Journal of Computer and System Sciences 119 (2021) 145–163

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Descriptive complexity of deterministic polylogarithmic time and space [☆]

Flavio Ferrarotti [a,*], Senén González [a], José María Turull Torres [b], Jan Van den Bussche [c], Jonni Virtema [c,d]

[a] *Software Competence Center Hagenberg, Austria*
[b] *Universidad Nacional de La Matanza, Argentina*
[c] *Hasselt University, Belgium*
[d] *Leibniz Universität Hannover, Germany*

A R T I C L E I N F O

Article history:
Received 20 August 2019
Received in revised form 10 February 2021
Accepted 18 February 2021
Available online 2 March 2021

Keywords:
Descriptive complexity
Polylogarithmic queries

A B S T R A C T

We propose logical characterizations of problems solvable in deterministic polylogarithmic time (PolylogTime) and polylogarithmic space (PolylogSpace). We introduce a novel two-sorted logic that separates the elements of the input domain from the bit positions needed to address these elements. We prove that the inflationary and partial fixed point variants of this logic capture PolylogTime and PolylogSpace, respectively. 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 a structure directly. We investigate whether an explicit predicate for the ordering of the domain is needed in our PolylogTime 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 [1–3] 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]. If this is the case, we say that ESO *captures* NP. The complexity class P is captured, on ordered finite structures, by a *fixed point logic*: the extensions of first-order logic with least fixed points [5,6].

After these two seminal results many more capturing results have been developed, and the benefits of this enterprise have been well articulated by several authors in the references given earlier and others [7]. We just mention here the

[☆] 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 Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH. The last author was partially suported by the DFG grant VI 1045/1-1.

[*] Corresponding author.
E-mail address: flavio.ferrarotti@scch.at (F. Ferrarotti).

https://doi.org/10.1016/j.jcss.2021.02.003
0022-0000/© 2021 Elsevier Inc. All rights reserved.

advantage of being able to specify properties of structures (e.g., data structures and databases) in a logical and 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 $(\log n)^k$ 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 [3]. A family $\mathcal{F}$ of ordered finite structures over some fixed finite vocabulary belongs to the complexity class PolylogTime if $\mathcal{F}$ is decided by some polylogarithmic-time random-access Turing machine. In this paper we show that PolylogTime can be captured on finite ordered structures by a new logic called *index logic*.

Index logic is a two-sorted logic whose semantics is only defined over finite ordered structures. Variables of the first sort range over the domain of the input structure, whereas variables of the second sort range over the initial segment of the natural numbers of length $\log(n)$, where n is the size of the domain of the input structure. Elements of the second sort can then be used to represent the bit positions needed to address the elements of the first sort. Index logic includes full fixed point 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 to 1. This “indexing mechanism” lends index logic its name.

In the course of proving our capturing result we introduce a new variant of random-access Turing machines called *direct-access Turing machines*. Random-access Turing machines read their inputs from a random-access tape in which input structures are encoded as binary strings. Direct-access machines do not require this encoding as they can access the different relations, functions and constants of the input structure directly. We show that both variants are equivalent in the sense that they lead to the same notion of PolylogTime. Direct-access Turing machines which compute directly on structures simplify the proofs of our main characterization theorems. The alternative of using random-access Turing machines results in much longer and cumbersome characterization proofs, mostly due to the complex logic formulae needed to model the machines random-access to the relevant parts of the binary encoded input. Note that in PolylogTime we cannot read the whole input as in Immerman and Vardi [5,6] logical characterization of PTIME.

A challenge was to develop a logic which enables the expression of PolylogTime problems in a relatively clean and natural way. The indexing mechanism in our logic is a key component to meet that challenge. The alternative of using fixed point operations and the BIT predicate[1] –which was very successfully used by Immerman and others to characterize related sublinear complexity classes [3] – to address values of the first sort leads in this case to a rather awkward logic that makes it difficult to express even simple queries.

We also devote attention to gaining a detailed understanding of the expressivity of index logic. In particular, we observe that order comparisons between quantified variables of the first sort can be expressed in terms of their addresses. In contrast, we show that this is not possible for constants of the first sort that are directly given by the structure. This implies that a variant of index logic without an explicit order predicate on the first sort does not capture PolylogTime on structures with constants.

Finally, we introduce a variant of index logic with partial fixed point operators and show that it captures PolylogSpace. This result is analogous to the classical result regarding the descriptive complexity of PSPACE, which is captured over ordered structures by first-order logic with the addition of partial fixed point operators [8]. For consistency we define PolylogSpace using the model of direct-access Turing machines, i.e., the variant of the random-access Turing machines introduced in this paper. As with PolylogTime, both models of computation lead to the same notion of PolylogSpace. Moreover we show that in the case of PolylogSpace random-access to the input-tape can be replaced with sequential-access without having any impact on the complexity class. Similar to PSPACE, the nondeterministic and deterministic PolylogSpace classes coincide. It is interesting to note that in addition to the problems in nondeterministic logarithmic space, there are well-known natural problems that belong to PolylogSpace (see related work below).

A preliminary version of this paper was presented at the 26th International Workshop in Logic, Language, Information and Computation [9]. This is an extended improved version which, in addition to the full proofs of the results on deterministic polylogarithmic time reported in [9], also considers polylogarithmic space and its descriptive characterization in terms of a variant of index logic.

Related work. Many natural fixed point computations such as transitive closure converge after a polylogarithmic number of steps. This motivated the study of a fragment of fixed point logic with counting (FPC) that only allows polylogarithmically many iterations of the fixed point operators (POLYLOG-FPC). As shown in [10], 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. An old result in [11] directly implies that POLYLOG-FPC is strictly weaker than FPC with regards to expressive power.

[1] BIT(x, i) holds iff the i-th bit of x in binary is 1.

It is well known that the (nondeterministic) logarithmic time hierarchy corresponds exactly to the set of first-order definable Boolean queries (see Theorem 5.30 in [3]). The relationship between uniform families of circuits within NC^1 and nondeterministic random-access logarithmic time machines was studied in [12]. However, the study of the descriptive complexity of classes of problems decidable by *deterministic* formal models of computation in polylogarithmic time, i.e., the central topic of this paper, appears not to have been considered 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 more attention [13,14]. A theorem which is analogous to Fagin's famous theorem [4] was recently proven for nondeterministic polylogarithmic time [14]. The logical characterization was obtained using a second-order logic with restricted second-order quantification ranging over relations of size at most polylogarithmic in the size of the structure and first-order universal quantification bounded to those relations. This latter work is closely related to the work on constant depth quasi-polynomial size AND/OR circuits and the corresponding restricted second-order logic in [13]. Both logics capture the full alternating polylogarithmic time hierarchy. However, the additional restriction in the first-order universal quantification in the second-order logic defined in [14] 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 Stockmeyer's result [15] on the polynomial-time hierarchy. Unlike the classical results of Fagin and Stockmeyer [4,15], the results on the descriptive complexity of nondeterministic polylogarithmic time classes only hold over ordered structures.

Up to the authors knowledge, little is known regarding the relationship of PolylogSpace with the main classical complexity classes (see [16] and [17]). Let L and NL denote deterministic and nondeterministic logarithmic space, respectively. Further, let L^{j} denote $\mathrm{DSPACE}[(\lceil \log n \rceil)^j]$. We do know however the following:

(i) PolylogSpace $\neq$ P, and it is *unknown* whether PolylogSpace $\subseteq$ P.
(ii) PolylogSpace $\neq$ NP, and it is *unknown* whether PolylogSpace $\subseteq$ NP.
(iii) Obviously: $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{L}^2 \subseteq$ PolylogSpace $\subseteq \mathrm{DTIME}[2^{(\lceil \log n \rceil)^{O(1)}}]$, the latter class being known as quasi-polynomial time (QuasiP).
(iv) For all $i \geq j \geq 1$, L^{j} uniform $\mathrm{NC}^{\mathrm{i}} \subseteq \mathrm{L}^{\mathrm{i}}$ (see [18]); hence we have that PolylogSpace uniform NC $\subseteq$ PolylogSpace.
(v) For all $i \geq 1$, let SC^{i} be the class of all languages that are decidable by deterministic Turing machines whose space is bounded by $O((\log n)^i)$ and whose time is simultaneously bounded by $n^{O(1)}$. Let SC (Steve's Class) be $\bigcup_{i \in \mathbb{N}} \mathrm{SC}^{\mathrm{i}}$ (see [19]). It follows that $\mathrm{SC} \subseteq \mathrm{P} \cap$ PolylogSpace.

Some interesting natural problems in PolylogSpace follow. From (iv) we get that division, exponentiation, iterated multiplication of integers [20] and integer matrix operations such as exponentiation, computation of the determinant, rank and the characteristic polynomial (see [21] and [22] for detailed algorithms in L^2) are all in PolylogSpace. Other well-known problems in the class are k-colorability of graphs of bounded tree-width [23], primality, 3NF test, BCNF test for relational schemas of bounded tree-width [24,25] and the circuit value problem of only EXOR gates [16]. Finally, in [26] an interesting family of problems is presented. It is shown there that for every $k \geq 1$ there is an algebra $(S; +, \cdot)$ over matrices such that the depth $O(\log n)^k$ straight linear formula problem over $M(S; +, \cdot)$ is NC^{k+1} complete under L reducibility. It then follows from (iv) that these problems are in $\mathrm{DSPACE}[(\log n)^{k+1}]$.

2. Preliminaries

In descriptive complexity it is a common practice to work only with relational structures since 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 $f(x)$ then we cannot spend the linear time required to find a y such that $\tilde{f}(x, y)$ holds. We therefore work with structures containing functions as well as relations and constants. We write $R_i^{r_i}$ and $f_i^{k_i}$ to denote relation and function symbols of arities r_i and k_i, respectively. Constant symbols are denoted by c_i. A *finite vocabulary* is a finite set of relation, function, and constant symbols. A *finite structure* $\mathbf{A}$ of vocabulary $\sigma = \{R_1^{r_1}, \ldots, R_p^{r_p}, c_1, \ldots c_q, f_1^{k_1}, \ldots, f_s^{k_s}\}$ is a tuple

$$(A, R_1^{\mathbf{A}}, \ldots, R_p^{\mathbf{A}}, c_1^{\mathbf{A}}, \ldots c_q^{\mathbf{A}}, f_1^{\mathbf{A}}, \ldots, f_s^{\mathbf{A}})$$

consisting 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 $f_i^{\mathbf{A}} : A^{k_i} \to A$. Throughout this paper $\leq$ will denote a binary relation symbol that is always interpreted as a linear order of the given finite structure. A *finite ordered σ-structure* $\mathbf{A}$ is a finite σ-structure, where the binary relation symbol $\leq \in \sigma$ and $\leq^{\mathbf{A}}$ is a linear order on A. In this paper, we consider only finite ordered structures. We emphasize that $\leq^{\mathbf{A}}$ is always the prescribed linear order of the ordered structure $\mathbf{A}$. Every finite ordered structure is isomorphic to one whose domain is an initial segment of the natural numbers. Thus, we assume 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 sometimes write $\log^k n$ as a shorthand for $(\lceil \log n \rceil)^k$. A tuple of elements $(a_1, \ldots, a_k)$ is frequently denoted as $\bar{a}$ and $\bar{a}[i]$ denotes the i-th element in it. Similarly, if s is a finite string then we denote by $s[i]$ the i-th letter of this string.

3. Deterministic polylogarithmic time

The restriction to sublinear time yields that a (sequential) Turing machine does not have, in general, enough time to access the entire input. Therefore, logarithmic time complexity classes are usually studied using models of computation that have random-access to their input, i.e., that can access every input address directly. Hence, we adopt a Turing machine model that has a *random-access* read-only input; similar to the logarithmic-time Turing machine in [12].

Our notion 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, are equipped with a *tape head* which scans the cells and are right-end infinite. The tape heads of the sequential access address-tape and work-tapes can move left or right. Whenever a tape 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 between the leftmost cell and the first blank cell of the address-tape (if the leftmost cell is blank then i is considered to be 0) as follows: If i is strictly smaller than the 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 $Q \cup \Sigma$.

Intuitively, $\delta(q, a_1, a_2, \ldots, a_{k+2}) = (p, b_2, D_2, \ldots, b_{k+2}, D_{k+2})$ means the following: If M is in the state q, the input-tape head is scanning a_1, the index-tape head is scanning a_2 and for every $i = 1, \ldots, k$ the head of the i-th work-tape is scanning a_{i+2}, then the next state is p, the index-tape head writes b_2 and moves in the direction indicated by D_2, and for every $i = 1, \ldots, k$ the head of the i-th work-tape writes b_{i+2} and moves in the direction indicated by D_{i+2}. Situations in which the transition function is undefined indicate that the computation must stop. Observe that δ cannot change the contents of the input tape.

