Descriptive Complexity of Deterministic Polylogarithmic Time[⋆]

Flavio Ferrarotti[1][0000−0003−2278−8233], Senén González[1], José María Turull Torres[2], Jan Van den Bussche[3][0000−0003−0072−3252], and Jonni Virtema[3][0000−0002−1582−3718]

[1] Software Competence Center Hagenberg, Hagenberg, Austria
{flavio.ferrarotti,senen.gonzalez}@scch.at
[2] Universidad Nacional de La Matanza, Buenos Aires, Argentina
jturull@unlam.edu.ar
[3] Hasselt University, Hasselt, Belgium
{jan.vandenbussche,jonni.virtema}@uhasselt.be

Abstract. We propose a logical characterization of problems solvable in deterministic polylogarithmic time (PolylogTime). We introduce a novel two-sorted logic that separates the elements of the input domain from the bit positions needed to address these elements. In the course of proving that our logic indeed captures PolylogTime on finite ordered structures, we introduce a variant of random-access Turing machines that can access the relations and functions of the structure directly. We investigate whether an explicit predicate for the ordering of the domain is needed in our logic. Finally, we present the open problem of finding an exact characterization of order-invariant queries in PolylogTime.

1 Introduction

The research area known as Descriptive Complexity [7, 11, 15] relates computational complexity to logic. For a complexity class of interest, one tries to come up with a natural logic such that a property of inputs can be expressed in the logic if and only if the problem of checking the property belongs to the complexity class. An exemplary result in this vein is that a family $\mathcal{F}$ of finite structures (over some fixed finite vocabulary) is definable in existential second-order logic (ESO), if and only if the membership problem for $\mathcal{F}$ belongs to NP [4]. We also say that ESO *captures* NP. The complexity class P is captured, on ordered finite structures, by a *fixpoint logic*: the extension of first-order logic with least-fixpoints [14, 22].

After these two seminal results, many more capturing results have been developed, and the benefits of this enterprise have been well articulated by several

[⋆] The research reported in this paper results from the project *Higher-Order Logics and Structures* supported by the Austrian Science Fund (FWF: **[I2420-N31]**) and the Research Foundation Flanders (FWO:**[G0G6516N]**). It was further supported by the Austrian Research Promotion Agency (FFG) through the COMET funding for the Software Competence Center Hagenberg.

authors in the references given earlier, and others [1]. We just mention here the advantage of being able to specify properties of structures (data structures, databases) in a logical, declarative manner; at the same time, we are guaranteed that our computational power is well delineated.

The focus of the present paper is on computations taking deterministic polylogarithmic time, i.e., time proportional to $\log^k n$ for some arbitrary but fixed k. Such computations are practically relevant and common on ordered structures. Well known examples are binary search in an array or search in a balanced search tree. Another natural example is the computation of $f(x_1, \ldots, x_r)$, where $x_1, \ldots, x_r$ are numbers taken from the input structure and f is a function computable in polynomial time when numbers are represented in binary.

Computations with sublinear time complexity can be formalized in terms of Turing machines with random access to the input [15]. When a family $\mathcal{F}$ of ordered finite structures over some fixed finite vocabulary is defined by some deterministic polylogarithmic-time random-access Turing machine, we say that $\mathcal{F}$ belongs to the complexity class PolylogTime. In this paper, we show how this complexity class can be captured by a new logic which we call *index logic*.

Index logic is two-sorted; variables of the first sort range over the domain of the input structure. Variables of the second sort range over an initial segment of the natural numbers; this segment is bounded by the logarithm of the size of the input structure. Thus, the elements of the second sort represent the bit positions needed to address elements of the first sort. Index logic includes full fixpoint logic on the second sort. Quantification over the first sort, however, is heavily restricted. Specifically, a variable of the first sort can only be bound using an address specified by a subformula that defines the positions of the bits of the address that are set. This "indexing mechanism" lends index logic its name.

In the course of proving our capturing result, we consider a new variant of random-access Turing machines. In the standard variant, the entire input structure is presented as one binary string. In our new variant, the different relations and functions of the structure can be accessed directly. We will show that both variants are equivalent, in the sense that they lead to the same notion of PolylogTime. We note that, in descriptive complexity, it is common practice to work only with relational structures, as functions can be identified with their graphs. In a sublinear-time setting, however, this does not work. Indeed, let f be a function and denote its graph by $\tilde{f}$. If we want to know the value of $f(x)$, we cannot spend the linear time needed to find a y such that $\tilde{f}(x, y)$ holds. Thus, in this work, we allow structures containing functions as well as relations.

At first glance, one might think that a simpler approach to ours, for the characterization of PolylogTime, could be to adapt the construction used by Immerman and Vardi [14, 22] to capture P. For instance, by querying binary representations of the indices with Immerman's BIT predicate, where $\mathrm{BIT}(x, i)$ holds iff the i-th bit of x in binary is 1, or avoiding our new variant of random-access Turing machine. In fact, that was our initial approach to the problem. This results, however, in a long and cumbersome characterization proof, mostly due to the need to express arithmetic operations within the logic to access the relevant parts of the input, since in PolylogTime we cannot read it in whole. A challenge, in this sense, was to develop a logic which enables the expression of PolylogTime problems in a relatively clean and natural way. For this, the indexing mechanism in our logic is a key contribution. The alternative of using fixed point operations and BIT to address values of the first sort leads to a logic which is rather awkward to define and to use.

We also devote attention to gaining a detailed understanding of the expressivity of index logic. Specifically, we observe that order comparisons between quantified variables of the first sort can be expressed in terms of their addresses. For constants of the first sort that are directly given by the structure, however, we show that this is not possible. In other words, index logic without an explicit order predicate on the first sort would no longer capture PolylogTime for structures with constants.

Related work. Many natural fixed point computations, such as transitive closure, converge after a polylogarithmic number of steps. This motivated the study in [10] of a fragment of fixed point logic with counting (FPC) that only allows polylogarithmically many iterations of the fixed point operators (POLYLOG-FPC). They noted that on ordered structures POLYLOG-FPC captures NC, i.e., the class of problems solvable in parallel polylogarithmic time. This holds even in the absence of counting, which on ordered structures can be simulated using fixed point operators. Moreover, an old result in [13] directly implies that POLYLOG-FPC is strictly weaker than FPC with regards to expressive power.

