Choiceless Logarithmic Space

Erich Grädel
RWTH Aachen University, Germany
graedel@logic.rwth-aachen.de

Svenja Schalthöfer
RWTH Aachen University, Germany
schalthoefer@logic.rwth-aachen.de

Abstract

One of the most important open problems in finite model theory is the question whether there is a logic characterising efficient computation. While this question usually concerns PTIME, it can also be applied to other complexity classes, and in particular to LOGSPACE which can be seen as a formalisation of efficient computation for big data. One of the strongest candidates for a logic capturing PTIME is Choiceless Polynomial Time (CPT). It is based on the idea of choiceless algorithms, a general model of symmetric computation over abstract structures (rather than their encodings by finite strings). However, there is currently neither a comparably strong candidate for a logic for LOGSPACE, nor a logic transferring the idea of choiceless computation to LOGSPACE.

We propose here a notion of Choiceless Logarithmic Space which overcomes some of the obstacles posed by LOGSPACE as a less robust complexity class. The resulting logic is contained in both LOGSPACE and CPT, and is strictly more expressive than all logics for LOGSPACE that have been known so far. Further, we address the question whether this logic can define all LOGSPACE-queries, and prove that this is not the case.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Finite Model Theory, Logics for Logspace, Choiceless Computation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.31

1 Introduction

The probably most fundamental problem of finite model theory concerns the question whether there exist logics that precisely characterise the efficiently computable queries on finite structures. Usually this problem is called the *quest for a logic for polynomial time* [15], a problem originally posed by Chandra and Harel [5] and later made more precise by Gurevich [19]. But PTIME is just one, although certainly the most common, complexity class to capture the intuitive notion of efficient computability. Another important such class is LOGSPACE, which formalises efficient computation over large data sets with significantly smaller working memory, and may thus be viewed as a notion of efficient computability for big data. When Chandra and Harel [5] enquired about a logical characterization of the PTIME-queries, they already asked about LOGSPACE as well, and since then, a number of logics have been proposed that capture relevant parts of LOGSPACE. We briefly compare the state of the art on the quests for logics for PTIME and LOGSPACE, respectively.

Logics for polynomial time. For PTIME, the logic of reference is *fixed-point logic with counting* (FPC), which has first been proposed informally by Immerman [20] and then made precise in [11]. It extends first-order logic by fixed-point operators and counting terms and actually comes rather close to being a logic for polynomial time. It is strong enough to express many of the algorithmic techniques leading to polynomial-time procedures and it captures PTIME on many interesting classes of structures, including planar graphs, structures of bounded tree width, and actually all classes of graphs with an excluded minor [16]. For a recent survey on FPC, see [6]. Nevertheless there are important problems that separate PTIME from FPC. The first of these, which still is an interesting benchmark problem for logics in PTIME, has been the CFI-query due to [4]. Other examples are efficiently solvable variants of the graph isomorphism problem or problems from linear algebra such as solving linear equation systems over finite fields or rings [2]. To handle such examples, several more powerful logics than FPC have been proposed, including for instance rank logic [7, 12].

However, the most promising candidate for a logic for PTIME is Choiceless Polynomial Time (CPT), introduced by Blass, Gurevich, and Shelah [3]. There are several different presentations of CPT (see e.g. [9, 10, 22, 25]). The original intention was to explore a model for efficient computations on finite structures which preserve symmetries at every step in the computation. This prohibits the explicit introduction of an ordering or, equivalently, arbitrary choices between indistinguishable elements of the input structure or of the current state. Such choices appear in many algorithms of fundamental importance, including depth-first search, Gaussian elimination, and many more. CPT is based on a computation model that avoids symmetry breaking choices, but allows essentially everything else, including parallelism and "fancy data structures", as long as all operations can be carried out in polynomial time. It works, given a finite structure, on its extension by all hereditarily finite sets over its universe, which may be seen as a powerful higher-order data structure. Choiceless Polynomial Time is the restriction of this model to polynomial-time resources. It is known that CPT is strictly more powerful than fixed-point logic with counting. In particular, CPT can express several variants of the CFI-queries that are not expressible in FPC and, more generally, CPT captures polynomial time over interesting classes of structures over which FPC fails to do so [1, 8, 23]. Nevertheless it is still open whether CPT captures PTIME on arbitrary finite structures. We refer to [9, 22, 26] for more information on CPT.

Logics for logarithmic space. The state of the art concerning logics for LOGSPACE is less advanced than for PTIME. On ordered structures, LOGSPACE can be captured by DTC, the extension of first-order logic by deterministic transitive closures [21]. A more elegant variant is the symmetric transitive closure logic STC, whose evaluation in LOGSPACE however depends on the highly non-trivial algorithm by Reingold [24] for undirected graph reachability. But even with an added counting operation, STC does not capture LOGSPACE on arbitrary structures, and again this can be proved by the CFI-query. Some of the shortcomings of transitive closure logics are elegantly countered by the logic LREC (short for L-recursion), especially in its stronger variant, called $\text{LREC}_=$ in the literature [17, 18]. The core of LREC is an elaborate recursion operator, augmented by a mechanism defining symmetric transitive closures to obtain closure under interpretations. However, although LREC captures LOGSPACE on a larger class of structures than transitive closure logics, it is still contained in FPC and thus does not capture LOGSPACE on arbitrary finite structures.

There has, up to now, not really emerged a convincing logspace-analogue of Choiceless Polynomial Time, and also no other really serious candidate for a logic for LOGSPACE. So some natural questions arise: *"What is Choiceless Logspace? What expressive power does it have, and could it actually capture all of* LOGSPACE*?"* Addressing these questions, we propose a notion of Choiceless Logspace. We prove that this provides a logic that is indeed contained in both CPT and LOGSPACE. Moreover, it is strictly more powerful than all LOGSPACE-logics known so far, including (the strong variant of) LREC. However, it turns out that even this logic fails to capture all LOGSPACE queries, which leads to the more general question of what would be a suitable notion of efficient choiceless computation for big data.

Sets of logarithmic size. Towards such a definition, we first note that the obvious, but naïve attempt fails. Since CPT permits polynomially many computation steps using only sets whose transitive closures contain polynomially many objects, the naïve approach would simply discard the time bound and allow sets with a transitive closure of logarithmically many objects. But this would make it possible to define sets containing logarithmically many atoms, which admits no straightforward evaluation in LOGSPACE: to represent such a set, one would store every atom as a number with logarithmically many bits, so one would actually have to represent logarithmically many numbers of logarithmic size. Unless LOGSPACE $=$ NC^2, this should not be possible.