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 corresponding 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 designated symbol for 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 (possibly infinite) sequence of configurations which starts with the initial configuration and for every two consecutive configurations the latter is obtained by applying the transition function of M to the former. If the transition function is not defined for the current configuration, then the machine stops and the configuration is called *final*. An input string is *accepted* if an accepting final configuration, i.e., a configuration with a state belonging to F, is reached. The input is *rejected* if a final configuration is reached and its state does not belong to F.

Example 1. Following a simple strategy, a random-access Turing machine M can compute 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 the computation is finished. 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 $\triangleleft$. It then rewrites the last 0 back to the blank symbol $\sqcup$. 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 then it writes a 1 and tests whether the input-tape head jumps to the cell with $\triangleleft$; if that is the case then 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:

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

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

When we want to give a finite ordered 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 [3]. Let $\sigma = \{R_1^{r_1}, \dots, R_p^{r_p}, c_1, \dots, c_q, f_1^{k_1}, \dots, f_s^{k_s}\}$ be a vocabulary and let $\mathbf{A}$ with $A = \{0, 1, \dots, n-1\}$ be a finite ordered structure of vocabulary σ. Note that the order $\leq^{\mathbf{A}}$ on A can be used to define an order for tuples of elements of A as well. Each relation $R_i^{\mathbf{A}} \subseteq A^{r_i}$ of $\mathbf{A}$ is encoded as a binary string $\mathrm{bin}(R_i^{\mathbf{A}})$ of length n^{r_i}, where 1 in a given position m indicates that the m-th tuple of A^{r_i} is in $R_i^{\mathbf{A}}$. Likewise, each constant number $c_j^{\mathbf{A}}$ is encoded as a binary string $\mathrm{bin}(c_j^{\mathbf{A}})$ of length $\lceil \log n \rceil$.

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

The encoding of the whole structure $\mathrm{bin}(\mathbf{A})$ is the concatenation of the binary strings encoding its relations, constants and functions. The length $\hat{n} = |\mathrm{bin}(\mathbf{A})|$ 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 $\log \hat{n} \in O(\lceil \log n \rceil)$, and hence $\mathrm{DTIME}[\log^k \hat{n}] = \mathrm{DTIME}[\log^k n]$.

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 also allows us to provide a less cumbersome proof of our main capturing result.

Let $\sigma = \{\leq\} \cup \{R_1^{r_1}, \dots, R_p^{r_p}, c_1, \dots c_q, f_1^{k_1}, \dots, f_s^{k_s}\}$ be a finite vocabulary for ordered structures. A *direct-access Turing machine that takes finite ordered σ-structures $\mathbf{A}$ as an 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 the answer (0 or 1) to the query $(n_1, \dots, n_{r_i}) \in R_i^{\mathbf{A}}$ for each relation $R_i \in \sigma$. Here n_j denotes the number written in binary in the jth distinguished tape of R_i. If one of the n_j is too large and does not denote any domain element, we stipulate that the answer to the query $(n_1, \dots, n_{r_i}) \in R_i^{\mathbf{A}}$ is 0. Note that we do not add the aforementioned construction for the order predicate $\leq$; the answer for the query $(n_1, n_2) \in \leq^{\mathbf{A}}$ can be computed directly from the binary representations n_1 and n_2.[2]

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 then we have

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

If l corresponds to one of the read-only tapes then 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 $f_i^{\mathbf{A}}(n_1, \dots n_{k_i})$ 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 then 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 $c_1, \dots, c_q$ hold their values in $\mathbf{A}$ in binary. The address-tapes, value-tapes and ordinary work-tapes hold only blanks. One additional constant-tape (there are $q+1$ of them) holds the size n of the domain of $\mathbf{A}$ in binary. Notice that a direct-access Turing machine cannot otherwise determine the size of the domain of a structure over a relational vocabulary, i.e., a vocabulary that does not have function symbols. On the other hand, if the vocabulary contains a function symbol, then the direct-access Turing machine can adapt the algorithm of the random-access Turing machine in Example 1 to determine the size of the domain, by checking whether the function value-tape has been blanked, instead of checking whether the input-tape head scans the end-marker $\triangleleft$ (i.e., the address in the index-tape is beyond the end of the input-tape).

The notions of an accepting and rejecting computation for direct-access machines, as well as the notion of deciding a class of structures, are defined in an analogous manner mirroring the definitions given in Section 3 for random-access machines.

[2] This design choice is purely esthetic and has no effect on the computational power of the direct-access machines in general.

Remark 2. Direct-access Turing machines M can be also used to decide properties of unordered structures. In this case, the unordered structure $\mathbf{A}$ is equipped with an arbitrary interpretation of the order predicate $\leq$, and the ordered expansion of $\mathbf{A}$ is given as an input to the direct-access Turing machine. Analogous to the way normal Turing machines work on unordered structures, the answer to whether M accepts the ordered expansion of $\mathbf{A}$ should be invariant on the interpretation of $\leq$.

Theorem 3. *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.*

Proof. We will first sketch how a random-access Turing machine M_r simulates a direct-access Turing machine M_d on an input $\mathbf{A}$. Let n denote the cardinality of A and $\hat{n}$ the length of $\text{bin}(\mathbf{A})$. We dedicate a work-tape of M_r to every tape of M_d. In addition, for each relation R of arity r we add one extra tape that will always contain the answer to the query $(n_1, \dots, n_r) \in R^{\mathbf{A}}$. We also use additional work-tapes for convenience. We then encode the initial configuration of M_d into the tapes of M_r:

1. On the 0th constant tape, write n in binary.
2. On each tape for a constant c_i, write $c_i^{\mathbf{A}}$ in binary.
3. For the answer-tapes of relations R_i, write the bit 0.

For encoding the transitions of M_d, we will in addition need two more constructs:

a. Updating the answer-tapes of relations after each transition.
b. Updating the answer-tapes of functions after each transition.

We now need to verify that these procedures (3. is trivial) can be performed by M_r in polylogarithmic time with respect to $\hat{n}$.

Step 1. On a fixed vocabulary σ, we have $\hat{n} = f(n)$ for some fixed function f of the form

$$n^{r_1} + \cdots + n^{r_p} + q\lceil \log n \rceil + \lceil \log n \rceil n^{k_1} + \cdots + \lceil \log n \rceil n^{k_s}.$$

We will find n by executing a binary search between the numbers 0 and $\hat{n}$; note that checking whether a binary representation of a number is at most $\hat{n}$ can be done by writing the representation to the index-tape and checking whether a bit or $\triangleleft$ is read from the input-tape. For each i between 0 and $\hat{n}$, $f(i)$ can be computed in polynomial time with respect to the length of $\hat{n}$ in binary, and thus in polylogarithmic time with respect to $\hat{n}$.

Step 2. The binary representation of a constant $c_i^{\mathbf{A}}$ is written in the input-tape between $g(n)$ and $g(n) + \lceil \log n \rceil$, where g is a fixed function of the form $n^{r_1} + \cdots + n^{r_p} + (i-1)\lceil \log n \rceil$. The numbers n and $g(n)$ are obtained as in case 1. Then $g(n)$ is written on the index tape and the next $\lceil \log n \rceil$ bits of the input are copied to the tape corresponding to c_i.

Steps a. and b. These cases are handled similar to each other and to the case 2. above. The main difference for b. is that the bits of the output are not in successive positions of the input, but the location of each bit needs to be calculated separately.

We next sketch how a direct-access Turing machine M_d simulates a random-access Turing machine M_r on an input $\mathbf{A}$. First note that an approach similar to the converse direction does not work here as we do not have enough time to directly construct the initial configuration of M_r inside M_d. For each work-tape of M_r, we dedicate a work-tape of M_d. For the index-tape of M_r, we dedicate a work-tape of M_d and call it the index-tape of M_d. Moreover, we use some additional work-tapes for convenience. The idea of the simulation is that the dedicated work-tapes and the index-tape of M_d copy exactly the behavior of the corresponding tapes of M_r. The additional work-tapes are used to calculate to which part of the input of M_r the index-tape refers to. After each transition of M_r this is checked so that the machine M_d can update its address-tapes accordingly.

Recall that given an input $\sigma = \{R_1^{r_1}, \dots, R_p^{r_p}, c_1, \dots c_q, f_1^{k_1}, \dots, f_s^{k_s}\}$ structure $\mathbf{A}$ of cardinality n, the input of M_r is of length

$$n^{r_1} + \cdots + n^{r_p} + q\lceil \log n \rceil + \lceil \log n \rceil n^{k_1} + \cdots + \lceil \log n \rceil n^{k_s}. \tag{1}$$

The number written in binary on the index-tape of M_r determines the position of the input that is read by M_r. From (1) we obtain fixed functions on n that we use in the simulation to check which part of the input is read when the index-tape holds a particular number. For example, if the index-tape holds $n_1^r + 1$, we can calculate that the head of the input-tape of M_r reads the bit answering the query: is $\vec{0} \in R_2^{\mathbf{A}}$. We can use an extra work-tape of M_d to always store the bit that M_r is reading from its input. The rest of the simulation is straightforward. $\square$

5. Index logic

In this section, we introduce a novel logic called *index logic*, which over finite ordered structures captures PolylogTime. Our definition of index logic is inspired by the second-order logic in [13], where relation variables take values from the sub-domain $\{0, \dots, \lceil \log n \rceil - 1\}$ (n being the size of the interpreting structure), as well as by the well known counting logics defined in [27].

Given a vocabulary σ, for every ordered σ-structure $\mathbf{A}$, we define a corresponding set of natural numbers $Num(\mathbf{A}) = \{0, \dots, \lceil \log n \rceil - 1\}$ where $n = |A|$. Note that $Num(\mathbf{A}) \subseteq A$, since we assume that A is an initial segment of the natural numbers. This simplifies the definitions, but it is otherwise unnecessary. Also for technical reasons we work with structures with at least two elements in the domain. Otherwise, if A has just one element then $Num(\mathbf{A})$ would be empty.

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 $Num(\mathbf{A})$. We denote variables of sort $\mathbf{v}$ with $x, y, z, \dots$, possibly with a subindex such as $x_0, x_1, x_2, \dots$, and variables of sort $\mathbf{n}$ with $\mathrm{x}, \mathrm{y}, \mathrm{z}$, also possibly with a subindex. Relation variables, denoted with uppercase letters $X, Y, Z, \dots$, are always of sort $\mathbf{n}$, and thus range over relations defined on $Num(\mathbf{A})$.

Definition 4 *(Numerical and first-order terms).* The only terms of sort $\boldsymbol{n}$ are the variables of sort $\boldsymbol{n}$. For a vocabulary σ, the σ-terms t of sort $\boldsymbol{v}$ are generated by the following grammar:

$$t ::= x \mid c \mid f(t, \dots, t),$$

where x is a variable of sort $\boldsymbol{v}$, c is a constant symbol in σ, and f is a function symbol in σ.

Definition 5 *(Syntax of index logic).* Let σ be a vocabulary. The formulae of *index logic* IL(IFP) is generated by the following grammar:

$$\varphi ::= \mathrm{x}_1 \le \mathrm{x}_2 \mid R(t_1, \dots, t_k) \mid X(\mathrm{x}_1, \dots, \mathrm{x}_k) \mid (\varphi \wedge \varphi) \mid \neg\varphi \mid [\mathrm{IFP}_{\bar{\mathrm{x}}, X}\varphi]\bar{\mathrm{y}} \mid$$
$$t = index\{\mathrm{x} : \varphi(\mathrm{x})\} \mid \exists x(x = index\{\mathrm{x} : \alpha(\mathrm{x})\} \wedge \varphi) \mid \exists \mathrm{x}\varphi,$$

where $t, t_1, \dots, t_k$ are σ-terms of sort $\boldsymbol{v}$, R is a relation symbol in σ, $\mathrm{x}, \mathrm{x}_1, \dots, \mathrm{x}_k$ are variables of sort $\boldsymbol{n}$, and $\bar{\mathrm{x}}$ and $\bar{\mathrm{y}}$ are tuples of variables of sort $\boldsymbol{n}$ whose length coincides with the arity of the relation variable X. Moreover, $\alpha(\mathrm{x})$ is a formula where the variable x of sort $\boldsymbol{v}$ does not occur as a free variable.