It is well known that the (nondeterministic) logarithmic time hierarchy corresponds exactly to the set of first-order definable Boolean queries (see [15], Theorem 5.30). The relationship between uniform families of circuits within NC^1 and nondeterministic random-access logarithmic time machines was studied in [19]. However, the study of descriptive complexity of classes of problems decidable by *deterministic* formal models of computation in polylogarithmic time, i.e., the topic of this paper, has been overlooked by previous works.

On the other hand, *nondeterministic* polylogarithmic time complexity classes, defined in terms of alternating random-access Turing machines and related families of circuits, have received some attention [18, 5]. Recently, a theorem analogous to Fagin's famous theorem [4], was proven for nondeterministic polylogarithmic time [5]. For this task, a restricted second-order logic for finite structures, where second-order quantification ranges over relations of size at most polylogarithmic in the size of the structure, and where first-order universal quantification is bounded to those relations, was exploited. This latter work, is closely related to the work on constant depth quasipolynomial size AND/OR circuits and the corresponding restricted second-order logic in [18]. Both logics capture the full alternating polylogarithmic time hierarchy, but the additional restriction in the first-order univesal quantification in the second-order logic defined in [5], enables a one-to-one correspondence between the levels of the polylogarithmic time hierarchy and the prenex fragments of the logic, in the style of a result of Stockmeyer [21] regarding the polynomial-time hierarchy. Unlike the classical results of Fagin and

Stockmeyer [4, 21], the results on the descriptive complexity of nondeterministic polylogarithmic time classes only hold over ordered structures.

2 Preliminaries

We allow structures containing functions as well as relations and constants. Unless otherwise stated, we work with finite ordered structures of finite vocabularies. A finite structure $\mathbf{A}$ of vocabulary $\sigma = \{R_1^{r_1}, \ldots, R_p^{r_p}, c_1, \ldots c_q, f_1^{k_1}, \ldots, f_s^{k_s}\}$, where each $R_i^{r_i}$ is an r_i-ary relation symbol, each c_i is a constant symbol, and each $f_i^{k_i}$ is a k_i-ary function symbol, consists of a finite domain A and interpretations for all relation, constant and function symbols in σ. An interpretation of a symbol $R_i^{r_i}$ is a relation $R_i^{\mathbf{A}} \subseteq A^{r_i}$, of a symbol c_i is a value $c_i^{\mathbf{A}} \in A$, and of a symbol $f_i^{k_i}$ is a function $f_i^{\mathbf{A}} : A^{k_i} \to A$. Every finite ordered structure has a corresponding isomorphic structure whose domain is an initial segment of the natural numbers. Thus, we assume as usual that $A = \{0, 1, \ldots, n-1\}$, where n is the cardinality $|A|$ of A.

In this paper, $\log n$ always refers to the binary logarithm of n, i.e., $\log_2 n$. We write $\log^k n$ as a shorthand for $(\lceil \log n \rceil)^k$.

3 Deterministic polylogarithmic time

The sequential access that Turing machines have to their tapes makes it impossible to do nontrivial computations in sub-linear time. Therefore, logarithmic time complexity classes are usually studied using models of computation that have random access[4] to their input, i.e., that can access every input address directly. As this also applies to the polylogarithmic complexity classes studied in this paper, we adopt a Turing machine model that has a *random access* read-only input, similar to the logarithmic-time Turing machine in [19].

Our concept of a *random-access Turing machine* is that of a multi-tape Turing machine which consists of: (1) a finite set of states, (2) a read-only random access *input-tape*, (3) a sequential access *address-tape*, and (4) one or more (but a fixed number of) sequential access *work-tapes*. All tapes are divided into cells, each equipped with a *tape head* which scans the cells, and are "semi-infinite" in the sense that they have no rightmost cell, but have a left-most cell. The tape heads of the sequential access address-tape and work-tapes can move left or right. When a head is in the leftmost cell, it is not allowed to move left. The address-tape alphabet only contains symbols 0, 1 and $\sqcup$ (for blank). The position of the input-tape head is determined by the number i stored in binary in between the left-most cell and the first blank cell of the address-tape (if the left-most cell is blank, then i is considered to be 0) as follows: If i is strictly smaller than the

[4] The term *random access* refers to the manner how *random-access memory* (RAM) is read and written. In contrast to sequential memory, the time it takes to read or write using RAM is almost independent of the physical location of the data in the memory. We want to emphasise that there is nothing *random* in random access.

length n of the input string, then the input-tape head is in the $(i+1)$-th cell. Otherwise, if $i \geq n$, then the input-tape head is in the $(n+1)$-th cell scanning the special end-marker symbol $\triangleleft$.

Formally, a *random-access Turing machine* M with k work-tapes is a five-tuple $(Q, \Sigma, \delta, q_0, F)$. Here Q is a finite set of *states*; $q_0 \in Q$ is the *initial state*. Σ is a finite set of symbols (the *alphabet* of M). For simplicity, we fix $\Sigma = \{0, 1, \sqcup\}$. $F \subseteq Q$ is the set of *accepting final states*. The *transition* function of M is of the form $\delta : Q \times (\Sigma \cup \{\triangleleft\}) \times \Sigma^{k+1} \to Q \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^{k+1}$. We assume that the tape head directions $\leftarrow$ for "left", $\rightarrow$ for "right" and $-$ for "stay", are not in $Q \cup \Sigma$.

A *configuration* of M on a fixed input w_0 is a $k+2$ tuple $(q, i, w_1, \ldots, w_k)$, where q is the current state of M, $i \in \Sigma^* \# \Sigma^*$ represents the current contents of the index-tape cells, and each $w_j \in \Sigma^* \# \Sigma^*$ represents the current contents of the j-th work-tape cells. We do not include the contents of the input-tape cells in the configuration since they cannot be changed. Further, the position of the input-tape head is uniquely determined by the contents of the index-tape cells. The symbol $\#$ (which we assume is not in Σ) marks the position of the tape head. By convention, the head scans the symbol immediately at the right of $\#$. All symbols in the infinite tapes not appearing in their corresponding strings $i, w_0, \ldots, w_k$ are assumed to be the special symbol blank $\sqcup$.

