2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science

Characterising Choiceless Polynomial Time with First-Order Interpretations

Erich Grädel, Wied Pakusa, and Svenja Schalthöfer
RWTH Aachen University
{graedel,schalthoefer,pakusa}@logic.rwth-aachen.de

Łukasz Kaiser
LIAFA, CNRS & Université Paris 7 (currently at Google Inc.)
lukaszkaiser@gmail.com

***Abstract*—Choiceless Polynomial Time (CPT) is one of the candidates in the quest for a logic for polynomial time. It is a strict extension of fixed-point logic with counting, but to date the question is open whether it expresses all polynomial-time properties of finite structures. We present here alternative characterisations of Choiceless Polynomial Time (with and without counting) based on iterated first-order interpretations.**

The fundamental mechanism of Choiceless Polynomial Time is the manipulation of hereditarily finite sets over the input structure by means of set-theoretic operations and comprehension terms. While this is very convenient and powerful for the design of abstract computations on structures, it makes the analysis of the expressive power of CPT rather difficult. We aim to reduce this functional framework operating on higher-order objects to an approach that evaluates formulae on less complex objects.

We propose a more model-theoretic formalism, called polynomial-time interpretation logic (PIL), that replaces the machinery of hereditarily finite sets and comprehension terms by traditional first-order interpretations, and handles counting by Härtig quantifiers. In our framework, computations on finite structures are captured by *iterations of interpretations*, and a run is a sequence of states, each of which is a finite structure of a fixed vocabulary. Our main result is that PIL has precisely the same expressive power as Choiceless Polynomial Time.

We also analyse the structure of PIL and show that many of the logical formalisms or database languages that have been proposed in the quest for a logic for polynomial time reappear as fragments of PIL, obtained by restricting interpretations in a natural way (e.g. by omitting congruences or using only one-dimensional interpretations).

I. Introduction

The quest for a logic for PTIME is one of the most important and fundamental challenges in the field of finite model theory (see e.g. [1], [2]). It is equivalent to the classical question in database theory, posed by Chandra and Harel [3], whether there is a query language on relational databases expressing precisely the polynomial time computable queries. [1] The study of this question motivated the definition of more and more expressive languages that capture interesting and natural levels of polynomial-time computability. A central logic of reference for this quest is fixed-point logic with counting (FPC) which captures PTIME on many important classes of finite structures such as planar graphs, structures of bounded tree-width, and, indeed, every class of graphs which excludes a fixed graph as a minor [4]. Further, this logic is powerful enough to express most of the common polynomial-time properties of finite structures as well as fundamental algorithmic techniques including, for instance, by a recent result of Anderson, Dawar and Holm, the ellipsoid method for linear programs [5]. More specifically, the aforementioned classes even admit FPC-*definable canonisation* which means that FPC can define, given an input structure, an isomorphic copy of that structure over a linearly ordered universe. Clearly, if a class of structures admits FPC-definable canonisations, then FPC captures PTIME on this class, since by the Immerman-Vardi Theorem (see e.g. [1]) fixed-point logic can define every polynomial-time query on ordered structures.

On the other hand, FPC fails to capture PTIME in general, which was shown by the CFI-construction of Cai, Fürer and Immerman [6]. This has motivated the search for logics with polynomial-time data complexity that are more expressive than FPC. Given our current knowledge, the two main sources of problems that separate PTIME from FPC are tractable cases of the graph isomorphism problem and queries from the field of linear algebra. First of all, although polynomial-time graph isomorphism tests are known for graphs of bounded degree or bounded colour class size, the CFI-construction shows that FPC cannot define the graph isomorphism problem even on classes of graphs where both the degree and the colour class size are simultaneously bounded by a constant. Recall that a graph of *colour class size* q is a graph coloured by an ordered set, say natural numbers, where at most q vertices get the same colour. Secondly, Atserias, Bulatov and Dawar [7] proved that FPC cannot express the solvability of linear equation systems over finite Abelian groups. Interestingly, also the CFI-query can be formulated using a linear equation system over $\mathbb{Z}_2$ [8].

Currently, *Rank Logic* (FPR) and *Choiceless Polynomial Time* (CPT) seem to be the two most promising candidates for logics that may capture PTIME. The first one, Rank Logic, was introduced by Dawar, Grohe, Holm and Laubner in [8] and is motivated by the fact that certain crucial problems from linear algebra cannot be expressed in FPC. It is defined as the extension of fixed-point logic by operators for the rank of definable matrices over finite fields. It was shown in [8] that Rank Logic is a *strict* extension of fixed-point logic with counting, and that it is powerful enough to define many of the known properties that separate FPC from PTIME. Perhaps

[1]Formally, a logic consists of a decidable set of sentences and an *isomorphism-closed* relation $\models$ between structures and sentences. A logic captures PTIME if it can define all PTIME-decidable Boolean queries and there is an algorithm that, for each sentence, outputs a PTIME algorithm deciding the corresponding query. For precise definitions, see e.g. [2].

1043-6871/15 $31.00 © 2015 IEEE
DOI 10.1109/LICS.2015.68

the most important ones are the solvability of linear equation systems over finite fields and the CFI-query. Similar extensions of FPC by operators that solve linear equation systems over finite rings (and not only over finite fields) have then been studied in [9]. For all of these extensions it remains open whether they suffice to capture PTIME.

In this paper we focus on the second candidate for a logic for PTIME, namely *Choiceless Polynomial Time* (CPT), an extension of FPC which has been proposed by Blass, Gurevich and Shelah in [10]. Instead of augmenting the power of FPC by operators for certain undefinable queries (such as the rank of a matrix), the basic idea of CPT is to combine the manipulation of higher order objects (namely hereditarily finite sets over the input structure) with a bounded amount of parallel computation. Technically, Choiceless Polynomial Time is based on BGS-machines (for Blass, Gurevich and Shelah), a computation model which directly works on structures (and not on their string encodings, like Turing machines do). Notice that although the definition of CPT is technically based on a machine model, it satisfies the formal requirements for a logic for PTIME, as formulated in a precise way by Gurevich in [11] (see also [2]). In particular, computations of BGS-machines respect symmetries of the input structure, and the set of states in a run of a BGS-program is closed under automorphisms of the input structure. More informally this means that BGS-computations are *choiceless*: it is impossible to implement statements like "pick an arbitrary element x and continue" which occur in many high-level descriptions of polynomial-time algorithms (e.g. Gaussian elimination, the Blossom algorithm for maximum matchings, and so on). On the other hand, BGS-machines are very powerful due to their ability to construct and manipulate hereditarily finite sets over the input structure (i.e., finite sets whose elements are either elements of the input structure or again hereditarily finite sets). If one imposes no further restriction on BGS-logic then *every* decidable class of structures can be defined in BGS-logic. Thus, to define CPT, a *polynomial-time* fragment of BGS-logic, one has to restrict the access of a BGS-program to hereditarily finite sets in the sense that only sets of polynomial complexity may be used. To be a realistic candidate for a logic for PTIME, Choiceless Polynomial Time also has, and needs, a built-in operator to define the cardinality of sets. As Blass, Gurevich and Shelah proved in [10], the variant CPT^- of Choiceless Polynomial Time without such an operator cannot express even very elementary counting properties such as the question whether a set has an even number of elements (although this is much more difficult to prove than for, say, fixed-point logic).

The ability of Choiceless Polynomial Time to create and manipulate hereditarily finite sets over the input structure can be used to compensate for the absence of a linear order by the parallel creation of *all* linear orders on a small subset of the input structure. This has been used by Blass, Gurevich and Shelah to show that on structures of size n, CPT captures polynomial time properties of fragments of size m if $m! \leq n$. Later, Laubner [12] was able to generalise this by proving that CPT even captures PTIME on fragments of logarithmic size (when restricted to graphs). Furthermore, Dawar, Richerby and Rossman [13] proved that CPT can define the CFI-query. Their clever construction uses the power of CPT to avoid arbitrary choices by finding succinct (polynomial-time representable) encodings of exponential-sized objects as hereditarily finite sets. In addition, they proved that these hereditarily finite sets have a high complexity in terms of their set-theoretic rank, and also that this is necessary: the fragment of CPT which can only access hereditarily finite sets of bounded rank cannot define the CFI-query. Further progress in our understanding of the power of CPT was recently made in [14], by the construction of canonisation procedures in CPT for structures of bounded colour class size with Abelian symmetry groups on the colour classes. This implies in particular that CPT captures polynomial time on structures of colour class size two. Since the CFI-graphs can be represented as structures of colour class size two, this can be seen as a strong generalisation of the CPT-definability result of Dawar, Richerby and Rossman discussed above. At the same time, this solves a problem posed by Blass, Gurevich, and Shelah in [15]: the isomorphism problem for multipedes (which, again, are structures of colour class size two) is definable in CPT.

While the definition of Choiceless Polynomial Time via the manipulation of hereditarily finite sets is very convenient and powerful for the design of abstract computations on structures, it makes the analysis of the expressive power of CPT rather difficult. Standard techniques for the analysis of logical systems as used in finite model theory, for instance those based on Ehrenfeucht-Fraïssé methods, are not directly available. In particular, applications of comprehension terms increase the rank of objects and are difficult to handle by the common logical tools, which are usually restricted to 'flat' objects. Thus we aim to reduce this rather functional framework operating on higher-order objects to an approach that evaluates formulae on less complex objects.

In this paper we shall present alternative characterisations of CPT (with and without counting) that are based on classical model-theoretic techniques. We replace the machinery of hereditarily finite sets and comprehension terms by traditional first-order interpretations, and handle counting by Härtig quantifiers (which are classical quantifiers for cardinality comparison). In our framework, computations on finite structures are captured by *iterations of interpretations*, and a run is a sequence of states, each of which is a finite structure of a fixed vocabulary. There is an initial interpretation that produces the initial state as a structure interpreted in the input structure, and a second interpretation $\mathcal{I}_{\text{step}}$ that always maps the current state to its successor state. Since interpretations need not be one-dimensional they can increase the size of the states. Although one application of an interpretation increases the size only polynomially, without imposing restrictions, the iterated application through a polynomial number of steps could produce states of exponential size. By imposing polynomial bounds on the length of such computations and the size of the states, we obtain polynomial-time interpretation logic. Our main result is that this logic, denoted PIL, has precisely the same expressive power as Choiceless Polynomial Time.

Theorem 1. $\mathrm{PIL} \equiv \mathrm{CPT}$.

The equivalence survives also in the absence of counting: polynomial-time interpretation logic without the Härtig quantifier PIL^- is equivalent to CPT^-.