The central question is thus what it means that a set is of logarithmic size. Even though choiceless computation is invariant under encodings, already the size of an atom depends on its encoding. On an ordered encoding of a structure, every atom can be referenced by a binary string with logarithmically many bits. On the basis of this straightforward encoding, a logspace algorithm could only store constantly many atoms at a time, so a set of logarithmic size would be a set whose transitive closure is bounded by a constant. In fact, many standard logspace algorithms operate with a bounded number of storage locations, each of which can store an object with logarithmically many bits, typically an element of the input structure or a number that is polynomially bounded by the size of the input structure. A previous approach to defining choiceless logarithmic space [13, 27] has operated on exactly these "bounded sets". It corresponds to a logic, denoted here BDTC, that provides an operator for deterministic transitive closures over bounded sets. In its original version, BDTC lacks the ability to count, but besides that it has more fundamental deficiencies. Indeed, the assumption that a set of logarithmic size can only contain constantly many atoms does not take into account that there are alternative ways to represent atoms, and logspace algorithms that operate in a more sophisticated manner, using an unbounded number of storage locations such as the algorithm for tree isomorphism or Reingold's algorithm for undirected graph reachability. It is, for instance, possible to store the unique root of a tree in constant space by just storing its defining property. In a structure with a unary predicate P, every atom in P can be represented as "the kth element of P", so the size of the atom representation is logarithmic in $|P|$ instead of the size of the whole structure. The size of an atom with respect to LOGSPACE-algorithms can hence vary depending on its representation.

Towards Choiceless Logspace. This observation leads to the main idea behind CLogspace, the logic we shall introduce: When evaluating formulae with logarithmic bounds, we do not assume a fixed size for every atom. Therefore, the sizes of atoms are part of the semantics and, consequently, the logic is evaluated over *size-annotated hereditarily finite sets.*

Atoms can be represented using logical formulae. We introduce terms "$\text{Atoms}\,.\varphi$" in our logic to make sure that, whenever a set of atoms is defined, that definition is guarded by a formula. The size of the atoms in that set will then be defined assuming that every atom is represented as "the ith entry of the kth tuple satisfying φ". So the size of every hereditarily finite object will depend on the terms of the form $\text{Atoms}\,.\varphi$ generating the atoms in its transitive closure. However, this way of defining atoms does not allow unbounded recursion. We therefore add (a modified version of) the recursion operator from LREC to our logic. But, just like in CPT, iterated *creation* of sets is necessary to obtain a formalism that is stronger than common logics. Consequently, our logic also incorporates CPT-style iteration with some modifications mimicking the behaviour of LOGSPACE-algorithms.

In a nutshell, Choiceless LOGSPACE in our sense means iteration and recursion over hereditarily finite sets of logarithmic size, where the size of each atom is derived from its representation through a formula.

2 Choiceless Logarithmic Space and Choiceless Polynomial Time

In this section we present the precise definition of CLogspace. Readers familiar with Choiceless Polynomial Time will recall that it is evaluated with the data structure of all *hereditarily finite objects* over the given input structure.

The elements of the universe A of an input structure $\mathbf{A}$ are assumed to be *atoms*, i. e. not sets, and an *object* is an atom or a set. A set a is *transitive* if $\in$ is a transitive relation on a, which means that all elements of a are either subsets of a or atoms. The transitive closure $\text{tc}(a)$ of an object a is the smallest transitive set b with $a \in b$. An object a is hereditarily finite if $\text{tc}(a)$ is finite. Given a set A of atoms, $\text{HF}(A)$ is the collection of hereditarily finite objects over A, i. e. the set of all objects x such that $x \in A$ or x is a hereditarily finite set such that all atoms in $\text{tc}(x)$ are elements of A.

Choiceless Logspace is evaluated over *size-annotated hereditarily finite sets* where a size annotation is a function assigning (in a consistent way) a size to every element of the transitive closure of the given set. The definition of sizes guarantees that a set of logarithmic size can be represented in LOGSPACE in a straightforward way.

▶ **Definition 1.** *Let* $\mathbf{A}$ *be a* σ*-structure and* $a \in \text{HF}(A)$*. A function* $s\colon \text{tc}(a) \to \mathbb{N}$ *is a* size annotation of *a if, for each set* $b \in \text{tc}(a)$*,* $s(b) = 1+\sum_{c \in b} s(c)$*. The set of* size-annotated hereditarily finite objects *is* $\text{SHF}(A) := \{(a, s_a) : a \in \text{HF}(A)$ *and* s_a *is a size annotation of* $a\}$*. For size annotations* $(a, s_a), (b, s_b)$*, we say that* $(b, s_b) \in (a, s_a)$ *if* $b \in a$ *and* $s_b = s_a{\restriction}\text{tc}(b)$*.*

Note that a size annotation is completely determined by the values it assigns to the atoms. In particular, for a *pure set* $a \in \text{HF}(A)$, with $\text{tc}(a) \cap A = \emptyset$, there is a unique size annotation, which we denote by ann_a. Further, the size annotation yields an upper bound for the size of the transitive closure. As atoms may have size 0, conclude that $|\,\text{tc}(a) \setminus A| \leq s(a)$ for any size annotation s of a.

We now formulate a high-level definition of the logic CLogspace, and then proceed to make precise, and explain, its ingredients step by step.

▶ **Definition 2** (Choiceless Logarithmic Space)**.** *The logic* CLogspace *is defined by rules for ordinary terms and formulae, iteration terms and recursion formulae as follows. If* t *is a term and* φ *is a formula according to these rules, and* f *is a function* $f\colon n \mapsto c \log n$ *for* $c \in \mathbb{N}$*, then* (t, f) *is a* CLogspace*-term, and* (φ, f) *is a* CLogspace*-formula.*

The function f only plays a role in defining the semantics of iteration terms. Otherwise the semantics is just given by the semantics of t and φ, respectively.

We first explain ordinary terms and formulae which provide the basic operations for constructing sets.

▶ **Definition 3** (Syntax)**.** *Ordinary terms and formulae over a vocabulary* σ *are defined inductively as follows:*

- $\emptyset$ *is a term, and every variable* x *is a term,*
- *if* $\varphi \in \text{FO}[\sigma]$ *with at least one free variable, then* $\text{Atoms}\,.\varphi$ *is a term,*
- *if* t_1, t_2 *are terms, then* $\text{Union}(t_1)$*,* $\text{Unique}(t_1)$*,* $\text{Pair}(t_1, t_2)$*, and* $\text{Card}(t_1)$ *are terms,*
- *for* $R \in \sigma$ *and terms* $t_1, \dots, t_k$*, the expressions* $t_1 = t_2$*,* $t_1 \in t_2$*, and* $Rt_1 \dots t_k$ *are formulae,*
- *Boolean combinations of formulae are formulae,*
- *if* p *and* q *are terms and* φ *is a formula, then* $\{p : x \in q : \varphi\}$ *is a (comprehension) term.*

The free variables $\text{free}(t)$ of a term or formula t are defined as usual, where, in the term $t := \{p : x \in q : \varphi\}$, x occurs free in p and φ and bound in t. We therefore write $\{p(x) : x \in q : \varphi(x)\}$.