At the beginning of a computation, all work-tapes are blank except the input-tape, that contains the input string, and the index-tape that contains a 0 (meaning that the input-tape head scans the first cell of the input-tape). Thus, the *initial configuration* of M is $(q_0, \#0, \#, \ldots, \#)$. A *computation* is a sequence of configurations which starts with the initial configuration and ends in a configuration in which no more steps can be performed, and such that each step from a configuration to the next obeys the transition function. An input string is *accepted* if an accepting configuration, i.e., a configuration in which the current state belongs to F, is reached.

Example 1. Following a simple strategy, a random-access Turing machine M can figure out the length n of its input as well as $\lceil \log n \rceil$ in polylogarithmic time. In its initial step, M checks whether the input-tape head scans the end-marker $\triangleleft$. If it does, then the input string is the empty string and its work is done. Otherwise, M writes 1 in the first cell of its address tape and keeps writing 0's in its subsequent cells right up until the input-tape head scans $\triangleleft$. At this point the resulting binary string in the index-tape is of length $\lceil \log n \rceil$. Next, M moves its address-tape head back to the first cell (i.e., to the only cell containing a 1 at this point). From here on, M repeatedly moves the index head one step to the right. Each time it checks whether the index-tape head scans a blank $\sqcup$ or a 0. If $\sqcup$ then M is done. If 0, it writes a 1 and tests whether the input-tape head jumps to the cell with $\triangleleft$; if so, it rewrites a 0, otherwise, it leaves the 1. The binary number left on the index-tape at the end of this process is $n-1$. Adding one in binary is now an easy task. $\square$

The *formal language accepted* by a machine M, denoted $L(M)$, is the set of strings accepted by M. We say that $L(M) \in \mathrm{DTIME}[f(n)]$ if M makes at most $O(f(n))$ steps before accepting or rejecting an input string of length n. We define the class of all formal languages decidable by (deterministic) random-access Turing machines in *polylogarithmic time* as follows:

$$\text{PolylogTime} = \bigcup_{k \in \mathbb{N}} \text{DTIME}[\log^k n]$$

It follows from Example 1 that a PolylogTime random-access Turing machine can check any numerical property that is polynomial time in the size of its input in binary. For instance, it can check whether the length of its input is even, by simply looking at the least-significant bit.

When we want to give a finite structure as an input to a random-access Turing machine, we encode it as a string, adhering to the usual conventions in descriptive complexity theory [15]. Let $\sigma{=}\{R_1^{r_1}, \ldots, R_p^{r_p}, c_1, \ldots, c_q, f_1^{k_1}, \ldots, f_s^{k_s}\}$ be a vocabulary, and let $\mathbf{A}$ with $A{=}\{0, 1, \ldots, n{-}1\}$ be an ordered structure of vocabulary σ. Each relation $R_i^{\mathbf{A}} \subseteq A^{r_i}$ of $\mathbf{A}$ is encoded as a binary string $\text{bin}(R_i^{\mathbf{A}})$ of length n^{r_i}, where 1 in a given position indicates that the corresponding tuple is in $R_i^{\mathbf{A}}$. Likewise, each constant number $c_j^{\mathbf{A}}$ is encoded as a binary string $\text{bin}(c_j^{\mathbf{A}})$ of length $\lceil \log n \rceil$.

We can also encode the functions in a structure. We view k-ary functions as consisting of $\lceil \log n \rceil$ k-ary relations, where the i-th relation indicates whether the i-th bit is 1. Thus, each function $f_i^{\mathbf{A}}$ is encoded as a binary string $\text{bin}(f_i^{\mathbf{A}})$ of length $\lceil \log n \rceil n^{k_i}$.

The encoding of the whole structure $\text{bin}(\mathbf{A})$ is the concatenation of the binary strings encoding its relations, constants and functions. The length $\hat{n} = |\text{bin}(\mathbf{A})|$ of this string is $n^{r_1} + \cdots + n^{r_p} + q\lceil \log n \rceil + \lceil \log n \rceil n^{k_1} + \cdots + \lceil \log n \rceil n^{k_s}$, where $n = |A|$ denotes the size of the input structure $\mathbf{A}$. Note that $\log \hat{n} \in O(\lceil \log n \rceil)$, so $\text{DTIME}[\log^k \hat{n}] = \text{DTIME}[\log^k n]$.

4 Direct-access Turing machines

In this section, we propose a new model of random-access Turing machines. In the standard model reviewed above, the entire input structure is assumed to be encoded as one binary string. In our new variant, the different relations and functions of the structure can be accessed directly. We then show that both variants are equivalent, in the sense that they lead to the same notion of PolylogTime. The direct-access model will then be useful to give a transparent proof of our capturing result.

Let our vocabulary $\sigma = \{R_1^{r_1}, \ldots, R_p^{r_p}, c_1, \ldots c_q, f_1^{k_1}, \ldots, f_s^{k_s}\}$. A *direct-access Turing machine that takes σ-structures $\mathbf{A}$ as input*, is a multitape Turing machine with $r_1 + \cdots + r_p + k_1 + \cdots + k_s$ distinguished work-tapes, called *address-tapes*, s distinguished read-only (function) *value-tapes*, $q + 1$ distinguished read-only *constant-tapes*, and one or more ordinary work-tapes.

Let us define a transition function δ_l for each tape l separately. These transition functions take as an input the current state of the machine, the bit read by each of the heads of the machine, and, for each relation $R_i \in \sigma$, the answer (0 or 1) to the query $(n_1, \ldots, n_{r_i}) \in R_i^{\mathbf{A}}$. Here, n_j denotes the number written in binary in the jth distinguished tape of R_i.