We hope that these alternative, and in a sense more model-theoretic, characterisations of CPT will help us build a more powerful toolkit of methods to analyse the structure and expressive power of Choiceless Polynomial Time and interpretation logic. In particular, the presentation in terms of first-order interpretations leads to natural fragments and stratifications of this logic along familiar syntactic parameters. For instance, one can consider the natural restrictions of PIL to k-dimensional interpretations, and/or to interpretations where the domain or equivalence formulae are trivial. We shall prove that the iteration of one-dimensional interpretations is in fact equivalent to the familiar relational iteration appearing in the partial fixed-point logic PFP, or equivalently, in the database language while. Thus, without the Härtig quantifier, one-dimensional polynomial-time interpretation logic turns out to be equivalent to the polynomial-time restriction of PFP, which by means of a classical result due to Abiteboul and Vianu implies that one-dimensional PIL^- is equivalent to LFP if, and only if, PTIME = PSPACE. On the other hand, it is known (see e.g. [16]) that the polynomial-time restriction of PFP with counting is actually equivalent to FPC. From this, we obtain the interesting result that one-dimensional PIL, when evaluated on the expansions of finite structures by an ordered numerical sort, has precisely the expressive power of FPC. One can thus view FPC as a one-dimensional fragment of PIL and CPT. In our view this confirms the intuition that the additional power of Choiceless Polynomial Time over FPC comes from the generalization of relational iteration in a fixed arity (as in fixed-point logics) to iterations of relations of changing arities. Already two-dimensional interpretations give us this additional flexibility of relational iteration and, indeed, two-dimensional PIL turns out to be equivalent to full PIL.

Another interesting question is whether the representation of equality by congruence relations and the passage to quotient structures are really necessary for obtaining the full expressive power of PIL. It turns out that in the absence of counting, PIL^- without congruences is equivalent to a previously studied extension of the database language while, called $\mathrm{while}_{\mathrm{new}}|_{\mathrm{PTIME}}$ which is known to be strictly weaker than CPT. In the presence of counting, the situation is even more intriguing. We shall prove that on any class of structures of bounded colour class size, PIL without congruences can be simulated by CPT-programs that access only hereditarily finite sets of bounded rank. In particular this holds for the class of CFI-graphs (which are, in fact, graphs of colour class size four). Since Dawar, Richerby, and Rossman prove in [13] that the CFI-query is definable in CPT, but not by programs of bounded rank, this separates also congruence-free PIL from full PIL. We conclude that, with or without counting, congruences are really essential for reaching the full power of PIL.

In Sect. II we recall the precise definition of CPT, in the variant presented by Rossman [17], and in Sect. III we introduce interpretation logic and its polynomial-time fragment, PIL. The following three sections are used to establish the equivalence of PIL and CPT. We shall first prove that the cardinality operator in CPT (which associates with every set the finite ordinal describing its cardinality) can be replaced by an equicardinality relation without reducing the expressive power. We obtain an equivalent variant $\mathrm{CPT}_{\mathrm{EqCard}}$ of CPT, and describe then simulations from PIL to $\mathrm{CPT}_{\mathrm{EqCard}}$ and vice versa in Sect. V and VI. Together, this proves Theorem 1. In the final section of the paper, we discuss several fragments of interpretation logic and relate them to logics and database query languages that have previously been proposed and studied in the context of the quest for a logic for polynomial time, and have turned out to be weaker than CPT.

II. Choiceless Polynomial Time

Choiceless Polynomial Time operates on hereditarily finite sets. Thus we first recall some basic notions of set theory.

We denote by $[n]$ the *von Neumann ordinal* associated with the natural number n, and by ω the first infinite ordinal. The notation $[n]$ is used to make the set-theoretic representation explicit. We identify the cardinality $|A|$ of a finite set A with the respective von Neumann ordinal. We write $\langle a, b\rangle$ to denote the Kuratowski encoding of the ordered pair (a, b), and $\langle a_1, \dots, a_k\rangle$ for $k \neq 2$ for the corresponding set-theoretic encoding of a k-tuple.

Sets can contain sets and *atoms*. Atoms are never sets, and $x \in y$ only holds if y is a set. An *object* is a set or an atom. A set y is *transitive* if $x \subseteq y$ for every set $x \in y$. The *transitive closure* $\mathrm{tc}(x)$ of a set x is the least transitive set y with $x \subseteq y$. An object is *hereditarily finite* if its transitive closure is finite. The collection $\mathrm{HF}(A)$ of hereditarily finite objects over a set A of atoms is defined as the set of all objects x such that $x \in A$ or x is a hereditarily finite set such that all atoms in $\mathrm{tc}(x)$ are elements of A. Alternatively, we say that each atom is hereditarily finite, and, recursively, each finite set of hereditarily finite objects is again hereditarily finite.

Choiceless Polynomial Time is a polynomial-time restriction of BGS-logic (named after Blass, Gurevich and Shelah). Essentially, BGS-logic permits iterated construction of hereditarily finite sets over the input structure. A fragment that lies inside of PTIME is obtained by restricting the complexity of the sets occurring in the computation.

Whereas the original definition of BGS-logic given by Blass, Gurevich and Shelah in [10] consists of instructions similar to those of a machine model, the equivalent definition introduced by Rossman in [17] is based on iterated evaluation of BGS-terms. The definition of BGS-logic and CPT we give in the following corresponds to the one in [17] except for some technical details.

Let τ be a relational signature. Then $\tau^{\mathrm{HF}} = \tau \cup \{\mathrm{Atoms}, \mathrm{Empty}, \in, \mathrm{TheUnique}, \mathrm{Pair}, \mathrm{Union}, \mathrm{Card}\}$, where

$\mathrm{Atoms}, \mathrm{Empty}$ are constant symbols, $\in$ is a binary relation symbol, $\mathrm{TheUnique}, \mathrm{Union}, \mathrm{Card}$ are unary function symbols and Pair is a binary function symbol.

Given a τ-structure $\mathfrak{A}$, the *hereditarily finite expansion* $\mathrm{HF}(\mathfrak{A})$ of $\mathfrak{A}$ is the τ^{HF}-structure over $\mathrm{HF}(A)$ where the relations in τ are defined as in $\mathfrak{A}$ and the auxiliary functions and relations are defined as follows:

- $\mathrm{Empty}^{\mathrm{HF}(\mathfrak{A})} = \emptyset$, $\mathrm{Atoms}^{\mathrm{HF}(\mathfrak{A})} = A$,
- $\in^{\mathrm{HF}(\mathfrak{A})} = \{(a,b) \mid a \in b\}$,
- $\mathrm{Pair}^{\mathrm{HF}(\mathfrak{A})}(a,b) = \{a,b\}$,
- $\mathrm{Union}^{\mathrm{HF}(\mathfrak{A})}(a) = \{b \in c \mid c \in a\}$
- $\mathrm{TheUnique}^{\mathrm{HF}(\mathfrak{A})}(a) = \begin{cases} b & a = \{b\}, \\ \emptyset & \text{otherwise}, \end{cases}$
- $\mathrm{Card}^{\mathrm{HF}(\mathfrak{A})}(a) = \begin{cases} |a| & a \notin A, \\ \emptyset & \text{otherwise}. \end{cases}$

BGS-*terms* (or simply *terms*) are variables and constants in τ^{HF} (these are the atomic terms), objects of the form $f\bar{t}$ for a function symbol $f \in \tau^{\mathrm{HF}}$ and terms $\bar{t}$, Boolean terms or *comprehension terms*. For BGS-terms $t_1, \dots, t_r$ and each r-ary relation symbol $R \in \tau^{\mathrm{HF}}$, $t_1 = t_2$ and $Rt_1 \dots t_r$ are Boolean terms. Boolean combinations of Boolean terms are again Boolean terms. $\{s(\bar{x}, y) : y \in r(\bar{x}) : \varphi(\bar{x}, y)\}$ is a comprehension term if s and r are terms and φ is a Boolean term. The value $[\![t]\!]^{\mathfrak{A}}$ of a term t is defined as an element of $\mathrm{HF}(\mathfrak{A})$ in the obvious way, where the value of a Boolean term is represented as $\mathrm{false} := \emptyset$ or $\mathrm{true} := \{\emptyset\}$ and the value of a comprehension term $\{s(\bar{x}, y) : y \in r(\bar{x}) : \varphi(\bar{x}, y)\}$ is the set $\{[\![s(\bar{a}, b)]\!]^{\mathfrak{A}} : b \in [\![r(\bar{a})]\!]^{\mathfrak{A}} : [\![\varphi(\bar{a}, b)]\!]^{\mathfrak{A}} = \{\emptyset\}\}$.

We say that each BGS-term t_{step} induces a BGS-program Π. For each τ-structure $\mathfrak{A}$, the *run* of Π is the sequence $(a_i)_{i<\kappa}$ ($\kappa \leq \omega$) such that $a_0 = \emptyset$, $a_{i+1} = [\![t_{\text{step}}(a_i)]\!]^{\mathfrak{A}}$ for all i, and κ is maximal such that $a_i \notin \{\emptyset, \{\emptyset\}\}$ for all $0 < i < \kappa$. If κ is finite, a_κ is the final state of the run of Π on $\mathfrak{A}$ and $\Pi(\mathfrak{A}) = a_{\kappa-1}$. In that case, $\mathfrak{A} \models \Pi$ if and only if $a_\kappa = \{\emptyset\}$, and if κ is infinite, the truth value of $\mathfrak{A} \models \Pi$ is *undefined*.

To obtain a polynomial bound, we restrict both the complexity of the states and the length of the run.

A CPT-*program* is a pair $\overline{\Pi} = (\Pi, p(n))$, where Π is a BGS-program and $p(n)$ is a polynomial. The run of $\overline{\Pi}$ on a structure $\mathfrak{A}$ is the maximal initial segment ϱ' of the run ϱ of Π on $\mathfrak{A}$, such that

- ϱ' is of length at most $p(|\mathfrak{A}|)$, and
- $|\mathrm{tc}(a_i)| \leq p(|\mathfrak{A}|)$ for each state a_i in ϱ'.

If the final states of ϱ and ϱ' coincide, then $\mathfrak{A} \models \overline{\Pi}$ if and only if $\mathfrak{A} \models \Pi$. Otherwise the truth value is undefined.

Note that, with the cardinality function, it is possible to count the number of iterations and to determine the sizes of $\mathrm{tc}(a_i)$. Therefore each CPT-program is equivalent to a CPT-program whose output is never undefined.

A variant of BGS and CPT is obtained by replacing the function Card in $\mathrm{HF}(\mathfrak{A})$ by the following 2-ary relation EqCard:

$$\mathrm{EqCard}^{\mathrm{HF}(\mathfrak{A})} = \{(a,b) \mid |a| = |b|\}.$$

We denote the resulting logics by $\mathrm{BGS}_{\mathrm{EqCard}}$ and $\mathrm{CPT}_{\mathrm{EqCard}}$, respectively. BGS and CPT refer to the variants of the logics with the cardinality function, and logics with neither Card nor EqCard are denoted BGS^- and CPT^-, respectively.

III. Interpretation Logic

Interpretation logic is based on first-order interpretations. To express counting operations in that logic, we extend first-order logic by the *Härtig quantifier* with the following rule for constructing formulae: If φ and ψ are formulae, then $\mathrm{H}xy(\varphi, \psi)$, where x occurs free in φ and y occurs free in ψ, is a formula. The semantics is defined by $\mathfrak{A} \models \mathrm{H}xy(\varphi, \psi)$ if and only if $|\{a \in A \mid \mathfrak{A} \models \varphi(a)\}| = |\{b \in A \mid \mathfrak{A} \models \psi(b)\}|$. The resulting logic is FO+H.

