

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

^a*Software Competence Center Hagenberg, Austria*

^b*Universidad Nacional de La Matanza, Argentina*

^c*Hasselt University, Belgium*

Abstract

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.

[☆]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 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.

*Corresponding author

Email address: `flavio.ferrarotti@scch.at` (Flavio Ferrarotti)

1. Introduction

The research area known as Descriptive Complexity [1, 2, 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 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 $f(x_1, \dots, x_r)$, where x_1, \dots, 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

31 finite ordered structures by a new logic called *index logic*.

32 Index logic is a two-sorted logic whose semantics is only defined over finite
33 ordered structures. Variables of the first sort range over the domain of the input
34 structure, whereas variables of the second sort range over the initial segment of
35 the natural numbers of length $\log(n)$, where n is the size of the domain of the
36 input structure. Elements of the second sort can then be used to represent the bit
37 positions needed to address the elements of the first sort. Index logic includes full
38 fixed point logic on the second sort. Quantification over the first sort, however,
39 is heavily restricted. Specifically, a variable of the first sort can only be bound
40 using an address specified by a subformula that defines the positions of the bits
41 of the address that are set to 1. This “indexing mechanism” lends index logic
42 its name.

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

57 A challenge was to develop a logic which enables the expression of PolylogTime
58 problems in a relatively clean and natural way. The indexing mechanism in our
59 logic is a key component to meet that challenge. The alternative of using fixed

60 point operations and the BIT predicate¹ –which was very successfully used by
61 Immerman and others to characterize related sublinear complexity classes [3]–
62 to address values of the first sort leads in this case to a rather awkward logic
63 that makes it difficult to express even simple queries.

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

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

85 A preliminary version of this paper was presented at the 26th International
86 Workshop in Logic, Language, Information and Computation [9]. This is an
87 extended improved version which, in addition to the full proofs of the results on
88 deterministic polylogarithmic time reported in [9], also considers polylogarithmic

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

89 space and its descriptive characterization in terms of a variant of index logic.

90 *Related work.* Many natural fixed point computations such as transitive closure
91 converge after a polylogarithmic number of steps. This motivated the study of
92 a fragment of fixed point logic with counting (FPC) that only allows polylog-
93 arithmically many iterations of the fixed point operators (POLYLOG-FPC). As
94 shown in [10], on ordered structures POLYLOG-FPC captures NC, i.e., the class
95 of problems solvable in parallel polylogarithmic time. This holds even in the
96 absence of counting, which on ordered structures can be simulated using fixed
97 point operators. An old result in [11] directly implies that POLYLOG-FPC is
98 strictly weaker than FPC with regards to expressive power.

99 It is well known that the (nondeterministic) logarithmic time hierarchy
100 corresponds exactly to the set of first-order definable Boolean queries (see
101 Theorem 5.30 in [3]). The relationship between uniform families of circuits within
102 NC^1 and nondeterministic random-access logarithmic time machines was studied
103 in [12]. However, the study of the descriptive complexity of classes of problems
104 decidable by *deterministic* formal models of computation in polylogarithmic
105 time, i.e., the central topic of this paper, appears not to have been considered
106 by previous works.

107 On the other hand, *nondeterministic* polylogarithmic time complexity classes
108 defined in terms of alternating random-access Turing machines and related
109 families of circuits have received more attention [13, 14]. A theorem which is
110 analogous to Fagin's famous theorem [4] was recently proven for nondeterministic
111 polylogarithmic time [14]. The logical characterization was obtained using
112 a second-order logic with restricted second-order quantification ranging over
113 relations of size at most polylogarithmic in the size of the structure and first-order
114 universal quantification bounded to those relations. This latter work is closely
115 related to the work on constant depth quasi-polynomial size AND/OR circuits
116 and the corresponding restricted second-order logic in [13]. Both logics capture
117 the full alternating polylogarithmic time hierarchy. However, the additional
118 restriction in the first-order universal quantification in the second-order logic

119 defined in [14] enables a one-to-one correspondence between the levels of the
 120 polylogarithmic time hierarchy and the prenex fragments of the logic; in the
 121 style of Stockmeyer’s result [15] on the polynomial-time hierarchy. Unlike the
 122 classical results of Fagin and Stockmeyer [4, 15], the results on the descriptive
 123 complexity of nondeterministic polylogarithmic time classes only hold over
 124 ordered structures.

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

- 129 (i) $\text{PolylogSpace} \neq P$, and it is *unknown* whether $\text{PolylogSpace} \subseteq P$.
- 130 (ii) $\text{PolylogSpace} \neq \text{NP}$, and it is *unknown* whether $\text{PolylogSpace} \subseteq \text{NP}$.
- 131 (iii) Obviously: $L \subseteq \text{NL} \subseteq L^2 \subseteq \text{PolylogSpace} \subseteq \text{DTIME}[2^{(\lceil \log n \rceil)^{O(1)}}]$, the
 132 latter class being known as quasi-polynomial time (QuasiP).
- 133 (iv) For all $i \geq j \geq 1$, L^j uniform $\text{NC}^i \subseteq L^i$ (see [18]); hence we have that
 134 PolylogSpace uniform $\text{NC} \subseteq \text{PolylogSpace}$.
- 135 (v) For all $i \geq 1$, let SC^i be the class of all languages that are decidable by
 136 deterministic Turing machines whose space is bounded by $O((\log n)^i)$ and
 137 whose time is simultaneously bounded by $n^{O(1)}$. Let SC (Steve’s Class) be
 138 $\bigcup_{i \in \mathbb{N}} \text{SC}^i$ (see [19]). It follows that $\text{SC} \subseteq P \cap \text{PolylogSpace}$.

139 Some interesting natural problems in PolylogSpace follow. From (iv) we get
 140 that division, exponentiation, iterated multiplication of integers [20] and integer
 141 matrix operations such as exponentiation, computation of the determinant, rank
 142 and the characteristic polynomial (see [21] and [22] for detailed algorithms in
 143 L^2) are all in PolylogSpace. Other well-known problems in the class are k -
 144 colorability of graphs of bounded tree-width [23], primality, 3NF test, BCNF
 145 test for relational schemas of bounded tree-width [24, 25] and the circuit value
 146 problem of only EXOR gates [16]. Finally, in [26] an interesting family of

147 problems is presented. It is shown there that for every $k \geq 1$ there is an algebra
 148 $(S; +, \cdot)$ over matrices such that the depth $O(\log n)^k$ straight linear formula
 149 problem over $M(S; +, \cdot)$ is NC^{k+1} complete under L reducibility. It then follows
 150 from (iv) that these problems are in $\text{DSPACE}[(\log n)^{k+1}]$.

151 2. Preliminaries

152 In descriptive complexity it is a common practice to work only with relational
 153 structures since functions can be identified with their graphs. In a sublinear-time
 154 setting, however, this does not work. Indeed, let f be a function and denote
 155 its graph by \tilde{f} . If we want to know the value of $f(x)$ then we cannot spend
 156 the linear time required to find a y such that $\tilde{f}(x, y)$ holds. We therefore work
 157 with structures containing functions as well as relations and constants. We
 158 write $R_i^{r_i}$ and $f_i^{k_i}$ to denote relation and function symbols of arities r_i and
 159 k_i , respectively. Constant symbols are denoted by c_i . A *finite vocabulary* is a
 160 finite set of relation, function, and constant symbols. A *finite structure* \mathbf{A} of
 161 vocabulary $\sigma = \{R_1^{r_1}, \dots, R_p^{r_p}, c_1, \dots, c_q, f_1^{k_1}, \dots, f_s^{k_s}\}$ is a tuple

$$162 \quad (A, R_1^{\mathbf{A}}, \dots, R_p^{\mathbf{A}}, c_1^{\mathbf{A}}, \dots, c_q^{\mathbf{A}}, f_1^{\mathbf{A}}, \dots, f_s^{\mathbf{A}})$$

163 consisting of a finite domain A and interpretations for all relation, constant
 164 and function symbols in σ . An *interpretation* of a symbol $R_i^{r_i}$ is a relation
 165 $R_i^{\mathbf{A}} \subseteq A^{r_i}$, 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
 166 $f_i^{\mathbf{A}} : A^{k_i} \rightarrow A$. Throughout this paper \leq will denote a binary relation symbol
 167 that is always interpreted as a linear order of the given finite structure. A *finite*
 168 *ordered σ -structure* \mathbf{A} is a finite σ -structure, where the binary relation symbol
 169 $\leq \in \sigma$ and $\leq^{\mathbf{A}}$ is a linear order on A . In this paper, we consider only finite
 170 ordered structures. We emphasize that $\leq^{\mathbf{A}}$ is always the prescribed linear order
 171 of the ordered structure \mathbf{A} . Every finite ordered structure is isomorphic to one
 172 whose domain is an initial segment of the natural numbers. Thus, we assume
 173 that $A = \{0, 1, \dots, n-1\}$, where n is the cardinality $|A|$ of A .

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

176 (a_1, \dots, a_k) is frequently denoted as \bar{a} and $\bar{a}[i]$ denotes the i -th element in it.
177 Similarly, if s is a finite string then we denote by $s[i]$ the i -th letter of this string.

178 3. Deterministic polylogarithmic time

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

185 Our notion of a *random-access Turing machine* is that of a multi-tape Turing
186 machine which consists of: (1) a finite set of states, (2) a read-only random access
187 *input-tape*, (3) a sequential access *address-tape* and (4) one or more (but a fixed
188 number of) sequential access *work-tapes*. All tapes are divided into cells, are
189 equipped with a *tape head* which scans the cells and are right-end infinite. The
190 tape heads of the sequential access address-tape and work-tapes can move left
191 or right. Whenever a tape head is in the leftmost cell it is not allowed to move
192 left. The address-tape alphabet only contains symbols 0, 1 and \sqcup (for blank).
193 The position of the input-tape head is determined by the number i stored in
194 binary between the leftmost cell and the first blank cell of the address-tape (if
195 the leftmost cell is blank then i is considered to be 0) as follows: If i is strictly
196 smaller than the length n of the input string then the input-tape head is in the
197 $(i + 1)$ -th cell. Otherwise, if $i \geq n$ then the input-tape head is in the $(n + 1)$ -th
198 cell scanning the special end-marker symbol \triangleleft .

199 Formally, a *random-access Turing machine* M with k work-tapes is a five-
200 tuple $(Q, \Sigma, \delta, q_0, F)$. Here Q is a finite set of *states*; $q_0 \in Q$ is the *initial state*. Σ
201 is a finite set of symbols (the *alphabet* of M). For simplicity, we fix $\Sigma = \{0, 1, \sqcup\}$.
202 $F \subseteq Q$ is the set of *accepting final states*. The *transition* function of M is of the
203 form $\delta : Q \times (\Sigma \cup \{\triangleleft\}) \times \Sigma^{k+1} \rightarrow Q \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^{k+1}$. We assume that the
204 tape head directions (\leftarrow for “left”, \rightarrow for “right” and $-$ for “stay”) are not in

205 $Q \cup \Sigma$.