Since we work always on ordered structures the prescribed order predicate $\le$ is in σ, and $t_1 \le t_2$ is an atomic formula as well. We also use the standard shorthand formulae $t_1 = t_2$, $\mathrm{x}_1 = \mathrm{x}_2$, $(\varphi \vee \psi)$ and $\forall \mathrm{y}\varphi$ with the obvious meanings.

As can be inferred already from the definition of its syntax, index logic includes full inflationary fixed point logic on the $\boldsymbol{n}$ sort. The logic, on the other hand, is heavily restricted on its access to the elements of the domain via variables of sort $\boldsymbol{v}$. Notice that existential quantification over the $\boldsymbol{v}$ sort is bounded by $x = index\{\mathrm{x} : \alpha(\mathrm{x})\}$. This informally means that x is equal to the element in the $\boldsymbol{v}$ sort (if it exists there) whose value written in binary has 1s only in those positions that appear in $\{\mathrm{x} : \alpha(\mathrm{x})\}$. Bounded universal quantification of the form $\forall x(x = index\{\mathrm{x} : \alpha(\mathrm{x})\} \to \varphi)$ would also be possible, but that is equivalent to $\neg\exists x(x = index\{\mathrm{x} : \alpha(\mathrm{x})\}) \vee \exists x(x = index\{\mathrm{x} : \alpha(\mathrm{x})\} \wedge \varphi)$ and thus already expressible in index logic.

A valuation is defined as usual for two-sorted logics. 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 sort $\mathbf{v}$, then $val(x) \in A$.
- If x is a variable of sort $\mathbf{n}$, then $val(\mathrm{x}) \in Num(\mathbf{A})$.
- If X is a relation variable with arity r, then $val(X) \subseteq (Num(\mathbf{A}))^r$.

If χ is a variable and B a permissible value for that variable, we write $val(B/\chi)$ to denote the valuation that maps χ to B and agrees with *val* for all other variables. Valuations extend to terms and tuples of terms in the usual way.

Fixed points are defined in the standard way (see, e.g., [28] and [29] for an in-depth presentation). 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 the *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 lfp(F). The *inflationary fixed point* of F, denoted by 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 $\varphi(X, \bar{\mathrm{x}})$ be a formula of vocabulary σ, where X is a relation variable of arity k and $\bar{\mathrm{x}}$ is a k-tuple of variables of sort $\mathbf{n}$. Let $\mathbf{A}$ be a σ-structure and *val* a variable valuation. The formula $\varphi(X, \bar{\mathrm{x}})$ gives rise to an operator $F^{\mathbf{A}, val}_{\varphi, \bar{\mathrm{x}}, X} : \mathcal{P}((Num(\mathbf{A}))^k) \to \mathcal{P}((Num(\mathbf{A}))^k)$ defined as follows:

$$F^{\mathbf{A}, val}_{\varphi, \bar{\mathrm{x}}, X}(S) := \{\bar{a} \in (Num(\mathbf{A}))^k \mid \mathbf{A}, val(S/X, \bar{a}/\bar{\mathrm{x}}) \models \varphi(X, \bar{\mathrm{x}})\}.$$

Definition 6. Let $\mathbf{A}$ be an ordered structure and *val* be a valuation over $\mathbf{A}$. Recall that x, x, X and t (possibly with subindices) denote individual variables of sort $\mathbf{v}$, individual variables of sort $\mathbf{n}$, relation variables of sort $\mathbf{n}$ and terms of sort $\mathbf{v}$, respectively. The formulae of IL(IFP) are interpreted as follows:

- $\mathbf{A}, val \models \mathrm{x}_1 \leq \mathrm{x}_2$ iff $val(\mathrm{x}_1) \leq val(\mathrm{x}_2)$.
- $\mathbf{A}, val \models R(t_1, \dots, t_k)$ iff $(val(t_1), \dots, val(t_k)) \in R^{\mathbf{A}}$.
- $\mathbf{A}, val \models X(\mathrm{x}_1, \dots, \mathrm{x}_k)$ iff $(val(\mathrm{x}_1), \dots, val(\mathrm{x}_k)) \in val(X)$.
- $\mathbf{A}, val \models t = index\{\mathrm{x} : \varphi(\mathrm{x})\}$ iff $val(t)$ in binary is $b_m b_{m-1} \cdots b_0$, where $m = \lceil \log |A| \rceil - 1$ and $b_j = 1$ iff $\mathbf{A}, val(j/\mathrm{x}) \models \varphi(\mathrm{x})$.
- $\mathbf{A}, val \models [\mathrm{IFP}_{\bar{\mathrm{x}},X}\varphi]\bar{\mathrm{y}}$ iff $val(\bar{\mathrm{y}}) \in \mathrm{ifp}(F^{\mathbf{A},val}_{\varphi,\bar{\mathrm{x}},X})$.
- $\mathbf{A}, val \models \neg\varphi$ iff $\mathbf{A}, val \not\models \varphi$.
- $\mathbf{A}, val \models \varphi \wedge \psi$ iff $\mathbf{A}, val \models \varphi$ and $\mathbf{A}, val \models \psi$.
- $\mathbf{A}, val \models \exists \mathrm{x}\varphi$ iff $\mathbf{A}, val(i/\mathrm{x}) \models \varphi$, for some $i \in Num(\mathbf{A})$.
- $\mathbf{A}, val \models \exists x(x = index\{\mathrm{x} : \alpha(\mathrm{x})\} \wedge \varphi)$ iff there exists $i \in A$ such that $\mathbf{A}, val(i/x) \models x = index\{\mathrm{x} : \alpha(\mathrm{x})\}$ and $\mathbf{A}, val(i/x) \models \varphi$.

It immediately follows from the famous result by Gurevich and Shelah regarding the equivalence between inflationary and least fixed points [30], that an equivalent index logic can be obtained if we (1) replace $[\mathrm{IFP}_{\bar{\mathrm{x}},X}\varphi]\bar{\mathrm{y}}$ by $[\mathrm{LFP}_{\bar{\mathrm{x}},X}\varphi]\bar{\mathrm{y}}$ in the formation rule for the fixed point operator in Definition 5, adding the restriction that every occurrence of X in φ is positive,[3] and (2) fix the interpretation $\mathbf{A}, val \models [\mathrm{LFP}_{\bar{\mathrm{x}},X}\varphi]\bar{y}$ iff $val(\bar{y}) \in \mathrm{lfp}(F^{\mathbf{A},val}_{\varphi,\bar{\mathrm{x}},X})$.

The use of *simultaneous fixed points* allows one to iterate several formulae at once in a single fixed point operator. In what follows, we make use of the convenient tool of simultaneous fixed points, for their addition to our logics does not increase their expressive powers. Following the syntax and semantics proposed by Ebbinghaus and Flum [28], a version of index logic with simultaneous inflationary fixed point operators can be obtained by replacing the clause corresponding to IFP in Definition 5 by the following:

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

Semantics for the simultaneous fixed point operator is defined such that $\mathbf{A}, val \models [\text{S-IFP}_{\bar{\mathrm{x}}_0,X_0,\dots,\bar{\mathrm{x}}_m,X_m}\varphi_0,\dots,\varphi_m]\bar{\mathrm{y}}$ iff $val(\bar{\mathrm{y}})$ 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.

In the next two subsections, we give two worked-out examples that illustrate the power of index logic. After that, the exact characterization of its expressive power is presented in Subsection 5.3.

5.1. Finding the binary representation of a term

Let t be a term of sort $\mathbf{v}$. In this example, we construct an index logic formula that expresses the well-known bit predicate $\mathrm{BIT}(t, \mathrm{x})$. The predicate $\mathrm{BIT}(t, \mathrm{x})$ states that the $(val(\mathrm{x}) + 1)$-th bit of $val(t)$ in binary is set to 1. Subsequently, the sentence $t = index\{\mathrm{x} : \mathrm{BIT}(t, \mathrm{x})\}$ is valid over the class of all finite ordered structures.

Informally, for a fixed term t, our implementation of $\mathrm{BIT}(t, \mathrm{x})$ works by iterating through the bit positions y from the most significant to the least significant. These bits are accumulated in a relation variable Z. For each y we set the corresponding bit on the condition that the resulting number does not exceed t. The set bits are collected in a relation variable Y.

In the formal description of $\mathrm{BIT}(t, \mathrm{x})$ below, we use the following abbreviations. We use M to denote the most significant bit position, i.e., $M = \lceil \log n \rceil$. Thus, formally, $\mathrm{z} = M$ abbreviates $\forall \mathrm{z}'\, \mathrm{z}' \leq \mathrm{z}$. Furthermore, for a unary relation variable Z, we use $\mathrm{z} = \min Z$ with the obvious meaning. We also use abbreviations such as $\mathrm{z} = \mathrm{z}' - 1$ with the obvious meaning.

Now $\mathrm{BIT}(t, \mathrm{x})$ is a simultaneous fixed point $[\text{S-IFP}_{\mathrm{y},Y,\mathrm{z},Z}\,\varphi_Y, \varphi_Z](\mathrm{x})$, where

$$\varphi_Z := (Z = \emptyset \wedge \mathrm{z} = M) \vee (Z \neq \emptyset \wedge \mathrm{z} = \min Z - 1),$$

$$\varphi_Y := Z \neq \emptyset \wedge \mathrm{y} = \min Z \wedge \exists x(x = index\{\mathrm{z} : Y(\mathrm{z}) \vee \mathrm{z} = \mathrm{y}\} \wedge t \geq x).$$

5.2. Binary search in an array of key values

To provide further insight on expressing properties with index logic, we develop an example showing how to express the useful procedure of binary search.

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

[3] This ensures that $F^{\mathbf{A},val}_{\varphi,\bar{\mathrm{x}},X}$ is a monotonic function and that the least fixed point $\mathrm{lfp}(F^{\mathbf{A},val}_{\varphi,\bar{\mathrm{x}},X})$ exists.

numbers. The constant $l := N^{\mathbf{A}}$ indicates the length of the array; the domain elements 0, 1, …, $l-1$ 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. Thus it is also defined on the domain elements that do not correspond to 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. Note that $\prec^{\mathbf{A}}$ 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., **A** must satisfy $\forall x \forall y \Big(x < y < N \rightarrow \big(K(x) \preceq K(y)\big)\Big)$. Finally, the constant $t := T^{\mathbf{A}}$ is the test value.

We want an index logic formula that determines whether the test value t is in the array. More formally, we want to express the following condition with an index logic formula.

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

We follow an approach which is close to the standard algorithm [31] for binary search:

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

We use a simultaneous fixed point with binary relation variables L and R, and a unary relation variable Z. Relation variables L and R encode integer values in binary and fulfill a similar role in the index logic formula than in the algorithm. More concretely, for each $i \in Num(\mathbf{A})$, the value of the term $index\{\mathtt{x} : L(i, \mathtt{x})\}$ will be the value of the integer variable L before the $(i+1)$-th iteration of the while loop (and similarly for R). The auxiliary variable Z is used to keep track of the current iteration, starting with 0 and adding in each step the current iteration number to Z until a fixed point is reached, i.e., until $Z = \{0, \ldots, \lceil \log n \rceil - 1\}$.

We further use the bit predicate from Section 5.1 and the following subformulae:

- $avg(X, Y, \mathtt{x})$ expressing that the bit $\mathtt{x}$ is set to 1 in the binary representation of $\lfloor (x+y)/2 \rfloor$ for x and y the numbers encoded in binary in X and Y, respectively.
- $minusone(X, \mathtt{y})$ expressing that the bit $\mathtt{y}$ is set to 1 in the binary representation of $x - 1$ for x the number encoded in binary in X.

Since the numeric **n** sort is only logarithmic in the size of the input structure, it follows from Immerman-Vardi theorem [5,6] that any PolylogTime time decidable query is expressible as a formula in index logic over the **n** sort. The fact that each of the above queries is computable in PTIME on the size of the numeric sort, or equivalently, in PolylogTime in the size of the input structure, determines that the required subformulae exist.

In the context in which we use $avg(X, Y, \mathtt{x})$, the variables X and Y are taken to be $L(\mathtt{z}, .)$ and $R(\mathtt{z}, .)$, respectively. Thus we write $avg'(\mathtt{z}, \mathtt{x})$ to denote the formula obtained by replacing in $avg(X, Y, \mathtt{x})$ each occurrence of $X(\mathtt{u})$ and $Y(\mathtt{u})$ by $L(\mathtt{z}, \mathtt{u})$ and $R(\mathtt{z}, \mathtt{u})$, respectively. Likewise, we write $minusone'(\mathtt{z}, \mathtt{u})$ to denote the formula obtained by replacing in $minusone(X, u)$ each occurrence of $X(\mathtt{u})$ by $avg'(\mathtt{z}, \mathtt{u})$. We also write $test(\mathtt{z})$ to denote the formula $\exists e(e = index\{\mathtt{x} : avg'(\mathtt{z}, \mathtt{x})\} \wedge K(e) \succ T)$.