Now let us recall the notion of interpretations over an extension of first-order logic.

Let $\mathcal{L}$ be an extension of FO and let τ, σ be relational signatures. A k-dimensional $\mathcal{L}[\tau, \sigma]$-*interpretation* is a sequence $\mathcal{I} = (\varphi_{\text{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \sigma})$ of $\mathcal{L}[\tau]$-formulae where

- φ_{dom}, called the *domain formula*, has exactly k free variables,
- $\varphi_{\approx}$, called the *equality formula*, has exactly $2k$ free variables,
- each φ_{R_i} has exactly $k \cdot r_i$ free variables (where r_i is the arity of R_i).

Note that we only consider interpretations without parameters, because we regard interpretations as a means of transforming structures within a computation.

A formula $\varphi_{\text{dom}}(x_1, \dots, x_k)$ defines the set $\varphi_{\text{dom}}^{\mathfrak{A}} = \{(a_1, \dots, a_k) \mid \mathfrak{A} \models \varphi_{\text{dom}}(a_1, \dots, a_k)\}$.

An $\mathcal{L}[\tau, \sigma]$-interpretation defines a mapping from τ- to σ-structures as follows: For a τ-structure $\mathfrak{A}$ and a σ-structure $\mathfrak{B}$, we say that $\mathfrak{B} = \mathcal{I}(\mathfrak{A})$ if there exists a surjective mapping $h : \varphi_{\text{dom}}^{\mathfrak{A}} \to B$ such that

- for all $\overline{a_1}, \overline{a_2} \in \varphi_{\text{dom}}^{\mathfrak{A}}$, $h(\overline{a_1}) = h(\overline{a_2})$ if and only if $\overline{a_1} = \overline{a_2}$ or $\mathfrak{A} \models \varphi_{\approx}(\overline{a_1}, \overline{a_2})$, and
- for every r-ary relation symbol $R \in \sigma$ and all tuples $\overline{a_1}, \dots, \overline{a_r} \in \varphi_{\text{dom}}^{\mathfrak{A}}$, $(h(\overline{a_1}, \dots, \overline{a_r})) \in R^{\mathfrak{B}}$ if and only if $\mathfrak{A} \models \varphi_R(\overline{a_1}, \dots, \overline{a_r})$.

$\mathcal{I}(\mathfrak{A})$ is usually identified with the structure over the domain $\varphi_{\text{dom}}^{\mathfrak{A}} / \varphi_{\approx}^{\mathfrak{A}}$.

We say that an interpretation $\mathcal{I}$ preserves the domain if $\mathcal{I} = (\varphi_{\text{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \sigma})$ is one-dimensional, φ_{dom} is valid and $\varphi_{\approx}(x, y)$ is equivalent to $x = y$. Furthermore, $\mathcal{I}$ preserves a relation R if $\mathcal{I}$ preserves the domain and $\varphi_R(x_1, \dots, x_r)$ is equivalent to $Rx_1 \dots x_r$.

The *concatenation* $\mathcal{I}_1 \circ \mathcal{I}_2$ of an $\mathcal{L}[\tau, \sigma]$-interpretation $\mathcal{I}_1$ and an $\mathcal{L}[\sigma, \sigma']$-interpretation is an interpretation with $\mathcal{I}_1 \circ \mathcal{I}_2(\mathfrak{A}) = \mathcal{I}_2(\mathcal{I}_1(\mathfrak{A}))$ for each τ-structure $\mathfrak{A}$.

Let τ, σ be relational signatures such that σ contains designated nullary relation symbols Halt and Out and let $\mathcal{L}$ be an extension of FO. The sentences of *interpretation logic (*IL*)* are defined as *programs*: An $\mathrm{IL}[\tau, \sigma]$-program over $\mathcal{L}$ is a pair $\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}})$, where $\mathcal{I}_{\text{init}}$ is an $\mathcal{L}[\tau, \sigma]$-interpretation and $\mathcal{I}_{\text{step}}$ is an $\mathcal{L}[\sigma, \sigma]$-interpretation. Π defines a mapping from τ-structures to σ-structures as follows:

For each τ-structure $\mathfrak{A}$, the *run* of Π on $\mathfrak{A}$ is the sequence $(\mathfrak{A}_i)_{i<\kappa}$, where κ is a finite ordinal or ω, such that $\mathfrak{A}_0 = \mathcal{I}_{\text{init}}(\mathfrak{A})$, $\mathfrak{A}_{i+1} = \mathcal{I}_{\text{step}}(\mathfrak{A}_i)$, and κ is maximal such that $\mathfrak{A}_i \not\models \text{Halt}$ for all $i < \kappa$. If κ is finite, $\mathfrak{A}_\kappa$ is the final state of the run of Π on $\mathfrak{A}$, $\Pi(\mathfrak{A}) = \mathfrak{A}_\kappa$ and $\mathfrak{A} \models \Pi$ if and only if $\Pi(\mathfrak{A}) \models \text{Out}$.

Analogously to CPT, we place polynomial bounds on the length of a run and the complexity of the states of an IL-program to obtain a fragment in PTIME.

A PIL-*program* is a pair $\overline{\Pi} = (\Pi, p(n))$, where Π is an IL-program and $p(n)$ is a polynomial. The run of $\overline{\Pi}$ on a structure $\mathfrak{A}$ is the maximal initial segment ϱ' of the run ϱ of Π on $\mathfrak{A}$, such that

- ϱ' is of length at most $p(|\mathfrak{A}|)$, and
- $|\mathfrak{A}_i| \leq p(|\mathfrak{A}|)$ for each state $\mathfrak{A}_i$ in ϱ'.

Again, if the final states of ϱ and ϱ' coincide, then $\mathfrak{A} \models \overline{\Pi}$ if and only if $\mathfrak{A} \models \Pi$. Otherwise the truth value is undefined.

In the following, we denote by IL and PIL interpretation logic and polynomial-time interpretation logic, respectively, assuming that the underlying logic is $\mathcal{L} = \text{FO+H}$, and we denote by IL^- and PIL^- these logics for the case $\mathcal{L} = \text{FO}$.

In order to ease the analysis of PIL, we observe that a single binary relation symbol for the step-interpretation $\mathcal{I}_{\text{step}}$ already suffices to obtain its full expressive power. This is because, as shown for instance in [18], there is an interpretation that uniquely maps each structure to a graph. Moreover, one can easily show that signatures consisting only of monadic predicates are not sufficient.

Remark 2. $\text{PIL}[\tau, \{E\}] \geq \text{PIL}[\tau, \sigma]$ *for all signatures* τ, σ. *Moreover, if* σ *only contains monadic predicates we have* $\text{PIL}[\{E\}, \{E\}] > \text{PIL}[\{E\}, \sigma]$.

IV. Expressing Counting with Equicardinality

In the following sections, we show that interpretation logic is indeed an alternative representation of Choiceless Polynomial Time.

PIL uses an equicardinality quantifier, whereas CPT is defined with an explicit counting function. Thus, we first show that the variant of CPT with an equicardinality relation is as expressive as CPT.

Proposition 3. $\text{CPT} \equiv \text{CPT}_{\text{EqCard}}$.

Proof. First, we observe that the function Card returns a von Neumann ordinal. Since BGS naturally has the ability to create hereditarily finite sets, it is possible to create finite ordinals. So we will construct, for each CPT program, an equivalent $\text{CPT}_{\text{EqCard}}$ program that first creates the set of all ordinals necessary for the computation and then simulates each term $\text{Card}\, t$ with a term defining the unique ordinal that has the same cardinality as the value of t.

In order to know in advance which ordinals have to be created, we use the polynomial bound on the complexity of states inherent in each CPT program. The following technical lemma will imply a bound on the complexity of all sets occurring in the computation of a CPT program.

Lemma 4. *For each* BGS-*term* $t(x_1, \ldots, x_k)$ *and each polynomial* p*, there is a polynomial* q *such that, for each input structure* $\mathfrak{A}$ *and each tuple* $\bar{a} \in \text{HF}(A)^k$ *where* $p(|\mathfrak{A}|)$ *bounds the transitive closure of each component,* $\text{tc}(\llbracket t(\bar{a}) \rrbracket)$ *is bounded by* $q(|\mathfrak{A}|)$.

Proof of Lemma 4. Induction on t. For atomic and Boolean terms, the statement is trivial. For terms $\text{Union}(t)$, $\text{TheUnique}(t)$, $\text{Pair}(t_1, t_2)$ it follows immediately from the induction hypothesis.

Now let $t = \{s(\bar{x}, y) : r(\bar{x}) : \varphi(\bar{x}, y)\}$. Then $\text{tc}(\llbracket t(\bar{a}) \rrbracket) \subseteq \bigcup\{\text{tc}(\llbracket s(\bar{a}, b) \rrbracket) : b \in \llbracket r(\bar{a}) \rrbracket\}$. By induction hypothesis, there are polynomials q_s and q_r that bound $\text{tc}(\llbracket s(\bar{a}, b) \rrbracket)$ and the transitive closure of all $b \in \llbracket r(\bar{a}) \rrbracket$ as well as the size of $\llbracket r(\bar{a}) \rrbracket$, respectively. By definition of $\llbracket t \rrbracket$, the polynomial for t can be constructed from the product of q_s and q_r. $\square$

In the resulting $\text{CPT}_{\text{EqCard}}$ program, the ordinals will be encoded in the value of the free variable of the term t_{step} inducing the program. So we define for each BGS term a $\text{BGS}_{\text{EqCard}}$ term that computes the same value and uses the encoded ordinal to express the cardinality function with the equicardinality relation.

Lemma 5. *For every* BGS *term* t*, there is a* $\text{BGS}_{\text{EqCard}}$ *term* t^{EqCard} *such that* $\llbracket t^{\text{EqCard}}(\langle a_1, [m] \rangle, a_2, \ldots, a_k) \rrbracket = \langle \llbracket t(a_1, \ldots, a_k) \rrbracket, [m] \rangle$ *for each* $m \in \mathbb{N}$ *that is greater than the cardinality of* $\llbracket t'(a_1, \ldots, a_k) \rrbracket$ *for all subterms* t' *of* t.

Proof of Lemma 5. Induction on t. For $t = \text{Card}(s(\bar{x}))$, let

$$t^{\text{EqCard}}(\bar{x}) = t_{\text{Pair}}\Big(\text{TheUnique}\left(\{y : y \in (x_1)_2 : \text{EqCard}\left(y, \left(s^{\text{EqCard}}(\bar{x})\right)_1\right)\}\right), (x_1)_2\Big),$$

where $t_{\text{Pair}}(a, b)$ is a term defining the ordered pair $\langle a, b \rangle$, $(x_1)_2$ defines the second component of x_1 if x_1 is an ordered pair, and, analogously, $\left(s^{\text{EqCard}}(\bar{x})\right)_1$ defines the first component of $s^{\text{EqCard}}(\bar{x})$. Since m is large enough, t^{EqCard} always defines the cardinality of $\llbracket s \rrbracket$. $\square$

Using the term $t_{\text{step}}^{\text{EqCard}}$ for a term t_{step} inducing a CPT program, we can simulate the associated program in $\text{CPT}_{\text{EqCard}}$.

