beta](X)$, where β is defined by:*

$$\beta ::= \alpha \mid (\alpha + \beta) \mid \Sigma Y.\varphi(X, \underline{Y}) \cdot f(Y) \tag{6}$$

where X, Y are second-order variables, φ is an SO *formula over σ, α is an x-free* $\mathtt{\Sigma SO(\underline{SO})}$ *formula over σ, and f is a second-order function symbol.*

▶ Remark 5.2. In the case of #PSPACE, we can attach a clock to non-deterministic poly-space TMs, and restrict the syntax of $\mathtt{R^r_{so}\Sigma_{so}(SO)}$ accordingly, as in Section 4. An alternative approach is the following: it can be proven that for every $\beta \in \mathtt{R^r_{so}\Sigma_{so}(SO)}$, $[\![\beta]\!]$ is in FPSPACE in the sense that there is a deterministic poly-space TM N such that on input $\mathrm{enc}(\mathcal{A}, v, V)$ outputs $[\![\beta]\!](\mathcal{A}, v, V)$, if $[\![\beta]\!](\mathcal{A}, v, V) \in \mathbb{N}$, and the symbol $\perp$, if $[\![\beta]\!](\mathcal{A}, v, V) = +\infty$. By Proposition 2.5, $\mathsf{R^r_{so}\Sigma_{so}(SO)} \subseteq \#\mathsf{PSPACE}$, where we consider a slightly different kind of a non-deterministic poly-space TM which on input x, if $f(x) = +\infty$, it outputs $\perp$ and halts, and if $f(x) = m \in \mathbb{N}$, it generates m accepting paths.

▶ **Theorem 5.3.** $\mathsf{R^r_{so}\Sigma_{so}(SO)} = \#\mathsf{PSPACE}$ *over finite ordered structures.*

6 $\mathtt{R}^{\mathtt{r}}_{\mathtt{so}}\Sigma^{\mathtt{r}}_{\mathtt{so}}(\mathtt{FO})$ captures TotP

We define a fragment of $\Sigma\mathtt{SO}(\underline{\mathtt{FO}})$ with recursion, which we call $\mathtt{R}^{\mathtt{r}}_{\mathtt{so}}\Sigma^{\mathtt{r}}_{\mathtt{so}}(\mathtt{FO})$. Definitions 6.1 and 6.2 will be used to restrict the use of Σ operator.

▶ **Definition 6.1.** *We say that a formula $\varphi(Y)$* syntactically defines *Y if $\varphi(Y)$ is of the form $\forall\vec{y}Y(\vec{y}) \leftrightarrow \psi(\vec{y})$, for some formula ψ.*

▶ **Definition 6.2.** *We say that a formula $\varphi(X,Y)$ (a)* extends *X to Y if it is of the form $\forall\vec{y}Y(\vec{y}) \leftrightarrow X(\vec{y}) \vee \psi(X,\vec{y})$, and (b)* strictly extends *X to Y if it is of the form $\forall\vec{y}\big(Y(\vec{y}) \leftrightarrow X(\vec{y})\vee\psi(X,\vec{y})\big)\wedge\exists\vec{y}\big(\neg X(\vec{y})\wedge Y(\vec{y})\big)$, for some formula ψ and $\mathsf{arity}(X) = \mathsf{arity}(Y)$.*

▶ Notation Remark 6.3. (a) $Y := \varphi \cdot \alpha$ denotes $\Sigma Y.\varphi(Y)\cdot\alpha$, where φ syntactically defines Y, and (b) $\underline{Y} := \varphi(X)\cdot f(Y)$ denotes $\Sigma Y.\varphi(X,Y)\cdot Y\cdot f(Y)$, where φ (strictly) extends X to Y.