Thus, with m the total number of tapes, the state transition function has the form

$$\delta_Q : Q \times \Sigma^m \times \{0,1\}^p \to Q.$$

If l corresponds to an address-tape or an ordinary work-tape, we get the form

$$\delta_l : Q \times \Sigma^m \times \{0,1\}^p \to \Sigma \times \{\leftarrow, \rightarrow, -\}.$$

If l corresponds to one of the read-only tapes, we have

$$\delta_l : Q \times \Sigma^m \times \{0,1\}^p \to \{\leftarrow, \rightarrow, -\}.$$

Finally we update the contents of the function value-tapes. If l is the function value-tape for a function f_i, then the content of the tape l is updated to $f_i(n_1, \ldots n_{k_i})$ written in binary. Here, n_j denotes the number written in binary in the jth distinguished address-tape of f_i *after* the execution of the above transition functions. If one of the n_j is too large, the tape l is updated to contain only blanks. Note that the head of the tape remains in place; it was moved by δ_l already.

In the initial configuration, read-only constant-tapes for the constant symbols $c_1, \ldots, c_q$ hold the values in binary of their values in $\mathbf{A}$. One additional constant-tape (there are $q+1$ of them) holds the size n of the domain of $\mathbf{A}$ in binary. Each address-tape, each value-tape, and each ordinary work-tape holds only blanks.

Theorem 2. *A class of finite ordered structures $\mathcal{C}$ of some fixed vocabulary σ is decidable by a random-access Turing machine working in* PolylogTime *with respect to $\hat{n}$, where $\hat{n}$ is the size of the binary encoding of the input structure, iff $\mathcal{C}$ is decidable by a direct-access Turing machine in* PolylogTime *with respect to n, where n is the size of the domain of the input structure.*

The proof (omitted) is based on computing precise locations in which bits can be found, and, for the other direction, on a binary search technique to compute n from $\hat{n}$.

5 Index logic

In this section we introduce *index logic*, a new logic which over ordered finite structures captures PolylogTime. Our definition of index logic is inspired by the second-order logic in [18], where relation variables are restricted to valuations on the sub-domain $\{0, \ldots, \lceil \log n \rceil - 1\}$ (n being the size of the interpreting structure), as well as by the well known counting logics as defined in [9].

Given a vocabulary σ, for every ordered σ-structure $\mathbf{A}$, we define a corresponding set of natural numbers $\mathit{Num}(\mathbf{A}) = \{0, \ldots, \lceil \log n \rceil - 1\}$ where $n = |A|$. Note that $\mathit{Num}(\mathbf{A}) \subseteq A$, since we assume that A is an initial segment of the natural numbers. This simplifies the definitions, but it is otherwise unnecessary.

Index logic is a two-sorted logic. Individual variables of the first sort $\mathbf{v}$ range over the domain A of $\mathbf{A}$, while individual variables of the second sort $\mathbf{n}$ range over $\mathit{Num}(\mathbf{A})$. We denote variables of sort $\mathbf{v}$ with $x, y, z, \ldots$, possibly with a subindex such as $x_0, x_1, x_2, \ldots$, and variables of type $\mathbf{n}$ with $\mathtt{x}, \mathtt{y}, \mathtt{z}$, also possibly with a subindex. Relation variables, denoted with uppercase letters $X, Y, Z, \ldots$, are always of sort $\mathbf{n}$, and thus range over relations defined on $\mathit{Num}(\mathbf{A})$.

Definition 3. *Let σ be a vocabulary, we inductively define terms and formulae of index logic as follows:*

- *Each individual variable of sort $\mathbf{v}$ and each constant symbol in σ is a term of sort $\mathbf{v}$.*
- *Each individual variable of sort $\mathbf{n}$ is a term of sort $\mathbf{n}$.*
- *If $t_1, \ldots, t_k$ are terms of sort $\mathbf{v}$ and f is a k-ary function symbol in σ, then $f(t_1, \ldots, t_k)$ is a term of sort $\mathbf{v}$.*
- *If t_1, t_2 are terms of a same sort, then $t_1 = t_2$ and $t_1 \leq t_2$ are (atomic) formulae.*
- *If $t_1, \ldots, t_k$ are terms of sort $\mathbf{v}$ and R is a k-ary relation symbol in σ, then $R(t_1, \ldots, t_k)$ is an (atomic) formula.*
- *If $t_1, \ldots, t_k$ are terms of sort $\mathbf{n}$ and X is a k-ary relation variable, then $X(t_1, \ldots, t_k)$ is an (atomic) formula.*
- *If t is a term of sort $\mathbf{v}$, φ is a formula and $\mathtt{x}$ is an individual variable of sort $\mathbf{n}$, then $t = \mathit{index}\{\mathtt{x} : \varphi(\mathtt{x})\}$ is an (atomic) formula.*
- *If $\bar{t}$ is tuple of terms of sort $\mathbf{n}$, $\bar{\mathtt{x}}$ is tuples of variables also of sort $\mathbf{n}$, X is a relation variable, the lengths of $\bar{t}$ and $\bar{\mathtt{x}}$ are the same and coincide with the arity of X, and φ is a formula, then $[\mathrm{IFP}_{\bar{\mathtt{x}}, X}\varphi]\bar{t}$ is an (atomic) formula.*
- *If φ, ψ are formulae, then $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\neg\psi$ are formulae.*
- *If $\mathtt{x}$ is a variable of type $\mathbf{n}$ and φ is a formula, then $\exists \mathtt{x}(\varphi)$ and $\forall \mathtt{x}(\varphi)$ are formulae.*
- *If $x = \mathit{index}\{\mathtt{x} : \alpha(\mathtt{x})\}$ is an atomic formula such that x does* not *appear free in $\alpha(\mathtt{x})$, and φ is a formula, then $\exists x(x = \mathit{index}\{\mathtt{x} : \alpha(\mathtt{x})\} \wedge \varphi)$ is a formula.*

The concept of a valuation is the standard for a two-sorted logic. Thus, a *valuation* over a structure $\mathbf{A}$ is any total function val from the set of all variables of index logic to values satisfying the following constraints:

- If x is a variable of type $\mathbf{v}$, then $\mathit{val}(x) \in A$.
- If $\mathtt{x}$ is a variable of type $\mathbf{n}$, then $\mathit{val}(\mathtt{x}) \in \mathit{Num}(\mathbf{A})$.
- If X is a relation variable with arity r, then $\mathit{val}(X) \subseteq (\mathit{Num}(\mathbf{A}))^r$.