Finally, the index logic formula expressing condition (γ) can be written as follows:

$$\exists x(x = index\{\mathtt{l} : \psi(\mathtt{l})\} \wedge K(x) = T)$$

where

$$\psi(\mathtt{l}) := \exists \mathtt{s} \forall \mathtt{s}'(\mathtt{s}' \leq \mathtt{s} \wedge [\text{S-IFP}_{\mathtt{z},\mathtt{x},L,\mathtt{z},\mathtt{x},R,\mathtt{z},Z}\, \varphi_L, \varphi_R, \varphi_Z](\mathtt{s}, \mathtt{l})),$$

$$\varphi_Z := (Z = \emptyset \wedge \mathtt{z} = 0) \vee (Z \neq \emptyset \wedge \mathtt{z} = \max Z + 1),$$

$$\varphi_L := Z \neq \emptyset \wedge \mathtt{z} = \max Z + 1 \wedge \\ \exists \mathtt{z}'(\mathtt{z}' = \max Z \wedge (test(\mathtt{z}') \rightarrow L(\mathtt{z}', \mathtt{x})) \wedge (\neg test(\mathtt{z}') \rightarrow avg'(\mathtt{z}', \mathtt{x}))),$$

$$\varphi_R := (Z = \emptyset \wedge \mathtt{z} = 0 \wedge \mathrm{BIT}(N-1, \mathtt{x})) \vee (Z \neq \emptyset \wedge \mathtt{z} = \max Z + 1 \wedge \\ \exists \mathtt{z}'(\mathtt{z}' = \max Z \wedge (test(\mathtt{z}') \rightarrow minusone'(\mathtt{z}', \mathtt{x})) \wedge (\neg test(\mathtt{z}') \rightarrow R(\mathtt{z}', \mathtt{x})))).$$

5.3. *The logical characterization theorem for* PolylogTime

We start by defining what it means for a logic to capture a complexity class over finite ordered structures.

Definition 7. A logic $\mathcal{L}$ captures PolylogTime on the class of finite ordered structures iff the following holds:

- For every $\mathcal{L}$-sentence φ of a vocabulary σ, for finite ordered structures, the language $\{\text{bin}(\mathbf{A}) \mid \mathbf{A} \models \varphi, \mathbf{A} \text{ is a finite ordered structure}\}$ is decidable by a random-access Turing machine in PolylogTime.
- For every property $\mathcal{P}$ of (binary encodings of) finite ordered structures of vocabulary σ that can be decided by a random-access Turing machine in PolylogTime, there is a sentence $\varphi_{\mathcal{P}}$ of $\mathcal{L}$ such that for every finite ordered structure $\mathbf{A}$ of vocabulary σ it holds that $\mathbf{A} \models \varphi_{\mathcal{P}}$ iff $\mathbf{A}$ has property $\mathcal{P}$.

Note that by Theorem 3 we can give an equivalent definition in terms of direct-access Turing machines, avoiding the need for binary encodings of structures.

The following result confirms that index logic serves our original purpose of characterizing PolylogTime on the class of finite ordered structures.

Theorem 8. *On finite ordered structures, index logic captures* PolylogTime.

Proof. *Formulae of index logic can be evaluated in polylogarithmic time*. Let VAR be a finite set of variables (of sort **n**, **v**, and relational). We use a Turing machine model that has a designated work-tape for each of the variables in VAR. The tape designated for a variable contains the value of that variable encoded as a binary string. We use induction on the structure of formulae to show that for every sentence φ of index logic, whose variables are from the set VAR, there exists a direct-access Turing machine M_φ such that, for every ordered structure $\mathbf{A}$ with $|A| = n$ and every valuation *val*, it decides in time $O(\lceil \log n \rceil^{O(1)})$ whether $\mathbf{A}, val \models \varphi$. Since VAR is an arbitrary finite set, this suffices.

In the proof variables v of sort $\boldsymbol{n}$ and $\boldsymbol{v}$ are treated in a similar way as constant symbols, meaning that their value $val(v)$ is written in binary in the first $\lceil \log n \rceil$ cells of their designated work-tapes. The work-tape designated to a relation variable X of arity k contains $val(X) \subseteq Num(\mathbf{A})^k$ encoded as a binary string in its first $\lceil \log n \rceil^k$ cells, where a 1 in the i-th cell indicates that the i-th tuple in the lexicographic order of $Num(\mathbf{A})^k$ is in $val(X)$.

Let t be a term, M be a direct-access Turing machine and *val* be a valuation such that for every variable χ that occurs in t the value $val(\chi)$ is the one encoded in binary in the designated work-tape for χ. We start by showing that $val(t)$ can then be computed by M in time $O(\lceil \log n \rceil^{O(1)})$. If t is a variable of sort $\boldsymbol{n}$, $\boldsymbol{v}$ or a constant symbol, then M only needs to read the first $\lceil \log n \rceil$ cells of the appropriate work-tape or constant-tape, respectively. If t is a term of the form $f_i(t_1, \ldots, t_k)$, we access and copy each $val(t_j)$ in binary in the corresponding address-tapes of f_i. By the induction hypothesis this takes time $O(\lceil \log n \rceil^{O(1)})$ each. Using $\lceil \log n \rceil$ additional steps the result of length $\lceil \log n \rceil$ will then be accessible in the value-tape of f_i.

We use induction to prove our main claim. Let φ be a formula with variables in VAR, *val* be a valuation and M be a direct-access Turing machine such that for every variable χ that occurs free in φ the value $val(\chi)$ is written in binary in the designated work-tape for χ. We show that $\mathbf{A}, val \models \psi$ can be decided by M in time $O(\lceil \log n \rceil^{O(1)})$.

If φ is an atomic formula of the form $t_1 \leq t_2$ then M can evaluate φ in polylogarithmic time by accessing the values of t_1 and t_2 in binary and then comparing their $\lceil \log n \rceil$ bits.

If φ is an atomic formula of the form $R_i(t_1, \ldots, t_k)$ then M can evaluate φ in polylogarithmic time by simply computing the values of the terms $t_1, \ldots, t_k$ and copying them to the corresponding address-tapes of R_i. Recall that each term's value can be computed in polylogarithmic time. They can also be copied in polylogarithmic time since each such value takes up to $\lceil \log n \rceil$ bits of space.

If φ is an atomic formula of the form $X(\mathrm{x}_1, \ldots, \mathrm{x}_k)$ then M can evaluate φ in polylogarithmic time as follows. First M accesses the values $\mathrm{x}_1, \ldots, \mathrm{x}_k$ and computes (in binary) the position i of the tuple $(\mathrm{x}_1, \ldots, \mathrm{x}_k)$ in the lexicographic order of $Num(\mathbf{A})^k$. Then M accesses the i-th cell of the work-tape which contains the encoding of $val(X)$ of length $\lceil \log n \rceil^k$. Computing i involves simple arithmetic operations on binary numbers of length bounded by $\log(\lceil \log n \rceil^k)$, which can clearly be done in time polynomial in $\log n$.

If φ is an atomic formula of the form $t = index\{\mathrm{x} : \psi(\mathrm{x})\}$ then M proceeds as follows. Let $s = \lceil \log n \rceil - 1$ and let $b_s b_{s-1} \cdots b_0$ be $val(t)$ in binary. For every i, $0 \leq i \leq s$, M writes i in binary in the work-tape designated for the variable x and checks whether $\mathbf{A}, val(i/\mathrm{x}) \models \psi(\mathrm{x})$ iff $b_i = 1$. Since by the induction hypothesis this check can be done in polylogarithmic time and $val(t)$ can also be computed in polylogarithmic time, we get that M decides $t = index\{\mathrm{x} : \varphi(\mathrm{x})\}$ in polylogarithmic time as well.

If φ is a formula of the form $[\text{IFP}_{\bar{\mathrm{x}}, X} \psi]\bar{y}$ where the arity of X is k. Let $F^{\mathbf{A}, val}_{\psi, \bar{\mathrm{x}}, X} : \mathcal{P}((Num(\mathbf{A}))^k) \to \mathcal{P}((Num(\mathbf{A}))^k)$ denote the related operator $F^0 := \emptyset$ and $F^{i+1} := F^i \cup F^{\mathbf{A}, val}_{\psi, \bar{\mathrm{x}}, X}(F^i)$ for each $i \geq 0$. The inflationary fixed point is reached on stage $|Num(\mathbf{A})^k|$ at the latest and thus $\text{ifp}(F^{\mathbf{A}, val}_{\psi, \bar{\mathrm{x}}, X}) = F^{\log^k n}$. Recall that

$$F^{\mathbf{A}, val}_{\psi, \bar{\mathrm{x}}, X}(S) := \{\bar{a} \in (Num(\mathbf{A}))^k \mid \mathbf{A}, val(S/X, \bar{a}/\bar{\mathrm{x}}) \models \psi(X, \bar{\mathrm{x}})\}.$$

Then M can proceed as follows. On each stage M computes the value of F^{i+1} in binary on a work-tape and then copy over this value to the work-tape designated for X. In stage $i = 0$ it only needs to write the string $0^{\lceil \log n \rceil^k}$ in the work-tape designated for X. To compute F^{i+1} from F^i it goes through all k-tuples $\bar{a} \in (Num(\mathbf{A}))^k$ in lexicographic order. For $1 \leq j \leq k$ it writes $\bar{a}[j]$ in binary on the designated work-tape for $\bar{\mathrm{x}}[j]$ and checks whether

$$\mathbf{A}, val(S/X, \bar{a}/\bar{\mathrm{x}}) \models \psi(X, \bar{\mathrm{x}}) \tag{2}$$

holds. By induction hypothesis, this can be checked in time $O(\lceil \log n \rceil^{O(1)})$. If (2) holds and $\bar{a}$ is the l-th k-tuple in the lexicographic ordering then M writes 1 in the l-th cell of the work-tape where the value of F^{i+1} is being constructed. Otherwise it writes 0. Hence the computation of F^{i+1} from F^i can be done in time $\log^k n \times O(\lceil \log n \rceil^{O(1)})$ which is still $O(\lceil \log n \rceil^{O(1)})$. Clearly $\mathrm{ifp}(F^{\mathbf{A}, val}_{\psi, \bar{\mathrm{x}}, X}) = F^{\log^k n}$ can be computed in time $O(\lceil \log n \rceil^{O(1)})$ as well. To determine whether $val(\bar{y})$ is included in the fixed point is also computable in $O(\lceil \log n \rceil^{O(1)})$ since M can just compute the position of $val(\bar{y})$ in the lexicographic order of k-tuples and then check whether that position has a 0 or 1 in the work-tape corresponding to X.

If φ is a formula of the form $\exists x(x = index\{\mathrm{x} : \alpha(\mathrm{x})\} \wedge \psi(x))$ then M proceeds as follows. For each $i \in \{0, \dots, \lceil \log n \rceil - 1\}$, M writes i in binary in the work-tape designated for x and checks whether $\mathbf{A}, val(i/\mathrm{x}) \models \alpha(\mathrm{x})$. Since by definition x does not appear free in $\alpha(\mathrm{x})$, it follows by the induction hypothesis that M can perform each of these checks in polylogarithmic time. In parallel M writes the bit string $b_s b_{s-1} \cdots b_0$ defined such that $b_i = 1$ iff $\mathbf{A}, val(i/\mathrm{x}) \models \alpha(\mathrm{x})$ to the work-tape designated to the variable x. Let the content of this work-tape at the end of this process be t in binary. M can now check whether $t < n$ (recall that by convention M has the value n in binary in one of its constant-tapes and thus this can be done in polylogarithmic time). If $t \geq n$ then $\mathbf{A}, val \not\models \varphi$. If $t < n$ then M checks whether $\mathbf{A}, val(t/x) \models \psi$. By the induction hypothesis this check can also be done in polylogarithmic time.

Finally, if φ is a formula of the form $\exists \mathrm{x}\, \psi$ then for each $i \in \{0, \dots, \lceil \log n \rceil - 1\}$ M writes i in binary to the work-tape designated for x and checks whether $\mathbf{A}, val(i/\mathrm{x}) \models \psi$. It follows by the induction hypothesis that M can perform each of these checks in polylogarithmic time. If the test is positive for some i then $\mathbf{A}, val \models \varphi$. The remaining cases are those corresponding to Boolean connectives and follow trivially from the induction hypothesis.