Clearly, the equicardinality relation can be expressed using the cardinality function, so $\text{CPT}_{\text{EqCard}} \leq \text{CPT}$. For the other direction, let (Π, p) be a CPT program, where Π is induced by the term t_{step}. We construct a $\text{CPT}_{\text{EqCard}}$ program $(\Pi_{\text{sim}}, p_{\text{sim}})$ that simulates (Π, p).

By definition of CPT, it holds for each structure $\mathfrak{A}$ that $|\text{tc}(a_i)|$ is bounded by $p(|\mathfrak{A}|)$ for each state a_i of the run of Π on $\mathfrak{A}$. So, since the value of the free variable of t_{step} is bounded by p, Lemma 4 implies that there is a polynomial q that bounds the value of each subterm of $t_{\text{step}}(a_i)$. Then q provides an upper bound for all cardinalities occurring in the runs of (Π, p).

So, on each structure $\mathfrak{A}$, Π_{sim} first constructs the ordinals up to $q(|\mathfrak{A}|)$. Since BGS terms can express basic set-theoretic operations, it is possible to compute successor ordinals until one of cardinality $|\mathfrak{A}|$ exists, compute a set of size $q(|\mathfrak{A}|)$ using cardinal arithmetic and then again compute successor ordinals to obtain $[q(|\mathfrak{A}|)+1]$.

Again with basic set-theoretic operations, the program constructs the ordered pair $\langle \emptyset, [q(|\mathfrak{A}|)+1]\rangle$.

By Lemma 5, there is a term $t_{\text{step}}^{\text{EqCard}}$ such that, for each state a_i of a run of Π on $\mathfrak{A}$, $\left[\!\left[t_{\text{step}}^{\text{EqCard}}(\langle a_i, [q(|\mathfrak{A}|)+1]\rangle)\right]\!\right] = \langle [\![t(a_i)]\!], [q(|\mathfrak{A}|)+1]\rangle$. Note that sequential execution of BGS-programs is possible, so there is a $\mathrm{BGS}_{\mathrm{EqCard}}$-program performing the operations described above.

After the initialisation of the ordinals, which needs $q(|\mathfrak{A}|)+1$ states, each state in the run of Π_{sim} corresponds to a state in the run of Π. Furthermore, the transitive closure of each state is bounded by $p(|\mathfrak{A}|)+q(|\mathfrak{A}|)+c$, where c is the number of elements necessary to encode the ordered pairs.

So there is a polynomial p_{sim} such that $(\Pi_{\text{sim}}, p_{\text{sim}})$ is a $\mathrm{CPT}_{\mathrm{EqCard}}$ program with the same output as (Π, p). ☐

V. Simulating Interpretation Logic in Choiceless Polynomial Time

To show our main result, we simulate $\mathrm{CPT}_{\mathrm{EqCard}}$-programs in PIL and vice versa. The elementary parts of PIL-programs are (FO+H)-formulae, so we start by simulating (FO+H)-formulae with $\mathrm{BGS}_{\mathrm{EqCard}}$-terms.

Note that, by Remark 2, we can w.l.o.g. consider only $\mathrm{PIL}[\tau, \{E\}]$-programs.

Since BGS-terms are evaluated in $\mathrm{HF}(\mathfrak{A})$ for each structure $\mathfrak{A}$, whereas FO formulae are evaluated in arbitrary structures, we first define a precise notion of simulation between (FO+H)-formulae and $\mathrm{BGS}_{\mathrm{EqCard}}$-terms.

Definition 6. *Let $\varphi(\bar{x})$ be a formula of vocabulary $\{E\}$. A* BGS*-term t_φ over τ* simulates *φ if for every finite τ-structure $\mathfrak{A}$ and every σ-structure $\mathfrak{B} = (B, E)$ with $B, E \in \mathrm{HF}(A)$ and all $\bar{b} \in B^k$ it holds that $\mathfrak{B} \models \varphi(\bar{b})$ if and only if $[\![t_\varphi(\bar{b}, B, E)]\!]^{\mathfrak{A}} = \mathrm{true}$.*

Next, we show that any (FO+H)-formula can be simulated in that sense.

Lemma 7. *For each (*FO+H*)-formula $\varphi(\bar{x})$ over $\{E\}$ and each signature τ, there is a* BGS*-term $t_\varphi(\bar{x}, y_B, y_E)$ over τ simulating φ.*

Proof. Induction on φ.
Induction Base. First note that the only FO-terms over a relational signature are variables. So for formulae of the form $t_1 = t_2$ there is nothing to show. For $\varphi = Et_1t_2$, let $t_\varphi = t_{\text{Pair}}(t_1, t_2) \in y_E$, where $t_{\text{Pair}}(a_1, a_2)$ defines the ordered pair $\langle a_1, a_2\rangle$.
Induction Step. Boolean connectives are translated directly. The formula $\varphi = \exists x\psi$ is translated to the term $t_\varphi = \emptyset \in \{\emptyset : x \in y_B : t_\psi(x)\}$, and for $\varphi = \mathrm{H}xy(\psi(x), \vartheta(y))$, we let $t_\varphi = \mathrm{EqCard}\left(\{x : x \in y_B : t_\psi(x)\}, \{y : y \in y_B : t_\vartheta(y)\}\right)$. ☐

The terms for the formulae in PIL-programs are combined to simulate PIL computations in BGS.

Proposition 8. $\mathrm{PIL} \leq \mathrm{CPT}_{\mathrm{EqCard}}$.

Proof. Let $\overline{\Pi} = (\Pi, p)$ be a $\mathrm{PIL}[\tau, \{E\}]$-program. We construct a $\mathrm{CPT}_{\mathrm{EqCard}}$-program Π_{sim} that represents each state $\mathfrak{A}_i = (A_i, E_i)$ as the pair $\langle A_i, E_i\rangle$. Note that the domain of each state is a set of equivalence classes of tuples from the domain of the previous state. Thus the domain of a state can be expressed with the term

$$\left\{\{y : y \in s_1^k : t_{\text{dom}}(y, s_1, s_2) \wedge t_{\approx}(x, y, s_1, s_2)\} \right. \\ \left. : x \in s_1^k : t_{\text{dom}}(x, s_1, s_2)\right\},$$

where s is a free variable that denotes the previous state, s_j defines the jth component of s whenever s is an ordered pair, s_1^k defines the set of k-tuples over s_1 if s_1 is a set, and t_{dom} and $t_\approx$ are the terms simulating the domain and equality formula of $\mathcal{I}_{\text{step}}$ according to Lemma 7.

The relation E_i of a state $\mathfrak{A}_i$ can then be defined using the term above and the term simulating φ_E in $\mathcal{I}_{\text{step}}$. Clearly, formulae for applying $\mathcal{I}_{\text{init}}$ can be defined analogously (note that, for that interpretation, the FO+H-formulae can be simulated in a more straightforward way because the BGS-terms can directly refer to the relations in the input signature).

The term t_{step} inducing Π_{sim} transforms each state into the pair representing the successor state, and uses the terms simulating the formulae defining Halt and Out to establish the halting condition. Then, clearly, $\mathfrak{A} \models \Pi_{\text{sim}}$ if and only if $\mathfrak{A} \models \Pi$ for each τ-structure $\mathfrak{A}$.

Let ϱ be the run of $\overline{\Pi}$ on a structure $\mathfrak{A}$, and let ϱ_{sim} be the corresponding run of Π_{sim} on $\mathfrak{A}$. By definition, each state in ϱ except for the initial and final state represents a state in ϱ_{sim}, so the length of ϱ_{sim} is bounded by $p(|\mathfrak{A}|)+2$. For the bound on the states, it suffices to consider the transitive closure of each set A_i, since the relation only consists of pairs of elements of A_i, so a polynomial bound on A_i implies a polynomial bound on each state $\langle A_i, E_i\rangle$.

We show by induction on i that, for each state $\langle A_i, E_i\rangle$ in ϱ_{sim}, $|\mathrm{tc}(A_i)| \leq p(|\mathfrak{A}|)^k \cdot c \cdot i + p(|\mathfrak{A}|) \cdot i$, where c is the number of additional sets necessary to encode a k-tuple. This clearly holds for A. In the induction step, the domain A_{i+1} consists of sets of tuples of elements from A_i. So

$$\begin{aligned} |\mathrm{tc}(A_{i+1})| &\leq |\mathrm{tc}(A_i)| + |\mathrm{tc}(A_i^k) \setminus \mathrm{tc}(A_i)| + |A_{i+1}| \\ &\leq (p(|\mathfrak{A}|) \cdot c \cdot i + p(|\mathfrak{A}|) \cdot i) + c \cdot p(|\mathfrak{A}|)^k + p(|\mathfrak{A}|) \\ &= p(|\mathfrak{A}|) \cdot c \cdot (i+1) + p(|\mathfrak{A}|) \cdot (i+1). \end{aligned}$$

$|\mathrm{tc}(A_i)| \leq c \cdot p(|\mathfrak{A}|)^k \cdot i + p(|\mathfrak{A}|) \cdot i$ holds by induction hypothesis, $|A_i^k| \leq c \cdot A_i^k$ by definition of c and $|A_{i+1}| \leq p(|\mathfrak{A}|)$ by definition of PIL.

So there is a polynomial p_{sim} such that $(\Pi_{\text{sim}}, p_{\text{sim}})$ is equivalent to (Π, p). ☐

VI. Simulating Choiceless Polynomial Time in Interpretation Logic

For the other direction of the equivalence, note that PIL computations are iterated interpretations, whereas CPT computations are iterated evaluations of BGS-terms. So it seems natural to simulate BGS-terms with interpretations in an appropriate way. Note, however, the following difference between CPT and PIL computations: The states of a CPT computation are arbitrary sets in $\mathrm{HF}(A)$, but any term can construct new sets over A, for instance with the constant Atoms. In contrast, PIL always modifies the current state without reference to the input structure. Therefore, we construct simulating interpretations in a way that the states of the PIL computation contain representations of all sets that may become necessary for the simulation of another term.

Additionally, the simulation should preserve polynomial bounds. Since, by definition of CPT, the transitive closure of each state is bounded polynomially, we can obtain a polynomial bound for the respective PIL-program by ensuring that not too many objects outside of that transitive closure are represented in the states of the PIL-program. This still causes a small technical difficulty: The polynomial bound on the value of a term occurring in a CPT-program is only guaranteed if the values assigned to the free variables actually occur in the run of the CPT-program. So, in the states of the simulating PIL-program, we require a relation that marks all possible values of the free variables.

Summing up, the states of the simulating PIL-program should represent the necessary elements of $\mathrm{HF}(\mathfrak{A})$ and make it possible to identify permitted values for the free variables. When a term is simulated by an interpretation, only necessary objects should be represented in the resulting state. A mapping from possible values of free variables to values of a term is represented by suitable relations.

These requirements are formalised in the following definitions.