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

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

225 At the beginning of a computation all work-tapes are blank, except the
226 input-tape that contains the input string and the index-tape that contains a 0
227 (meaning that the input-tape head scans the first cell of the input-tape). Thus,
228 the *initial configuration* of M is $(q_0, \#0, \#, \dots, \#)$. A *computation* is a (possibly
229 infinite) sequence of configurations which starts with the initial configuration and
230 for every two consecutive configurations the latter is obtained by applying the
231 transition function of M to the former. If the transition function is not defined
232 for the current configuration, then the machine stops and the configuration is
233 called *final*. An input string is *accepted* if an accepting final configuration, i.e., a
234 configuration with a state belonging to F , is reached. The input is *rejected* if a
235 final configuration is reached and its state does not belong to F .

236 **Example 1.** Following a simple strategy, a random-access Turing machine M
237 can compute the length n of its input as well as $\lceil \log n \rceil$ in polylogarithmic time.
238 In its initial step M checks whether the input-tape head scans the end-marker
239 \triangleleft . If it does, then the input string is the empty string and the computation is
240 finished. Otherwise, M writes 1 in the first cell of its address tape and keeps
241 writing 0's in its subsequent cells right up until the input-tape head scans \triangleleft . It
242 then rewrites the last 0 back to the blank symbol \sqcup . At this point the resulting
243 binary string in the index-tape is of length $\lceil \log n \rceil$. Next, M moves its address-
244 tape head back to the first cell (i.e., to the only cell containing a 1 at this point).
245 From here on, M repeatedly moves the index head one step to the right. Each
246 time it checks whether the index-tape head scans a blank \sqcup or a 0. If \sqcup then M
247 is done. If 0 then it writes a 1 and tests whether the input-tape head jumps to
248 the cell with \triangleleft ; if that is the case then it rewrites a 0, otherwise it leaves the
249 1. The binary number left on the index-tape at the end of this process is $n - 1$.
250 Adding one in binary is now an easy task. \square

251 The *formal language accepted* by a machine M , denoted $L(M)$, is the set of
252 strings accepted by M . We say that $L(M) \in \text{DTIME}[f(n)]$ if M makes at most
253 $O(f(n))$ steps before accepting or rejecting an input string of length n . We define
254 the class of all formal languages decidable by (deterministic) random-access
255 Turing machines in *polylogarithmic time* as follows:

$$256 \quad \text{PolylogTime} := \bigcup_{k \in \mathbb{N}} \text{DTIME}[\log^k n]$$

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

262 When we want to give a finite ordered structure as an input to a random-access
263 Turing machine we encode it as a string adhering to the usual conventions in
264 descriptive complexity theory [3]. Let $\sigma = \{R_1^{r_1}, \dots, R_p^{r_p}, c_1, \dots, c_q, f_1^{k_1}, \dots, f_s^{k_s}\}$

265 be a vocabulary and let \mathbf{A} with $A = \{0, 1, \dots, n-1\}$ be a finite ordered structure
 266 of vocabulary σ . Note that the order $\leq^{\mathbf{A}}$ on A can be used to define an order
 267 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
 268 binary string $\text{bin}(R_i^{\mathbf{A}})$ of length n^{r_i} , where 1 in a given position m indicates that
 269 the m -th tuple of A^{r_i} is in $R_i^{\mathbf{A}}$. Likewise, each constant number $c_j^{\mathbf{A}}$ is encoded
 270 as a binary string $\text{bin}(c_j^{\mathbf{A}})$ of length $\lceil \log n \rceil$.

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

275 The encoding of the whole structure $\text{bin}(\mathbf{A})$ is the concatenation of the binary
 276 strings encoding its relations, constants and functions. The length $\hat{n} = |\text{bin}(\mathbf{A})|$
 277 of this string is $n^{r_1} + \dots + n^{r_p} + q \lceil \log n \rceil + \lceil \log n \rceil n^{k_1} + \dots + \lceil \log n \rceil n^{k_s}$, where
 278 $n = |A|$ denotes the size of the input structure \mathbf{A} . Note that $\log \hat{n} \in O(\lceil \log n \rceil)$,
 279 and hence $\text{DTIME}[\log^k \hat{n}] = \text{DTIME}[\log^k n]$.

280 4. Direct-access Turing machines

281 In this section, we propose a new model of random-access Turing machines.
 282 In the standard model reviewed above the entire input structure is assumed
 283 to be encoded as one binary string. In our new variant the different relations
 284 and functions of the structure can be accessed directly. We then show that
 285 both variants are equivalent in the sense that they lead to the same notion of
 286 PolylogTime. The direct-access model also allow us to provide a less cumbersome
 287 proof of our main capturing result.

288 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
 289 for ordered structures. A *direct-access Turing machine that takes finite ordered*
 290 *σ -structures \mathbf{A} as an input* is a multitape Turing machine with $r_1 + \dots + r_p +$
 291 $k_1 + \dots + k_s$ distinguished work-tapes (called *address-tapes*), s distinguished
 292 read-only (function) *value-tapes*, $q + 1$ distinguished read-only *constant-tapes*
 293 and one or more ordinary *work-tapes*.

294 Let us define a transition function δ_l for each tape l separately. These
 295 transition functions take as an input the current state of the machine, the bit
 296 read by each of the heads of the machine and the answer (0 or 1) to the query
 297 $(n_1, \dots, n_{r_i}) \in R_i^{\mathbf{A}}$ for each relation $R_i \in \sigma$. Here n_j denotes the number written
 298 in binary in the j th distinguished tape of R_i . If one of the n_j is too large
 299 and does not denote any domain element, we stipulate that the answer to the
 300 query $(n_1, \dots, n_{r_i}) \in R_i^{\mathbf{A}}$ is 0. Note that we do not add the aforementioned
 301 construction for the order predicate \leq ; the answer for the query $(n_1, n_2) \in \leq^{\mathbf{A}}$
 302 can be computed directly from the binary representations n_1 and n_2 .²

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

$$305 \quad \delta_Q : Q \times \Sigma^m \times \{0, 1\}^p \rightarrow Q.$$

306 If l corresponds to an address-tape or an ordinary work-tape then we have

$$307 \quad \delta_l : Q \times \Sigma^m \times \{0, 1\}^p \rightarrow \Sigma \times \{\leftarrow, \rightarrow, -\}.$$

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

$$309 \quad \delta_l : Q \times \Sigma^m \times \{0, 1\}^p \rightarrow \{\leftarrow, \rightarrow, -\}.$$

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

317 In the initial configuration, read-only constant-tapes for the constant symbols
 318 c_1, \dots, c_q hold their values in \mathbf{A} in binary. The address-tapes, value-tapes and
 319 ordinary work-tapes hold only blanks. One additional constant-tape (there

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

320 are $q + 1$ of them) holds the size n of the domain of \mathbf{A} in binary. Notice
 321 that a direct-access Turing machine cannot otherwise determine the size of the
 322 domain of a structure over a relational vocabulary, i.e., a vocabulary that does
 323 not have function symbols. On the other hand, if the vocabulary contains a
 324 function symbol, then the direct-access Turing machine can adapt the algorithm
 325 of the random-access Turing machine in Example 1 to determine the size of the
 326 domain, by checking whether the function value-tape has been blanked, instead
 327 of checking whether the input-tape head scans the end-marker \triangleleft (i.e., the address
 328 in the index-tape is beyond the end of the input-tape).

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

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

340 **Theorem 3.** *A class of finite ordered structures \mathcal{C} of some fixed vocabulary σ*
 341 *is decidable by a random-access Turing machine working in PolylogTime with*
 342 *respect to \hat{n} , where \hat{n} is the size of the binary encoding of the input structure, iff*
 343 *\mathcal{C} is decidable by a direct-access Turing machine in PolylogTime with respect to*
 344 *n , where n is the size of the domain of the input structure.*

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

350 additional work-tapes for convenience. We then encode the initial configuration
351 of M_d into the tapes of M_r :

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

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

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

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

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

$$362 \quad n^{r_1} + \dots + n^{r_p} + q \lceil \log n \rceil + \lceil \log n \rceil n^{k_1} + \dots + \lceil \log n \rceil n^{k_s}.$$

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

369 Step 2. The binary representation of a constant $c_i^{\mathbf{A}}$ is written in the input-
370 tape between $g(n)$ and $g(n) + \lceil \log n \rceil$, where g is a fixed function of the form
371 $n^{r_1} + \dots + n^{r_p} + (i - 1) \lceil \log n \rceil$. The numbers n and $g(n)$ are obtained as in case
372 1. Then $g(n)$ is written on the index tape and the next $\lceil \log n \rceil$ bits of the input
373 are copied to the tape corresponding to c_i .

374 Steps a. and b. These cases are handled similar to each other and to
375 the case 2. above. The main difference for b. is that the bits of the output are

376 not in successive positions of the input, but the location of each bit needs to be
 377 calculated separately.

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

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

$$392 \quad n^{r_1} + \dots + n^{r_p} + q \lceil \log n \rceil + \lceil \log n \rceil n^{k_1} + \dots + \lceil \log n \rceil n^{k_s}. \quad (1)$$

393 The number written in binary on the index-tape of M_r determines the position
 394 of the input that is read by M_r . From (1) we obtain fixed functions on n that
 395 we use in the simulation to check which part of the input is read when the
 396 index-tape holds a particular number. For example, if the index-tape holds
 397 $n_1^{r_1} + 1$, we can calculate that the head of the input-tape of M_r reads the bit
 398 answering the query: is $\vec{0} \in R_2^{\mathbf{A}}$. We can use an extra work-tape of M_d to always
 399 store the bit that M_r is reading from its input. The rest of the simulation is
 400 straightforward. \square

401 5. Index logic

402 In this section, we introduce a novel logic called *index logic*, which over finite
 403 ordered structures captures PolylogTime. Our definition of index logic is inspired

404 by the second-order logic in [13], where relation variables take values from the
 405 sub-domain $\{0, \dots, \lceil \log n \rceil - 1\}$ (n being the size of the interpreting structure),
 406 as well as by the well known counting logics defined in [27].