Every polylogarithmic time property can be expressed in index logic. Suppose we are given a class $\mathcal{C}$ of ordered σ-structures which can be decided by a deterministic polylogarithmic time direct-access Turing machine $M = (Q, \Sigma, \delta, q_0, F, \sigma)$ that has m tapes (including ordinary work-tapes, address-tapes, (function) value-tapes and constant-tapes). We assume w.l.o.g. that $F = \{q_a\}$ (there is only one accepting state), $|Q| = a + 1$ and $Q = \{q_0, q_1, \dots, q_a\}$.

Let M run in time $O(\lceil \log n \rceil^k)$. Note that a finite number of small inputs (up to some fixed constant size) may require more time than $\lceil \log n \rceil^k$. Such small inputs can however be dealt with separately since each finite structure can be defined by an index logic sentence. Hence we do not consider them here. By using the order relation $\leq^{\mathbf{A}}$ of the structure $\mathbf{A}$ we can define the lexicographic order $\leq^{\mathbf{A}}_k$ for the k-tuples in $Num(\mathbf{A})^k$. We use this order to model time and positions of the tape heads of M. This is possible since the number of k-tuples in $Num(\mathbf{A})^k$ is $\lceil \log n \rceil^k$. Expressions of the form $\bar{t} \triangleright t'$ denote that $val(\bar{t})$ is the $(val(t') + 1)$-th tuple in the order $\leq^{\mathbf{A}}_k$. This is clearly expressible in index logic since it is a polynomial time property on the $\mathbf{n}$ sort.

Adapting the construction used by Immerman and Vardi [5,6] to capture P we use the following relations to encode the configurations of polylogarithmic time direct-access Turing machines.

- A k-ary relation S_q for every state $q \in Q$ such that $S_q(\bar{t})$ holds iff M is in state q at time $\bar{t}$.
- $2k$-ary relations $T^0_i, T^1_i, T^\sqcup_i$ for every tape $i = 1, \dots, m$ such that $T^s_i(\bar{p}, \bar{t})$ holds iff at the time $\bar{t}$ the cell $\bar{p}$ of the tape i contains the symbol s.
- $2k$-ary relations H_i for every tape $i = 1, \dots, m$ such that $H_i(\bar{p}, \bar{t})$ holds iff at the time $\bar{t}$ the head of the tape i is on the cell $\bar{p}$.

We show that these relations are definable in index logic by means of a simultaneous inflationary fixed point formula. The following sentence of index logic is satisfied by a structure $\mathbf{A}$ iff $\mathbf{A} \in \mathcal{C}$. Note that the simultaneous fixed point operator in the formula reconstructs step-by-step the computation of M for the given input.

$$\exists \mathrm{x}_0 \dots \mathrm{x}_{k-1} \big([\text{S-IFP}_{\bar{t}, S_{q_a}, \mathrm{A}, \mathrm{B}_1, \mathrm{B}_2, \mathrm{B}_3, \mathrm{C}}\, \varphi_{q_a}, \Phi_\mathrm{A}, \Phi_{\mathrm{B}_1}, \Phi_{\mathrm{B}_2}, \Phi_{\mathrm{B}_3}, \Phi_\mathrm{C}](\mathrm{x}_0, \dots, \mathrm{x}_{k-1}) \big)$$

where

$$\mathrm{A} = \bar{t}, S_{q_0}, \dots, \bar{t}, S_{q_{a-1}} \quad \mathrm{B}_1 = \bar{p}\,\bar{t}, T^0_1, \dots, \bar{p}\,\bar{t}, T^0_m \quad \mathrm{B}_2 = \bar{p}\,\bar{t}, T^1_1, \dots, \bar{p}\,\bar{t}, T^1_m$$

$$\mathrm{B}_3 = \bar{p}\,\bar{t}, T^\sqcup_1, \dots, \bar{p}\,\bar{t}, T^\sqcup_m \quad \mathrm{C} = \bar{p}\,\bar{t}, H_1, \dots, \bar{p}\,\bar{t}, H_m$$

$$\Phi_\mathrm{A} = \varphi_{q_0}, \dots, \varphi_{q_{a-1}} \quad \Phi_{\mathrm{B}_1} = \psi_{01}, \dots, \psi_{0m} \quad \Phi_{\mathrm{B}_2} = \psi_{11}, \dots, \psi_{1m}$$

$$\Phi_{\mathrm{B}_3} = \psi_{\sqcup 1}, \dots, \psi_{\sqcup m} \quad \Phi_\mathrm{C} = \gamma_1, \dots, \gamma_m.$$

Here $\bar{p}$ and $\bar{t}$ denote k-tuples of variables of sort $\mathbf{n}$.

The formula builds the required relations S_{q_i}, T_i^0, T_i^1, $T_i^{\sqcup}$ and H_i (for $1 \leq i \leq m$) in stages, where the j-th stage represents the configuration at time steps up to $j-1$. The subformulae φ_{q_i}, ψ_{0i}, ψ_{1i}, $\psi_{\sqcup i}$ and γ_i define S_{q_i}, T_i^0, T_i^1, $T_i^{\sqcup}$ and H_i, respectively.

To simplify the presentation of the subformulae we assume w.l.o.g. that in every non-initial state of a computation each address-tape only contains a single binary number between 0 and $n-1$. This number has at most $\lceil \log n \rceil$ bits. Hence we encode positions of address-tapes (and function value-tapes) with a single variable of sort $\mathbf{n}$ (instead of a tuple of variables).

The formulae φ_{q_i}, ψ_{0i}, ψ_{1i}, $\psi_{\sqcup i}$ and γ_i are defined according to M. Next we define ψ_{0i} in detail. ψ_{1i} and $\psi_{\sqcup i}$ can be defined in a similar way. The intuition is that the formulae describe both the initial configuration of the computation and how each subsequent configuration is computed from the previous one in the sequence. The formula $\psi_{0i}(\bar{p}, \bar{t})$ for instance defines whether the i-th tape at cell position $\bar{p}$ at time $\bar{t}$ contains the symbol 0. If i is an address-tape or an ordinary work-tape then in the initial configuration of the computation the tape i contains the blank symbol $\sqcup$ in all its cells. In this case the formula ψ_{0i} is of the form $\neg(\bar{t} \rhd 0) \wedge \alpha_i^0(\bar{p}, \bar{t}-1)$ where $\alpha_i^0(\bar{p}, \bar{t}-1)$ list conditions under which at the following time instant $\bar{t}$ the position $\bar{p}$ of the tape i contains 0. In the general case the formula has the form

$$(\bar{t} \rhd 0 \wedge \xi_{T_i^0}) \vee (\neg(\bar{t} \rhd 0) \wedge \alpha_i^0(\bar{p}, \bar{t}-1))$$

where $\xi_{T_i^0}$ is used to define the initial configuration related to the relation T_i^0.

Assume i denotes an address-tape or an ordinary work-tape. We define $\alpha_i^0(\bar{p}, \bar{t}-1)$ as a disjunction over all cases for tape i to have a 0 in position $\bar{p}$ at time $\bar{t}$. There are two possibilities: (a) at time $\bar{t}-1$ the head of tape i is not in position $\bar{p}$ and position $\bar{p}$ already has a 0, (b) at time $\bar{t}-1$ the head of tape i is in position $\bar{p}$ and writes a 0. Let $\tau^R_{l,1}, \ldots, \tau^R_{l,r_l}$ denote the r_l address-tapes corresponding to the r_l-ary relation R_l. Let $check(R_l(x_1, \ldots, x_{r_l}), b_l)$ be $R_l(x_1, \ldots, x_{r_l})$ or $\neg R_l(x_1, \ldots, x_{r_l})$ depending on whether $b_l = 1$ or $b_l = 0$, respectively. Let $\bar{p} = \bar{p}_i$. The disjunct of $\alpha_i^0(\bar{p}, \bar{t}-1)$ corresponding to case (b) for a transition of the form $\delta_i(q, a_1, \ldots, a_m, b_1, \ldots, b_p) = (0, \rightarrow)$ is as follows.

At time $\bar{t}-1$ M is in the state q and the head of the tape j is in position $\bar{p}_j$ reading a_j.

At time $\bar{t}-1$ the tuple of values in the address-tapes of R_l is in $R^{\mathbf{A}}$ iff $b_l = 1$.

$$\exists \bar{p}_1 \ldots \bar{p}_{i-1} \bar{p}_{i+1} \ldots \bar{p}_m \Big(S_q(\bar{t}-1) \wedge \big(\bigwedge_{1 \leq j \leq m} H_j(\bar{p}_j, \bar{t}-1) \wedge T_j^{a_j}(\bar{p}_j, \bar{t}-1) \big) \wedge$$
$$\bigwedge_{1 \leq l \leq p} \exists x_1 \ldots x_{r_l} \big(check(R_l(x_1, \ldots, x_{r_l}), b_l) \wedge \bigwedge_{1 \leq k \leq r_l} x_k = index\{\mathrm{x} \mid (T^1_{\tau^R_{l,k}}(\mathrm{x}, \bar{t}-1))\}\big)\Big),$$

Assume i denotes a value-tape of a function f_j of arity k_j. Let $\tau^f_{j,1}, \ldots, \tau^f_{j,k_j}$ refer to its address-tapes. Then $\psi_{0i}(p, \bar{t})$ can be defined as follows.

$$\exists x_1 \ldots x_{k_j} \Big(\big(\bigwedge_{1 \leq l \leq k_j} x_l = index\{\mathrm{x} \mid T^1_{\tau^f_{j,l}}(\mathrm{x}, \bar{t})\}\big) \wedge \neg \mathrm{BIT}(f_j(x_1, \ldots, x_{k_j}), p)\Big),$$

Here $\mathrm{BIT}(f_j(x_1, \ldots, x_{k_j}), p)$ expresses that the bit of position p of $f_j(x_1, \ldots, x_{k_j})$ in binary is 1. As shown in Section 5.1 this is definable in index logic. Note that the content of the value-tape of a function at time $\bar{t}$ depends only on the contents of its address-tapes at time $\bar{t}$. This however does not result in a circular definition since the contents of such address-tapes at time $\bar{t}$ are all defined based only in the configuration of the machine at time $\bar{t}-1$.

We let φ_{q_0} be $\bar{t} \rhd 0 \vee (\neg(\bar{t} \rhd 0) \wedge \alpha_{q_0}(\bar{t}-1))$. For $q \neq q_0$ we let φ_q be $\neg(\bar{t} \rhd 0) \wedge \alpha_q(\bar{t}-1)$ where $\alpha_q(\bar{t}-1)$ list conditions under which M will enter state q at time $\bar{t}$.

Finally, we define γ_i as follows.

$$(\bar{t} \rhd 0 \wedge \bar{p} \rhd 0) \vee \big(\neg(\bar{t} \rhd 0) \wedge \alpha_i(\bar{p}, \bar{t}-1)\big)$$

Here $\alpha_i(\bar{p}, \bar{t}-1)$ describe the conditions for the head of tape i to be in position $\bar{p}$ at time $\bar{t}$.

We omit the remaining subformulae since it should be sufficiently clear at this point that they can indeed be written as index logic formulae. It is also not difficult to see that in the j-th stage of the simultaneous inflationary fixed point computation, the relations S_q, $(T_i^0, T_i^1, T_i^{\sqcup})_{1 \leq i \leq m}$ and $(H_i)_{1 \leq i \leq m}$ encode the configuration of M at every time $\leq j-1$. This completes our proof. □

As pointed out in [32] among others, from the point of view of applications such as database queries it is also desirable that there is a computable function that associates with every sentence φ of the logic a machine M such that M decides φ within the required time bound (PolylogTime in our case). The constructive proof of Theorem 8 directly implies that such a computable function exists for the index logic.

6. Definability in deterministic PolylogTime

We observe here that very simple properties of structures are not definable 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 **v**) in our logic. Of course, the semantics of index terms only makes sense on ordered structures. However that does not mean that formulae that do not mention the order predicate are useless. For example, the following formula expresses that the cardinality of the domain is a power of two:

$$\exists x(x = index\{\mathrm{x} : true\})$$

Index terms are based on sets of bit positions which can be compared as binary numbers. Hence, it is reasonable to consider the logic with the index terms, but without the built-in order predicate available, and ask whether this actually results in a strictly weaker logic. We prove that in the presence of constant or function symbols this is indeed the case.