Definition 9. *Let $\mathfrak{A}$ be a τ-structure and let t be a* BGS*-term over τ with k free variables. A structure $\mathfrak{S} = (S, \tau_{state})$ for $\tau_{state} \supseteq \tau \cup \{\mathrm{Atoms}, \mathrm{Empty}, \in, \in^*, T_t, V_t\}$ is an $(\mathfrak{A}, t)$-*state *if, up to isomorphism,*

- *$A \cup \{A, \emptyset, \{\emptyset\}\} \subseteq S \subseteq \mathrm{HF}(A)$,*
- *$\mathfrak{S} \restriction \tau \cup \{\mathrm{Atoms}, \mathrm{Empty}, \in\} \subseteq \mathrm{HF}(\mathfrak{A}) \restriction \tau \cup \{\mathrm{Atoms}, \mathrm{Empty}, \in\}$ (where the constants are replaced by unary relation symbols),*
- *${\in^*}^{\mathfrak{S}}$ is the transitive closure of $\in^{\mathfrak{S}}$,*
- *V_t is a k-ary relation symbol and T_t is a $(k+1)$-ary relation symbol.*

Note that the signature τ_{state} may contain additional relation symbols. This will be necessary when simulating terms inductively, because the relations T_t and V_t corresponding to different subterms of the same term have to be preserved during the simulation.

Definition 10. *Let $t(\bar{v})$ be a* BGS*-term over an input signature τ with free variables $\bar{v} = v_1, \dots, v_k$ and let τ_{state} be a signature of $(\mathfrak{A}, t)$-states. A $[\tau_{state}, \tau_{state}]$-interpretation $\mathcal{I}_t$* simulates *t if for all τ-structures $\mathfrak{A}$ and all $(\mathfrak{A}, t)$-states $\mathfrak{S}$ of vocabulary τ_{state}, it holds that, up to isomorphism,*

- *$\mathcal{I}_t(\mathfrak{S})$ is an $(\mathfrak{A}, t)$-state,*
- *the domain of $\mathcal{I}_t(\mathfrak{S})$ is $S \cup \bigcup_{\bar{a} \in V_t^{\mathfrak{S}}} (\{[\![t(\bar{a})]\!]\} \cup \mathrm{tc}([\![t(\bar{a})]\!]))$,*
- *$\mathfrak{S} \subseteq \mathcal{I}_t(\mathfrak{S})$,*
- *$T_t^{\mathcal{I}_t(\mathfrak{S})} = \{(a_1, \dots, a_k, [\![t(a_1, \dots, a_k)]\!]) \mid (a_1, \dots, a_k) \in V_t^{\mathfrak{S}}\}$,*
- *$V_t^{\mathcal{I}_t(\mathfrak{S})} = V_t^{\mathfrak{S}}$.*

Simulation of terms is the crucial ingredient to the simulation of BGS-programs. Thus, we now show the following main lemma:

Lemma 11. *For each* BGS*-term t, there is an interpretation $\mathcal{I}_t$ simulating t.*

Proof. Let t be a BGS-term over the signature τ.

In most cases, the simulating interpretation is the concatenation of interpretations that create objects representing the relevant values of t. Therefore, we define the following interpretations:

- $\mathcal{I}_{\text{add}}$ enriches the domain by an element $a_{t,\bar{v}}$ for each $\bar{v} \in V_t$, where $a_{t,\bar{v}}$ represents the value $[\![t(\bar{v})]\!]$, and T_t is defined as the set of tuples $(\bar{v}, a_{t,\bar{v}})$,
- $\mathcal{I}_{\in}$ defines the relation $\in$ for the new objects and updates $\in^*$ accordingly,
- $\mathcal{I}_{\text{merge}}$ identifies all newly created objects that represent the same set.

$\mathcal{I}_{\text{add}}$ is a $k+2$-dimensional interpretation, where k is the arity of V_t. In $\mathcal{I}_{\text{add}}(\mathfrak{A})$ for a structure $\mathfrak{A}$, each element a of $\mathfrak{A}$ is represented by the equivalence class containing all tuples $(b_1, \dots, b_k, a, a)$ for all $b_1, \dots, b_k \in A$, and each $a_{t,(v_1,\dots,v_k)}$ is represented by the class of all tuples $(v_1, \dots, v_k, a_1, a_2)$ for $a_1 \neq a_2$.

$\mathcal{I}_{\in}$ preserves the domain and the relations except for $\in$ and $\in^*$. $\in$ is extended to the newly created sets (depending on the semantics of the term t), and $\varphi_{\in^*}(x, y) = x = y \lor x \in^* y \lor x \in y \lor \exists z (x \in^* z \land z \in y)$.

In $\mathcal{I}_{\text{merge}}$, the equality formula identifies all elements containing the same set of elements according to the relation $\in$. The element relation is only changed for newly created elements, so both the operation of $\mathcal{I}_{\text{merge}}$ and the update of $\in^*$ in $\mathcal{I}_{\text{add}}$ are possible without recursion.

With these auxiliary interpretations, the simulation of terms becomes a rather straightforward induction. In the following, $\mathfrak{S}$ denotes an $(\mathfrak{A}, t)$-state for a τ-structure $\mathfrak{A}$ and we fix a signature τ_{state} for the term t containing relation symbols T_s, V_s for all subterms s of t.

Induction on t:

1) $t(x) = x$: By definition, the relation T_t in $\mathcal{I}_t(\mathfrak{S})$ should be the set of all pairs $(a, [\![t(a)]\!]) = (a, a)$ for $a \in V_t^{\mathfrak{S}}$. So let $\mathcal{I}_t$ be the $\mathrm{FO}[\tau_{\text{state}}, \tau_{\text{state}}]$-interpretation that preserves the domain and relations of the input structure, except for
$$\varphi_{T_t}(x, y) = V_t x \land x = y.$$
2) $t = c$: Analogous to the previous case. Note that the interpretations use a relational signature, so $\varphi_{T_t}(x) = Cx$, for a suitable relation C. Since the input signature is relational, $c \in \{\mathrm{Atoms}, \mathrm{Empty}\}$, so the value of c is already an element of each $(\mathfrak{A}, t)$-state.

3) $t(\bar{x}) = Rt_1(\bar{x})\dots t_r(\bar{x})$ for some $R \in \tau$: By induction hypothesis, there are interpretations $\mathcal{I}_{t_1}, \dots, \mathcal{I}_{t_r}$ simulating $t_1, \dots, t_r$. Let $\mathcal{I}_t = \mathcal{I}_{t_1} \circ \dots \circ \mathcal{I}_{t_r} \circ \mathcal{I}_R$, where $\mathcal{I}_R$ is the interpretation preserving the domain and the relations, except for

$$\varphi_{T_t}(\bar{x}, y) = V_t\bar{x} \wedge ((\varphi_{\text{true}}(\bar{x}) \wedge y = \{\emptyset\}) \vee (\neg\varphi_{\text{true}}(\bar{x}) \wedge y = \emptyset)),$$

where

$$\varphi_{\text{true}}(\bar{x}) = \exists y_1 \dots \exists y_r\,(\,T_{t_1}\bar{x}y_1 \wedge \dots \wedge T_{t_r}\bar{x}y_r \wedge Ry_1\dots y_r)\,.$$

Note that the parameter t in the definition of $(\mathfrak{A}, t)$-states only influences the relations V_t and T_t, so each $(\mathfrak{A}, t)$-state is also an $(\mathfrak{A}, t_1)$-state and an $(\mathfrak{A}, t_2)$-state and vice versa (assuming w.l.o.g. that the relations T_t, T_{t_1} and T_{t_2} are already present), so the concatenation is possible and the interpreted structure is again an $(\mathfrak{A}, t)$-state.

4) $t(\bar{x}) = t_1(\bar{x}) = t_2(\bar{x})$: Analogous to Case 3.

5) $t(\bar{x}) = t_1(\bar{x}) \wedge t_2(\bar{x})$, $t(\bar{x}) = t_1(\bar{x}) \vee t_2(\bar{x})$ or $t(\bar{x}) = \neg t_1(\bar{x})$: Analogous to Case 3.

6) $t(\bar{x}) = \text{EqCard}(t_1(\bar{x}), t_2(\bar{x}))$: Similarly to Case 3, let $\mathcal{I}_t = \mathcal{I}_{t_1} \circ \mathcal{I}_{t_2} \circ \mathcal{I}_{\text{EqCard}}$, where $\mathcal{I}_{\text{EqCard}}$ defines T_t using the following formula in φ_{T_t}:

$$\varphi_{\text{true}}(\bar{x}) = \exists t_1 \exists t_2 (T_{t_1}\bar{x}t_1 \wedge T_{t_2}\bar{x}t_2 \wedge \text{H}y_1y_2(y_1 \in t_1, y_2 \in t_2)).$$

7) $t(\bar{x}) = ft_1(\bar{x})\dots t_r(\bar{x})$: Since we assume the input signature to be relational, $f \in \{\text{Pair}, \text{Union}, \text{TheUnique}\}$.

a) $t(\bar{x}) = \text{Pair}(t_1(\bar{x}), t_2(\bar{x}))$: Let $\mathcal{I}_t = \mathcal{I}_{t_1} \circ \mathcal{I}_{t_2} \circ \mathcal{I}_{\text{add}} \circ \mathcal{I}_\in \circ \mathcal{I}_{\text{merge}}$, where $\mathcal{I}_{t_1}$ and $\mathcal{I}_{t_2}$ are obtained from the induction hypothesis, and $\mathcal{I}_{\text{add}}$, $\mathcal{I}_\in$ and $\mathcal{I}_{\text{merge}}$ are defined as above. $\mathcal{I}_\in$ ensures that $a_{t,\bar{v}}$ contains exactly the elements of $[\![t(\bar{v})]\!] = \{[\![t_1(\bar{v})]\!], [\![t_2(\bar{v})]\!]\}$, so the formula $\varphi_\in$ in $\mathcal{I}_\in$ is

$$\varphi_\in(x, y) = x \in y \vee \exists\bar{v}((T_{t_1}\bar{v}x \vee\ T_{t_2}\bar{v}x) \wedge\ T_t\bar{v}y).$$

By definition of simulation, $(\mathcal{I}_{t_1} \circ \mathcal{I}_{t_2})(\mathfrak{S})$ is an $(\mathfrak{A}, t)$-state over the domain $S \cup \bigcup_{\bar{a} \in V_t^{\mathfrak{S}}}(\{[\![t_1(\bar{a})]\!]\} \cup \{[\![t_2(\bar{a})]\!]\} \cup \text{tc}([\![t_1(\bar{a})]\!]) \cup \text{tc}([\![t_2(\bar{a})]\!]))$. $\mathcal{I}_{\text{add}}$ creates an element for each value $[\![t(\bar{a})]\!]$ for $\bar{a} \in V_t^{\mathfrak{S}}$, and $\mathcal{I}_{\text{merge}}$ makes sure that $\mathcal{I}_t(\mathfrak{S})$ contains exactly one element for each distinct set. So the domain of $\mathcal{I}_t(\mathfrak{S})$ is as required by the definition of simulation.
Since the relation $\in$ is defined correctly by $\mathcal{I}_\in$ and the relations remain otherwise unchanged, the interpreted structure is again an $(\mathfrak{A}, t)$-state. T_t is defined appropriately by definition of $\mathcal{I}_{\text{add}}$, so $\mathcal{I}_t$ simulates t.