Counting. The terms $\mathrm{Card}(t)$ define cardinalities of sets. Since the cardinality of sets of polynomial size can be computed in LOGSPACE, von Neumann ordinals, which represent cardinalities in Choiceless Polynomial Time, are in general too large for our purposes. Therefore cardinalities will be denoted by a set encoding of their binary representation. For this purpose, we assume the short form $\langle a, b\rangle = \{a, \{a, b\}\}$ of Kuratowski pairs. This will make sure that every word of length two is represented by a set with two elements and can thus be distinguished from $\mathrm{bitset}(0)$ and $\mathrm{bitset}(1)$.

▶ **Definition 4.** *We define the function* $\mathrm{bitset}\colon \{0,1\}^+ \to \mathrm{HF}(\emptyset)$ *by setting* $\mathrm{bitset}(0) := \emptyset$, $\mathrm{bitset}(1) := \{\emptyset\}$, *and* $\mathrm{bitset}(bw) = \langle \mathrm{bitset}(b), \mathrm{bitset}(w)\rangle$ *for* $b \in \{0,1\}$ *and* $w \in \{0,1\}^+$.

If $n \in \mathbb{N}$, *let* $\mathrm{bitset}(n)$ *denote* $\mathrm{bitset}(\mathrm{bin}(n))$, *where* $\mathrm{bin}(n)$ *is the binary representation of* n, *with* 2^0 *as the right-most bit and without leading zeroes.*

It follows from the definition that the function bitset is injective. This representation of numbers keeps their size logarithmic with respect to their unique size annotation. More precisely, a simple induction on $|w|$ shows:

▶ **Lemma 5.** *Let* $s\colon \mathrm{tc}(\mathrm{bitset}(w)) \to \mathbb{N}$ *be a size annotation for* $w \in \{0,1\}^+$. *Then* $s(\mathrm{bitset}(w)) \le 6|w|$.

Semantics of ordinary terms and formulae. Evaluation over size-annotated objects means that the value of every term is equipped with a size annotation. Hence, also the values of free variables have to incorporate size annotations. Recall that a size annotation defines a size for every element of the transitive closure of an object, which is uniquely determined by the sizes associated with the atoms. Unless the size of an atom is given externally through a free variable, its size originates from the term defining it. The only way to define atoms directly will be with terms of the form $\mathrm{Atoms}\,.\varphi$.

We assume every atom to be represented as "the ith entry of the kth tuple satisfying φ". Both φ and i are of constant size, since φ is a part of the term. So we can assume that, up to an additive constant, such an atom can be written as a number with $\log k$ bits where k is bounded by the number of tuples satisfying φ. This number is the size we assign to every atom defined that way. If an atom occurs in the values of multiple subterms, it may have multiple sizes that contribute to the value of the term. In that case, the minimal size is picked, which is formalised as follows:

▶ **Definition 6.** *Let* $S \subseteq \mathrm{SHF}(A)$ *be a set of size-annotated hereditarily finite sets. We denote by* S_{SHF} *the pair* (s, ann_s) *where* $s = \{s' : (s', \mathrm{ann}_{s'}) \in S\}$ *and* ann_s *is the unique size annotation of* s *with* $\mathrm{ann}_s(a) = \min\{\mathrm{ann}_{s'}(a) : (s', \mathrm{ann}_{s'}) \in S\}$ *for every atom* $a \in \mathrm{tc}(s)$.

▶ **Definition 7** (Semantics of ordinary terms). *Let* $\mathbf{A}$ *be a* σ*-structure and let* $\beta\colon X \to \mathrm{SHF}(A)$ *be a variable assignment. For* $\alpha \in \mathrm{SHF}(A)$, *we write* $\beta[x \mapsto \alpha]\colon X \cup \{x\} \to \mathrm{SHF}(A)$ *to denote the function that behaves like* β *on* $X \setminus \{x\}$ *and maps* x *to* α. *For an ordinary* σ*-term* t *(with* $\mathrm{free}(t) \subseteq X$*), we define the value* $t^{\mathbf{A},\beta} = (\llbracket t\rrbracket^{\mathbf{A},\beta}, \mathrm{ann}_t^{\mathbf{A},\beta}) \in \mathrm{SHF}(A)$ *as follows:*

- $\emptyset^{\mathbf{A},\beta} = (\emptyset, \emptyset \mapsto 1)$, *and* $x^{\mathbf{A},\beta} = \beta(x)$ *for any variable* x.
- *For* $t = \mathrm{Atoms}\,.\varphi$ *with* $\varphi \in \mathrm{FO}[\sigma]$, *let* $\llbracket t\rrbracket^{\mathbf{A},\beta} = \{a : \mathbf{A} \models \varphi(a)\}$ *if* $|\mathrm{free}(\varphi)| = 1$, *and* $\llbracket t\rrbracket^{\mathbf{A},\beta} = \{\{\langle \mathrm{bitset}(1), a_1\rangle, \dots, \langle \mathrm{bitset}(k), a_k\rangle\} : \mathbf{A} \models \varphi(a_1, \dots, a_k)\}$ *otherwise. In both cases* $\mathrm{ann}_t^{\mathbf{A},\beta}$ *is generated by mapping every occurring atom* a *to* $\log|\llbracket t\rrbracket^{\mathbf{A},\beta}|$.
- *If* $t = \mathrm{Pair}(t_1, t_2)$, *then* $t^{\mathbf{A},\beta} = \{{t_1}^{\mathbf{A},\beta}, {t_2}^{\mathbf{A},\beta}\}_{\mathrm{SHF}}$.
- *If* $t = \mathrm{Unique}(t')$, *then* $t^{\mathbf{A},\beta} = (a, \mathrm{ann}_{t'}^{\mathbf{A},\beta} \restriction \mathrm{tc}(a))$ *if* $\llbracket t'\rrbracket^{\mathbf{A},\beta} = \{a\}$ *and* $t^{\mathbf{A},\beta} = (\emptyset, \emptyset \mapsto 1)$ *otherwise.*

- *If* $t = \text{Union}(t')$*, then* $[\![t]\!]^{\mathbf{A},\beta} = \bigcup_{b \in [\![t']\!]^{\mathbf{A},\beta}} b$ *and* $\text{ann}_t^{\mathbf{A},\beta} = \text{ann}_{t'}^{\mathbf{A},\beta} \upharpoonright [\![t]\!]^{\mathbf{A},\beta}$.
- *If* $t = \text{Card}(t')$*, then* $[\![t]\!]^{\mathbf{A},\beta} = \text{bitset}\left(\left|[\![t']\!]^{\mathbf{A},\beta}\right|\right)$, *and* $\text{ann}_t^{\mathbf{A},\beta} = \text{ann}_{[\![t]\!]^{\mathbf{A},\beta}}$.
- *If* $t = \{p : x \in q : \varphi\}$*, then* $t^{\mathbf{A},\beta} = \{p^{\mathbf{A},\beta[x \mapsto a]} : a \in q^{\mathbf{A},\beta} : \mathbf{A}, \beta[x \mapsto a] \models \varphi\}_{\text{SHF}}$.