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

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

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

$$422 \quad t ::= x \mid c \mid f(t, \dots, t),$$

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

425 **Definition 5** (Syntax of index logic). Let σ be a vocabulary. The formulae of
 426 *index logic* $\mathbb{IL}(\text{IFP})$ is generated by the following grammar:

$$427 \quad \varphi ::= \mathbf{x}_1 \leq \mathbf{x}_2 \mid R(t_1, \dots, t_k) \mid X(\mathbf{x}_1, \dots, \mathbf{x}_k) \mid (\varphi \wedge \varphi) \mid \neg \varphi \mid [\text{IFP}_{\bar{\mathbf{x}}, X} \varphi] \bar{\mathbf{y}} \mid$$

$$428 \quad t = \text{index}\{\mathbf{x} : \varphi(\mathbf{x})\} \mid \exists x (x = \text{index}\{\mathbf{x} : \alpha(\mathbf{x})\} \wedge \varphi) \mid \exists \mathbf{x} \varphi,$$

429 where t, t_1, \dots, t_k are σ -terms of sort \mathbf{v} , R is a relation symbol in σ , $\mathbf{x}, \mathbf{x}_1, \dots, \mathbf{x}_k$
 430 are variables of sort \mathbf{n} , and $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ are tuples of variables of sort \mathbf{n} whose length

433 coincides with the arity of the relation variable X . Moreover, $\alpha(\mathbf{x})$ is a formula
 434 where the variable x of sort \mathbf{v} does not occur as a free variable.

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

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

448 A valuation is defined as usual for two-sorted logics. Thus, a *valuation* over
 449 a structure \mathbf{A} is any total function val from the set of all variables of index logic
 450 to values satisfying the following constraints:

- 451 • If x is a variable of sort \mathbf{v} , then $val(x) \in A$.
- 452 • If \mathbf{x} is a variable of sort \mathbf{n} , then $val(\mathbf{x}) \in Num(\mathbf{A})$.
- 453 • If X is a relation variable with arity r , then $val(X) \subseteq (Num(\mathbf{A}))^r$.

454 If χ is a variable and B a permissible value for that variable, we write
 455 $val(B/\chi)$ to denote the valuation that maps χ to B and agrees with val for all
 456 other variables. Valuations extend to terms and tuples of terms in the usual way.

457 Fixed points are defined in the standard way (see, e.g., [28] and [29] for an
 458 in-depth presentation). Given an operator $F : \mathcal{P}(B) \rightarrow \mathcal{P}(B)$, a set $S \subseteq B$ is a
 459 *fixed point* of F if $F(S) = S$. A set $S \subseteq B$ is the *least fixed point* of F if it is a
 460 fixed point and, for every other fixed point S' of F , we have $S \subseteq S'$. We denote

461 the least fixed point of F as $\text{lfp}(F)$. The *inflationary fixed point* of F , denoted
 462 by $\text{ifp}(F)$, is the union of all sets S^i where $S^0 := \emptyset$ and $S^{i+1} := S^i \cup F(S^i)$.

463 Let $\varphi(X, \bar{x})$ be a formula of vocabulary σ , where X is a relation variable
 464 of arity k and \bar{x} is a k -tuple of variables of sort \mathbf{n} . Let \mathbf{A} be a σ -structure
 465 and val a variable valuation. The formula $\varphi(X, \bar{x})$ gives rise to an operator
 466 $F_{\varphi, \bar{x}, X}^{\mathbf{A}, val} : \mathcal{P}((Num(\mathbf{A}))^k) \rightarrow \mathcal{P}((Num(\mathbf{A}))^k)$ defined as follows:

$$467 \quad F_{\varphi, \bar{x}, X}^{\mathbf{A}, val}(S) := \{\bar{a} \in (Num(\mathbf{A}))^k \mid \mathbf{A}, val(S/X, \bar{a}/\bar{x}) \models \varphi(X, \bar{x})\}.$$

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

- 472 • $\mathbf{A}, val \models \mathbf{x}_1 \leq \mathbf{x}_2$ iff $val(\mathbf{x}_1) \leq val(\mathbf{x}_2)$.
- 473 • $\mathbf{A}, val \models R(t_1, \dots, t_k)$ iff $(val(t_1), \dots, val(t_k)) \in R^{\mathbf{A}}$.
- 474 • $\mathbf{A}, val \models X(\mathbf{x}_1, \dots, \mathbf{x}_k)$ iff $(val(\mathbf{x}_1), \dots, val(\mathbf{x}_k)) \in val(X)$.
- 475 • $\mathbf{A}, val \models t = \text{index}\{\mathbf{x} : \varphi(\mathbf{x})\}$ iff $val(t)$ in binary is $b_m b_{m-1} \dots b_0$, where
 476 $m = \lceil \log |A| \rceil - 1$ and $b_j = 1$ iff $\mathbf{A}, val(j/\mathbf{x}) \models \varphi(\mathbf{x})$.
- 477 • $\mathbf{A}, val \models [\text{IFP}_{\bar{x}, X} \varphi] \bar{y}$ iff $val(\bar{y}) \in \text{ifp}(F_{\varphi, \bar{x}, X}^{\mathbf{A}, val})$.
- 478 • $\mathbf{A}, val \models \neg \varphi$ iff $\mathbf{A}, val \not\models \varphi$.
- 479 • $\mathbf{A}, val \models \varphi \wedge \psi$ iff $\mathbf{A}, val \models \varphi$ and $\mathbf{A}, val \models \psi$.
- 480 • $\mathbf{A}, val \models \exists \mathbf{x} \varphi$ iff $\mathbf{A}, val(i/\mathbf{x}) \models \varphi$, for some $i \in Num(\mathbf{A})$.
- 481 • $\mathbf{A}, val \models \exists x(x = \text{index}\{\mathbf{x} : \alpha(\mathbf{x})\} \wedge \varphi)$ iff there exists $i \in A$ such that
 482 $\mathbf{A}, val(i/x) \models x = \text{index}\{\mathbf{x} : \alpha(\mathbf{x})\}$ and $\mathbf{A}, val(i/x) \models \varphi$.

483 It immediately follows from the famous result by Gurevich and Shelah re-
 484 garding the equivalence between inflationary and least fixed points [30], that an
 485 equivalent index logic can be obtained if we (1) replace $[\text{IFP}_{\bar{x}, X} \varphi] \bar{y}$ by $[\text{LFP}_{\bar{x}, X} \varphi] \bar{y}$

486 in the formation rule for the fixed point operator in Definition 5, adding the
 487 restriction that every occurrence of X in φ is positive³, and (2) fix the interpre-
 488 tation $\mathbf{A}, val \models [\text{LFP}_{\bar{x}, X} \varphi] \bar{y}$ iff $val(\bar{y}) \in \text{lfp}(F_{\varphi, \bar{x}, X}^{\mathbf{A}, val})$.

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

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

501 Semantics for the simultaneous fixed point operator is defined such that $\mathbf{A}, val \models$
 502 $[\text{S-IFP}_{\bar{x}_0, X_0, \dots, \bar{x}_m, X_m} \varphi_0, \dots, \varphi_m] \bar{y}$ iff $val(\bar{y})$ belongs to the first (here X_0) compo-
 503 nent of the simultaneous inflationary fixed point.

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

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

509 5.1. Finding the binary representation of a term

510 Let t be a term of sort \mathbf{v} . In this example, we construct an index logic formula
 511 that expresses the well-known bit predicate $\text{BIT}(t, \mathbf{x})$. The predicate $\text{BIT}(t, \mathbf{x})$

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

512 states that the $(val(\mathbf{x}) + 1)$ -th bit of $val(t)$ in binary is set to 1. Subsequently,
 513 the sentence $t = index\{\mathbf{x} : BIT(t, \mathbf{x})\}$ is valid over the class of all finite ordered
 514 structures.

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

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

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

$$526 \quad \varphi_Z := (Z = \emptyset \wedge \mathbf{z} = M) \vee (Z \neq \emptyset \wedge \mathbf{z} = \min Z - 1),$$

$$527 \quad \varphi_Y := Z \neq \emptyset \wedge \mathbf{y} = \min Z \wedge \exists x (x = index\{\mathbf{z} : Y(\mathbf{z}) \vee \mathbf{z} = \mathbf{y}\} \wedge t \geq x).$$

529 5.2. Binary search in an array of key values

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

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

540 The simplicity of the above abstraction gives rise to two peculiarities which,
 541 however, pose no problems. First, the array cells belong to the range of the

542 function K . Thus array cells are allowed to play a double role as key values.
 543 Second, the function K is total. Thus it is also defined on the domain elements
 544 that do not correspond to array cells. We will simply ignore K on that part of
 545 the domain.

546 We still need to discuss \prec and T . We assume $\prec^{\mathbf{A}}$ to be a total order used
 547 to compare key values. Note that $\prec^{\mathbf{A}}$ can be different from the built-in order
 548 $<^{\mathbf{A}}$. For the binary search procedure to work the array needs to be sorted, i.e.,
 549 \mathbf{A} must satisfy $\forall x \forall y (x < y < N \rightarrow (K(x) \preceq K(y)))$. Finally, the constant
 550 $t := T^{\mathbf{A}}$ is the test value.

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

$$554 \quad \exists x (x < N \wedge K(x) = T). \quad (\gamma)$$

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

```

L := 0
R := N - 1
while L ≠ R do
  I := ⌊(L + R)/2⌋
  if K(I) > T then R := I - 1 else L := I
if K(L) = T return 'found' else return 'not found'

```

557 We use a simultaneous fixed point with binary relation variables L and R ,
 558 and a unary relation variable Z . Relation variables L and R encode integer
 559 values in binary and fulfill a similar role in the index logic formula than in
 560 the algorithm. More concretely, for each $i \in \text{Num}(\mathbf{A})$, the value of the term
 561 $\text{index}\{\mathbf{x} : L(i, \mathbf{x})\}$ will be the value of the integer variable L before the $(i + 1)$ -th
 562 iteration of the while loop (and similarly for R). The auxiliary variable Z is
 563 used to keep track of the current iteration, starting with 0 and adding in each
 564 step the current iteration number to Z until a fixed point is reached, i.e., until
 565 $Z = \{0, \dots, \lceil \log n \rceil - 1\}$.

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

- 568 • $avg(X, Y, \mathbf{x})$ expressing that the bit \mathbf{x} is set to 1 in the binary representation
 569 of $\lfloor (x + y)/2 \rfloor$ for x and y the numbers encoded in binary in X and Y ,
 570 respectively.
- 571 • $minusone(X, \mathbf{y})$ expressing that the bit \mathbf{y} is set to 1 in the binary repre-
 572 sentation of $x - 1$ for x the number encoded in binary in X .