▶ **Definition 6.4.**

(a) *The $\Sigma\mathtt{SO}^{\mathtt{r}}(\underline{\mathtt{FO}})$ formulae over σ are the x-free $\Sigma\mathtt{SO}(\underline{\mathtt{FO}})$ formulae with the restriction that the second-order sum operator only appears as $Y := \varphi\cdot\alpha$, $\varphi\in\mathtt{FO}$.*

(b) *$\mathtt{R}^{\mathtt{r}}_{\mathtt{so}}\Sigma^{\mathtt{r}}_{\mathtt{so}}(\mathtt{FO})$ over σ is the set of formulae $[\mathrm{lfp}_f\,\beta](X)$, where β is defined by:*

$$\beta ::= \ \alpha \ \mid\ \underline{Y} := \psi(X)\cdot f(Y) \ \mid\ \alpha+\beta \ \mid\ \varphi\cdot\beta \ \mid\ \beta+\beta+\top \ \mid\ \varphi\cdot\beta+\neg\varphi\cdot\beta \qquad (7)$$

where α is a $\Sigma\mathtt{SO}^{\mathtt{r}}(\underline{\mathtt{FO}})$ formula, $\varphi,\psi\in\mathtt{FO}$, ψ strictly extends X to Y, and f is a second-order function symbol.

To express the generic TotP problem in $\mathtt{R}^{\mathtt{r}}_{\mathtt{so}}\Sigma^{\mathtt{r}}_{\mathtt{so}}(\mathtt{FO})$, we first describe how an NPTM run can be encoded. Let $\mathcal{A}$ be of size n and $N = (\mathcal{Q},\Sigma,\delta,q_0,q_F)$ be an NPTM that uses at most n^d-1 time. W.l.o.g. assume that N has a single tape. We define $\Gamma = \Sigma\cup\{\textvisiblespace\} = \{0,1,\textvisiblespace\}$, $\Gamma_{\mathcal{Q}} = \Gamma\times\mathcal{Q}$, and $k = \max\{d, \lceil\log(3+3|\mathcal{Q}|)\rceil\}$. To encode cells, time steps, and symbols in $\Gamma\cup\Gamma_{\mathcal{Q}}$, we use k-tuples over A. Let S be a relation of arity $3k$, such that, if $\vec{r}$ represents the symbol $\gamma\in\Gamma$, then $S(\vec{c},\vec{t},\vec{r})$ signifies that cell $\vec{c}$ contains symbol γ at time step $\vec{t}$. If $\vec{r}$ represents the symbol-state pair $(\gamma,q)\in\Gamma_{\mathcal{Q}}$, then $S(\vec{c},\vec{t},\vec{r})$ signifies that $\vec{c}$ contains symbol γ, the head is at cell $\vec{c}$, and N is in state q at time step $\vec{t}$. We use the FO expressible formulae $\vec{x}+1$ and min to describe the successor of $\vec{x}$ and the minimum k-tuple, respectively.

We say that a *relation* S of arity $3k$ on $\mathcal{A}$ *describes a partial run* $c_0c_1\cdots c_m$ of N, when (a) there is some $\vec{t}\in A^k$, such that for every $\vec{t'}\le\vec{t}$, there are $\vec{c},\vec{r}\in A^k$, such that $S(\vec{c},\vec{t'},\vec{r})$, and for every $\vec{t'}>\vec{t}$ and $\vec{c},\vec{r}\in A^k$, not $S(\vec{c},\vec{t'},\vec{r})$, (b) $S(-,\min,-)$ describes the encoding of the starting configuration c_0, and (c) if $S(-,\vec{t},-)$ describes the encoding of c_i, then $S(-,\vec{t}+1,-)$ either describes the encoding of c_{i+1} or is empty. We say that *formula* $\varphi(\vec{c},\vec{t},\vec{r})$ *describes a partial run* $c_0c_1\cdots c_m$, when φ defines in $\mathcal{A}$ a relation that does so. We use the standard notion of definability, where $\varphi(\vec{x})$ defines R in $\mathcal{A}$, if for every $\vec{a}\in A^k$, $R(\vec{a})$ iff $\mathcal{A}, v[\vec{a}/\vec{x}]\models\varphi(\vec{x})$. For example, let S_0 be a relation of arity $3k$ that describes the beginning of a run by N on $\mathrm{enc}(\mathcal{A})$. S_0 can be defined in FO by $\vec{y} = \min\wedge\varphi_{c_0}(\vec{x},\vec{z})$, where φ_{c_0} encodes the starting configuration, as, for instance, in [21].