We start with an inexpressibility result which shows that checking emptiness (or non-emptiness) of a unary relation is not decidable in PolylogTime and thus not expressible in index logic.

Proposition 9. *Let $\mathcal{C}$ be the class of ordered $\{\leq, P\}$-structures that interprets the unary relation symbol P as the empty set. The language $\mathcal{L} = \{\mathrm{bin}(\mathbf{A}) : \mathbf{A} \in \mathcal{C}\}$ is* not *decidable in* PolylogTime.

Proof. For a contradiction, assume that $\mathcal{L}$ is decidable in PolylogTime. Consider ordered first-order structures over the vocabulary $\{\leq, P\}$, where P is a unary relation symbol. Let M be some random-access Turing machine that given a binary encoding of a $\{\leq, P\}$-structure $\mathbf{A}$ decides in PolylogTime 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 $\{\leq, P\}$-structure with domain $\{0, \ldots, n-1\}$ where $P^{\mathbf{A}_\emptyset} = \emptyset$. The encoding of $\mathbf{A}_\emptyset$ to the Turing machine M is the sequence $s := \mathrm{bin}(\leq^{\mathbf{A}})\underbrace{0\ldots0}_{n\text{ times}}$. Note that the running time of M with input s is strictly less than n. This means that there must exist an index $i \geq n^2$ of s that was not read in the computation $M(s)$. Define

$$s' := \mathrm{bin}(\leq^{\mathbf{A}})\underbrace{0\ldots0}_{i\text{ times}}1\underbrace{0\ldots0}_{n-i-1\text{ times}}.$$

Clearly the output of the computations $M(s)$ and $M(s')$ are identical, which is a contradiction since s' is an encoding of a $\{\leq, P\}$-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. These kinds of lower bounds are well known by researchers working in the area of sublinear time algorithms [33].

We can develop this technique further to show that the order predicate on terms of sort **v** is a primitive in the logic. The proof of the following lemma is quite a bit more complicated though.

Lemma 10. *Let P and Q be unary relation symbols. There does not exist an index logic formula φ such that for all ordered $\{\leq, P, Q\}$-structures $\mathbf{A}$ such that $P^{\mathbf{A}}$ and $Q^{\mathbf{A}}$ are disjoint singleton sets $\{l\}$ and $\{m\}$, respectively, it holds that*

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

Proof. We will show that the property described above cannot be decided in PolylogTime; the claim then follows from Theorem 8. For a contradiction, suppose that the property can be decided in PolylogTime, and let M and $f : \mathbb{N} \to \mathbb{N}$ be the related random-access Turing machine and polylogarithmic function, respectively, such that, for all ordered $\{\leq, P, Q\}$-structures $\mathbf{A}$ that satisfy the conditions of the claim, $M(\mathrm{bin}(\mathbf{A}))$ decides the property in at most $f(|\mathrm{bin}(\mathbf{A})|)$ steps. Let k be a natural number such that $f(2k) < k - 1$.

Consider a computation $M(s)$ of M with an input string s. We say that an index i is *inspected* in the computation, if at some point during the computation i is written in the index tape in binary. Let $\mathrm{Ins}_M(s)$ denote the set of inspected indices of the computation of $M(s)$ and $\mathrm{Ins}^j_M(s)$ denote the set of inspected indices during the first j steps of the computation. We say that s and t are *M-j-equivalent* if the lengths of t and s are equal and $t[i] = s[i]$, for each $i \in \mathrm{Ins}^j_M(s)$. We say that $\mathbf{A}$ and $\mathbf{B}$ are *M-j-equivalent* whenever $\mathrm{bin}(\mathbf{A})$ and $\mathrm{bin}(\mathbf{B})$ are. Note that if two structures $\mathbf{A}$ and $\mathbf{B}$ are M-j-equivalent, then the computations $M(\mathrm{bin}(\mathbf{A}))$ and $M(\mathrm{bin}(\mathbf{B}))$ are at the same configuration after j steps of computation. Hence if $\mathbf{A}$ and $\mathbf{B}$ are M-$f(|\mathrm{bin}(\mathbf{A})|)$-equivalent, then outputs of $M(\mathbf{A})$ and $M(\mathbf{B})$ are identical.

Let $\mathfrak{C}$ be the class of all ordered $\{\leq, P, Q\}$-structures $\mathbf{A}$ of domain $\{0, \ldots k-1\}$, for which $P^{\mathbf{A}}$ and $Q^{\mathbf{A}}$ are disjoint singleton sets. The encodings of these structures are bit strings of the form $\mathrm{bin}(\leq^{\mathbf{A}})b_1 \ldots b_k c_1 \ldots c_k$, where exactly one b_i and one c_j, $i \neq j$, is 1. The computation of $M(\mathrm{bin}(\mathbf{A}))$ takes at most $f(2k)$ steps.

We will next construct a subclass $\mathfrak{C}^*$ of $\mathfrak{C}$ that consists of exactly those structures $\mathbf{A}$ in $\mathfrak{C}$ for which the indices $i \geq 2^n$ that are in $\mathrm{Ins}(\mathrm{bin}(\mathbf{A}))$ hold only the bit 0. We present an inductive process that will in the end produce $\mathfrak{C}^*$. Each step i of this process produces a subclass $\mathfrak{C}_i$ of $\mathfrak{C}$ for which the following hold:

a) The structures in $\mathfrak{C}_i$ are M-i-equivalent.
b) There exists $\mathbf{A}_i \in \mathfrak{C}_i$ and

$$\mathfrak{C}_i = \{\mathbf{B} \in \mathfrak{C} \mid \forall j \in \mathrm{Ins}^i(\mathrm{bin}(\mathbf{A}_i)),\ \text{if } j \geq 2^n, \text{the } j\text{th bit of } \mathrm{bin}(\mathbf{B}) \text{ is } 0\}.$$

Define $\mathfrak{C}_0 := \mathfrak{C}$; clearly $\mathfrak{C}_0$ satisfies the properties above. For $i < f(2k)$, we define $\mathfrak{C}_{i+1}$ to be the subclass of $\mathfrak{C}_i$ consisting of those structures $\mathbf{A}$ that on time step $i+1$ inspects an index that holds the bit 0, or that inspects an index that is at most $2^n - 1$.[4]

Assume that a) and b) hold for $\mathfrak{C}_i$, we will show that the same holds for $\mathfrak{C}_{i+1}$. Proof of a): Let $\mathbf{A}, \mathbf{B} \in \mathfrak{C}_{i+1}$. By construction and by the induction hypothesis, $\mathbf{A}$ and $\mathbf{B}$ are M-i-equivalent, and on step $i+1$ $M(\mathrm{bin}(\mathbf{A}))$ and $M(\mathrm{bin}(\mathbf{B}))$ inspect the same index, and if the index is at least 2^n, it holds 0. Remember that the first 2^n indices of both $\mathbf{A}$ and $\mathbf{B}$ hold the same string (namely $\mathrm{bin}(\leq^{\mathbf{A}})$). Thus $\mathbf{A}$ and $\mathbf{B}$ are M-$(i+1)$-equivalent. Proof of b): It suffices to show that $\mathfrak{C}_{i+1}$ is nonempty; the claim then follows by construction and the property b) of $\mathfrak{C}_i$. By the induction hypothesis, there is a structure $\mathbf{A}_i \in \mathfrak{C}_i$. Let j be the index that $M(\mathrm{bin}(\mathbf{A}_i))$ inspects on step $i+1$. If $j < 2^n$ then $\mathbf{A}_i \in \mathfrak{C}_{i+1}$. Suppose $j \geq 2^n$. Since $i+1 \leq f(2k) < k-1$, there exists a structure $\mathbf{A}'_i \in \mathfrak{C}_i$ such that the jth bit of $\mathrm{bin}(\mathbf{A}'_i)$ is 0. Clearly $\mathbf{A}'_i \in \mathfrak{C}_{i+1}$.

Consider the class $\mathfrak{C}_{k-2}$ (this will be our $\mathfrak{C}^*$) and $\mathbf{B} \in \mathfrak{C}_{k-2}$ and recall that $\mathrm{bin}(\mathbf{B})$ is of the form $\mathrm{bin}(\leq^{\mathbf{A}})b_1 \ldots b_k c_1 \ldots c_k$. Since $|\mathrm{Ins}^{k-2}(\mathbf{B})| \leq k-2$, there exists two distinct indices i and j, $2^n \leq i < j < 2^n + k$, such that $i, j, i+k, j+k \notin \mathrm{Ins}^{k-2}(\mathrm{bin}(\mathbf{A}))$. Let $\mathbf{B}_{P<Q}$ denote the structure such that $\mathrm{bin}(\mathbf{B}_{P<Q})$ is a bit string with prefix $\mathrm{bin}(\leq^{\mathbf{A}})$, and where the ith and $j+k$th bits are 1 and all other bits are 0. Similarly, let $\mathbf{B}_{Q<P}$ denote the structure such that $\mathrm{bin}(\mathbf{B}_{Q<P})$ is a bit string with prefix $\mathrm{bin}(\leq^{\mathbf{A}})$, and where the jth and $i+k$th bits are 1 and all other bits are 0. Clearly the structures $\mathbf{B}_{P<Q}$ and $\mathbf{B}_{Q<P}$ are in $\mathfrak{C}_{k-2}$ and M-$(k-2)$-equivalent. Since $(k-2)$ bounds above the length of computations of $M(\mathrm{bin}(\mathbf{B}_{P<Q}))$ and $M(\mathrm{bin}(\mathbf{B}_{Q<P}))$, it follows that the outputs of the computations are identical. This is a contradiction, for $\mathbf{B}_{P<Q}$ and $\mathbf{B}_{Q<P}$ are such that M should accept the first and reject the second. □

We are now in a position to show, as announced, that the order predicate of sort $\mathbf{v}$ is primitive.

Theorem 11. *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 to the formula $c \leq d$.*

Proof. For the sake of a contradiction, assume that φ is a formula as stated in the theorem. We will derive a contradiction with Lemma 10. Without loss of generality, we may assume that the only symbols of σ that occur in φ are c and d, and that φ is a sentence (i.e., φ has no free variables).

We define the translation φ^* of φ inductively. In addition to the cases below, we also have the cases where the roles of c and d are swapped.

- For ψ that does not include c or d, let $\psi^* := \psi$.
- For ψ of the form $(\alpha_1 \wedge \alpha_2)$, let $\psi^* := (\alpha_1^* \wedge \alpha_2^*)$.
- For ψ of the form $(\neg\alpha)$, let $\psi^* := (\neg\alpha^*)$.
- For ψ of the form $(\exists \mathrm{x}\alpha)$, let $\psi^* := (\exists \mathrm{x}\alpha^*)$
- For ψ of the form $(\exists x(x = \mathit{index}\{\mathrm{x} : \alpha(\mathrm{x})\} \wedge \varphi))$, let

$$\psi^* = (\exists x(x = \mathit{index}\{\mathrm{x} : \alpha(\mathrm{x})\} \wedge \varphi)^*)$$

- For ψ of the form $\left[\mathrm{IFP}_{\bar{\mathrm{x}},X}\theta\right]\bar{y}$, let $\psi^* := \left[\mathrm{IFP}_{\bar{\mathrm{x}},X}\theta^*\right]\bar{y}$.
- For ψ of the form $c = d$, let $\psi^* := \bot$.[5]
- For ψ of the form $c = x$ or $x = c$, let $\psi^* := C(x)$.
- For ψ of the form $x = \mathit{index}\{\mathrm{x} : \theta(\mathrm{x})\}$, define ψ^* as $x = \mathit{index}\{\mathrm{x} : \theta^*(\mathrm{x})\}$.
- For ψ of the form $c = \mathit{index}\{\mathrm{x} : \theta(\mathrm{x})\}$, let

  $$\psi^* := \exists z(z = \mathit{index}\{\mathrm{x} : \theta^*(\mathrm{x})\} \wedge C(z)),$$

  where z is a fresh variable.

[4] If the machine already halted on an earlier time step t, we stipulate that the machine inspects on time step $i+1$ the same index that it inspected on time step t.
[5] By $\bot$ we denote some formula that is always false, e.g., $\exists \mathrm{x}\, \mathrm{x} \neq \mathrm{x}$.