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

579 In the context in which we use $avg(X, Y, \mathbf{x})$, the variables X and Y are
 580 taken to be $L(\mathbf{z}, \cdot)$ and $R(\mathbf{z}, \cdot)$, respectively. Thus we write $avg'(\mathbf{z}, \mathbf{x})$ to denote
 581 the formula obtained by replacing in $avg(X, Y, \mathbf{x})$ each occurrence of $X(\mathbf{u})$ and
 582 $Y(\mathbf{u})$ by $L(\mathbf{z}, \mathbf{u})$ and $R(\mathbf{z}, \mathbf{u})$, respectively. Likewise, we write $minusone'(\mathbf{z}, \mathbf{u})$ to
 583 denote the formula obtained by replacing in $minusone(X, \mathbf{u})$ each occurrence of
 584 $X(\mathbf{u})$ by $avg'(\mathbf{z}, \mathbf{u})$. We also write $test(\mathbf{z})$ to denote the formula $\exists e(e = index\{\mathbf{x} :$
 585 $avg'(\mathbf{z}, \mathbf{x})\} \wedge K(e) \succ T)$.

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

$$588 \quad \exists x(x = index\{1 : \psi(1)\} \wedge K(x) = T)$$

589 where

$$590 \psi(\mathbf{1}) := \exists \mathbf{s} \forall \mathbf{s}' (\mathbf{s}' \leq \mathbf{s} \wedge [\text{S-IFP}_{\mathbf{z}, \mathbf{x}, L, \mathbf{z}, \mathbf{x}, R, \mathbf{z}, Z} \varphi_L, \varphi_R, \varphi_Z](\mathbf{s}, \mathbf{1})),$$

$$591 \varphi_Z := (Z = \emptyset \wedge \mathbf{z} = 0) \vee (Z \neq \emptyset \wedge \mathbf{z} = \max Z + 1),$$

$$592 \varphi_L := Z \neq \emptyset \wedge \mathbf{z} = \max Z + 1 \wedge$$

$$\exists \mathbf{z}' (\mathbf{z}' = \max Z \wedge (\text{test}(\mathbf{z}') \rightarrow L(\mathbf{z}', \mathbf{x})) \wedge (\neg \text{test}(\mathbf{z}') \rightarrow \text{avg}'(\mathbf{z}', \mathbf{x}))),$$

$$593 \varphi_R := (Z = \emptyset \wedge \mathbf{z} = 0 \wedge \text{BIT}(N - 1, \mathbf{x})) \vee (Z \neq \emptyset \wedge \mathbf{z} = \max Z + 1 \wedge$$

$$594 \exists \mathbf{z}' (\mathbf{z}' = \max Z \wedge (\text{test}(\mathbf{z}') \rightarrow \text{minuse}'(\mathbf{z}', \mathbf{x})) \wedge (\neg \text{test}(\mathbf{z}') \rightarrow R(\mathbf{z}', \mathbf{x}))).$$

595 5.3. The logical characterization theorem for PolylogTime

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

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

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

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

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

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

613 *Proof.*

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

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

629 Let t be a term, M be a direct-access Turing machine and val be a valuation
630 such that for every variable χ that occurs in t the value $val(\chi)$ is the one encoded
631 in binary in the designated work-tape for χ . We start by showing that $val(t)$
632 can then be computed by M in time $O(\lceil \log n \rceil^{O(1)})$. If t is a variable of sort \mathbf{n} ,
633 \mathbf{v} or a constant symbol, then M only needs to read the first $\lceil \log n \rceil$ cells of the
634 appropriate work-tape or constant-tape, respectively. If t is a term of the form
635 $f_i(t_1, \dots, t_k)$, we access and copy each $val(t_j)$ in binary in the corresponding
636 address-tapes of f_i . By the induction hypothesis this takes time $O(\lceil \log n \rceil^{O(1)})$
637 each. Using $\lceil \log n \rceil$ additional steps the result of length $\lceil \log n \rceil$ will then be
638 accessible in the value-tape of f_i .

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

644 If φ is an atomic formula of the form $t_1 \leq t_2$ then M can evaluate φ in

645 polylogarithmic time by accessing the values of t_1 and t_2 in binary and then
 646 comparing their $\lceil \log n \rceil$ bits.

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

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

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

666 If φ is a formula of the form $[IFP_{\bar{\mathbf{x}}, X} \psi] \bar{y}$ where the arity of X is k . Let
 667 $F_{\psi, \bar{\mathbf{x}}, X}^{\mathbf{A}, val} : \mathcal{P}((Num(\mathbf{A}))^k) \rightarrow \mathcal{P}((Num(\mathbf{A}))^k)$ denote the related operator $F^0 := \emptyset$
 668 and $F^{i+1} := F^i \cup F_{\psi, \bar{\mathbf{x}}, X}^{\mathbf{A}, val}(F^i)$ for each $i \geq 0$. The inflationary fixed point is
 669 reached on stage $|Num(\mathbf{A})^k|$ at the latest and thus $\text{ifp}(F_{\psi, \bar{\mathbf{x}}, X}^{\mathbf{A}, val}) = F^{\log^k n}$. Recall
 670 that

$$671 \quad F_{\psi, \bar{\mathbf{x}}, X}^{\mathbf{A}, val}(S) := \{\bar{a} \in (Num(\mathbf{A}))^k \mid \mathbf{A}, val(S/X, \bar{a}/\bar{\mathbf{x}}) \models \psi(X, \bar{\mathbf{x}})\}.$$

672 Then M can proceed as follows. On each stage M computes the value of F^{i+1} in
 673 binary on a work-tape and then copy over this value to the work-tape designated
 674 for X . In stage $i = 0$ it only needs to write the string $0^{\lceil \log n \rceil^k}$ in the work-

675 tape designated for X . To compute F^{i+1} from F^i it goes through all k -tuples
676 $\bar{a} \in (\text{Num}(\mathbf{A}))^k$ in lexicographic order. For $1 \leq j \leq k$ it writes $\bar{a}[j]$ in binary on
677 the designated work-tape for $\bar{\mathbf{x}}[j]$ and checks whether

$$678 \quad \mathbf{A}, \text{val}(S/X, \bar{a}/\bar{\mathbf{x}}) \models \psi(X, \bar{\mathbf{x}}) \quad (2)$$

679 holds. By induction hypothesis, this can be checked in time $O(\lceil \log n \rceil^{O(1)})$. If
680 (2) holds and \bar{a} is the l -th k -tuple in the lexicographic ordering then M writes 1
681 in the l -th cell of the work-tape where the value of F^{i+1} is being constructed.
682 Otherwise it writes 0. Hence the computation of F^{i+1} from F^i can be done in time
683 $\log^k n \times O(\lceil \log n \rceil^{O(1)})$ which is still $O(\lceil \log n \rceil^{O(1)})$. Clearly $\text{ifp}(F_{\psi, \bar{\mathbf{x}}, X}^{\mathbf{A}, \text{val}}) = F^{\log^k n}$
684 can be computed in time $O(\lceil \log n \rceil^{O(1)})$ as well. To determine whether $\text{val}(\bar{y})$ is
685 included in the fixed point is also computable in $O(\lceil \log n \rceil^{O(1)})$ since M can just
686 compute the position of $\text{val}(\bar{y})$ in the lexicographic order of k -tuples and then
687 check whether that position has a 0 or 1 in the work-tape corresponding to X .

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

700 Finally, if φ is a formula of the form $\exists \mathbf{x} \psi$ then for each $i \in \{0, \dots, \lceil \log n \rceil - 1\}$
701 M writes i in binary to the work-tape designated for \mathbf{x} and checks whether
702 $\mathbf{A}, \text{val}(i/\mathbf{x}) \models \psi$. It follows by the induction hypothesis that M can perform
703 each of these checks in polylogarithmic time. If the test is positive for some
704 i then $\mathbf{A}, \text{val} \models \varphi$. The remaining cases are those corresponding to Boolean

705 connectives and follow trivially from the induction hypothesis.

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

712 Let M run in time $O(\lceil \log n \rceil^k)$. Note that a finite number of small inputs
 713 (up to some fixed constant size) may require more time than $\lceil \log n \rceil^k$. Such
 714 small inputs can however be dealt with separately since each finite structure can
 715 be defined by an index logic sentence. Hence we do not consider them here. By
 716 using the order relation $\leq^{\mathbf{A}}$ of the structure \mathbf{A} we can define the lexicographic
 717 order $\leq_k^{\mathbf{A}}$ for the k -tuples in $Num(\mathbf{A})^k$. We use this order to model time and
 718 positions of the tape heads of M . This is possible since the number of k -tuples
 719 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
 720 $(val(t') + 1)$ -th tuple in the order $\leq_k^{\mathbf{A}}$. This is clearly expressible in index logic
 721 since it is a polynomial time property on the \mathbf{n} sort.

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

- 725 • A k -ary relation S_q for every state $q \in Q$ such that $S_q(\bar{t})$ holds iff M is in
 726 state q at time \bar{t} .
- 727 • $2k$ -ary relations $T_i^0, T_i^1, T_i^{\sqcup}$ for every tape $i = 1, \dots, m$ such that $T_i^s(\bar{p}, \bar{t})$
 728 holds iff at the time \bar{t} the cell \bar{p} of the tape i contains the symbol s .
- 729 • $2k$ -ary relations H_i for every tape $i = 1, \dots, m$ such that $H_i(\bar{p}, \bar{t})$ holds iff
 730 at the time \bar{t} the head of the tape i is on the cell \bar{p} .

731 We show that these relations are definable in index logic by means of a
 732 simultaneous inflationary fixed point formula. The following sentence of index
 733 logic is satisfied by a structure \mathbf{A} iff $\mathbf{A} \in \mathcal{C}$. Note that the simultaneous fixed

734 point operator in the formula reconstructs step-by-step the computation of M
 735 for the given input.

736 $\exists \mathbf{x}_0 \dots \mathbf{x}_{k-1} ([\text{S-IFP}_{\bar{t}, S_{q_a}, A, B_1, B_2, B_3, C} \varphi_{q_a}, \Phi_A, \Phi_{B_1}, \Phi_{B_2}, \Phi_{B_3}, \Phi_C](\mathbf{x}_0, \dots, \mathbf{x}_{k-1}))$

737 where

738 $A = \bar{t}, S_{q_0}, \dots, \bar{t}, S_{q_{a-1}} \quad B_1 = \bar{p}\bar{t}, T_1^0, \dots, \bar{p}\bar{t}, T_m^0 \quad B_2 = \bar{p}\bar{t}, T_1^1, \dots, \bar{p}\bar{t}, T_m^1$

739

740 $B_3 = \bar{p}\bar{t}, T_1^\sqcup, \dots, \bar{p}\bar{t}, T_m^\sqcup \quad C = \bar{p}\bar{t}, H_1, \dots, \bar{p}\bar{t}, H_m$

741

742 $\Phi_A = \varphi_{q_0}, \dots, \varphi_{q_{a-1}} \quad \Phi_{B_1} = \psi_{01}, \dots, \psi_{0m} \quad \Phi_{B_2} = \psi_{11}, \dots, \psi_{1m}$

743

744 $\Phi_{B_3} = \psi_{\sqcup 1}, \dots, \psi_{\sqcup m} \quad \Phi_C = \gamma_1, \dots, \gamma_m.$

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

746 The formula builds the required relations S_{q_i} , T_i^0 , T_i^1 , T_i^\sqcup and H_i (for
 747 $1 \leq i \leq m$) in stages, where the j -th stage represents the configuration at time
 748 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 ,
 749 T_i^\sqcup and H_i , respectively.