b) $t(\bar{x}) = \text{Union}(t_1(\bar{x}))$: Let $\mathcal{I}_t = \mathcal{I}_{t_1} \circ \mathcal{I}_{\text{add}} \circ \mathcal{I}_\in \circ \mathcal{I}_{\text{merge}}$, where $\mathcal{I}_\in$ defines $\in$ with the formula

$$\varphi_\in(x, y) = x \in y\ \vee\ \exists\bar{v}\,(T_t\bar{v}y \wedge \exists t_1 \exists z\,(T_{t_1}\bar{v}t_1 \wedge\ x \in z \wedge z \in t_1))\,.$$

With the same reasoning as in Case 7a, it follows that $\mathcal{I}_t$ simulates t.

c) $t(\bar{x}) = \text{TheUnique}(t_1(\bar{x}))$: Let $\mathcal{I}_t = \mathcal{I}_{t_1} \circ \mathcal{I}_{T_t}$, where $\mathcal{I}_{T_t}$ preserves the domain and the relations except for T_t, and defines T_t with the following formula:

$$\varphi_{T_t}(\bar{x}, y) = \exists t_1(T_{t_1}\bar{x}t_1 \wedge ((\varphi_{\text{sing}}(t_1) \wedge y \in t_1) \vee (\neg\, \varphi_{\text{sing}}(t_1) \wedge \text{Empty}\, y))) \wedge V_t\bar{x},$$

where $\varphi_{\text{sing}}(x)$ is a formula defining that x is a singleton.
Since $\mathcal{I}_{t_1}$ simulates t_1, the domain of $\mathcal{I}_{t_1}(\mathfrak{S})$ already contains all elements of $[\![t_1(\bar{a})]\!]$, so, by definition of $\mathcal{I}_{T_t}$, $\mathcal{I}_t$ simulates t.

8) $t(\bar{x}) = \{s(\bar{x}, y) : y \in r(\bar{x}) : \varphi(\bar{x}, y)\}$: Let $\mathcal{I}_t = \mathcal{I}_{\text{mark}} \circ \mathcal{I}_r \circ \mathcal{I}_V \circ \mathcal{I}_s \circ \mathcal{I}_\varphi \circ \mathcal{I}_{\text{add}} \circ \mathcal{I}_\in \circ \mathcal{I}_{\text{merge}} \circ \mathcal{I}_{\text{remove}}$, where $\mathcal{I}_V$ defines V_s and V_φ with the formula $\varphi_{V_s}(\bar{x}, y) = \varphi_{V_\varphi}(\bar{x}, y) = \exists r(T_r\bar{x}r \wedge y \in r)$ and $\mathcal{I}_\in$ defines $\in$ with the formula

$$\varphi_\in(x, y) = x \in y\ \vee\ \exists\bar{v}(T_t\bar{v}y \wedge \exists z(T_s\bar{v}zx \wedge \exists rT_r\bar{v}r \wedge z \in r \wedge \exists x_\varphi(T_\varphi\bar{v}zx_\varphi \wedge \neg\,\text{Empty}\, x_\varphi))).$$

The interpretation $\mathcal{I}_{\text{mark}}$ adds the whole domain to a unary relation P to mark the elements of $\mathfrak{S}$, and $\mathcal{I}_{\text{remove}}$ uses this information to remove all unnecessary elements from the domain with the formula

$$\varphi_{\text{dom}}(x) = Px \vee \exists y \exists z_1 \dots \exists z_k(T_tz_1\dots z_ky \wedge x \in^* y).$$

By induction hypothesis, $\mathcal{I}_r$ defines T_r appropriately. By definition, $\mathcal{I}_V$ makes sure that the resulting structure is an $(\mathfrak{A}, s)$-state and an $(\mathfrak{A}, \varphi)$-state, where V_s and V_φ contain exactly those tuples that are substituted for the free variables of s and φ when computing t. So $\mathcal{I}_s$ and $\mathcal{I}_\varphi$ define T_s and T_φ as required. In particular, $\mathcal{I}_s$ creates an object for each element of a value of t. Thus $\mathcal{I}_\in$ indeed defines the values of t to be the correct sets, and $\mathcal{I}_{\text{merge}}$ restores the property that the interpreted structure is an $(\mathfrak{A}, t)$-state. By definition, $\mathcal{I}_{\text{remove}}$ ensures that the domain only contains elements of $S \cup \bigcup_{\bar{a} \in V_t^{\mathfrak{S}}}(\{[\![t(\bar{a})]\!]\} \cup \text{tc}(t(\bar{a})))$. So $\mathcal{I}_t$ simulates t. $\square$

In the following, we combine the interpretations simulating the terms in a PIL-program to show that each CPT-program can be simulated in PIL:

Proposition 12. $\text{CPT}_{\text{EqCard}} \leq \text{PIL}$

Proof. Let $\overline{\Pi}_{\text{CPT}} = (\Pi_{\text{CPT}}, p_{\text{CPT}})$ be a CPT-program, where Π is the BGS program induced by the term t_{step}. We define a PIL-program (Π, p) with $\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}})$ that is equivalent to $\overline{\Pi}_{\text{CPT}}$.

The interpretation $\mathcal{I}_{\text{step}}$ will simulate the term t_{step}. Therefore, $\mathcal{I}_{\text{init}}$ transforms the input structure $\mathfrak{A}$ to an $(\mathfrak{A}, t_{\text{step}})$-state. So $\mathcal{I}_{\text{init}}$ is the concatenation of interpretations that create objects representing the sets $\emptyset$, $\{\emptyset\}$ and A, define the relations Atoms, Empty and $\in$ accordingly, initialise relations T_t for all subterms t of t_{step}, and define a relation Cur which contains exactly $\emptyset$. This relation will contain the current value of the free variable of t_{step} during the remaining execution of the program, so Cur represents $V_{t_{\text{step}}}$.

$\mathcal{I}_{\text{step}}$ simulates t_{step} and updates that relation. So let $\mathcal{I}_{\text{step}} = \mathcal{I}_{t_{\text{step}}} \circ \mathcal{I}_{\text{Cur}}$, where $\mathcal{I}_{t_{\text{step}}}$ is the interpretation simulating t_{step} according to Lemma 11. $\mathcal{I}_{\text{Cur}}$ defines Cur with the formula $\varphi_{\text{Cur}} = \exists y(\text{Cur}y \wedge T_{t_{\text{step}}}yx)$.

The program should halt whenever the current state is $\emptyset$ or $\{\emptyset\}$. So the relation Halt is defined by the formula that is true if and only if the element identified by Cur is $\emptyset$ or $\{\emptyset\}$. Analogously, the formula defining Out is true if and only if that element is $\{\emptyset\}$.

Thus, by construction, Π is equivalent to Π_{CPT}.

It remains to translate the polynomial bound. Let ϱ_{CPT} be the run of Π_{CPT} on a structure $\mathfrak{A}$ and let ϱ be the corresponding run of Π. Since each state in ϱ corresponds to a state in ϱ_{CPT}, the length of both runs is the same.

By definition of simulation, the following holds for each state $\mathfrak{A}_{i+1}$ in ϱ:

$$|\mathfrak{A}_{i+1}| \leq |\mathfrak{A}| + 4 + |\text{tc}([\![t_{\text{step}}(a_i)]\!])|,$$

where a_i is the ith state of ϱ_{CPT} (by the observations above, a_i is the only element of Cur in each state of ϱ). Since each $\text{tc}([\![t_{\text{step}}(a_i)]\!])$ is bounded by $p_{\text{CPT}}(|\mathfrak{A}|)$, the maximal size of a state in ϱ is bounded by $p_{\text{CPT}} \cdot p_{\text{CPT}}(|\mathfrak{A}|) + 4$. So if we let $p = p_{\text{CPT}} \cdot p_{\text{CPT}} + 4$, $\overline{\Pi}$ is equivalent to $\overline{\Pi}_{\text{CPT}}$. This completes the proof of Proposition 12. $\square$

VII. Fragments and Structure of Interpretation Logic

One of the motivations for the characterisation of CPT in terms of iterated first-order interpretations is the hope that this may help to build a more powerful toolkit of methods to analyse the structure and expressive power of Choiceless Polynomial Time and interpretation logic. In fact, for PIL we obtain natural fragments and stratifications along familiar syntactic parameters. In this section we follow these lines and study some important stratifications of PIL with respect to

- the signature of the interpretation $\mathcal{I}_{\text{step}}$,
- the dimension of the interpretations $\mathcal{I}_{\text{init}}$ and $\mathcal{I}_{\text{step}}$, and
- the possibility to use congruences in the interpretations $\mathcal{I}_{\text{init}}$ and $\mathcal{I}_{\text{step}}$.

As mentioned in Remark 2, a single binary relation symbol suffices to obtain the full expressive power of PIL. Monadic predicates, however, do not suffice.

Interpretations of small dimension. We next consider the restrictions of PIL along the dimension of the interpretations. We denote by k-dimensional PIL the set of all PIL-programs $((\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}), p)$ where both $\mathcal{I}_{\text{init}}$ and $\mathcal{I}_{\text{step}}$ are at most k-dimensional. It turns out that by restricting PIL to one-dimensional interpretations (with and without counting) we obtain strict fragments of PIL which are equivalent to important logical formalisms that have been studied before. On the other hand, already two-dimensional interpretations suffice to obtain the full expressive power of PIL.

A technical point that is relevant for the study of one-dimensional PIL is that, even with a Härtig quantifier, one-dimensional interpretations cannot create the additional objects (i.e. the ordinals) that are necessary to simulate counting. We therefore consider one-dimensional PIL over two-sorted structures. More precisely, for any logic $\mathcal{L}$, we write $\mathcal{L}^*$ for the application of $\mathcal{L}$ to finite structures extended by a disjoint linear order of the size of the input structure (as in the definition of FPC, but without the connection of the two sorts by counting terms or additional counting quantifiers).

It turns out that fixed-point logic with counting reappears as a one-dimensional fragment of CPT and PIL.

Theorem 13. *one-dimensional* $\text{PIL}^* \equiv \text{FPC}$

The main step in the proof is an equivalence between the iteration of one-dimensional interpretations and iterations in partial-fixed-point logic (for a definition, see [1]). Let PFPC denote partial fixed-point logic with counting. The restriction $(\text{PFPC})^* \mid_{\text{PTIME}}$ of PFPC to PTIME is defined analogously to that of interpretation logic, where we put a polynomial bound on the number of steps necessary to reach a fixed-point. Clearly, PFPC is equivalent to $(\text{PFP}+\text{H})^*$, where $\text{PFP}+\text{H}$ denotes the extension of PFP by the Härtig quantifier.