If $\mathbf{A}$ is a $\{\leq, C, D\}$-structure such that $C^{\mathbf{A}}$ and $D^{\mathbf{A}}$ are disjoint singleton sets, we denote by $\mathbf{A}'$ the $\{\leq, c, d\}$-structure with the same domain such that $\{c^{\mathbf{A}'}\} = C^{\mathbf{A}}$ and $\{d^{\mathbf{A}'}\} = D^{\mathbf{A}}$. We claim that for every $\{\leq, C, D\}$-structure $\mathbf{A}$ such that $C^{\mathbf{A}}$ and $D^{\mathbf{A}}$ are disjoint singleton sets $\{l\}$ and $\{m\}$ and every valuation *val* the following holds:

$$l < m \quad \Leftrightarrow \quad c^{\mathbf{A}'} < d^{\mathbf{A}'} \quad \Leftrightarrow \quad \mathbf{A}', val \models \varphi \quad \Leftrightarrow \quad \mathbf{A}, val \models \varphi^*.$$

This is a contradiction with Lemma 10. It suffices to prove the last equivalence as the first two are reformulations of our assumptions. The proof is by induction on the structure of φ. The cases that do not involve the constants c and d are immediate. Note that by assumption, $c^{\mathbf{A}}$ and $d^{\mathbf{A}}$ are never equal and thus the subformula $c = d$ is equivalent to $\bot$. The case $c = x$ is also easy:

$$\mathbf{A}', val \models c = x \quad \Leftrightarrow \quad val(x) = c^{\mathbf{A}'} \quad \Leftrightarrow \quad val(x) \in C^{\mathbf{A}} \quad \Leftrightarrow \quad \mathbf{A}, val \models C(x).$$

The case for $c = index\{x : \theta(x)\}$ is similar:

$$\begin{aligned} \mathbf{A}', val \models c = index\{x : \theta(x)\} \quad &\Leftrightarrow \quad \mathbf{A}', val \models \exists z(z = index\{x : \theta(x)\} \wedge c = z) \\ &\Leftrightarrow \quad \mathbf{A}, val \models \exists z(z = index\{x : \theta(x)\} \wedge C(z)). \end{aligned}$$

All other cases follow similarly. □

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.

Theorem 12. *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 $\mathbf{v}$.*

Proof. We will define the translation φ' of φ inductively. Without loss of generality, we may assume that each variable that occurs in φ is quantified exactly once (for this purpose, we stipulate that the variable $\mathtt{x}$ is quantified by the term $index\{\mathtt{x} : \alpha(\mathtt{x})\}$). For every variable x of sort $\mathbf{v}$ that occurs in φ, let $\alpha_x(\mathtt{x})$ denote the unique subformula such that $\exists x(x = index\{\mathtt{x} : \alpha_x(\mathtt{x})\} \wedge \psi)$ is a subformula of φ for some ψ. Note that $\mathtt{x}$ occurs only in $index\{\mathtt{x} : \alpha_x(\mathtt{x})\}$. We define the following shorthands for variables $\mathtt{x}$ and $\mathtt{y}$ of sort $\mathbf{n}$:

$$\begin{aligned} &\varphi_{\mathtt{x}=\mathtt{y}}(\psi(\mathtt{x}), \theta(\mathtt{y})) := \forall \mathtt{z}\big(\psi(\mathtt{z}/\mathtt{x}) \leftrightarrow \theta(\mathtt{z}/\mathtt{y})\big), \\ &\varphi_{\mathtt{x}<\mathtt{y}}(\psi(\mathtt{x}), \theta(\mathtt{y})) := \exists \mathtt{z}\Big(\big(\neg\psi(\mathtt{z}/\mathtt{x}) \wedge \theta(\mathtt{z}/\mathtt{y})\big) \wedge \\ &\qquad\qquad \forall \mathtt{z}'\Big(\mathtt{z} < \mathtt{z}' \rightarrow \big(\psi(\mathtt{z}'/\mathtt{x}) \leftrightarrow \theta(\mathtt{z}'/\mathtt{y})\big)\Big)\Big), \end{aligned}$$

where $\mathtt{z}$ and $\mathtt{z}'$ are fresh distinct variables of sort $\mathbf{n}$. In the formulae above, $\psi(\mathtt{z}/\mathtt{x})$ denotes the formula that is obtained from ψ by substituting each free occurrence of $\mathtt{x}$ in ψ by $\mathtt{z}$. The translation $\varphi \mapsto \varphi'$ is defined as follows:

- For formulae that do not include variables of sort $\mathbf{v}$ as well as for formulae of the form $R(x_1, \ldots, x_r)$ where R is an r-ary relation symbol, the translation is the identity.
- For ψ of the form $(\alpha_1 \wedge \alpha_2)$, let $\psi' := (\alpha_1' \wedge \alpha_2')$.
- For ψ of the form $\neg\alpha$, let $\psi' := \neg\alpha'$.
- For ψ of the form $\exists \mathtt{x}\alpha$, let $\psi' := \exists \mathtt{x}\alpha'$
- For ψ of the form $\big[\mathrm{IFP}_{\bar{\mathtt{x}},X}\theta\big]\bar{y}$, let $\psi' := \big[\mathrm{IFP}_{\bar{\mathtt{x}},X}\theta'\big]\bar{y}$.
- For ψ of the form $x \leq y$, let

$$\psi' := \Big(\varphi_{\mathtt{x}=\mathtt{y}}\big(\alpha_x(\mathtt{x}), \alpha_y(\mathtt{y})\big) \vee \varphi_{\mathtt{x}<\mathtt{y}}\big(\alpha_x(\mathtt{x}), \alpha_y(\mathtt{y})\big)\Big)'.$$

- For ψ of the form $x = index\{\mathtt{y} : \theta(\mathtt{y})\}$, define $\psi' := x = index\{\mathtt{y} : \theta'(\mathtt{y})\}$.
- For ψ of the form $\exists x(x = index\{\mathtt{x} : \alpha(\mathtt{x})\} \wedge \theta)$, define

$$\psi' := \exists x((x = index\{\mathtt{x} : \alpha(\mathtt{x})\})' \wedge \theta').$$

To see that the translation is well-defined and always produces a sentence of index logic, let $x_1, \ldots, x_n$ be a list of all variables of sort $\mathbf{v}$ that occur in a given index logic sentence φ such that, if x_i is quantified before x_j in φ then $i < j$. We now show that the case $(x \leq y)'$ does not lead to a cycle and that the translation terminates. All other cases are clear. Recall that

$$(x_i \leq x_j)' = \Big(\varphi_{\mathtt{x_i}=\mathtt{x_j}}\big(\alpha_{x_i}(\mathtt{x_i}), \alpha_{x_j}(\mathtt{x_j})\big) \vee \varphi_{\mathtt{x_i}<\mathtt{x_j}}\big(\alpha_{x_i}(\mathtt{x_i}), \alpha_{x_j}(\mathtt{x_j})\big)\Big)'.$$

Since φ is a sentence, it follows from the index logic syntax that the only variables of sort **v** that may occur in $\varphi_{\mathtt{x_i}=\mathtt{x_j}}\big(\alpha_{x_i}(\mathtt{x_i}), \alpha_{x_j}(\mathtt{x_j})\big) \vee \varphi_{\mathtt{x_i}<\mathtt{x_j}}\big(\alpha_{x_i}(\mathtt{x_i}), \alpha_{x_j}(\mathtt{x_j})\big)$ are $x_1, \ldots, x_{\max\{i,j\}-1}$. Hence, while the translation $(x_i \leq x_j)'$ might introduce additional occurrences of $\leq$, the variables in subformulae of the form $x_l \leq x_{l'}$ are such that $l, l' < \max\{i, j\}$. Hence the translation is well-defined.

By a straightforward inductive argument it can be verified that the translation preserves equivalence. $\square$

7. Index logic with partial fixed points

In this section we introduce a variant of index logic defined in Section 5. This logic, which we denote as IL(PFP), is defined by simply replacing the inflationary fixed point operator IFP in the definition of index logic by the partial fixed point operator PFP. We stick to the standard semantics of the PFP operator. We define that

$$\mathbf{A}, val \models [\mathrm{PFP}_{\bar{\mathtt{x}},X}\varphi]\bar{\mathtt{y}} \text{ iff } val(\bar{\mathtt{y}}) \in \mathrm{pfp}(F^{\mathbf{A},val}_{\varphi,\bar{\mathtt{x}},X}),$$

where $\mathrm{pfp}(F^{\mathbf{A},val}_{\varphi,\bar{\mathtt{x}},X})$ denotes the *partial* fixed point of the operator $F^{\mathbf{A},val}_{\varphi,\bar{\mathtt{x}},X}$ (see the description above Definition 6). The *partial fixed point* $\mathrm{pfp}(F)$ of an operator $F : \mathcal{P}(B) \to \mathcal{P}(B)$ is defined as the fixed point of F obtained from the sequence $(S^i)_{i\in\mathbb{N}}$, where $S^0 := \emptyset$ and $S^{i+1} := F(S^i)$, if such a fixed point exists. If it does not exist then $\mathrm{pfp}(F) := \emptyset$.

It is well known that first-order logic extended with partial fixed point operators captures PSPACE. As a counterpart for this result we show that IL(PFP) captures the complexity class polylogarithmic space (PolylogSpace). Recall that in IL(PFP) the relation variables bounded by the PFP operators range over (tuples of) *Num*(**A**), where **A** is the interpreting structure. Thus, the maximum number of iterations before reaching a fixed point (or concluding that it does not exist) is *not* exponential in the size n of **A** as in FO(PFP). It is instead *quasi-polynomial*, i.e., of size $O(2^{\log^k n})$ for some constant k. This observation is the main reason why IL(PFP) characterizes PolylogSpace. Finally, an analogous argument to the one that proves the well-known relationship PSPACE $\subseteq$ DTIME$(2^{n^{O(1)}})$ proves that PolylogSpace $\subseteq$ DTIME$(2^{\log^{O(1)} n})$.

7.1. *The Complexity Class* PolylogSpace

Let $L(M)$ denote the class of structures of a given vocabulary σ accepted by a direct-access Turing machine M. We say that $L(M) \in \mathrm{DSPACE}[f(n)]$ if M visits at most $O(f(n))$ cells in each work-tape before accepting or rejecting an input structure whose domain is of size n. We define the class of all languages decidable by a deterministic direct-access Turing machines in *polylogarithmic space* as follows:

$$\mathrm{PolylogSpace} := \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}[(\lceil \log n \rceil)^k].$$

Note that it is equivalent whether we define the class PolylogSpace by means of direct-access Turing machines or random-access Turing machines. Indeed, from Theorem 3 and the fact that the (standard) binary encoding of a structure **A** is of size polynomial with respect to the cardinality of its domain A, the following corollary is immediate.

Corollary 13. *A class of finite ordered structures $\mathcal{C}$ of some fixed vocabulary σ is decidable by a random-access Turing machine working in* PolylogSpace *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* PolylogSpace *with respect to n, where n is the size of the domain of the input structure.*

Moreover, in the context of PolylogSpace, there is no need for random-access address-tape for the input; PolylogSpace defined with random-access Turing machines coincide with PolylogSpace defined with (standard) Turing machines that have sequential access to the input.

Proposition 14. *A class of finite ordered structures $\mathcal{C}$ of some fixed vocabulary σ is decidable by a random-access Turing machine working in* PolylogSpace *with respect to $\hat{n}$ iff $\mathcal{C}$ is decidable by a standard (sequential-access) Turing machine in* PolylogSpace *with respect to $\hat{n}$, where $\hat{n}$ is the size of the binary encoding of the input structure.*

Proof. We give the idea behind the proof; the proof itself is straightforward. We take as the definition of the standard (sequential-access) Turing machine the definition of the random-access Turing machine given in Section 3, except that we suppose a sequential-access read-only-head for the input tape and remove the address-tape.

A random-access Turing machine M_r can simulate a sequential-access Turing machine M_s directly by using its address-tape to simulate the movement of the head of the sequential-access input-tape. In the simulation, when the head of the input-tape of M_s is on the $i+1$-th cell, the address-tape of M_r holds the number i in binary and hence refers to the $i+1$-th cell of the input. When the head of the input-tape of M_s moves right, the machine M_r will increase the binary number in its address-tape by one. Similarly, when the head of the input-tape of M_s moves left, the machine M_r will decrease the binary number in its address-tape by one. A total of $\lceil \log n \rceil$ bits suffices to access any bit of an input of length n. Clearly increasing or decreasing a binary number of length at most $\lceil \log n \rceil$ by one can be done in PolylogSpace. The rest of the simulation is straightforward.