For a σ*-formula* φ*, the truth value of* $\mathbf{A}, \beta \models \varphi$ *is defined in the obvious way.*

We denote the size $\text{ann}_t^{\mathbf{A},\beta}([\![t]\!]^{\mathbf{A},\beta})$ assigned to the value of a term t by $\|t\|^{\mathbf{A},\beta}$. Further, we write true^k to denote a tautology with k free variables, true for true^1 and Atoms instead of Atoms . true. Then Atoms . true^k defines the set of all k-tuples over the input structure.

Iteration terms. Ordinary terms and formulae cannot define arbitrarily nested sets. This is achieved by constructing sets using fixed-point iteration.

▶ **Definition 8** (Syntax of iteration terms). *If* u *and* x *are variables and* t *is a term, then* $[t_u]^*(x)$ *is an* iteration term *with free variables* $\{x\} \cup \text{free}(t) \setminus \{u\}$.

The fixed-point iteration starts with the object given by the free variable x, and the variable u is usually a free variable of t, whose value is the previous stage of the iteration. To ensure LOGSPACE-computability, the fixed point is computed only if the logarithmic bound given as part of the term is satisfied by all intermediate stages. The value of an iteration term is not the fixed point itself, but the set of all stages up to that point.

▶ **Definition 9** (Semantics of iteration terms). *The* i*-fold iteration of a term* s *with free variable* u *is defined by induction as* $s^0 = s[u/x]$ *and* $s^{i+1} = s[u/s^i]$*, where* $s[u/t]$ *results from replacing every occurrence of* u *in* s *by the term* t.

Now let $t = [s_u]^*(x)$ *be an iteration term, let* $\mathbf{A}$ *be a structure and* $\beta\colon \text{free}(t) \to \text{SHF}(A)$ *a variable assignment. Then*

$$[\![t]\!]^{\mathbf{A},\beta} = \begin{cases} \{(s^i)^{\mathbf{A},\beta} : i \leq \ell\}_{\text{SHF}}, & \text{for the least } \ell \text{ with } s^\ell = s^{\ell+1} \\ & \text{and } \|s^i\|^{\mathbf{A},\beta} \leq f(|A|) \text{ for all } i \leq \ell \\ & \text{if such an } \ell \text{ exists,} \\ (\emptyset, \emptyset \mapsto 1), & \text{otherwise.} \end{cases}$$

The differences between this definition and iteration in CPT are explained by two important properties of LOGSPACE-algorithms: Firstly, a LOGSPACE-algorithm can be sequentially executed for different input values, e.g. in a `forall`-loop. Iteration terms allow to do this in parallel for different values of the free variable x using suitable comprehension terms. Secondly, defining the value of the iteration term as the aggregation of all intermediate stages allows to generate outputs of polynomial size, which can be the input for other iteration terms.

Recursion formulae. With recursion formulae, we basically add to our logic the lrec-operator from [17, 18]. This operator permits a restricted kind of recursion over a graph interpretable in the input structure. Since our logic operates on hereditarily finite sets, we modify the interpretation such that the domain of the interpreted structure can be any definable set. Recursion formulae contain a term t_δ that restricts the domain to finitely many objects and defines their size annotations. Further, the vertices of the graph are labelled by sets of natural numbers, which we represent as bitsets instead of elements of a distinct number sort.

▶ **Definition 10** (Syntax of recursion formulae). *If* t_δ *and* t_C *are terms and* φ_E *and* $\varphi_=$ *are formulae, then* $[\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x, y)$ *is a* recursion formula.

For the definition of the semantics, fix a σ-structure $\mathbf{A}$, a variable assignment β, and a recursion formula $\varphi = [\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x, y)$. We first describe how t_δ, $\varphi_=$, φ_E and t_C define a labelled graph $G = (V, E, C)$ interpreted in the input structure. The formulae $\varphi_=$ and φ_E will be evaluated for pairs of elements from $[\![t_\delta]\!]^{\mathbf{A},\beta}$. For that purpose, we define the extension $\beta_{a,b} := \beta[u \mapsto (a, \mathrm{ann}_{t_\delta}^{\mathbf{A},\beta} \restriction \mathrm{tc}(a)), v \mapsto (b, \mathrm{ann}_{t_\delta}^{\mathbf{A},\beta} \restriction \mathrm{tc}(b))]$ of β for $a, b \in [\![t_\delta]\!]^{\mathbf{A},\beta}$.

The vertex set V consists of the equivalence classes of the reflexive, symmetric, transitive closure of the relation $\{(a, b) \in [\![t_\delta]\!]^{\mathbf{A},\beta} \times [\![t_\delta]\!]^{\mathbf{A},\beta} : \mathbf{A}, \beta_{a,b} \models \varphi_=\}$. Let $[a]$ denote the equivalence class of a. The edge set of G is $E := \{([a], [b]) : \mathbf{A}, \beta_{a,b} \models \varphi_E\}$. Further, the label of $[a] \in V$ is the set $C([a]) = \{c :$ there exists $a' \in [a]$ with $\mathrm{bitset}(c) \in [\![t_C]\!]^{\mathbf{A},\beta_{a'}}\}$, where $\beta_{a'} = \beta[u \mapsto (a', \mathrm{ann}_{t_\delta}^{\mathbf{A},\beta} \restriction \mathrm{tc}(a'))]$.

The truth value of the recursion formula depends on a recursive property X of vertices of a DAG unfolding of G. This unfolding is obtained by augmenting every vertex $[a] \in V$ with a resource $\ell \in \mathbb{N}$ restricting the space necessary to process paths through the DAG (for a detailed explanation, see [17, 18]). Then $([a], \ell) \in X \subseteq V \times \mathbb{N}$ if, and only if,

$$\ell > 0 \text{ and } \mathrm{bitset}\left(\left|\left\{[b] \in [a]E : \left([b], \left\lfloor \frac{\ell - 1}{|E[b]|} \right\rfloor\right) \in X\right\}\right|\right) \in C([a]),$$

If the values of the free variables x, y define a pair in $V \times \mathbb{N}$ (i.e. a vertex of the DAG), then the membership of that pair in X determines the truth value of the recursion formula:

▶ **Definition 11** (Semantics of recursion formulae). *Let $\varphi, \mathbf{A}, \beta$ and X as above. Then $\mathbf{A}, \beta \models \varphi$, where $\beta(x) = (a, \mathrm{ann}_{t_\delta}^{\mathbf{A},\beta} \restriction \mathrm{tc}(a))$ and $\beta(y) = (\mathrm{bitset}(\ell), \mathrm{ann}_{\mathrm{bitset}(\ell)})$ for $\ell \in \mathbb{N}$, if, and only if, $([a], \ell) \in X$.*

This completes the definition of CLogspace. We illustrate it with two short examples.