Below we define formula $\mathsf{tot}(X,f)$, the least fixed point of which applied on S_0 is equal to the number of branchings of N on input $\mathrm{enc}(\mathcal{A})$:

$$\mathsf{branch}(X)\Big(\sum_{i=0,1}\underline{Y} := \mathsf{ndet}_i(X)\cdot f(Y)+\top\Big) + \neg\mathsf{branch}(X)\big(\mathsf{nfinal}(X)\cdot\underline{Y} := \mathsf{det}(X)\cdot f(Y)\big).$$

Let X be interpreted as a relation S_p that describes a partial run $c_0\ldots c_m$ of N. Formula branch checks whether the current configuration c_m creates a branching. Formulae ndet_i, $i=0,1$, and det extend S_p to a relation S_{new}, that describes the run $c_0\ldots c_mc_{m+1}$, where

c_{m+1} is the configuration that N reaches from c_m by making non-deterministic choice i or a deterministic transition, respectively. The evaluation continues recursively on S_{new}. Finally, if c_m is a configuration where N halts, nfinal becomes false and recursion stops. Moreover, $\mathsf{ndet}_i(X,Y)$, $i=0,1$, and $\mathsf{det}(X,Y)$ are $\mathtt{FO}$ formulae that strictly extend X to Y. As a result, there is a bijection between the strings in $\mathrm{Expl}[\,[\mathrm{lfp}_f\,\mathsf{tot}](X)\,](\mathcal{A},v,V)$ and branchings of $N(\mathrm{enc}(\mathcal{A}))$. Assume that c_m is a configuration that is not the initial configuration c_0 and leads to a non-deterministic choice. Then, c_m can be mapped to a string $S_1 \circ \ldots \circ S_i \in (\mathcal{R}_{3k})^*$ in $\mathrm{Expl}[\,[\mathrm{lfp}_f\,\mathsf{tot}](X)\,](\mathcal{A},v,V)$, where S_j extends S_{j-1}, for every $2 \leq j \leq i$, and S_i describes $c_0 \ldots c_m$. If c_0 leads to a non-deterministic choice, it is mapped to string ε.

▶ **Proposition 6.5.** *Given an NPTM N, $[\![\,[\mathrm{lfp}_f\,\mathsf{tot}](X)\,]\!](\mathcal{A},v,V) =$ $\#$(branchings of $N(\mathrm{enc}(\mathcal{A}))$, where $V(X)$ encodes the initial configuration of N.*

The specific form of any $[\mathrm{lfp}_f\,\beta](X) \in \mathtt{R}^{\mathtt{r}}_{\mathtt{so}}\Sigma^{\mathtt{r}}_{\mathtt{so}}(\mathtt{FO})$ guarantees that there is an NPTM that generates a number of paths equal to $[\![\,[\mathrm{lfp}_f\,\beta](X)\,]\!](\mathcal{A},v,V)+1$.

▶ **Theorem 6.6.** $\mathsf{R}^{\mathsf{r}}_{\mathsf{so}}\Sigma^{\mathsf{r}}_{\mathsf{so}}(\mathsf{LFP}) = \mathsf{TotP}$ *over finite ordered structures.*