The simulation of the other direction is a bit more complicated. Each time the content of the address-tape of the random-access machine is updated, we need to calculate the corresponding position of the head of the input-tape of the sequential-access machine. This computation however can be clearly done in PolylogSpace: We use a work-tape of the sequential-access machine to mimic the address-tape of the sequential-access machine and an additional work-tape as a binary counter. After each computation step of the random-access machine, the sequential-access machine moves the head of its input tape to its leftmost cell and formats the work-tape working as a binary counter to have exactly the binary number that is written on the address-tape. Then the sequential-access machine moves the head of its input-tape right step-by-step simultaneously decreasing the binary counter by 1. Once the binary counter reaches 0, the head of the input tape is in correct position. The rest of the simulation is straightforward. $\square$

Since the function $\lceil \log n \rceil$ is *space constructible* (s.c. for short), or equivalently *proper* as called in [16], and for any two s.c. functions their product is also s.c., we get that for any $k \geq 1$ the function $(\lceil \log n \rceil)^k$ is s.c. Hence, from Savitch's theorem we get the following result.

Fact 15. *For any $k \geq 1$, it holds that* $\mathrm{NSPACE}[(\lceil \log n \rceil)^k] \subseteq \mathrm{DSPACE}[(\lceil \log n \rceil)^{2k}]$. *Thus, nondeterministic and deterministic* PolylogSpace *coincide.*

7.2. Index logic with partial fixed point operators captures PolylogSpace

To encode a configuration of polylogarithmic size, we follow a similar strategy as in Theorem 8, i.e., in the proof of the characterization of PolylogTime by IL(IFP). The difference here is that there is no reason to encode the whole history of a computation in the fixed point. At a time step t it suffices that the configuration of the machine at time step $t-1$ is encoded; hence we may drop the variables $\bar{t}$ from the fixed point formula defined on page 155. Moreover, we make a small alteration to the Turing machines so that acceptance on an input structure will correspond to the existence of a partial fixed point.

Theorem 16. *Over ordered finite structures,* IL(PFP) *captures* PolylogSpace.

Proof. The direction of the proof that argues that IL(PFP) can indeed be evaluated in PolylogSpace is straightforward. Let ψ be an IL(PFP)-sentence. We only need to show that there exists a direct-access Turing machine M_ψ that works in $O(\log^d n)$ space for some constant d and that for every structure $\mathbf{A}$ and valuation *val* it holds that $\mathbf{A} \in L(M_\psi)$ iff $\mathbf{A}, val \models \psi$. Note that in an induction on the structure of ψ, all cases except the case for the PFP operator are as in the proof of Theorem 8. Clearly if a formula can be evaluated in PolylogTime it can also be evaluated in PolylogSpace. For the case of the PFP operator (using a similar strategy as in [28]) we set a counter to $2^{\log^r n}$ using exactly $\log^r n$ cells in a work-tape, where r is the arity of the relation variable X bounded by the PFP operator. To evaluate the PFP operator, say on a formula $\varphi(\bar{x}, X)$, M will iterate evaluating φ and decrease the counter in each iteration. When the counter gets to 0, M checks whether the contents of the relation X is equal to its contents in the following cycle and whether the tuple given in the PFP application belongs to it. If both answers are positive then M accepts. Otherwise it rejects. This suffices to find the fixed point (or to conclude that it does not exist) as there are $2^{\log^r n}$ many relations of arity r with domain $\{0, \dots, \lceil \log n \rceil - 1\}$.

For the converse, let $M = (Q, \Sigma, \delta, q_0, F, \sigma)$ be an m-tape direct-access Turing machine that works in PolylogSpace. Same as in the proof of Theorem 8 we can assume w.l.o.g. that $F = \{q_a\}$ (i.e., there is only one accepting state), $|Q| = a+1$ and $Q = \{q_0, q_1, \dots, q_a\}$. We additionally assume here that once the machine reaches an accepting state, it will not change its configuration any longer. That is, all of its heads stay put and write the same symbol that they read. Note that the machine M accepts if and only if M is in the same accepting configuration during two consecutive time steps.

We build an IL(PFP)-sentence ψ_M such that for every structure $\mathbf{A}$ and valuation *val*, it holds that $\mathbf{A} \in L(M)$ iff $\mathbf{A}, val \models \psi_M$. The formula is a derivative of that of Theorem 8 and is defined using a simultaneous PFP operator. In the formula below, $S_{q_0}, \dots, S_{q_a}$ denote 0-ary relation variables that range over the values *true* and *false*. We define

$$\psi_M := [\text{S-PFP}_{S_{q_a}, \mathrm{A}, \mathrm{B}_1, \mathrm{B}_2, \mathrm{B}_3, \mathrm{C}}\, \varphi_{q_a}, \Phi_{\mathrm{A}}, \Phi_{\mathrm{B}_1}, \Phi_{\mathrm{B}_2}, \Phi_{\mathrm{B}_3}, \Phi_{\mathrm{C}}],$$

where

$$\begin{aligned}
&\mathrm{A} = S_{q_0}, \dots, S_{q_{a-1}} \quad \mathrm{B}_1 = \bar{p}, T_1^0, \dots, \bar{p}, T_m^0 \quad \mathrm{B}_2 = \bar{p}, T_1^1, \dots, \bar{p}, T_m^1 \\
&\mathrm{B}_3 = \bar{p}, T_1^{\sqcup}, \dots, \bar{p}, T_m^{\sqcup} \quad \mathrm{C} = \bar{p}, H_1, \dots, \bar{p}, H_m \\
&\Phi_{\mathrm{A}} = \varphi_{q_0}, \dots, \varphi_{q_{a-1}} \quad \Phi_{\mathrm{B}_1} = \psi_{01}, \dots, \psi_{0m} \quad \Phi_{\mathrm{B}_2} = \psi_{11}, \dots, \psi_{1m} \\
&\Phi_{\mathrm{B}_3} = \psi_{\sqcup 1}, \dots, \psi_{\sqcup m} \quad \Phi_{\mathrm{C}} = \gamma_1, \dots, \gamma_m.
\end{aligned}$$

The formulae used in the PFP operator are defined in the same way as in Theorem 8; with the following two exceptions.

1. The formulae of the form $\alpha_i^0(\bar{p}, \bar{t}-1)$ are replaced with the analogous formulae $\alpha_i^0(\bar{p})$ obtained by simply removing the variables referring to time steps.
2. Subformulae of the form $\bar{t} \rhd 0$ are replaced with $\neg S_{q_0} \wedge \ldots \wedge \neg S_{q_{a-1}}$, which are true only on the first iteration of the fixed point calculation.

Following the proof of Theorem 8 it is now easy to show that $\mathbf{A}, val \models \psi_M$ if and only if *M accepts* $\mathbf{A}$. $\square$

8. Discussion

The natural question left open by our work is to find a logic capturing PolylogTime over structures that are not necessarily ordered. Since index logic captures PolylogTime on ordered structures, this question is equivalent to finding an effective syntax for the sentences in index logic that are order-invariant. As explored in Section 6, it appears that actually very few properties of unordered structures are in fact decidable in PolylogTime. Then again, any polynomial-time numerical property of the size of the domain is clearly decidable in PolylogTime. So, there seems to be a delicate balance of weakness on the one hand, and power on the other hand, and it seems interesting to pursue this open question further.

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 [34]. 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 provide useful syntactic sugar. Since this remains in PolylogTime, it follows from our capturing result that such extension would not increase the expressive power of index logic.

CRediT authorship contribution statement

Each named author has substantially contributed to conducting the underlying research and drafting this manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the two anonymous reviewers whose comments have significantly helped to improve the manuscript.

References

[1] E. Grädel, P. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema, S. Weinstein, Finite Model Theory and Its Applications, Springer, 2007.
[2] Y. Gurevich, Toward logic tailored for computational complexity, in: M. Richter, et al. (Eds.), Computation and Proof Theory, in: Lecture Notes in Mathematics, vol. 1104, Springer-Verlag, 1984, pp. 175–216.
[3] N. Immerman, Descriptive Complexity, Springer, 1999.
[4] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: R. Karp (Ed.), Complexity of Computation, in: SIAM-AMS Proceedings, vol. 7, American Mathematical Society, 1974, pp. 43–73.
[5] N. Immerman, Relational queries computable in polynomial time, Inf. Control 68 (1986) 86–104.
[6] M. Vardi, The complexity of relational query languages, in: Proceedings 14th ACM Symposium on the Theory of Computing, 1982, pp. 137–146.
[7] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.
[8] M.Y. Vardi, The complexity of relational query languages, in: Proceedings of the 14th Annual ACM Symposium on Theory of Computing, ACM, 1982, pp. 137–146.
[9] F. Ferrarotti, S. González, J.M. Turull Torres, J. Van den Bussche, J. Virtema, Descriptive complexity of deterministic polylogarithmic time, in: Logic, Language, Information, and Computation – Proceedings of the 26th International Workshop, WoLLIC 2019, in: Lecture Notes in Computer Science, vol. 11541, Springer, 2019, pp. 208–222.
[10] M. Grohe, W. Pakusa, Descriptive complexity of linear equation systems and applications to propositional proof complexity, in: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, IEEE Computer Society, 2017, pp. 1–12.
[11] N. Immerman, Number of quantifiers is better than number of tape cells, J. Comput. Syst. Sci. 22 (3) (1981) 384–406.
[12] D.A. Mix Barrington, N. Immerman, H. Straubing, On uniformity within NC^1, J. Comput. Syst. Sci. 41 (3) (1990) 274–306.
[13] D.A. Mix Barrington, Quasipolynomial size circuit classes, in: Proceedings of the Seventh Annual Structure in Complexity Theory Conference, IEEE Computer Society, 1992, pp. 86–93.
[14] F. Ferrarotti, S. González, K. Schewe, J.M. Turull Torres, The polylog-time hierarchy captured by restricted second-order logic, in: 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE, 2018, pp. 133–140.
[15] L. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1) (1976) 1–22.
[16] C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[17] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 1979.
[18] A. Borodin, On relating time and space to size and depth, SIAM J. Comput. 6 (4) (1977) 733–744.
[19] R. Greenlaw, H.J. Hoover, W.L. Ruzzo, Limits to Parallel Computation: P-Completeness Theory, Oxford University Press, 1995.
[20] J.H. Reif, Logarithmic depth circuits for algebraic functions, SIAM J. Comput. 15 (1) (1986) 231–242.

[21] G. Matera, J.M. Turull Torres, The space complexity of elimination theory: upper bounds, in: Foundations of Computational Mathematics, Springer, 1997, pp. 267–276.
[22] A. Grosso, N. Herrera, G. Matera, M.E. Stefanoni, J.M. Turull Torres, An algorithm for the computation of the rank of integer matrices in polylogarithmic space, Electron. J. Chil. Soc. Comput. Sci. 4 (1) (2000), 45 pages, in Spanish.
[23] G. Gottlob, N. Leone, F. Scarcello, Computing LOGCFL certificates, Theor. Comput. Sci. 270 (1–2) (2002) 761–777.
[24] G. Gottlob, R. Pichler, F. Wei, Tractable database design through bounded treewidth, in: Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM, 2006, pp. 124–133.
[25] G. Gottlob, R. Pichler, F. Wei, Tractable database design and datalog abduction through bounded treewidth, Inf. Syst. 35 (3) (2010) 278–298.
[26] M. Beaudry, P. McKenzie, Circuits, matrices, and nonassociative computation, J. Comput. Syst. Sci. 50 (3) (1995) 441–455.
[27] M. Grohe, Descriptive Complexity, Canonisation, and Definable Graph Structure Theory, Cambridge University Press, 2017.
[28] H.-D. Ebbinghaus, J. Flum, Finite Model Theory, 2nd edition, Springer, 1999.
[29] L. Libkin, Elements of Finite Model Theory, Springer, 2004.
[30] Y. Gurevich, S. Shelah, Fixed-point extensions of first-order logic, Ann. Pure Appl. Log. 32 (1986) 265–280.
[31] D. Knuth, Sorting and Searching, 2nd edition, The Art of Computer Programming, vol. 3, Addison-Wesley, 1998.
[32] M. Grohe, The quest for a logic capturing PTIME, in: Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24–27 June 2008, Pittsburgh, PA, USA, IEEE Computer Society, 2008, pp. 267–271.
[33] R. Rubinfeld, A. Shapira, Sublinear time algorithms, SIAM J. Discrete Math. 25 (4) (2011) 1562–1588.
[34] E. Grandjean, F. Olive, Graph properties checkable in linear time in the number of vertices, J. Comput. Syst. Sci. 68 (2004) 546–597.