▶ **Example 12** (Size annotations). To see that the size annotation of a term can map the atoms to different values, consider the term $\mathrm{Union}(\mathrm{Pair}(\mathrm{Atoms}\,.\,\mathrm{true}, \mathrm{Atoms}\,.Px))$. The value of that term in a structure $\mathbf{A}$ is its whole domain. But the size annotation maps every element of $P^{\mathbf{A}}$ to $\log |P^{\mathbf{A}}|$, and the other atoms to $\log |A|$.

To demonstrate the use of iteration terms, we construct a formula $\varphi_{\mathrm{dtc}}(x, y)$ defining that there is a deterministic path from x to y.

▶ **Example 13** (Deterministic paths). The core of the formula is an iteration term $[t_u]^*(x)$ progressing along the deterministic path starting from x:

$$t(u) = \mathrm{Unique}\left(\{z : z \in \mathrm{Atoms}\,.\,\mathrm{true} : Euz\}\right).$$

Note that the free variable x determines the value of the first stage. So, by definition of Unique, the ith stage of $[t_u]^*(x)$ is the ith element on the deterministic path starting at x. Every stage is added to the value of the term, so the desired formula is $\varphi_{\mathrm{dtc}}(x, y) = y \in [t_u]^*(x)$. Since every atom is initially defined with the term $\mathrm{Atoms}\,.\,\mathrm{true}$, it is mapped to $\log |A|$ by the size annotation for the structure $\mathbf{A}$. Thus $n \mapsto \log n$ is a suitable bound for the formula.

Choiceless Polynomial Time. The newly introduced features that distinguish Choiceless Logspace from Choiceless Polynomial Time are size annotations, the addition of recursion formulae and the semantics of iteration terms. Further, CPT-formulae clearly possess polynomial instead of logarithmic bounds.

Consequently, the definition of CPT-formulae can be recovered from our definition of CLogspace by the following modifications: The terms of the form "$\mathrm{Atoms}\,.\varphi$" are replaced by an atomic term "Atoms", defining the full set of atoms from the input structure. The rule for recursion formulae is omitted. In terms (t, f) and formulae (φ, f), the function f is a polynomial.

In the definition of the semantics, the size annotations are eliminated completely. The polynomial bound for iteration terms is applied to the cardinality of the transitive closure of the current stage. For the computation to remain within polynomial time, the number of stages is also bounded by the polynomial. The value of an iteration term is the stage inducing the fixed-point instead of the accumulation of all intermediate stages. For more detailed definitions of Choiceless Polynomial Time, we refer to, for instance, [25, 9].

3 Inclusion in Logspace and CPT

We next prove the two essential properties of our logic that justify it to be called Choiceless Logspace: Firstly, every formula in our logic can be evaluated in logarithmic space. Secondly, it is choiceless in the sense that every formula can be translated to Choiceless Polynomial Time, the prototypical model of choiceless computation.

▶ **Theorem 14.** *Every query definable in* CLogspace *is* LOGSPACE*-computable.*

The explicit logarithmic bound in CLogspace-sentences is intended to enable evaluation in LOGSPACE. It ensures that every value occurring during an iterated computation has a sufficiently small size annotation. The size annotations induced by the semantics of terms and formulae are defined with a certain encoding of hereditarily finite sets in mind. We first define this encoding, which is the core of our evaluation algorithm.

Recall that elements of the input structure are always defined by means of terms $\text{Atoms}\,.\varphi$. Since the algorithm will take as input string encodings of structures, we now operate on ordered structures. Hence one can represent an atom as "the ith entry of the kth tuple satisfying φ" using the linear order on the input structure. We denote the set of k-tuples from a structure $\mathbf{A}$ satisfying an FO-formula φ with k free variables by $\varphi^{\mathbf{A}}$.

▶ **Definition 15** (Representations)**.** *Let $\Phi \subseteq \mathrm{FO}[\sigma]$ for some vocabulary σ. The alphabet Σ_Φ consists of 0,1, "(", ")", "{", "}", "," and an alphabet symbol for each $\varphi \in \Phi$, which we also denote by φ. A* Φ-representation *is a word in the alphabet Σ_Φ that is either a set representation or an atom representation:*

- *An* atom representation *is a word (φ, i, m), where $\varphi \in \Phi$ with $k \geq 1$ free variables, and i and m are binary encodings of natural numbers $\leq k$ and $\leq |\varphi^{\mathbf{A}}|$, respectively. If $|\varphi^{\mathbf{A}}| = 0$, then m is the empty string.*
- *If $r_1, \dots, r_k$ are representations, then the word $\{r_1, \dots, r_k\}$ is a* set representation.

Let $\mathbf{A}$ be the expansion of a σ-structure $\mathbf{A}'$ by a linear order $<^{\mathbf{A}}$. The value *$r^{\mathbf{A}}$ of a representation r is defined as*

- *$(\varphi, i, m)^{\mathbf{A}} = a_i$ where $(a_1, \dots, a_k)$ is the m-th tuple (w.r.t. the lexicographical order extending $<$) in $\varphi^{\mathbf{A}}$.*
- *$\{r_1, \dots, r_k\}^{\mathbf{A}} = \{r_1^{\mathbf{A}}, \dots, r_k^{\mathbf{A}}\}$.*

A representation r is $\mathbf{A}$-minimal if

1. *for every atom a in $\mathrm{tc}(r^{\mathbf{A}})$, there is a unique atom representation of a that occurs as a substring of r,*
2. *if $r = \{r_1, \dots, r_k\}$, then $r_1, \dots, r_k$ are minimal representations and the values $r_i^{\mathbf{A}}$ are pairwise distinct.*

For an $\mathbf{A}$*-minimal representation* r*, we define the size annotation* $\text{ann}_r^{\mathbf{A}}$ *by induction on the representation* r*:*

- $\text{ann}_{(\varphi,i,m)}^{\mathbf{A}} \colon (\varphi,i,m)^{\mathbf{A}} \mapsto \log|\varphi^{\mathbf{A}}|$,
- $\text{ann}_{\{r_1\dots r_k\}}^{\mathbf{A}}$ *is the size annotation of* $\{(r_1^{\mathbf{A}}, \text{ann}_{r_1}^{\mathbf{A}}),\dots,(r_k^{\mathbf{A}}, \text{ann}_{r_k}^{\mathbf{A}})\}_{\text{SHF}}$.

We say that r *is a representation of* $(r^{\mathbf{A}}, \text{ann}_r^{\mathbf{A}}) \in \text{SHF}(A)$.