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

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

764 at the following time instant \bar{t} the position \bar{p} of the tape i contains 0. In the
 765 general case the formula has the form

$$766 \quad (\bar{t} \triangleright 0 \wedge \xi_{T_i^0}) \vee (\neg(\bar{t} \triangleright 0) \wedge \alpha_i^0(\bar{p}, \bar{t} - 1))$$

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

768 Assume i denotes an address-tape or an ordinary work-tape. We define
 769 $\alpha_i^0(\bar{p}, \bar{t} - 1)$ as a disjunction over all cases for tape i to have a 0 in position \bar{p}
 770 at time \bar{t} . There are two possibilities: (a) at time $\bar{t} - 1$ the head of tape i is
 771 not in position \bar{p} and position \bar{p} already has a 0, (b) at time $\bar{t} - 1$ the head of
 772 tape i is in position \bar{p} and writes a 0. Let $\tau_{l,1}^R, \dots, \tau_{l,r_l}^R$ denote the r_l address-
 773 tapes corresponding to the r_l -ary relation R_l . Let $check(R_l(x_1, \dots, x_{r_l}), b_l)$ be
 774 $R_l(x_1, \dots, x_{r_l})$ or $\neg R_l(x_1, \dots, x_{r_l})$ depending on whether $b_l = 1$ or $b_l = 0$,
 775 respectively. Let $\bar{p} = \bar{p}_i$. The disjunct of $\alpha_i^0(\bar{p}, \bar{t} - 1)$ corresponding to case (b)
 776 for a transition of the form $\delta_i(q, a_1, \dots, a_m, b_1, \dots, b_p) = (0, \rightarrow)$ is as follows.

$$\begin{aligned}
 & \exists \bar{p}_1 \dots \bar{p}_{i-1} \bar{p}_{i+1} \dots \bar{p}_m \left(S_q(\bar{t} - 1) \wedge \right. & \text{At time } \bar{t} - 1 \text{ } M \text{ is in the} \\
 & \left(\bigwedge_{1 \leq j \leq m} H_j(\bar{p}_j, \bar{t} - 1) \wedge T_j^{a_j}(\bar{p}_j, \bar{t} - 1) \right) \wedge & \text{state } q \text{ and the head of} \\
 & \bigwedge_{1 \leq l \leq p} \exists x_1 \dots x_{r_l} \left(check(R_l(x_1, \dots, x_{r_l}), b_l) \wedge & \text{the tape } j \text{ is in position } \bar{p}_j \\
 & \bigwedge_{1 \leq k \leq r_l} x_k = index\{\mathbf{x} \mid (T_{\tau_{l,k}^1}^1(\mathbf{x}, \bar{t} - 1))\} \right) \Big), & \text{reading } a_j. \\
 & & \text{At time } \bar{t} - 1 \text{ the tuple of} \\
 & & \text{values in the address-tapes} \\
 & & \text{of } R_l \text{ is in } R^A \text{ iff } b_l = 1.
 \end{aligned}$$

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

$$780 \quad \exists x_1 \dots x_{k_j} \left(\left(\bigwedge_{1 \leq l \leq k_j} x_l = index\{\mathbf{x} \mid T_{\tau_{j,l}^f}^1(\mathbf{x}, \bar{t})\} \right) \wedge \neg BIT(f_j(x_1, \dots, x_{k_j}), p) \right),$$

782 Here $BIT(f_j(x_1, \dots, x_{k_j}), p)$ expresses that the bit of position p of $f_j(x_1, \dots, x_{k_j})$
 783 in binary is 1. As shown in Section 5.1 this is definable in index logic. Note
 784 that the content of the value-tape of a function at time \bar{t} depends only on the
 785 contents of its address-tapes at time \bar{t} . This however does not results in a circular

786 definition since the contents of such address-tapes at time \bar{t} are all defined based
 787 only in the configuration of the machine at time $\bar{t} - 1$.

788 We let φ_{q_0} be $\bar{t} \triangleright 0 \vee (\neg(\bar{t} \triangleright 0) \wedge \alpha_{q_0}(\bar{t} - 1))$. For $q \neq q_0$ we let φ_q be
 789 $\neg(\bar{t} \triangleright 0) \wedge \alpha_q(\bar{t} - 1)$ where $\alpha_q(\bar{t} - 1)$ list conditions under which M will enter
 790 state q at time \bar{t} .

791 Finally, we define γ_i as follows.

$$792 \quad (\bar{t} \triangleright 0 \wedge \bar{p} \triangleright 0) \vee (\neg(\bar{t} \triangleright 0) \wedge \alpha_i(\bar{p}, \bar{t} - 1))$$

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

795 We omit the remaining subformulae since it should be sufficiently clear at
 796 this point that they can indeed be written as index logic formulae. It is also
 797 not difficult to see that in the j -th stage of the simultaneous inflationary fixed
 798 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
 799 the configuration of M at every time $\leq j - 1$. This completes our proof. \square

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

806 6. Definability in Deterministic PolylogTime

807 We observe here that very simple properties of structures are not definable
 808 in index logic. Moreover, we provide an answer to a fundamental question
 809 on the primitivity of the built-in order predicate (on terms of sort \mathbf{v}) in our
 810 logic. Of course, the semantics of index terms only makes sense on ordered
 811 structures. However that does not mean that formulae that do not mention the
 812 order predicate are useless. For example, the following formula expresses that

813 the cardinality of the domain is a power of two:

$$814 \quad \exists x(x = \text{index}\{\mathbf{x} : \text{true}\})$$

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

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

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

826 *Proof.* For a contradiction, assume that \mathcal{L} is decidable in PolylogTime. Consider
 827 ordered first-order structures over the vocabulary $\{\leq, P\}$, where P is a unary
 828 relation symbol. Let M be some random-access Turing machine that given a
 829 binary encoding of a $\{\leq, P\}$ -structure \mathbf{A} decides in PolylogTime whether $P^{\mathbf{A}}$ is
 830 empty. Let f be a polylogarithmic function that bounds the running time of M .
 831 Let n be a natural number such that $f(n) < n$.

832 Let \mathbf{A}_\emptyset be the $\{\leq, P\}$ -structure with domain $\{0, \dots, n-1\}$ where $P^{\mathbf{A}_\emptyset} = \emptyset$.
 833 The encoding of \mathbf{A}_\emptyset to the Turing machine M is the sequence $s := \text{bin}(\leq^{\mathbf{A}_\emptyset}) \underbrace{0 \dots 0}_{n \text{ times}}$.
 834 Note that the running time of M with input s is strictly less than n . This means
 835 that there must exist an index $i \geq n^2$ of s that was not read in the computation
 836 $M(s)$. Define

$$837 \quad s' := \text{bin}(\leq^{\mathbf{A}_\emptyset}) \underbrace{0 \dots 0}_i 1 \quad \underbrace{0 \dots 0}_{n-i-1} \quad .$$

838 Clearly the output of the computations $M(s)$ and $M(s')$ are identical, which is
 839 a contradiction since s' is an encoding of a $\{\leq, P\}$ -structure where the interpre-
 840 tation of P is a singleton. \square

841 The technique of the above proof can be adapted to prove a plethora of unde-
842 finability results, e.g., it can be shown that k -regularity of directed graphs cannot
843 be decided in PolylogTime, for any fixed k . These kinds of lower bounds are
844 well known by researchers working in the area of sublinear time algorithms [33].

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

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

$$851 \quad \mathbf{A}, val \models \varphi \text{ if and only if } l < m.$$

852 *Proof.* We will show that the property described above cannot be decided in
853 PolylogTime; the claim then follows from Theorem 8. For a contradiction,
854 suppose that the property can be decided in PolylogTime, and let M and
855 $f : \mathbb{N} \rightarrow \mathbb{N}$ be the related random-access Turing machine and polylogarithmic
856 function, respectively, such that, for all ordered $\{\leq, P, Q\}$ -structures \mathbf{A} that
857 satisfy the conditions of the claim, $M(\text{bin}(\mathbf{A}))$ decides the property in at most
858 $f(|\text{bin}(\mathbf{A})|)$ steps. Let k be a natural number such that $f(2k) < k - 1$.

859 Consider a computation $M(s)$ of M with an input string s . We say that an
860 index i is *inspected* in the computation, if at some point during the computation
861 i is written in the index tape in binary. Let $\text{Ins}_M(s)$ denote the set of inspected
862 indices of the computation of $M(s)$ and $\text{Ins}_M^j(s)$ denote the set of inspected
863 indices during the first j steps of the computation. We say that s and t are
864 M - j -*equivalent* if the lengths of t and s are equal and $t[i] = s[i]$, for each
865 $i \in \text{Ins}_M^j(s)$. We say that \mathbf{A} and \mathbf{B} are M - j -*equivalent* whenever $\text{bin}(\mathbf{A})$ and
866 $\text{bin}(\mathbf{B})$ are. Note that if two structures \mathbf{A} and \mathbf{B} are M - j -equivalent, then the
867 computations $M(\text{bin}(\mathbf{A}))$ and $M(\text{bin}(\mathbf{B}))$ are at the same configuration after
868 j steps of computation. Hence if \mathbf{A} and \mathbf{B} are M - $f(|\text{bin}(\mathbf{A})|)$ -equivalent, then
869 outputs of $M(\mathbf{A})$ and $M(\mathbf{B})$ are identical.

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

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

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

$$881 \quad \mathfrak{C}_i = \{\mathbf{B} \in \mathfrak{C} \mid \forall j \in \text{Ins}^i(\text{bin}(\mathbf{A}_i)), \text{ if } j \geq 2^n, \text{ the } j\text{th bit of } \text{bin}(\mathbf{B}) \text{ is } 0\}.$$

882 Define $\mathfrak{C}_0 := \mathfrak{C}$; clearly \mathfrak{C}_0 satisfies the properties above. For $i < f(2k)$, we define
883 \mathfrak{C}_{i+1} to be the subclass of \mathfrak{C}_i consisting of those structures \mathbf{A} that on time step
884 $i + 1$ inspects an index that holds the bit 0, or that inspects an index that is at
885 most $2^n - 1$.⁴

886 Assume that a) and b) hold for \mathfrak{C}_i , we will show that the same holds for \mathfrak{C}_{i+1} .
887 Proof of a): Let $\mathbf{A}, \mathbf{B} \in \mathfrak{C}_{i+1}$. By construction and by the induction hypothesis,
888 \mathbf{A} and \mathbf{B} are M - i -equivalent, and on step $i + 1$ $M(\text{bin}(\mathbf{A}))$ and $M(\text{bin}(\mathbf{B}))$
889 inspect the same index, and if the index is at least 2^n , it holds 0. Remember
890 that the first 2^n indices of both \mathbf{A} and \mathbf{B} hold the same string (namely $\text{bin}(\leq^{\mathbf{A}})$).
891 Thus \mathbf{A} and \mathbf{B} are M - $(i + 1)$ -equivalent. Proof of b): It suffices to show that
892 \mathfrak{C}_{i+1} is nonempty; the claim then follows by construction and the property b) of
893 \mathfrak{C}_i . By the induction hypothesis, there is a structure $\mathbf{A}_i \in \mathfrak{C}_i$. Let j be the index
894 that $M(\text{bin}(\mathbf{A}_i))$ inspects on step $i + 1$. If $j < 2^n$ then $\mathbf{A}_i \in \mathfrak{C}_{i+1}$. Suppose
895 $j \geq 2^n$. Since $i + 1 \leq f(2k) < k - 1$, there exists a structure $\mathbf{A}'_i \in \mathfrak{C}_i$ such that

⁴If 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 .