In order to simulate PFP formulae in PIL, we introduce a notion of free variables of PIL programs. Whereas, usually, the concept of free variables is not meaningful within the framework of PIL, the elements of the output structure in the one-dimensional case are already present in the input structure. We say that a one-dimensional PIL-program has free variables $x_1, \ldots, x_k$ if Out is a k-ary relation. Then $\mathfrak{A} \models \Pi(a_1, \ldots, a_k)$ if and only if $\Pi(\mathfrak{A}) \models \text{Out}a_1 \ldots a_k$.

Lemma 14. *one-dimensional* $\text{PIL}^* \equiv (\text{PFP} + \text{H})^* \mid_{\text{PTIME}}$

Proof. First note that, whenever all relations reach a fixed-point, a PIL-program has to halt because otherwise it would enter an infinite loop. So the formulae defining the relations in a PIL-program can be modified such that the relations remain unchanged whenever the formula for Halt is true. A program of that form can be translated to $(\text{PFP}+\text{H})^*$ inductively, using simultaneous fixed-points to modify all relations at once.

Because a PFP-formula cannot modify the domain of a structure, we represent the domain of the states of the PIL-program by a unary relation. Instead of joining elements with the equality formula, the PFP formula maintains a congruence relation. The only interesting step is the translation of counting formulae, since these formulae have to count equivalence classes. However, it is well-known (c.f. [16]) that counting of equivalence classes is already definable in FPC. Thus, one-dimensional PIL^*-programs can be translated to $(\text{PFP} + \text{H})^* \mid_{\text{PTIME}}$-formulae.

To show the other direction of the equivalence, we translate PFP-formulae to one-dimensional PIL-programs:

- Trivial for atomic formulae.
- For Boolean combinations, consider the concatenation $\Pi_1 \circ \Pi_2$ of two one-dimensional PIL-programs, which can easily be defined using an auxiliary relation that indi-

cates which interpretation is currently applied. A program for formulae $\varphi \wedge \psi$, $\varphi \vee \psi$ or $\neg\varphi$ is obtained by defining the Out-relation appropriately in the concatenation of the programs for the subformulae.

- For $\exists x\varphi(x)$, decrease the arity of Out and define it with $\varphi_{\text{Out}}{}' = \exists x \varphi_{\text{Out}}(x)$.
- For $\text{H}xy(\varphi(x), \psi(y))$, we define the Out-relation of the concatenation of the programs for φ and ψ using the Härtig quantifier.
- For $[\text{PFP}_{X,\bar{x}}\varphi](\bar{x})$, we add a relation X_{prev} to the output signature and define it with $\varphi_{X_{\text{prev}}}(\bar{y}) = X\bar{y}$. The program for φ is then iterated until $X = X_{\text{prev}}$. □

Theorem 13 now follows from the well-known fact (see [16]) that, in the presence of counting, polynomial-time partial fixed-point inductions are equivalent to ordinary fixed-point inductions: $\text{PFPC} \mid_{\text{PTIME}} \equiv \text{FPC}$.

Thus, with the Härtig quantifier, one-dimensional PIL over two-sorted structures is equivalent to FPC.

The arguments in the previous lemma survive on common finite structures in the absence of counting, to show that one-dimensional $\text{PIL}^- \equiv \text{PFP} \mid_{\text{PTIME}}$. However, in this case the relationship between the polynomial-time restrictions of partial fixed-point logic and least (or inflationary) fixed-point logic is (probably) a different one, due to a result by Abiteboul and Vianu [19]: $\text{PFP} \mid_{\text{PTIME}} \equiv \text{LFP}$ if and only if $\text{PTIME} = \text{PSPACE}$.

Corollary 15. *one-dimensional* $\text{PIL}^- \equiv \text{LFP}$ *if and only if* $\text{PTIME} = \text{PSPACE}$.

We see that both with and without counting, fixed-point logic appears as a one-dimensional fragment of PIL and thus CPT.

On the other side, already two-dimensional interpretations suffice for the full power of PIL.

Theorem 16.

$$\textit{two-dimensional}\ \text{PIL} \equiv \textit{two-dimensional}\ \text{PIL}^* \equiv \text{PIL}$$

Proof. It can be shown that each k-dimensional interpretation can be simulated by a sequence of two-dimensional interpretations. An interpretation logic program can apply these interpretations sequentially. So two-dimensional PIL ≡ PIL.

The number sort of the input structure in two-dimensional PIL^* can be constructed by subsequently adding elements to a specific unary relation. This only requires two-dimensional interpretations. □

Finally, let us remark that in the definition of PIL one could also use $\mathcal{L}$-interpretations for logics $\mathcal{L}$ which are more expressive than FO+H. However, it turns out that PIL over any logic $\mathcal{L}$ with $\text{FO+H} \leq \mathcal{L} \leq \text{PIL}$ has the same expressive power as PIL.

Interpretations without congruences. Another interesting fragment of PIL arises by omitting the congruences in the interpretations. Let us first discuss the case without counting, i.e. the logic PIL^-.

Clearly, in the absence of the Härtig quantifier, it is always possible to maintain the congruence relation as a part of the interpreted structures in order to simulate the behaviour of a program with congruences by a program without congruences. It does, however, seem necessary to join elements in order to preserve the polynomial bounds. Indeed we show that PIL^- without congruences is equivalent to the polynomial-time restriction of $\text{while}_{\text{new}}$. The language $\text{while}_{\text{new}}$ is an extension of while (which, in turn, is equivalent to PFP) that is strictly weaker than CPT, as shown by Blass, Gurevich and van den Bussche in [20].

The language while permits statements of the form $X := \{(x_1, \dots, x_j) \mid \varphi\}$ for FO-formulae φ, and, for while-programs P_1, P_2, the concatenation $P_1; P_2$ and the loop **while** φ **do** P_1 **od** are while-programs. The extension $\text{while}_{\text{new}}$ additionally allows to construct new elements with expressions of the form $Y := \textbf{tup-new}\{(x_1, \dots, x_j) \mid \varphi\}$. This expression creates a new element b for each tuple $(a_1, \dots, a_j)$ satisfying φ, and defines Y as the relation containing all tuples $(a_1, \dots, a_j, b)$ where b is the new element associated with $(a_1, \dots, a_j)$. A program (P, q) is in $\text{while}_{\text{new}} \mid_{\text{PTIME}}$, if both the number of atomic expressions executed and the number of new elements created during the run of the program P on each structure $\mathfrak{A}$ is bounded by $q(|\mathfrak{A}|)$.

In the following, we denote by $\approx$ -free PIL the fragment of PIL without congruences.

Lemma 17. $(\approx\text{-free PIL}^-) \equiv \text{while}_{\textit{new}} \mid_{\text{PTIME}}$.

Proof. Let (Π, p) be a $\text{PIL}[\tau, \sigma]$-program. A $\text{while}_{\text{new}}$-program P_{sim} can simulate Π by modifying relations D, N and R_i for $R_i \in \sigma$, where D represents the current domain of the state of Π, and N is the relation identifying the elements created by **tup-new**-statements. For every application of an interpretation in Π, P_{sim} creates new elements for all tuples satisfying the domain formula, assigns these elements to D and redefines the relations in σ according to the interpretation. $\mathcal{I}_{\text{step}}$ is applied in that way until Halt is true.

Then each iteration of P_{sim} corresponds to a state in the run of Π, so there is a polynomial bound on the number of atomic expressions executed in each run. Since each state in a run of Π on $\mathfrak{A}$ is bounded by $p(|\mathfrak{A}|)$, p^2 bounds the number of elements created during the run of P_{sim}. So there is a polynomial q such that (P_{sim}, q) is in $\text{while}_{\text{new}} \mid_{\text{PTIME}}$, and each $\approx$ -free PIL-program can be simulated by a $\text{while}_{\text{new}} \mid_{\text{PTIME}}$-program.

Conversely, programs of $\text{while}_{\text{new}} \mid_{\text{PTIME}}$ can be translated to $\approx$ -free PIL inductively. Relations can of course be modified with PIL-programs. Sequential execution and loops of PIL-programs are again PIL-programs (with and without congruences). So the only interesting case is the creation of new elements. As in the simulation of BGS-programs in PIL before, we want to represent each element a that is already in the domain by the new element $(a, \dots, a)$. However, this approach represents the new element associated with a tuple $(a_1, \dots, a_j)$ as the equivalence class of tuples

$(a_1, \dots, a_j, b)$ for all $b \neq a_1$. This representation cannot be directly transferred to the case without congruences: Even if equalities are encoded in a relation, the number of elements can quickly grow exponentially with a naive approach.

Therefore, we create in an initial step a sufficiently small set of elements marked by a new predicate P, and, whenever the $\text{while}_{\text{new}}$-program would create new elements, introduce one copy of the new elements for each element of P. Then a statement $Y := \textbf{tup-new}\{(x_1, \dots, x_j) \mid \varphi\}$ is simulated in a way that the new element $(a_1, \dots, a_j, b)$ is created if and only if b is in P, each a_i is either in the original domain or in the copy associated with b, and $(a_1, \dots, a_j)$ satisfies φ.

Then, whenever the $\text{while}_{\text{new}}$-program creates m new elements, the simulating PIL-program creates $|P| \cdot m$ new elements, so we can find polynomial bounds for the simulating program. □

Since $\text{while}_{\text{new}} \mid_{\text{PTIME}}$ is strictly weaker than CPT^-, we conclude that, without counting, congruences are necessary to obtain the full expressive power of PIL^-.

Theorem 18. $(\approx\text{-free PIL}^-) < \text{PIL}^-$.

Additionally, it is shown in [20] that CPT^- is equivalent to the polynomial-time restriction of $\text{while}^{\text{sets}}_{\text{new}}$, an extension of $\text{while}_{\text{new}}$ that also allows the construction of new elements associated with sets instead of just tuples. Together with the equivalence between while/PFP and one-dimensional PIL, this yields a noticeable stratification of CPT^- along natural parameters of PIL^-.

Next we consider the case with counting. Here we obtain a similar result saying that

Theorem 19. $\text{FPC} < (\approx\text{-free PIL}) < \text{PIL}$.

To embed FPC into $(\approx\text{-free PIL})$ the only difficulty is that, in order to simulate counting terms by the Härtig quantifier, we first have to construct the counting sort in $(\approx\text{-free PIL})$. The problem is that, since we do not have congruences available, it is no longer possible to add single new objects to the universe of the interpreted structures. We can, however, easily overcome this obstacle by creating not only *one*, but a polynomial number of linear orderings whose lengths correspond to the size of the input structure (by using techniques similar to the ones we used in the proof of Lemma 17).

Moreover, to see that $\text{FPC} < (\approx\text{-free PIL})$ we apply a standard padding argument. It is easy to see that also in the absence of congruences, a $(\approx\text{-free IL})$-program can create all linear orderings on the universe of the input structure. In particular, if structures come suitably padded by irrelevant elements, we can define every polynomial-time property on sufficiently small parts of the input structure in $(\approx\text{-free PIL})$. Hence, a suitably padded version of the CFI-query is definable in $(\approx\text{-free PIL})$ but not in FPC (cf. [10], [15]).

To establish the inclusion $(\approx\text{-free PIL}) < \text{PIL}$ we in fact show a more general result: on every class of structures of *bounded colour class size* each program of $(\approx\text{-free PIL})$ can be simulated by a CPT-program which only uses hereditarily finite sets of bounded rank. Before we proceed, let us first recall the notion of structures of bounded colour class size.