We omit the structure $\mathbf{A}$ and speak about minimal representations if $\mathbf{A}$ can be inferred from the context. Note that there are representations that are not minimal, since the same set may be listed twice or an atom may occur encoded by different atom representations throughout the transitive closure of the value. Thus minimal representations ensure that the size annotation is well-defined. Note further that the size annotation does not agree with the word length of the representation. It does, however, provide an upper bound on the word length up to constant factors. Since the size annotation does not depend on the specific numbers i and m, this upper bound is independent of the linear order. Moreover, the size annotation of any given minimal representation is LOGSPACE-computable.

Towards the evaluation algorithm, we show how to check whether two representations have the same value and, using the former as a subroutine, how to compute minimal representations.

▶ **Lemma 16.** *There is an algorithm that, given a* $\sigma \cup \{<\}$*-structure* $\mathbf{A}$ *(where* $<^{\mathbf{A}}$ *is obtained from the string representation of* $\mathbf{A}$*) and* $\mathbf{A}$*-minimal representations* r_1, r_2*, decides in logarithmic space whether* $r_1^{\mathbf{A}} = r_2^{\mathbf{A}}$*.*

Proof. We reduce equality of representations to isomorphism of coloured trees, where the colours are numbers $1,\dots,|A|$. An atom representation is encoded by a single node coloured by its value, and a set representation is encoded by a tree where the subtrees below the root encode the elements of the set representation. Note that, since the representations are minimal, every node has at most one child of any isomorphism type, so equality of the representations indeed implies isomorphism of the corresponding trees. ◀

▶ **Lemma 17.** *There is an algorithm that, given a* $\sigma \cup \{<\}$*-structure* $\mathbf{A}$ *(where* $<^{\mathbf{A}}$ *is obtained from the encoding of* $\mathbf{A}$*) and* $\mathbf{A}$*-minimal representations* $r_1,\dots,r_k$*, computes in logarithmic space a minimal representation* r *of the size-annotated set* $\{(r_1^{\mathbf{A}}, \text{ann}_{r_1}^{\mathbf{A}}),\dots,(r_k^{\mathbf{A}}, \text{ann}_{r_k}^{\mathbf{A}})\}_{\text{SHF}}$.

Proof. We sequentially execute two LOGSPACE-algorithms. The first one copies those representations to the output tape whose value did not occur before (using the algorithm from the previous lemma). The second algorithm replaces every atom representation by the shortest representation of its value occurring in the original string. ◀

With these subroutines, the evaluation of terms and formulae becomes rather straightforward. To simplify the statement of the following lemma, we do not distinguish between terms and formulae, but assume formulae to be terms with values $\emptyset$ and $\{\emptyset\}$.

▶ **Lemma 18.** *For every term* t *with free variables* $x_1,\dots,x_k$ *and every function* $f \colon n \mapsto c\log n$*, there is a* LOGSPACE *algorithm that, given a* σ*-structure* $\mathbf{A}$ *and representations* $r_1,\dots,r_k$ *of* $\alpha_1,\dots,\alpha_k \in \text{SHF}(A)$*, computes a representation of* $(t,f)^{\mathbf{A},\beta\colon x_i\mapsto\alpha_i}$*.*

Proof. We proceed by induction on the construction of terms. Evaluation of atomic terms $\emptyset$ and x is trivial. For terms of the form $\text{Atoms}\,.\varphi$, it suffices to compute the number n of tuples satisfying the FO-formulae φ and enumerate all tuples with entries (φ,i,m) for $0 \le m \le n$.

In the induction step, most cases follow directly from the induction hypothesis, using the fact that LOGSPACE is closed under sequential execution and thus the output of previous computations can be processed further regardless of its size. For Pair, Union and comprehension, the results of the subcomputations can be converted to a minimal representation with the algorithm from Lemma 17. Formulae $t_1 = t_2$ and $t_1 \in t_2$ can be evaluated using the algorithm from Lemma 16. For recursion formulae, the lemma follows from inclusion of LREC in LOGSPACE.

Let $[t_u]^*$ be an iteration term with free variable x_j. In its ith iteration, the algorithm computes a representation of $(t^i)^{\mathbf{A},\beta}$ from $(t^{i-1})^{\mathbf{A},\beta}$ (which is possible by induction hypothesis) and checks whether it satisfies the size bound. This is again possible using sequential execution. The new stage is compared to the previous one. Whenever a new value has been produced, it is additionally written to the output tape. A postprocessing step converts the result to a minimal representation and replaces it by the empty set if the bound has been exceeded at some point.

The bound on the size of each stage induces a logarithmic bound on the word length of the representations that are written to the work tape. In particular, this implies that the number of possible representations occurring as stages is polynomial. This ensures termination, since a counter can track the number of values that have been produced. ◀

We have established that CLogspace is a fragment of LOGSPACE. To verify that this fragment is *choiceless*, we embed it in CPT.

▶ **Theorem 19.** CLogspace $\leq$ CPT.

The straightforward way to prove the theorem is to inductively translate CLogspace-formulae to CPT. But the translation of formulae with free variables bears two technical difficulties: Firstly, CLogspace is evaluated over size-annotated hereditarily finite set. Secondly, standard definitions of CPT do not allow free variables in iteration terms.

We can, however, assume a non-standard variant of CPT where iteration terms may have free variables. This does not increase its expressive power because iteration terms can be simulated for all possible values of the free variables at once without exceeding the polynomial bound.

We address the first difficulty by the key concept of the translation to CPT: Expressing size annotations as hereditarily finite sets.

▶ **Definition 20.** *Let $\mathbf{A}$ be a σ-structure, and let $(a, s) \in \mathrm{SHF}(A)$. The* set representation $\mathrm{set}((a, s)) \in \mathrm{HF}(A)$ *is the set $\langle a, \{\langle b, [s(b)]\rangle : b \in \mathrm{tc}(a) \cap A\}\rangle$, where $[s(b)]$ is the ordinal corresponding to $s(b)$. We say that $\{\langle b, s(b)\rangle : b \in \mathrm{tc}(a) \cap A\}$* represents the size annotation *s.*

Proof of Theorem 19. Set representations of size annotations induce a translation between variable assignments, and thus a meaningful definition of translation between formulae with free variables. It remains to show that every CLogspace-formula can be (inductively) translated to CPT in that sense.

Most cases follow directly from the induction hypothesis, using CPT-terms that define combinations of size annotations. Size annotations are combined by using the minimal known size for every atom, which is definable in CPT.

In case a size annotation is not derived from the subterms, it originates from a term of the form "Atoms $.\varphi$". So the corresponding CPT-term has to define the size annotation that assigns the value $\log|\varphi^{\mathbf{A}}|$ to every atom from the structure $\mathbf{A}$ occurring in the value of the term. This is possible since logarithms are CPT-definable.