896 the j th bit of $\text{bin}(\mathbf{A}'_i)$ is 0. Clearly $\mathbf{A}'_i \in \mathfrak{C}_{i+1}$.

897 Consider the class \mathfrak{C}_{k-2} (this will be our \mathfrak{C}^*) and $\mathbf{B} \in \mathfrak{C}_{k-2}$ and recall
 898 that $\text{bin}(\mathbf{B})$ is of the form $\text{bin}(\leq^{\mathbf{A}})b_1 \dots b_k c_1 \dots c_k$. Since $|\text{Ins}^{k-2}(\mathbf{B})| \leq k-2$,
 899 there exists two distinct indices i and j , $2^n \leq i < j < 2^n + k$, such that
 900 $i, j, i+k, j+k \notin \text{Ins}^{k-2}(\text{bin}(\mathbf{A}))$. Let $\mathbf{B}_{P<Q}$ denote the structure such that
 901 $\text{bin}(\mathbf{B}_{P<Q})$ is a bit string with prefix $\text{bin}(\leq^{\mathbf{A}})$, and where the i th and $j+k$ th
 902 bits are 1 and all other bits are 0. Similarly, let $\mathbf{B}_{Q<P}$ denote the structure
 903 such that $\text{bin}(\mathbf{B}_{Q<P})$ is a bit string with prefix $\text{bin}(\leq^{\mathbf{A}})$, and where the j th and
 904 $i+k$ th bits are 1 and all other bits are 0. Clearly the structures $\mathbf{B}_{P<Q}$ and
 905 $\mathbf{B}_{Q<P}$ are in \mathfrak{C}_{k-2} and M -($k-2$)-equivalent. Since $(k-2)$ bounds above the
 906 length of computations of $M(\text{bin}(\mathbf{B}_{P<Q}))$ and $M(\text{bin}(\mathbf{B}_{Q<P}))$, it follows that
 907 the outputs of the computations are identical. This is a contradiction, for $\mathbf{B}_{P<Q}$
 908 and $\mathbf{B}_{Q<P}$ are such that M should accept the first and reject the second. \square

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

911 **Theorem 11.** *Let c and d be constant symbols in a vocabulary σ . There does
 912 not exist an index logic formula φ that does not use the order predicate \leq on
 913 terms of sort \mathbf{v} and that is equivalent to the formula $c \leq d$.*

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

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

- 920 • For ψ that does not include c or d , let $\psi^* := \psi$.
- 921 • For ψ of the form $(\alpha_1 \wedge \alpha_2)$, let $\psi^* := (\alpha_1^* \wedge \alpha_2^*)$.
- 922 • For ψ of the form $(\neg\alpha)$, let $\psi^* := (\neg\alpha^*)$.
- 923 • For ψ of the form $(\exists x\alpha)$, let $\psi^* := (\exists x\alpha^*)$

924 • For ψ of the form $(\exists x(x = \text{index}\{\mathbf{x} : \alpha(\mathbf{x})\} \wedge \varphi))$, let

$$925 \quad \psi^* = (\exists x(x = \text{index}\{\mathbf{x} : \alpha(\mathbf{x})\} \wedge \varphi)^*)$$

926 • For ψ of the form $[\text{IFP}_{\bar{x}, X} \theta] \bar{y}$, let $\psi^* := [\text{IFP}_{\bar{x}, X} \theta^*] \bar{y}$.

927 • For ψ of the form $c = d$, let $\psi^* := \perp$.⁵

928 • For ψ of the form $c = x$ or $x = c$, let $\psi^* := C(x)$.

929 • For ψ of the form $x = \text{index}\{\mathbf{x} : \theta(\mathbf{x})\}$, define ψ^* as $x = \text{index}\{\mathbf{x} : \theta^*(\mathbf{x})\}$.

930 • For ψ of the form $c = \text{index}\{\mathbf{x} : \theta(\mathbf{x})\}$, let

$$931 \quad \psi^* := \exists z(z = \text{index}\{\mathbf{x} : \theta^*(\mathbf{x})\} \wedge C(z)),$$

932 where z is a fresh variable.

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

$$938 \quad l < m \quad \Leftrightarrow \quad c^{\mathbf{A}'} < d^{\mathbf{A}'} \quad \Leftrightarrow \quad \mathbf{A}', val \models \varphi \quad \Leftrightarrow \quad \mathbf{A}, val \models \varphi^*.$$

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

$$944 \quad \mathbf{A}', val \models c = x \quad \Leftrightarrow \quad val(x) = c^{\mathbf{A}'} \quad \Leftrightarrow \quad val(x) \in C^{\mathbf{A}} \quad \Leftrightarrow \quad \mathbf{A}, val \models C(x).$$

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

$$946 \quad \mathbf{A}', val \models c = \text{index}\{x : \theta(x)\} \quad \Leftrightarrow \quad \mathbf{A}', val \models \exists z(z = \text{index}\{x : \theta(x)\} \wedge c = z)$$

$$947 \quad \Leftrightarrow \quad \mathbf{A}, val \models \exists z(z = \text{index}\{x : \theta(x)\} \wedge C(z)).$$

948

⁵By \perp we denote some formula that is always false, e.g. $\exists \mathbf{x} \mathbf{x} \neq \mathbf{x}$.

949 All other cases follow similarly. \square

950 We conclude this section by affirming that, on purely relational vocabularies,
 951 the order predicate on sort \mathbf{v} is redundant. The intuition for this result was
 952 given in the beginning of this section.

953 **Theorem 12.** *Let σ be a vocabulary without constant or function symbols. For*
 954 *every sentence φ of index logic of vocabulary σ there exists an equivalent sentence*
 955 *φ' that does not use the order predicate on terms of sort \mathbf{v} .*

956 *Proof.* We will define the translation φ' of φ inductively. Without loss of
 957 generality, we may assume that each variable that occurs in φ is quantified
 958 exactly once (for this purpose, we stipulate that the variable \mathbf{x} is quantified by
 959 the term $index\{\mathbf{x} : \alpha(\mathbf{x})\}$). For every variable x of sort \mathbf{v} that occurs in φ , let
 960 $\alpha_x(\mathbf{x})$ denote the unique subformula such that $\exists x(x = index\{\mathbf{x} : \alpha_x(\mathbf{x})\} \wedge \psi)$ is
 961 a subformula of φ for some ψ . Note that \mathbf{x} occurs only in $index\{\mathbf{x} : \alpha_x(\mathbf{x})\}$. We
 962 define the following shorthands for variables \mathbf{x} and \mathbf{y} of sort \mathbf{n} :

$$\begin{aligned}
 963 \quad \varphi_{\mathbf{x}=\mathbf{y}}(\psi(\mathbf{x}), \theta(\mathbf{y})) &:= \forall \mathbf{z}(\psi(\mathbf{z}/\mathbf{x}) \leftrightarrow \theta(\mathbf{z}/\mathbf{y})), \\
 964 \quad \varphi_{\mathbf{x}<\mathbf{y}}(\psi(\mathbf{x}), \theta(\mathbf{y})) &:= \exists \mathbf{z} \left((\neg\psi(\mathbf{z}/\mathbf{x}) \wedge \theta(\mathbf{z}/\mathbf{y})) \wedge \right. \\
 965 \quad &\quad \left. \forall \mathbf{z}' \left(\mathbf{z} < \mathbf{z}' \rightarrow (\psi(\mathbf{z}'/\mathbf{x}) \leftrightarrow \theta(\mathbf{z}'/\mathbf{y})) \right) \right), \\
 966
 \end{aligned}$$

967 where \mathbf{z} and \mathbf{z}' are fresh distinct variables of sort \mathbf{n} . In the formulae above,
 968 $\psi(\mathbf{z}/\mathbf{x})$ denotes the formula that is obtained from ψ by substituting each free
 969 occurrence of \mathbf{x} in ψ by \mathbf{z} . The translation $\varphi \mapsto \varphi'$ is defined as follows:

- 970 • For formulae that do not include variables of sort \mathbf{v} as well as for formulae of
 971 the form $R(x_1, \dots, x_r)$ where R is an r -ary relation symbol, the translation
 972 is the identity.
- 973 • For ψ of the form $(\alpha_1 \wedge \alpha_2)$, let $\psi' := (\alpha'_1 \wedge \alpha'_2)$.
- 974 • For ψ of the form $\neg\alpha$, let $\psi' := \neg\alpha'$.
- 975 • For ψ of the form $\exists \mathbf{x}\alpha$, let $\psi' := \exists \mathbf{x}\alpha'$

976 • For ψ of the form $[\text{IFP}_{\bar{x}, X} \theta] \bar{y}$, let $\psi' := [\text{IFP}_{\bar{x}, X} \theta'] \bar{y}$.

977 • For ψ of the form $x \leq y$, let

$$978 \quad \psi' := \left(\varphi_{x=y}(\alpha_x(\mathbf{x}), \alpha_y(\mathbf{y})) \vee \varphi_{x < y}(\alpha_x(\mathbf{x}), \alpha_y(\mathbf{y})) \right)'$$

979 • For ψ of the form $x = \text{index}\{\mathbf{y} : \theta(\mathbf{y})\}$, define $\psi' := x = \text{index}\{\mathbf{y} : \theta'(\mathbf{y})\}$.