Valuations extend to terms and tuples of terms in the usual way. Further, we say that a valuation val is v-equivalent to a valuation val' if $\mathit{val}(v') = \mathit{val}'(v')$ for all variables v' other than v.

Fixed points are defined in the standard way (see [2] and [17] among others). Given an operator $F : \mathcal{P}(B) \to \mathcal{P}(B)$, a set $S \subseteq B$ is a *fixed point* of F if $F(S) = S$. A set $S \subseteq B$ is a *least fixed point* of F if it is a fixed point and for every other fixed point S' of F we have $S \subseteq S'$. We denote the least fixed point of F as $\mathrm{lfp}(F)$. The *inflationary fixed point* of F, denoted by $\mathrm{ifp}(F)$, is the union of all sets S^i where $S^0 = \emptyset$ and $S^{i+1} = S^i \cup F(S^i)$.

Let $\varphi(X, \bar{\mathtt{x}})$ be a formula of vocabulary σ, where X is a relation variable of arity k and $\mathtt{x}$ is a k-tuple of variables of type $\mathbf{n}$. Let $\mathbf{A}$ be a σ-structure. The formula $\varphi(X, \bar{\mathtt{x}})$ gives rise to an operator $F^{\mathbf{A}}_{\varphi,\bar{\mathtt{x}},X} : \mathcal{P}((\mathit{Num}(\mathbf{A}))^k) \to \mathcal{P}((\mathit{Num}(\mathbf{A}))^k)$ defined as follows:

$F^{\mathbf{A}}_{\varphi,\bar{\mathtt{x}},X}(S) := \{\bar{a} \in (\mathit{Num}(\mathbf{A}))^k \mid \mathbf{A}, \mathit{val} \models \varphi(X, \bar{\mathtt{x}})$ for some valuation val with $\mathit{val}(X) = S$ and $\mathit{val}(\bar{\mathtt{x}}) = \bar{a}\}$.

Definition 4. *The formulae of* IFP^{plog} *are interpreted as follows:*

- $\mathbf{A}, \mathit{val} \models t_1 = t_2$ *iff* $\mathit{val}(t_1) = \mathit{val}(t_2)$.
- $\mathbf{A}, \mathit{val} \models t_1 \leq t_2$ *iff* $\mathit{val}(t_1) \leq \mathit{val}(t_2)$.
- $\mathbf{A}, \mathit{val} \models R(t_1, \ldots, t_k)$ *iff* $(\mathit{val}(t_1), \ldots, \mathit{val}(t_k)) \in R^{\mathbf{A}}$.
- $\mathbf{A}, \mathit{val} \models X(t_1, \ldots, t_k)$ *iff* $(\mathit{val}(t_1), \ldots, \mathit{val}(t_k)) \in \mathit{val}(X)$.
- $\mathbf{A}, \mathit{val} \models t = \mathit{index}\{\mathtt{x} : \varphi(\mathtt{x})\}$ *iff* $\mathit{val}(t)$ *in binary is* $b_m b_{m-1} \cdots b_0$, *where* $m = \lceil \log |A| \rceil - 1$ *and* $b_j = 1$ *iff* $\mathbf{A}, \mathit{val}' \models \varphi(\mathtt{x})$ *for* val' $\mathtt{x}$*-equivalent to* val *and* $\mathit{val}'(\mathtt{x}) = j$.
- $\mathbf{A}, \mathit{val} \models [\mathrm{IFP}_{\bar{\mathtt{x}},X}\varphi]\bar{t}$ *iff* $\mathit{val}(\bar{t}) \in \mathrm{ifp}(F^{\mathbf{A}}_{\varphi,\bar{\mathtt{x}},X})$.
- $\mathbf{A}, \mathit{val} \models \neg\varphi$ *iff* $\mathbf{A}, \mathit{val} \not\models \varphi$.
- $\mathbf{A}, \mathit{val} \models \varphi \wedge \psi$ *iff* $\mathbf{A}, \mathit{val} \models \varphi$ *and* $\mathbf{A}, \mathit{val} \models \psi$.
- $\mathbf{A}, \mathit{val} \models \varphi \vee \psi$ *iff* $\mathbf{A}, \mathit{val} \models \varphi$ *or* $\mathbf{A}, \mathit{val} \models \psi$.
- $\mathbf{A}, \mathit{val} \models \exists \mathtt{x}(\varphi)$ *iff there is a* val' $\mathtt{x}$*-equivalent to* val *such that* $\mathbf{A}, \mathit{val}' \models \varphi$.
- $\mathbf{A}, \mathit{val} \models \forall \mathtt{x}(\varphi)$ *iff for all* val' $\mathtt{x}$*-equivalent to* val, *it holds that* $\mathbf{A}, \mathit{val}' \models \varphi$.
- $\mathbf{A}, \mathit{val} \models \exists x(x = \mathit{index}\{\mathtt{x} : \alpha(\mathtt{x})\} \wedge \varphi)$ *iff there is a* val' x*-equivalent to* val *such that* $\mathbf{A}, \mathit{val}' \models x = \mathit{index}\{\mathtt{x} : \alpha(\mathtt{x})\}$ *and* $\mathbf{A}, \mathit{val}' \models \varphi$.