7 Conclusions and open questions

Inspired by the two-step semantics developed in the context of weighted logics, we introduced two-step semantics that enriches the existing framework of quantitative logics, i.e. logics for expressing counting problems. We provided logical characterisations of SpanL and TotP, answering an open question of [5]. Furthermore, we determined logics that capture SpanPSPACE and FPSPACE. Compared to the other classes, the logic that captures TotP was defined in a more complicated way that is related to the properties of TotP problems: recursion of the logic expresses self-reducibility and the restricted form of the recursion captures the easy-decision property. It is worth investigating whether TotP is captured by a simpler, more elegant logic. The intermediate semantics can express sets of computation paths of TMs, different valid outputs of transducers, or solutions to computational problems. In particular, in the case of SpanL and SpanPSPACE, union and concatenation of sets are more suitable than addition and multiplication of $\mathtt{QSO}$; when the union (resp. concatenation) of two sets of strings is computed, identical outputs will contribute one string to the resulting set. In general, using the intermediate semantics, it becomes possible to keep track of paths, outputs, and solutions, apply operations on them, and then count them. Another difference between our logics and quantitative logics from [5], is that in [5], only first-order function symbols were considered and interpreted as functions $h : A^k \to \mathbb{N}$. Then, the respective second lattice $(\mathcal{F}, \leq_F)$ is not complete and the least fixed point was defined by considering the supports of functions in $\mathcal{F}$ [5, Section 6]. By defining here, functions whose values are sets of strings, the lattice $(\mathcal{F}, \leq_F)$, where $\mathcal{F}$ is one of $\mathcal{FOF}$, $\mathcal{SOF}$, or $\mathcal{RSOF}$, becomes complete, and the definition of the least fixed point is straightforward.

The two-step semantics we propose in this work is noteworthy for reasons beyond its primary objective. For instance, by specifying the concrete semantics such that any non-empty set maps to 1 and the empty set to 0, our results yield least-fixed-point logical characterisations of NL and PSPACE, the decision variants of SpanL and FPSPACE, respectively. It is known that these two classes are captured by $\mathtt{FO}$ and $\mathtt{SO}$, equipped with the transitive closure operator, respectively [21]. Our logics combine the least fixed point with quite natural syntactic definitions, without resorting to different fixed-point operators for each logic.

We believe that the logical characterisation of SpanL can yield more direct ways to approximate its problems. $\mathtt{R}_{\mathtt{fo}}\Sigma_{\mathtt{fo}}(\mathtt{FO})$ formulae bear some resemblance to regular grammars, (or, equivalently, to NFAs), since the syntax of the logic, at each recursive call, concatenates

a string of fixed length from the left with $f(\vec{x})$. An interesting question is whether one can adjust the fpras for #NFA and apply it directly to the syntax of $\mathtt{R_{fo}\Sigma_{fo}(FO)}$, giving an fpras metatheorem for the logic. Moreover, it is only natural to investigate the class that results from allowing arbitrary concatenations of recursive calls, and to expect a natural connection to context-free languages. Note that the problem of counting the strings of a specific length accepted by a context-free grammar admits a quasi-polynomial randomized approximation algorithm [19] and it is open whether it has an fpras.

Another interesting question remains the logical characterisation of a class for which computing the permanent of a matrix is complete under parsimonious reductions. This was the first problem shown in [32] to be #P-complete under Turing reductions, and it has an fpras [22]. Therefore, such a result would provide a new subclass of FPRAS and refine the complexity of the well-studied PERMANENT problem.

References

1 Antonis Achilleos and Mathias Ruggaard Pedersen. Axiomatizations and computability of weighted monadic second-order logic. In *Proc. of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470615.

2 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107(1):3–30, 1993. doi:10.1016/0304-3975(93)90252-O.

3 Antonis Antonopoulos, Eleni Bakali, Aggeliki Chalki, Aris Pagourtzis, Petros Pantavos, and Stathis Zachos. Completeness, approximability and exponential time results for counting problems with easy decision version. *Theoretical Computer Science*, 915:55–73, 2022. doi:10.1016/j.tcs.2022.02.030.

4 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. *SIGMOD Record*, 49(1):52–59, 2020. doi:10.1145/3422648.3422661.

5 Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. *Logical Methods in Computer Science*, 16(1), 2020. doi:10.23638/LMCS-16(1:9)2020.