980 • For ψ of the form $\exists x(x = \text{index}\{\mathbf{x} : \alpha(\mathbf{x})\} \wedge \theta)$, define

$$981 \quad \psi' := \exists x((x = \text{index}\{\mathbf{x} : \alpha(\mathbf{x})\})' \wedge \theta').$$

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

$$987 \quad (x_i \leq x_j)' = \left(\varphi_{x_i=x_j}(\alpha_{x_i}(\mathbf{x}_i), \alpha_{x_j}(\mathbf{x}_j)) \vee \varphi_{x_i < x_j}(\alpha_{x_i}(\mathbf{x}_i), \alpha_{x_j}(\mathbf{x}_j)) \right)'$$

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

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

995 7. Index logic with partial fixed points

996 In this section we introduce a variant of index logic defined in Section 5. This
 997 logic, which we denote as $\text{IL}(\text{PFP})$, is defined by simply replacing the inflationary
 998 fixed point operator IFP in the definition of index logic by the partial fixed point
 999 operator PFP . We stick to the standard semantics of the PFP operator. We
 1000 define that

$$1001 \quad \mathbf{A}, \text{val} \models [\text{PFP}_{\bar{x}, X} \varphi] \bar{y} \text{ iff } \text{val}(\bar{y}) \in \text{pfp}(F_{\varphi, \bar{x}, X}^{\mathbf{A}, \text{val}}),$$

1002 where $\text{pfp}(F_{\varphi, \bar{x}, X}^{\mathbf{A}, \text{val}})$ denotes the *partial* fixed point of the operator $F_{\varphi, \bar{x}, X}^{\mathbf{A}, \text{val}}$ (see the
1003 description above Definition 6). The *partial fixed point* $\text{pfp}(F)$ of an operator
1004 $F : \mathcal{P}(B) \rightarrow \mathcal{P}(B)$ is defined as the fixed point of F obtained from the sequence
1005 $(S^i)_{i \in \mathbb{N}}$, where $S^0 := \emptyset$ and $S^{i+1} := F(S^i)$, if such a fixed point exists. If it does
1006 not exist then $\text{pfp}(F) := \emptyset$.

1007 It is well known that first-order logic extended with partial fixed point
1008 operators captures PSPACE. As a counterpart for this result we show that
1009 $\text{IL}(\text{PFP})$ captures the complexity class polylogarithmic space (PolylogSpace).
1010 Recall that in $\text{IL}(\text{PFP})$ the relation variables bounded by the PFP operators
1011 range over (tuples of) $\text{Num}(\mathbf{A})$, where \mathbf{A} is the interpreting structure. Thus,
1012 the maximum number of iterations before reaching a fixed point (or concluding
1013 that it does not exist) is *not* exponential in the size n of \mathbf{A} as in $\text{FO}(\text{PFP})$. It
1014 is instead *quasi-polynomial*, i.e., of size $O(2^{\log^k n})$ for some constant k . This
1015 observation is the main reason why $\text{IL}(\text{PFP})$ characterizes PolylogSpace. Finally,
1016 an analogous argument to the one that proves the well-known relationship
1017 $\text{PSPACE} \subseteq \text{DTIME}(2^{n^{O(1)}})$ proves that $\text{PolylogSpace} \subseteq \text{DTIME}(2^{\log^{O(1)} n})$.

1018 7.1. The Complexity Class PolylogSpace

1019 Let $L(M)$ denote the class of structures of a given vocabulary σ accepted by
1020 a direct-access Turing machine M . We say that $L(M) \in \text{DSPACE}[f(n)]$ if M
1021 visits at most $O(f(n))$ cells in each work-tape before accepting or rejecting an
1022 input structure whose domain is of size n . We define the class of all languages
1023 decidable by a deterministic direct-access Turing machines in *polylogarithmic*
1024 *space* as follows:

$$1025 \quad \text{PolylogSpace} := \bigcup_{k \in \mathbb{N}} \text{DSPACE}[(\lceil \log n \rceil)^k].$$

1026 Note that it is equivalent whether we define the class PolylogSpace by means of
1027 direct-access Turing machines or random-access Turing machines. Indeed, from
1028 Theorem 3 and the fact that the (standard) binary encoding of a structure \mathbf{A} is
1029 of size polynomial with respect to the cardinality of its domain A , the following
1030 corollary is immediate.

1031 **Corollary 13.** *A class of finite ordered structures \mathcal{C} of some fixed vocabulary σ*
 1032 *is decidable by a random-access Turing machine working in PolylogSpace with*
 1033 *respect to \hat{n} , where \hat{n} is the size of the binary encoding of the input structure, iff*
 1034 *\mathcal{C} is decidable by a direct-access Turing machine in PolylogSpace with respect to*
 1035 *n , where n is the size of the domain of the input structure.*

1036 Moreover, in the context of PolylogSpace, there is no need for random-access
 1037 address-tape for the input; PolylogSpace defined with random-access Turing
 1038 machines coincide with PolylogSpace defined with (standard) Turing machines
 1039 that have sequential access to the input.

1040 **Proposition 14.** *A class of finite ordered structures \mathcal{C} of some fixed vocabulary*
 1041 *σ is decidable by a random-access Turing machine working in PolylogSpace with*
 1042 *respect to \hat{n} iff \mathcal{C} is decidable by a standard (sequential-access) Turing machine*
 1043 *in PolylogSpace with respect to \hat{n} , where \hat{n} is the size of the binary encoding of*
 1044 *the input structure.*

1045 *Proof.* We give the idea behind the proof; the proof itself is straightforward. We
 1046 take as the definition of the standard (sequential-access) Turing machine the
 1047 definition of the random-access Turing machine given in Section 3, except that
 1048 we suppose a sequential-access read-only-head for the input tape and remove
 1049 the address-tape.

1050 A random-access Turing machine M_r can simulate a sequential-access Turing
 1051 machine M_s directly by using its address-tape to simulate the movement of the
 1052 head of the sequential-access input-tape. In the simulation, when the head of
 1053 the input-tape of M_s is on the $i + 1$ -th cell, the address-tape of M_r holds the
 1054 number i in binary and hence refers to the $i + 1$ -th cell of the input. When
 1055 the head of the input-tape of M_s moves right, the machine M_r will increase
 1056 the binary number in its address-tape by one. Similarly, when the head of the
 1057 input-tape of M_s moves left, the machine M_r will decrease the binary number
 1058 in its address-tape by one. A total of $\lceil \log n \rceil$ bits suffices to access any bit of an
 1059 input of length n . Clearly increasing or decreasing a binary number of length at
 1060 most $\lceil \log n \rceil$ by one can be done in PolylogSpace. The rest of the simulation is

1061 straightforward.

1062 The simulation of the other direction is a bit more complicated. Each time
1063 the content of the address-tape of the random-access machine is updated, we
1064 need to calculate the corresponding position of the head of the input-tape of the
1065 sequential-access machine. This computation however can be clearly done in
1066 PolylogSpace: We use a work-tape of the sequential-access machine to mimic the
1067 address-tape of the sequential-access machine and an additional work-tape as a
1068 binary counter. After each computation step of the random-access machine, the
1069 sequential-access machine moves the head of its input tape to its leftmost cell and
1070 formats the work-tape working as a binary counter to have exactly the binary
1071 number that is written on the address-tape. Then the sequential-access machine
1072 moves the head of its input-tape right step-by-step simultaneously decreasing the
1073 binary counter by 1. Once the binary counter reaches 0, the head of the input
1074 tape is in correct position. The rest of the simulation is straightforward. \square

1075 Since the function $\lceil \log n \rceil$ is *space constructible* (s.c. for short), or equivalently
1076 *proper* as called in [16], and for any two s.c. functions their product is also s.c.,
1077 we get that for any $k \geq 1$ the function $(\lceil \log n \rceil)^k$ is s.c. Hence, from Savitch's
1078 theorem we get the following result.

1079 **Fact 15.** *For any $k \geq 1$, it holds that $\text{NSPACE}[(\lceil \log n \rceil)^k] \subseteq \text{DSpace}[(\lceil \log n \rceil)^{2k}]$.*
1080 *Thus, nondeterministic and deterministic PolylogSpace coincide.*

1081 7.2. Index logic with partial fixed point operators captures PolylogSpace

1082 To encode a configuration of polylogarithmic size, we follow a similar strategy
1083 as in Theorem 8, i.e., in the proof of the characterization of PolylogTime by
1084 $\text{IL}(\text{IFP})$. The difference here is that there is no reason to encode the whole
1085 history of a computation in the fixed point. At a time step t it suffices that the
1086 configuration of the machine at time step $t - 1$ is encoded; hence we may drop
1087 the variables \bar{t} from the fixed point formula defined on page 28. Moreover, we
1088 make a small alteration to the Turing machines so that acceptance on an input
1089 structure will correspond to the existence of a partial fixed point.

1090 **Theorem 16.** *Over ordered finite structures, IL(PFP) captures PolylogSpace.*

1091 *Proof.* The direction of the proof that argues that IL(PFP) can indeed be
1092 evaluated in PolylogSpace is straightforward. Let ψ be an IL(PFP)-sentence.
1093 We only need to show that there exists a direct-access Turing machine M_ψ that
1094 works in $O(\log^d n)$ space for some constant d and that for every structure \mathbf{A}
1095 and valuation val it holds that $\mathbf{A} \in L(M_\psi)$ iff $\mathbf{A}, val \models \psi$. Note that in an
1096 induction on the structure of ψ , all cases except the case for the PFP operator
1097 are as in the proof of Theorem 8. Clearly if a formula can be evaluated in
1098 PolylogTime it can also be evaluated in PolylogSpace. For the case of the PFP
1099 operator (using a similar strategy as in [28]) we set a counter to $2^{\log^r n}$ using
1100 exactly $\log^r n$ cells in a work-tape, where r is the arity of the relation variable X
1101 bounded by the PFP operator. To evaluate the PFP operator, say on a formula
1102 $\varphi(\bar{x}, X)$, M will iterate evaluating φ and decrease the counter in each iteration.
1103 When the counter gets to 0, M checks whether the contents of the relation X
1104 is equal to its contents in the following cycle and whether the tuple given in
1105 the PFP application belongs to it. If both answers are positive then M accepts.
1106 Otherwise it rejects. This suffices to find the fixed point (or to conclude that
1107 it does not exist) as there are $2^{\log^r n}$ many relations of arity r with domain
1108 $\{0, \dots, \lceil \log n \rceil - 1\}$.

1109 For the converse, let $M = (Q, \Sigma, \delta, q_0, F, \sigma)$ be an m -tape direct-access Turing
1110 machine that works in PolylogSpace. Same as in the proof of Theorem 8 we can
1111 assume w.l.o.g. that $F = \{q_a\}$ (i.e., there is only one accepting state), $|Q| = a + 1$
1112 and $Q = \{q_0, q_1, \dots, q_a\}$. We additionally assume here that once the machine
1113 reaches an accepting state, it will not change its configuration any longer. That
1114 is, all of its heads stay put and write the same symbol that they read. Note that
1115 the machine M accepts if and only if M is in the same accepting configuration
1116 during two consecutive time steps.