It immediately follows from the famous result by Gurevich and Shelah regarding the equivalence between inflationary and least fixed points [12], that an equivalent index logic can be obtained if we (1) replace $[\mathrm{IFP}_{\bar{\mathtt{x}},X}\varphi]\bar{t}$ by $[\mathrm{LFP}_{\bar{\mathtt{x}},X}\varphi]\bar{t}$ in the formation rule for the fixed point in Definition 3, adding the restriction that every occurrence of X in φ is positive[5], and (2) fix the interpretation $\mathbf{A}, \mathit{val} \models [\mathrm{LFP}_{\bar{\mathtt{x}},X}\varphi]\bar{t}$ iff $\mathit{val}(\bar{t}) \in \mathrm{lfp}(F^{\mathbf{A}}_{\varphi,\bar{\mathtt{x}},X})$.

Moreover, the convenient tool of *simultaneous fixed points*, which allows one to iterate several formulae at once, can still be used here since it does not increase the expressive power of the logic. Following the syntax and semantics proposed by Ebbinghaus and Flum [2], a version of index logic with simultaneous inflationary fixed point can be obtained by replacing the clause corresponding to IFP in Definition 3 by the following:

- If $\bar{t}$ is tuple of terms of sort $\mathbf{n}$, and for $m \geq 0$ and $0 \leq i \leq m$, we have that $\bar{\mathtt{x}}_i$ is a tuple of variables of sort $\mathbf{n}$, X_i is a relation variable whose arity coincides with the length of $\bar{\mathtt{x}}_i$, the lengths of $\bar{t}$ and $\bar{\mathtt{x}}_0$ are the same, and φ_i is a formula, then $[\text{S-IFP}_{\bar{\mathtt{x}}_0,X_0,\ldots,\bar{\mathtt{x}}_m,X_m}\varphi_0, \ldots, \varphi_m]\bar{t}$ is an atomic formula.

[5] This ensures that $F^{\mathbf{A}}_{\varphi,\bar{\mathtt{x}},X}$ is monotonous and thus that the least fixed point $\mathrm{lfp}(F^{\mathbf{A}}_{\varphi,\bar{\mathtt{x}},X})$ is guaranteed to exists

The interpretation is that $\mathbf{A}, val \models [\text{S-IFP}_{\bar{\mathtt{x}}_0, X_0, \ldots, \bar{\mathtt{x}}_m, X_m} \varphi_0, \ldots, \varphi_m]\bar{t}$ iff $val(\bar{t})$ belongs to the first (here X_0) component of the simultaneous inflationary fixed point.

Thus, we can use index logic with the operators IFP, LFP, S-IFP or S-LFP interchangeably.

The following result confirms that our logic serves our purpose.

Theorem 5. *Over ordered structures, index logic captures* PolylogTime*.*

The proof of the theorem can be found in the full arXiv version of this article [6]; instead we give two worked-out examples illustrating the power of index logic.

5.1 Finding the binary representation of a constant

Assume a constant symbol c of sort $\mathbf{v}$. In this example, we show a formula $\beta_c(\mathtt{x})$ such that the sentence $c = \mathit{index}\{\mathtt{x} : \beta_c\}$ is valid over the class of all finite ordered structures. In other words, β_c defines the binary representation of the number c.

Informally, β_c works by iterating through the bit positions $\mathtt{y}$ from the most significant to the least significant. These bits are accumulated in a relation variable Z. For each $\mathtt{y}$ we set the corresponding bit, on the condition that the resulting number does not exceed c. The set bits are collected in a relation variable Y.

In the formal description of β_c below, we use the following abbreviations. We use M to denote the most significant bit position. Thus, formally, $\mathtt{z} = M$ abbreviates $\forall \mathtt{z}'\, \mathtt{z}' \leq \mathtt{z}$. Furthermore, for a unary relation variable Z, we use $\mathtt{z} = \min Z$ with the obvious meaning. We also use abbreviations such as $\mathtt{z} = \mathtt{z}' - 1$ with the obvious meaning.

Now β_c is a simultaneous fixpoint $[\text{S-IFP}_{\mathtt{y}, Y, \mathtt{z}, Z}\, \varphi_Y, \varphi_Z](\mathtt{x})$ where

$$\varphi_Z := (Z = \emptyset \wedge \mathtt{z} = M) \vee (Z \neq \emptyset \wedge \mathtt{z} = \min Z - 1)$$
$$\varphi_Y := Z \neq \emptyset \wedge \mathtt{y} = \min Z \wedge \exists x (x = \mathit{index}\{\mathtt{z} : Y(\mathtt{z}) \vee \mathtt{z} = \mathtt{y}\} \wedge c \geq x).$$

5.2 Binary search in an array of key values

In order to develop insight in how index logic works, we develop in detail an example showing how binary search in an array of key values can be expressed in the logic.

We represent the data structure as an ordered structure $\mathbf{A}$ over the vocabulary consisting of a unary function K, a constant symbol N, a constant symbol T, and a binary relation $\prec$. The domain of $\mathbf{A}$ is an initial segment of the natural numbers. The constant $l := N^{\mathbf{A}}$ indicates the length of the array; the domain elements $0, 1, \ldots, l-1$ represent the cells of the array. The remaining domain elements represent key values. Each array cell holds a key value; the assignment of key values to array cells is given by the function $K^{\mathbf{A}}$.

The simplicity of the above abstraction gives rise to two peculiarities, which, however, pose no problems. First, the array cells belong to the range of the function K. Thus, array cells are allowed to play a double role as key values. Second, the function K is total, so it is also defined on the domain elements that are not array cells. We will simply ignore K on that part of the domain.

We still need to discuss $\prec$ and T. We assume $\prec^{\mathbf{A}}$ to be a total order, used to compare key values. So $\prec^{\mathbf{A}}$ can be different from the built-in order $<^{\mathbf{A}}$. For the binary search procedure to work, the array needs to be sorted, i.e., $\mathbf{A}$ must satisfy $\forall x \forall y (x < y \rightarrow (K(x) \preceq K(y)))$. Finally, the constant $t := T^{\mathbf{A}}$ is the test value. Specifically, we are going to exhibit an index logic formula that expresses that t is a key value stored in the array. In other words, we want to express the condition

$$\exists x (x < N \wedge K(x) = T). \qquad (\gamma)$$

Note that, we express here condition (γ) by a first-order formula that is not an index formula. So, our aim is to show that γ is still expressible, over all sorted arrays, by an index formula.