Further, CPT can check the logarithmic bound of iteration terms using the size annotation at each stage. Finally, recursion formulae can be translated to CPT because LREC $\leq$ FPC $\leq$ CPT. This concludes the proof of Theorem 19. ◀

4 The expressive power of CLogspace

To substantiate the idea that the rich syntax of CLogspace leads to greater expressive power, we compare it to known logics below LOGSPACE. As classical benchmarks, we consider the logics DTC and STC, which augment FO by deterministic and symmetric transitive closure operators. For precise definitions, we refer to [14, Sect. 3.5.2]. These logics are known [17] to be included in LREC, which is the strongest logic inside LOGSPACE known so far. Hence our main theorem of this kind is:

▶ **Theorem 21.** CLogspace $\gneqq$ LREC.

LREC is a logic with counting terms, so it features a linearly ordered number sort and terms defining the cardinality of definable sets of tuples. Its core is the recursion operator, a modified version of which appears in our logic. We describe the lrec-operator in terms of the differences to our version, and refer to [17] for the full definition. Recall that the recursion operator interprets a graph in the input structure. In the original version, the elements of that graph are k-tuples from the input structure instead of hereditarily finite sets. Consequently, the domain term t_δ does not occur, and the subformulae can have multiple free variables. Further, the labels of the graph are defined as tuples over the number sort instead of bitsets. Both for counting of tuples and defining the labels of the graph, LREC uses a LOGSPACE-computable encoding of numbers polynomial in the size of the input structure by fixed-size tuples over the number sort. To verify that the LOGSPACE-computable arithmetic operations used in LREC are CLogspace-definable, we show:

▶ **Proposition 22.** *Let $R \subseteq \{0, \dots, |A|^k\}^\ell$ be any* LOGSPACE*-computable relation. Then there is a* CLogspace*-term defining $\{\langle \text{bitset}(n_1), \dots, \text{bitset}(n_\ell)\rangle : (n_1, \dots, n_\ell) \in R\}$.*

Proof. By implementing basic bit operations in CLogspace, the successor of any natural number is definable in the bitset encoding. This makes it possible to define a linear order on all bitsets up to any definable number. The elements of the linear order are bitsets of logarithmic size, so DTC-formulae can be simulated on that order analogously to Example 13. As DTC captures LOGSPACE on ordered structures, the proposition follows. ◀

The different encodings of numbers also pose a difficulty when translating formulae with free variables, since these variables can be domain or number variables. Defining a translation between variable assignments, inclusion of LREC in CLogspace can be made precise on the level of formulae with free variables. The translation to CLogspace is then rather straightforward. In particular, the only necessary iteration terms are those defining the operations on the number sort, so there exists an appropriate logarithmic bound.

So LREC is included in CLogspace. The inclusion is strict because of a property that CLogspace shares with CPT but not with traditional logics such as FPC and LREC: It benefits from padding of the input structure, i.e. it can define all LOGSPACE-properties of sufficiently small, definable substructures.

▶ **Definition 23.** *A $\sigma \cup \{U\}$-structure $\mathbf{A}$ is a* padded σ-structure *if $U \notin \sigma$ is a unary predicate. The* underlying structure *of $\mathbf{A}$ is the σ-reduct of $U^{\mathbf{A}}$.*

▶ **Theorem 24.** *Let $\mathcal{C}$ be the class of padded structures such that $2^{u!(u^2(6\log u+2)+1)} \leq n$ for every structure in $\mathcal{C}$ of size n with underlying structure of size u. For every* Logspace*-computable property $\mathcal{P}$, there is a* CLogspace*-sentence (φ, f) such that any structure in $\mathcal{C}$ satisfies (φ, f) if, and only if, the underlying structure satisfies $\mathcal{P}$.*

Proof. We define the set of all linear orders on the underlying structure. Then, since DTC $\leq$ LREC $\leq$ CLogspace and DTC captures Logspace on ordered structures, every Logspace-computable property can be defined on the resulting ordered structures.

Initially, all linear orders on two elements are created, defining pairs of atoms with the term $\text{Atoms}\,.(Ux \wedge Uy \wedge x \neq y)$. Then an iteration term extends every linear order by every atom that does not occur yet, defined by $\text{Atoms}\,.Uy$. It remains to show that there is a logarithmic bound for the iteration term. Its value in the ith stage is the set of all orders of the form $\{\langle a, b\rangle : a, b \in V : a < b\}$ for subsets $V \subseteq U$ of size $i + 2$. All atoms are defined by the terms $\text{Atoms}\,.Ux$ and $\text{Atoms}\,.(Ux \wedge Uy \wedge x \neq y)$, so every atom is annotated with a size $\leq 2\log|U|$. Then the pair $\langle a, b\rangle = \{a, \{a, b\}\}$ is mapped to $6\log|U| + 2$. Every linear order contains $< |V|^2 \leq |U|^2$ many pairs of that form, and there are $|V|! \leq |U|!$ many linear orders in the ith iteration. So $\|t^i\|^{\mathbf{A},\beta} \leq (|U|^2 \cdot (6\log|U| + 2) + 1) \cdot |U|! + 1$. By assumption, this is logarithmic in the size of the full input structure. ◀

Since the CFI-query is in Logspace, it is CLogspace-definable on padded structures. But LREC is included in FPC [18], so it cannot define this query even in the presence of padding. Thus Theorem 24 implies that CLogspace is strictly more expressive than LREC. It follows that also DTC and STC, and their extensions with counting, are strictly included in CLogspace. The same holds for all models of choiceless computation with a constant bound on the number of objects in the transitive closure of a set.

Even though our results demonstrate that CLogspace is stronger than all previously studied logics for Logspace, we can show that it does not capture all of Logspace. To establish this result we use a technique for proving inexpressibility results for fragments of CPT that is based on *supports* of hereditarily finite objects (see e.g. [3, 8, 26]).

▶ **Definition 25.** *Let $\text{Aut}(\mathbf{A})$ be the automorphism group of a finite structure $\mathbf{A}$. For $a \in \text{HF}(A)$, let $\text{Stab}(a)$ be the stabiliser of a with respect to $\text{Aut}(\mathbf{A})$, and let $\text{Stab}^\bullet(a)$ be its point-wise stabiliser. An object $s \in \text{HF}(A)$ is a* support *of a if $\text{Stab}^\bullet(s) \subseteq \text{Stab}(a)$.*

Every CPT-formula can be translated to an FPC-formula over a substructure of $\text{HF}(\mathbf{A})$ containing the sets *activated* by the formula. For general CPT-formulae, this can be an arbitrary substructure of $\text{HF}(\mathbf{A})$, depending on both the CPT-formula and the input structure. But for some fragments of CPT, the substructure can be over-approximated by those hereditarily finite objects that have a sufficiently small support. In particular, CLogspace-formulae can be translated to FPC-formulae that are evaluated over hereditarily finite sets with a support of logarithmic size.