1117 We build an IL(PFP)-sentence ψ_M such that for every structure \mathbf{A} and
1118 valuation val , it holds that $\mathbf{A} \in L(M)$ iff $\mathbf{A}, val \models \psi_M$. The formula is a
1119 derivative of that of Theorem 8 and is defined using a simultaneous PFP operator.

1120 In the formula below, S_{q_0}, \dots, S_{q_a} denote 0-ary relation variables that range
 1121 over the values *true* and *false*. We define

$$1122 \quad \psi_M := [\text{S-PFP}_{S_{q_a}, \mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, \mathbf{C}} \varphi_{q_a}, \Phi_{\mathbf{A}}, \Phi_{\mathbf{B}_1}, \Phi_{\mathbf{B}_2}, \Phi_{\mathbf{B}_3}, \Phi_{\mathbf{C}}],$$

1123 where

$$1124 \quad \mathbf{A} = S_{q_0}, \dots, S_{q_{a-1}} \quad \mathbf{B}_1 = \bar{p}, T_1^0, \dots, \bar{p}, T_m^0 \quad \mathbf{B}_2 = \bar{p}, T_1^1, \dots, \bar{p}, T_m^1$$

1125

$$1126 \quad \mathbf{B}_3 = \bar{p}, T_1^{\sqcup}, \dots, \bar{p}, T_m^{\sqcup} \quad \mathbf{C} = \bar{p}, H_1, \dots, \bar{p}, H_m$$

1127

$$1128 \quad \Phi_{\mathbf{A}} = \varphi_{q_0}, \dots, \varphi_{q_{a-1}} \quad \Phi_{\mathbf{B}_1} = \psi_{01}, \dots, \psi_{0m} \quad \Phi_{\mathbf{B}_2} = \psi_{11}, \dots, \psi_{1m}$$

1129

$$1130 \quad \Phi_{\mathbf{B}_3} = \psi_{\sqcup 1}, \dots, \psi_{\sqcup m} \quad \Phi_{\mathbf{C}} = \gamma_1, \dots, \gamma_m.$$

1131 The formulae used in the PFP operator are defined in the same way as in
 1132 Theorem 8; with the following two exceptions.

- 1133 1. The formulae of the form $\alpha_i^0(\bar{p}, \bar{t} - 1)$ are replaced with the analogous
 1134 formulae $\alpha_i^0(\bar{p})$ obtained by simply removing the variables referring to time
 1135 steps.
- 1136 2. Subformulae of the form $\bar{t} \triangleright 0$ are replaced with $\neg S_{q_0} \wedge \dots \wedge \neg S_{q_{a-1}}$, which
 1137 are true only on the first iteration of the fixed point calculation.

1138 Following the proof of Theorem 8 it is now easy to show that $\mathbf{A}, \text{val} \models \psi_M$ if
 1139 and only if M *accepts* \mathbf{A} . □

1140 8. Discussion

1141 The natural question left open by our work is to find a logic capturing
 1142 PolylogTime over structures that are not necessarily ordered. Since index logic
 1143 captures PolylogTime on ordered structures, this question is equivalent to finding
 1144 an effective syntax for the sentences in index logic that are order-invariant. As
 1145 explored in Section 6, it appears that actually very few properties of unordered
 1146 structures are in fact decidable in PolylogTime. Then again, any polynomial-time
 1147 numerical property of the size of the domain is clearly decidable in PolylogTime.

1148 So, there seems to be a delicate balance of weakness on the one hand, and power
1149 on the other hand, and it seems interesting to pursue this open question further.

1150 Another natural direction is to get rid of Turing machines altogether and work
1151 with a RAM model working directly on structures, as proposed by Grandjean
1152 and Olive [34]. Plausibly by restricting their model to numbers bounded in value
1153 by a polynomial in n (the size of the structure), we would get an equivalent
1154 PolylogTime complexity notion.

1155 In this vein, we would like to note that extending index logic with numeric
1156 variables that can hold values up to a polynomial in n , with arbitrary polynomial-
1157 time functions on these, would provide useful syntactic sugar. Since this remains
1158 in PolylogTime, it follows from our capturing result that such extension would
1159 not increase the expressive power of index logic.

1160 Acknowledgements

1161 We thank the two anonymous reviewers whose comments have significantly
1162 helped to improve the manuscript.

1163 [1] E. Grädel, P. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema,
1164 S. Weinstein, *Finite Model Theory and Its Applications*, Springer, 2007.

1165 [2] Y. Gurevich, Toward logic tailored for computational complexity, in:
1166 M. Richter, et al. (Eds.), *Computation and Proof Theory*, Vol. 1104 of
1167 *Lecture Notes in Mathematics*, Springer-Verlag, 1984, pp. 175–216.

1168 [3] N. Immerman, *Descriptive Complexity*, Springer, 1999.

1169 [4] R. Fagin, Generalized first-order spectra and polynomial-time recognizable
1170 sets, in: R. Karp (Ed.), *Complexity of Computation*, Vol. 7 of *SIAM-AMS*
1171 *Proceedings*, American Mathematical Society, 1974, pp. 43–73.

1172 [5] N. Immerman, Relational queries computable in polynomial time, *Informa-*
1173 *tion and Control* 68 (1986) 86–104.

- 1174 [6] M. Vardi, The complexity of relational query languages, in: Proceedings
1175 14th ACM Symposium on the Theory of Computing, 1982, pp. 137–146.
- 1176 [7] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley,
1177 1995.
- 1178 [8] M. Y. Vardi, The complexity of relational query languages, in: Proceedings
1179 of the 14th Annual ACM Symposium on Theory of Computing, ACM, 1982,
1180 pp. 137–146.
- 1181 [9] F. Ferrarotti, S. González, J. M. Turull Torres, J. Van den Bussche,
1182 J. Virtema, Descriptive complexity of deterministic polylogarithmic time,
1183 in: Logic, Language, Information, and Computation - 26th International
1184 Workshop, WoLLIC 2019, Proceedings, Vol. 11541 of Lecture Notes in
1185 Computer Science, Springer, 2019, pp. 208–222.
- 1186 [10] M. Grohe, W. Pakusa, Descriptive complexity of linear equation systems and
1187 applications to propositional proof complexity, in: 32nd Annual ACM/IEEE
1188 Symposium on Logic in Computer Science, LICS, IEEE Computer Society,
1189 2017, pp. 1–12.
- 1190 [11] N. Immerman, Number of quantifiers is better than number of tape cells, J.
1191 Comput. Syst. Sci. 22 (3) (1981) 384–406.
- 1192 [12] D. A. Mix Barrington, N. Immerman, H. Straubing, On uniformity within
1193 NC^1 , J. Comput. Syst. Sci. 41 (3) (1990) 274–306.
- 1194 [13] D. A. Mix Barrington, Quasipolynomial size circuit classes, in: Proceedings
1195 of the Seventh Annual Structure in Complexity Theory Conference, IEEE
1196 Computer Society, 1992, pp. 86–93.
- 1197 [14] F. Ferrarotti, S. González, K. Schewe, J. M. Turull Torres, The polylog-time
1198 hierarchy captured by restricted second-order logic, in: 20th International
1199 Symposium on Symbolic and Numeric Algorithms for Scientific Computing,
1200 IEEE, 2018, pp. 133–140.

- 1201 [15] L. Stockmeyer, The polynomial-time hierarchy, *Theor. Comput. Sci.* 3 (1)
1202 (1976) 1–22.
- 1203 [16] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
- 1204 [17] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the*
1205 *Theory of NP-Completeness*, Freeman, 1979.
- 1206 [18] A. Borodin, On relating time and space to size and depth, *SIAM J. Comput.*
1207 6 (4) (1977) 733–744.
- 1208 [19] R. Greenlaw, H. J. Hoover, W. L. Ruzzo, *Limits to Parallel Computation:*
1209 *P-completeness Theory*, Oxford University Press, 1995.
- 1210 [20] J. H. Reif, Logarithmic depth circuits for algebraic functions, *SIAM J.*
1211 *Comput.* 15 (1) (1986) 231–242.
- 1212 [21] G. Matera, J. M. Turull Torres, The space complexity of elimination theory:
1213 Upper bounds, in: *Foundations of Computational Mathematics*, Springer,
1214 1997, pp. 267–276.
- 1215 [22] A. Grosso, N. Herrera, G. Matera, M. E. Stefanoni, J. M. Turull Torres,
1216 An algorithm for the computation of the rank of integer matrices in poly-
1217 logarithmic space, *Electronic Journal of the Chilean Society of Computer*
1218 *Science* 4 (1), 45 pages, in Spanish.
- 1219 [23] G. Gottlob, N. Leone, F. Scarcello, Computing LOGCFL certificates, *Theor.*
1220 *Comput. Sci.* 270 (1-2) (2002) 761–777.
- 1221 [24] G. Gottlob, R. Pichler, F. Wei, Tractable database design through bounded
1222 treewidth, in: *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-*
1223 *SIGART Symposium on Principles of Database Systems*, ACM, 2006, pp.
1224 124–133.
- 1225 [25] G. Gottlob, R. Pichler, F. Wei, Tractable database design and datalog
1226 abduction through bounded treewidth, *Inf. Syst.* 35 (3) (2010) 278–298.

- 1227 [26] M. Beaudry, P. McKenzie, Circuits, matrices, and nonassociative computa-
1228 tion, *J. Comput. Syst. Sci.* 50 (3) (1995) 441–455.
- 1229 [27] M. Grohe, *Descriptive Complexity, Canonisation, and Definable Graph*
1230 *Structure Theory*, Cambridge University Press, 2017.
- 1231 [28] H.-D. Ebbinghaus, J. Flum, *Finite Model Theory*, 2nd Edition, Springer,
1232 1999.
- 1233 [29] L. Libkin, *Elements of Finite Model Theory*, Springer, 2004.
- 1234 [30] Y. Gurevich, S. Shelah, Fixed-point extensions of first-order logic, *Annals*
1235 *of Pure and Applied Logic* 32 (1986) 265–280.
- 1236 [31] D. Knuth, *Sorting and Searching*, 2nd Edition, Vol. 3 of *The Art of Computer*
1237 *Programming*, Addison-Wesley, 1998.
- 1238 [32] M. Grohe, The quest for a logic capturing PTIME, in: *Proceedings of the*
1239 *Twenty-Third Annual IEEE Symposium on Logic in Computer Science*,
1240 *LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, IEEE Computer Society,
1241 2008, pp. 267–271.
- 1242 [33] R. Rubinfeld, A. Shapira, Sublinear time algorithms, *SIAM J. Discret. Math.*
1243 25 (4) (2011) 1562–1588.
- 1244 [34] E. Grandjean, F. Olive, Graph properties checkable in linear time in the
1245 number of vertices, *J. Comput. Syst. Sci.* 68 (2004) 546–597.