We recall the procedure for binary search [16] in the following form, using integer variables L, R and I:

```
$L := 0$
$R := N - 1$
**while** $L \neq R$ **do**
    $I := \lfloor (L + R)/2 \rfloor$
    **if** $K(I) \succ T$ **then** $R := I - 1$ **else** $L := I$
**od**
**if** $K(L) = T$ **return** ‘found’ **else return** ‘not found’
```

We are going to express the above procedure as a simultaneous fixpoint, using binary relation variables L and R and a unary relation variable Z. We collect the iteration numbers in Z, thus counting until the logarithm of the size of the structure. Relation variables L and R are used to store the values, in binary representation, of the integer variables L and R during all iterations. Specifically, for each $i \in \mathit{Num}(\mathbf{A})$, the value of the term $\mathit{index}\{\mathtt{x} : L(i, \mathtt{x})\}$ will be the value of the integer variable L before the i-th iteration of the while loop (and similarly for R).

In the formal expression of γ below, we use the formula β_c from Section 5.1, with $N - 1$ playing the role of c. We also assume the following formulas:

- A formula avg that expresses, for unary relation variables X and Y and a numeric variable $\mathtt{x}$, that the bit $\mathtt{x}$ is set in the binary representation of $\lfloor x + y \rfloor / 2$, where x and y are the numbers represented in binary by X and Y.
- A formula $\mathit{minusone}(X, \mathtt{y})$, expressing that the bit $\mathtt{y}$ is set in the binary representation of $x - 1$, where x is the number represented in binary by X.

These formulas surely exist because index logic includes full fixpoint logic on the numeric sort; fixpoint logic captures PTIME on the numeric sort; and computing the average, or subtracting one, are PTIME operations on binary numbers.

We are going to apply the formula avg where X and Y are given by $L(\mathtt{z}, .)$ and $R(\mathtt{z}, .)$. So, formally, below, we use $\mathit{avg}'(\mathtt{z}, \mathtt{x})$ for the formula obtained from formula *avg* by replacing each subformula of the form $X(\mathtt{u})$ by $L(\mathtt{z}, \mathtt{u})$, and $Y(\mathtt{u})$ by $R(\mathtt{z}, \mathtt{u})$.

Furthermore, we are going to apply formula *minusone* where X is given by avg'. So, formally, $minusone'$ will denote the formula obtained from *minusone* by replacing each subformula of the form $X(\mathtt{u})$ by $avg'(\mathtt{z}, \mathtt{u})$.

A last abbreviation we will use is *test*, which will denote the formula $\exists e(e = index\{\mathtt{x} : avg'\} \wedge K(e) \succ T)$.

Now γ is expressed by $\exists x(x = index\{\mathtt{l} : \psi(\mathtt{l})\} \wedge K(x) = T)$, where

$$\begin{aligned}
\psi(\mathtt{l}) &:= \exists \mathtt{s} \forall \mathtt{s}'(\mathtt{s}' \leq \mathtt{s} \wedge [\text{S-IFP}_{\mathtt{z},\mathtt{x},L,\mathtt{z},\mathtt{x},R,\mathtt{z},Z}\, \varphi_L, \varphi_R, \varphi_Z](\mathtt{s}, \mathtt{l})) \\
\varphi_Z &:= (Z = \emptyset \wedge \mathtt{z} = 0) \vee (Z \neq \emptyset \wedge \mathtt{z} = \max Z + 1) \\
\varphi_L &:= Z \neq \emptyset \wedge \mathtt{z} = \max Z + 1 \wedge \\
&\qquad \exists \mathtt{z}'(\mathtt{z}' = \max Z \wedge (test \rightarrow L(\mathtt{z}', \mathtt{x})) \wedge (\neg test \rightarrow avg'(\mathtt{z}', \mathtt{x}))) \\
\varphi_R &:= (Z = \emptyset \wedge \mathtt{z} = 0 \wedge \beta_{N-1}(\mathtt{x})) \vee (Z \neq \emptyset \wedge \mathtt{z} = \max Z + 1 \wedge \\
&\qquad \exists \mathtt{z}'(\mathtt{z}' = \max Z \wedge (test \rightarrow minusone'(\mathtt{z}', \mathtt{x})) \wedge (\neg test \rightarrow R(\mathtt{z}', \mathtt{x}))))
\end{aligned}$$

6 Definability in Deterministic PolylogTime

We observe here that very simple properties of structures are nondefinable in index logic. Moreover, we provide an answer to a fundamental question on the primitivity of the built-in order predicate (on terms of sort $\mathbf{v}$) in our logic. Indeed, we are working with ordered structures, and variables of sort $\mathbf{v}$ can only be introduced by binding them to an index term. Index terms are based on sets of bit positions which can be compared as binary numbers. Hence, it is plausible to suggest that the built-in order predicate can be removed from our logic without losing expressive power. We prove, however, that this does not work in the presence of constant or function symbols in the vocabulary.

Proposition 6. *Assume that the vocabulary includes a unary relation symbol P. Checking emptiness (or non-emptiness) of $P^{\mathbf{A}}$ in a given structure $\mathbf{A}$ is not computable in* PolylogTime.

Proof. We will show that emptiness is not computable in PolylogTime. For a contradiction, assume that it is. Consider first-order structures over the vocabulary $\{P\}$, where P is a unary relation symbol. Let M be some Turing machine that decides in PolylogTime, given a $\{P\}$-structure $\mathbf{A}$, whether $P^{\mathbf{A}}$ is empty. Let f be a polylogarithmic function that bounds the running time of M. Let n be a natural number such that $f(n) < n$.