6 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proc. of the 33rd Annual Symposium on Foundations of Computer Science, FOCS 1992*, pages 14–23. IEEE Computer Society, 1992. doi:10.1109/SFCS.1992.267823.

7 Albert Atserias, Anuj Dawar, and Joanna Ochremiak. On the power of symmetric linear programs. *Journal of the ACM*, 68(4):26:1–26:35, 2021. doi:10.1145/3456297.

8 Eleni Bakali, Aggeliki Chalki, and Aris Pagourtzis. Characterizations and approximability of hard counting classes below #P. In *Proc. of the 16th International Conference on Theory and Applications of Models of Computation, TAMC 2020*, volume 12337 of *Lecture Notes in Computer Science*, pages 251–262, 2020. doi:10.1007/978-3-030-59267-7_22.

9 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6:66–92, 1960. doi:10.1002/malq.19600060105.

10 Kevin J. Compton and Erich Grädel. Logical definability of counting functions. *Journal of Computer and System Sciences*, 53(2):283–297, 1996. doi:10.1006/jcss.1996.0069.

11 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1):69–86, 2007. doi:10.1016/j.tcs.2007.02.055.

12 Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of #P functions: A new perspective. *Journal of Computer and System Sciences*, 116:40–54, 2021. doi:10.1016/j.jcss.2020.04.002.

13 Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/s00453-003-1073-y.

14 Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for boolean #CSP. *Journal of Computer and System Sciences*, 76(3-4):267–277, 2010. doi:10.1016/j.jcss.2009.08.003.

15 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–51, 1962. doi:10.2307/2270940.

16 Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972. URL: https://books.google.is/books?id=DeLuAAAAMAAJ.

17 Ronald Fagin. Generalized first-order spectra, and polynomial. time recognizable sets. *SIAM-AMS Proceedings*, 7:43–73, 1974. URL: http://www.researchgate.net/publication/242608657_Generalized_first-order_spectra_and_polynomial._time_recognizable_sets.

18 Paul Gastin and Benjamin Monmege. A unifying survey on weighted logics and weighted automata - core weighted logic: minimal and versatile specification of quantitative properties. *Soft Computing*, 22(4):1047–1065, 2018. doi:10.1007/s00500-015-1952-6.

19 Vivek Gore, Mark Jerrum, Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 1997. doi:10.1006/inco.1997.2621.

20 Erich Grädel and Wied Pakusa. Rank logic is dead, long live rank logic! *The Journal of Symbolic Logic*, 84(1):54–87, 2019. doi:10.1017/jsl.2018.33.

21 Neil Immerman. *Descriptive complexity*. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

22 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, July 2004. doi:10.1145/1008731.1008738.

23 Richard M Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989. doi:10.1016/0196-6774(89)90038-2.

24 Johannes Köbler, Uwe Schöning, and Jacobo Torán. On counting and approximation. *Acta Informatica*, 26(4):363–379, 1989. doi:10.1007/BFb0026095.

25 Richard E. Ladner. Polynomial space counting problems. *SIAM Journal on Computing*, 18(6):1087–1097, 1989. doi:10.1137/0218073.

26 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.

27 Aris Pagourtzis and Stathis Zachos. The complexity of counting functions with easy decision version. In *Proc. of the 31st International Symposium on Mathematical Foundations of Computer Science 2006, MFCS 2006*, pages 741–752. Springer, 2006. doi:10.1007/11821069_64.

28 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. URL: https://books.google.is/books?id=JogZAQAAIAAJ.

29 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

30 Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. Descriptive complexity of #P functions. *Journal of Computer and System Sciences*, 50(3):493–505, 1995. doi:10.1006/jcss.1995.1039.

31 Boris A. Trakhtenbrot. Finite automata and the logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329, 1961.

32 Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

33 Nils Vortmeier and Thomas Zeume. Dynamic complexity of parity exists queries. *Logical Methods in Computer Science*, 17(4), 2021. doi:10.46298/lmcs-17(4:9)2021.