▶ **Lemma 26.** *For a structure $\mathbf{A}$ and $f : \mathbb{N} \to \mathbb{N}$, let $\text{HF}_f(\mathbf{A})$ be the substructure of $\text{HF}(\mathbf{A})$ containing exactly those sets that have a support of size $f(n)$, where n is the size of the domain of $\mathbf{A}$. For every* CLogspace*-formula (φ, f), there is an* FPC*-formula ψ such that, for every structure $\mathbf{A}$, $\mathbf{A} \models (\varphi, f)$ if, and only if, $\text{HF}_f(\mathbf{A}) \models \psi$.*

Proof. Since the stages of all iteration terms are of size at most $f(n)$ with respect to their size annotations, they are elements of $\text{HF}_f(\mathbf{A})$. Analogously to the translation of CPT-terms to interpretations in an extension of FO in [10], every ordinary term or formula in CLogspace can be translated to an LREC-interpretation. Iteration terms can be translated to FPC-interpretations that create new objects representing the value of the iteration term, i.e. the aggregation of all stages. The element relation between that new element and the stages can then be defined by a fixed-point formula, maintaining the size annotations as a separate relation. ◀

By the same arguments as used in the proof of Theorem 40 in [8], this implies that the Cai-Fürer-Immerman query cannot be defined in CLogspace. As a consequence we get

▶ **Theorem 27.** CLogspace *is a strict fragment of* Logspace.

5 Discussion

We have introduced Choiceless Logarithmic Space formalising the notion of symmetric, choiceless computations using only logarithmic workspace. Through the notion of size-annotated hereditarily finite sets, our logic takes into account that sizes of objects in Logspace are sensitive to their encoding. The logic is a fragment of both Choiceless Polynomial Time and Logspace, and it captures Logspace on certain classes of structures with padding. This demonstrates the similarity to CPT, and, more importantly, makes it stronger than all previously known logics in Logspace, in particular LREC.

However, we have seen that CLogspace does not capture Logspace because it can only define sets with a support of logarithmic size, which, as shown in [8], makes it impossible to define the Cai-Fürer-Immerman query. It remains open whether the concepts used in CLogspace can be used to obtain a stronger logic that could capture all queries in Logspace.

Of course, our definition can be tuned in several ways, addressing issues such as closure under interpretations, or trying to avoid the use of two different kinds of recursion operators (iteration terms and recursion formulae). Even for the current definition of CLogspace, it is open whether the lrec-operator can be eliminated without losing expressive power. It can, however, be shown that iteration terms are necessary (see [26]). However, contrary to polynomial time, logspace complexity is a much less robust notion, and the conflicting goals of providing sufficient power for choiceless computation and remaining inside Logspace necessarily seem to result in a rather involved construction of CLogspace. It seems difficult to overcome, with further tunings of the definition, the barrier that the definable sets in CLogspace have a (far too) small support. In the light of the research carried out here, we therefore consider it improbable that some variant of choiceless computation can capture precisely all (symmetry-invariant) Logspace-queries.

We may take a step back, and reconsider our general objectives. One of the reasons why we are looking for a logic that captures logspace complexity on arbitrary finite structures is that Logspace, other than being a well-studied complexity class for standard computation models on ordered objects, is a reasonable formalisation of efficient computation for big data. But there are also other, less restrictive, complexity classes that can serve a similar purpose, for instance on the basis of polylogarithmic space bounds. Such variants may be more robust and make it possible to assume a standard encoding of atoms with small space and might grant more freedom in defining tree-like recursion using only iteration terms. Rather than trying to tune the definition of CLogspace to find stronger notions of choiceless computation inside Logspace, we may thus also go beyond the logspace bound and ask more generally: *"What is efficient choiceless computation for big data?"*.

References

1 F. Abu Zaid, E. Grädel, M. Grohe, and W. Pakusa. Choiceless Polynomial Time on structures with small Abelian colour classes. In *MFCS 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 50–62. Springer, 2014.

2 A. Atserias, A. Bulatov, and A. Dawar. Affine Systems of Equations and Counting Infinitary Logic. *Theoretical Computer Science*, 410:1666–1683, 2009.

3 A. Blass, Y. Gurevich, and S. Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1-3), 1999.

4 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

5 A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1), 1982.

6 A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

7 A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with Rank Operators. In *Proc. 24th IEEE Symp. on Logic in Computer Science (LICS 09)*, pages 113–122, 2009.

8 A. Dawar, D. Richerby, and B. Rossman. Choiceless Polynomial Time, Counting and the Cai-Fürer-Immerman Graphs. *Annals of Pure and Applied Logic*, 152:31–50, 2009.

9 E. Grädel and M. Grohe. Is Polynomial Time Choiceless? In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, pages 193–209, 2015.

10 E. Grädel, Ł. Kaiser, W. Pakusa, and S. Schalthöfer. Characterising Choiceless Polynomial Time with First-Order Interpretations. In *LICS*, 2015.

11 E. Grädel and M. Otto. Inductive Definability with Counting on Finite Structures. In *Computer Science Logic, CSL 92*, volume 702 of *Lecture Notes in Computer Science*, pages 231–247. Springer, 1992.

12 E. Grädel and W. Pakusa. Rank logic is dead, long live rank logic! *Journal of Symbolic Logic*, To appear, 2019. http://arxiv.org/abs/1503.05423.

13 E. Grädel and M. Spielmann. Logspace Reducibility via Abstract State Machines. In J. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods (FM '99)*, volume 1709 of *LNCS*. Springer, 1999. URL: `http://www.logic.rwth-aachen.de/pub/graedel/GrSp-fm99.ps`.

14 E. Grädel et al. *Finite Model Theory and Its Applications*. Springer-Verlag, 2007.

15 M. Grohe. The quest for a logic capturing PTIME. In *Proceedings of the 23rd IEEE Symposium on Logic in Computer Science (LICS'08)*, pages 267–271, 2008.

16 M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press, 2017.

17 M. Grohe, B. Grußien, A. Hernich, and B. Laubner. L-Recursion and a new Logic for Logarithmic Space. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 12. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2011.

18 B. Grußien. *Capturing Polynomial Time and Logarithmic Space using Modular Decompositions and Limited Recursion*. PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2017. `doi:10.18452/18548`.

19 Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Current Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, 1988.

20 N. Immerman. Expressibility as a complexity measure: results and directions. In *Structure in Complexity Theory*, pages 194–202, 1987.

21 N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.

22 W. Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen University, 2016.

23 W. Pakusa, S. Schalthöfer, and E. Selman. Definability of Cai-Fürer-Immerman Problems in Choiceless Polynomial Time. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, pages 19:1–19:17, 2016.
24 O. Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.
25 B. Rossman. Choiceless Computation and Symmetry. In A. Blass, N. Dershowitz, and W. Reisig, editors, *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *LNCS*, pages 565–580. Springer, 2010.
26 S. Schalthöfer. *Choiceless Computation and Logic.* PhD thesis, RWTH Aachen University, 2018.
27 M. Spielmann. *Abstract state machines: Verification problems and complexity.* PhD thesis, RWTH Aachen University, 2000.