Let $\mathbf{A}_\emptyset$ be the $\{P\}$-structure with domain $\{0, \ldots, n-1\}$, where $P^{\mathbf{A}} = \emptyset$. The encoding of $\mathbf{A}_\emptyset$ to the Turing machine M is the sequence $s := \underbrace{0 \ldots 0}_{n \text{ times}}$. Note that the running time of M with input s is strictly less than n. This means that there must exist an index i of s that was not read in the computation $M(s)$. Define

$$s' := \underbrace{0 \ldots 0}_{i \text{ times}} 1 \underbrace{0 \ldots 0}_{n-i-1 \text{ times}} .$$

Clearly the output of the computations $M(s)$ and $M(s')$ are identical, which is a contradiction since s' is an encoding of a $\{P\}$-structure where the interpretation of P is a singleton. □

The technique of the above proof can be adapted to prove a plethora of undefinability results, e.g., it can be shown that k-regularity of directed graphs cannot be decided in PolylogTime, for any fixed k.

We can develop this technique further to show that the order predicate on terms of sort $\mathbf{v}$ is a primitive in the logic. The proof of the following lemma is quite a bit more complicated and can be found in the full arXiv version [6] of this article.

Lemma 7. *Let P and Q be unary relation symbols. There does not exist an index logic formula φ such that for all $\{P, Q\}$-structures $\mathbf{A}$ such that $P^{\mathbf{A}}$ and $Q^{\mathbf{A}}$ are disjoint singleton sets $\{l\}$ and $\{m\}$, respectively, it holds that*

$$\mathbf{A}, val \models \varphi \text{ if and only if } l < m.$$

Theorem 8. *Let c and d be constant symbols in a vocabulary σ. There does not exist an index logic formula φ that does not use the order predicate $\leq$ on terms of sort $\mathbf{v}$ and that is equivalent with the formula $c \leq d$.*

The proof, by contradiction, shows that a formula φ as stated in the theorem would contradict the above lemma. We give the translation in the full arXiv version [6] of this article.

We conclude this section by affirming that, on purely relational vocabularies, the order predicate on sort $\mathbf{v}$ is redundant. The intuition for this result was given in the beginning of this section and we omit the formal proof.

Theorem 9. *Let σ be a vocabulary without constant or function symbols. For every sentence φ of index logic of vocabulary σ there exists an equivalent sentence φ' that does not use the order predicate on terms of sort $\mathbf{v}$.*

7 Discussion

An interesting open question concerns order-invariant queries. Indeed, while index logic is defined to work on ordered structures, it is natural to try to understand which queries about ordered structures that are actually invariant of the order, are computable in PolylogTime. Results of the kind given by Proposition 6 already suggest that very little may be possible. Then again, any polynomial-time numerical property of the size of the domain is clearly computable. We would love to have a logical characterization of the order-invariant queries computable in PolylogTime.

Another natural direction is to get rid of Turing machines altogether and work with a RAM model working directly on structures, as proposed by Grandjean and Olive [8]. Plausibly by restricting their model to numbers bounded in value by a polynomial in n (the size of the structure), we would get an equivalent PolylogTime complexity notion.

In this vein, we would like to note that extending index logic with numeric variables that can hold values up to a polynomial in n, with arbitrary polynomial-time functions on these, would be useful syntactic sugar that would, however, not increase the expressive power.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Ebbinghaus, H.D., Flum, J.: Finite Model Theory. Springer, second edn. (1999)
3. Fagin, R.: Contributions to Model Theory of Finite Structures. Ph.D. thesis, U. C. Berkeley (1973)
4. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: Karp, R. (ed.) Complexity of Computation, SIAM-AMS Proceedings, vol. 7, pp. 43–73 (1974)
5. Ferrarotti, F., González, S., Schewe, K.D., Turull Torres, J.M.: The polylog-time hierarchy captured by restricted second-order logic. In: Post-Proceedings of the 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (To appear). IEEE Computer Society (2019)
6. Ferrarotti F., González S., Turull Torres J.M., Van den Bussche J., Virtema J.: Descriptive Complexity of Deterministic Polylogarithmic Time. CoRR abs/1903.03413 (2019)
7. Grädel, E., Kolaitis, P., Libkin, L., Marx, M., Spencer, J., Vardi, M., Venema, Y., Weinstein, S.: Finite Model Theory and Its Applications. Springer (2007)
8. Grandjean, E., Olive, F.: Graph properties checkable in linear time in the number of vertices. J. Comput. Syst. Sci. **68**, 546–597 (2004)
9. Grohe, M.: Descriptive Complexity, Canonisation, and Definable Graph Structure Theory. Lecture Notes in Logic, Cambridge University Press (2017).
10. Grohe, M., Pakusa, W.: Descriptive complexity of linear equation systems and applications to propositional proof complexity. In: Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017. pp. 1–12. IEEE Computer Society (2017).
11. Gurevich, Y.: Toward logic tailored for computational complexity. In: Richter, M., et al. (eds.) Computation and Proof Theory, Lecture Notes in Mathematics, vol. 1104, pp. 175–216. Springer-Verlag (1984)
12. Gurevich, Y., Shelah, S.: Fixed-point extensions of first-order logic. Ann. Pure Appl. Logic **32**, 265–280 (1986).
13. Immerman, N.: Number of Quantifiers is Better Than Number of Tape Cells. J. Comput. Syst. Sci. 22(3): 384-406 (1981)
14. Immerman, N.: Relational queries computable in polynomial time. Information and Control **68**, 86–104 (1986)
15. Immerman, N.: Descriptive Complexity. Springer (1999)
16. Knuth, D.: Sorting and Searching, The Art of Computer Programming, vol. 3. Addison-Wesley, second edn. (1998)
17. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
18. Mix Barrington, D.A.: Quasipolynomial size circuit classes. In: Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992. pp. 86–93. IEEE Computer Society (1992).

19. Mix Barrington, D.A., Immerman, N., Straubing, H.: On uniformity within NC^1. J. Comput. Syst. Sci. **41**(3), 274–306 (1990)
20. Ramakrishnan, R., Gehrke, J.: Database Management Systems. McGraw-Hill, Inc., New York, NY, USA, 3 edn. (2003)
21. Stockmeyer, L.J.: The polynomial-time hierarchy. Theor. Comput. Sci. **3**(1), 1–22 (1976)
22. Vardi, M.: The complexity of relational query languages. In: Proceedings 14th ACM Symposium on the Theory of Computing. pp. 137–146 (1982)