Definition 20. *Let* $\tau = \{R_1, \dots, R_d\}$*. A* q-bounded τ-structure $\mathcal{H}$ *is a* $\tau \uplus \{\preceq\}$*-structure* $\mathcal{H} = (H, R_1^{\mathcal{H}}, \dots, R_d^{\mathcal{H}}, \preceq)$ *where* $\preceq$ *is a preorder on* H *of width* $\leq q$*, i.e. where sets of pairwise* $\preceq$*-equivalent elements are of size at most* q*. We write* $H = C_1 \preceq \dots \preceq C_m$ *where* C_i *denotes the* i*-th* colour class.

Let us fix a class $\mathcal{K}$ of q-bounded structures (over a common vocabulary τ). Moreover, let $\overline{\Pi} = (\Pi, n^\ell)$ be a PIL-program such that $\Pi = \big(\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}\big)$ consists of two k-dimensional interpretations $\mathcal{I}_{\text{init}}$ and $\mathcal{I}_{\text{step}}$ with trivial equality formulae. In what follows we explain how we can simulate the computation of $\overline{\Pi}$ over the class $\mathcal{K}$ by a CPT-program which only accesses hereditarily finite sets of bounded rank.

The main idea is to represent the elements of the structures $\mathcal{H}_i$ in the run $(\mathcal{H}_i)$ of $\overline{\Pi}$ on a structure $\mathcal{H} \in \mathcal{K}$ where $H = C_1 \preceq \dots \preceq C_m$ by tuples in $(n^\ell \times H^{q\cdot m})$ where $n = |H|$. Since q is a constant and since we have given the preorder $\preceq$ on H of length m it is easy to see that such objects can be represented over $\mathcal{H}$ as sets of constant rank. It directly follows that also relations over such objects are sets of bounded rank. Hence, it suffices to show that such a representation for the elements of the structures $\mathcal{H}_i$ is CPT-definable.

To see this, let us consider the step from $\mathcal{H}_i$ to $\mathcal{H}_{i+1}$ in the simulation of $\overline{\Pi}$ on $\mathcal{H}$. We assume that we have already represented the elements of $\mathcal{H}_i$ as elements in $(n^\ell \times H^{q\cdot m})$. First of all, since the interpretations in $\overline{\Pi}$ are of dimension k and since the domain of the structure $\mathcal{H}_{i+1}$ is of size $\leq n^\ell$ we can easily represent the elements of $\mathcal{H}_{i+1}$ as tuples in $(n^\ell \times H^{k\cdot q\cdot m})$. To again obtain elements in $(n^\ell \times H^{q\cdot m})$ we perform the following steps. For convenience let $\lambda := k\cdot q\cdot m$.

(i) First of all we define the following strict preorder $\prec$ on the set H^λ. For $\bar a, \bar b \in H^\lambda$ we set $\bar a \prec \bar b$
   - if there exists a position $1 \leq j \leq \lambda$ such that the entries $\bar a(j)$ and $\bar b(j)$ of the tuples $\bar a$ and $\bar b$ at position j belong to different colour classes, and if for the minimal such j we have $\bar a(j) \in C_{j_a}$ and $\bar b(j) \in C_{j_b}$ with $j_a < j_b$,
   - or, in case that $\bar a$ and $\bar b$ agree component-wise on the colour classes, we set $\bar a \prec \bar b$ if there exists a colour class C_j such that the equality type t_a^j of the tuple $\bar a$ restricted to entries in C_j and the equality type t_b^j of the tuple $\bar b$ restricted to entries in C_j are different, and if for the minimal such colour class C_j we have $t_a^j < t_b^j$ (for some fixed linear order on equality types).

   It is easy to see that this preorder can be defined in CPT.

(ii) From the definition of $\prec$ it follows that each set of $\prec$-incomparable elements $[\bar a]$ can be characterised by
   - the list of colour classes to which the entries $\bar a(1), \dots, \bar a(\lambda)$ belong, and
   - the list of equality types $t_1, \dots, t_m$ of the sub-tuples $(\bar a \upharpoonright C_1), \dots, (\bar a \upharpoonright C_m)$ of $\bar a$ which arise by

restricting to entries in $C_1, \dots, C_m$, respectively.

Having fixed this information we can identify the elements from one class $[\bar{a}]$ by an element in $H^{q \cdot m}$: since we have given the equality type t_j it suffices to specify for each colour class C_j a tuple of length $\leq |C_j| \leq q$ in order to reconstruct the whole sub-tuple $(\bar{a} \upharpoonright C_j)$. To see this observe that the equality type t_j can specify at most $|C_j| \leq q$-many position to be pairwise distinct and thus it suffices to define the values for a maximal initial set of distinct positions.

Altogether, and since $\mathcal{H}_{i+1}$ contains at most n^ℓ elements, we can identify an element $(x, \bar{a}) \in (n^\ell \times H^\lambda)$ of $\mathcal{H}_{i+1}$ by a number $\leq n^\ell$ (which simultaneously encodes the first component x and the position of the $\prec$-class $[\bar{a}]$ of the second component $\bar{a}$) together with an element in $H^{q \cdot m}$. It is easy to see that all required operations can be defined in CPT by using sets of bounded rank.

Theorem 21. *On every class $\mathcal{K}$ of q-bounded structures, each* ($\approx$ -free PIL)*-program can be simulated by a* CPT*-program which only accesses sets of bounded rank.*

In [13] Dawar, Richerby and Rossman show that CPT can define the well-known isomorphism problem for the Cai, Fürer, Immerman graphs (which are 2-bounded structures). Moreover, they show that this query cannot be expressed by a CPT-program which only uses sets of bounded rank. Together with Theorem 21 this implies that ($\approx$ -free PIL) $<$ PIL $=$ CPT and thus finishes our proof of Theorem 19.

VIII. Conclusion

We presented polynomial-time interpretation logic PIL as an alternative characterisation of Choiceless Polynomial Time. PIL is based on the iteration of logical interpretations and is equivalent to CPT in the variants with and without counting. First applications of our result give further evidence that Choiceless Polynomial Time is a very natural extension of fixed-point logic with counting. We presume that our, in a sense more model-theoretic, approach makes Choiceless Polynomial Time more accessible to the tools of finite model theory.

Of course, the fundamental open question about CPT (or PIL) remains: can this logic express *all* polynomial-time decidable properties of finite structures? A natural way to proceed is to identify richer and richer classes of structures on which PIL suffices to capture PTIME. This study is closely connected with the quest for efficient graph isomorphism tests and graph canonisation algorithms, see e.g. [14]. To express such algorithms as procedures in CPT it seems necessary to invent novel CPT-data structures which can represent certain algebraic objects (more precisely, cosets of permutation groups) in a succinct way. Our characterisation of CPT by polynomial-time interpretation logic might lead to new ideas for the design of such data structures as it gives a completely new picture on the kind of objects that CPT can create and manipulate.

Moreover, as we saw in Section VII, by the equivalence of CPT and PIL we obtain new and surprising insights into the structure of the logic Choiceless Polynomial Time by considering stratifications of PIL along natural parameters. Besides those fragments of PIL that we covered in Section VII, we aim to study other such fragments, like the k-variable fragment or fragments with bounded quantifier rank. Of course, it is also very interesting to clarify the relationship between the expressive power of (such fragments of) PIL and Rank Logic.

Another intriguing question is whether the model comparison problem for such fragments $\mathcal{L} \leq$ PIL can be decided in polynomial time, i.e. whether equivalence in $\mathcal{L}$ can be used as an efficient approximation for the structure isomorphism problem. Finally, it would also be very interesting to develop game-theoretic methods which characterise these relations in the style of Ehrenfeucht-Fraïssé games.

References

[1] E. G. et. al., *Finite Model Theory and Its Applications*. Springer, 2007.

[2] M. Grohe, "The quest for a logic capturing PTIME," in *LICS 2008*, 2008, pp. 267–271.

[3] A. Chandra and D. Harel, "Structure and complexity for relational queries," *Journal of Computer and System Sciences*, vol. 25, pp. 99–128, 1982.

[4] M. Grohe, "Fixed-point definability and polynomial time on graphs with excluded minors," *Journal of the ACM (JACM)*, vol. 59, no. 5, p. 27, 2012.

[5] M. Anderson, A. Dawar, and B. Holm, "Maximum matching and linear programming in fixed-point logic with counting," in *LICS 2013*, 2013, pp. 173–182.

[6] J. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identification," *Combinatorica*, vol. 12, no. 4, pp. 389–410, 1992.

[7] A. Atserias, A. Bulatov, and A. Dawar, "Affine systems of equations and counting infinitary logic," *Theoretical Computer Science*, vol. 410, no. 18, pp. 1666–1683, 2009.

[8] A. Dawar, M. Grohe, B. Holm, and B. Laubner, "Logics with rank operators," in *LICS 2009*, 2009, pp. 113–122.

[9] A. Dawar, E. Grädel, B. Holm, E. Kopczynski, and W. Pakusa, "Definability of linear equation systems over groups and rings," *LMCS*, vol. 9, no. 4, 2013.

[10] A. Blass, Y. Gurevich, and S. Shelah, "Choiceless polynomial time," *Annals of Pure and Applied Logic*, vol. 100, no. 1, pp. 141–187, 1999.

[11] Y. Gurevich, "Logic and the challenge of computer science," in *Current Trends in Theoretical Computer Science*, E. Börger, Ed. Computer Science Press, 1988, pp. 1–57.

[12] B. Laubner, "The structure of graphs and new logics for the characterization of polynomial time," Ph.D. dissertation, Humboldt Universität zu Berlin, 2011.

[13] A. Dawar, D. Richerby, and B. Rossman, "Choiceless polynomial time, counting and the Cai–Fürer–Immerman graphs," *Annals of Pure and Applied Logic*, vol. 152, no. 1–3, pp. 31 – 50, 2008.

[14] F. Abu Zaid, E. Grädel, M. Grohe, and W. Pakusa, "Choiceless Polynomial Time on structures with small Abelian colour classes," in *Mathematical Foundations of Computer Science 2014*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8634, pp. 50–62.

[15] A. Blass, Y. Gurevich, and S. Shelah, "On polynomial time computation over unordered structures," *Journal of Symbolic Logic*, vol. 67, no. 3, pp. 1093–1125, 2002.

[16] M. Otto, *Bounded Variable Logics and Counting*. Springer, 1997.

[17] B. Rossman, "Choiceless computation and symmetry," in *Fields of logic and computation*. Springer, 2010, pp. 565–580.

[18] W. Hodges, *Model Theory*. Cambridge University Press, 1993.

[19] S. Abiteboul and V. Vianu, "Computing with first-order logic," *Journal of computer and System Sciences*, vol. 50, no. 2, pp. 309–335, 1995.

[20] A. Blass, Y. Gurevich, and J. V. den Bussche, "Abstract state machines and computationally complete query languages," in *Abstract State Machines-Theory and Applications*. Springer, 2000, pp. 22–33.