Rheinisch-Westfälische Technische Hochschule Aachen
Mathematische Grundlagen der Informatik

Computing on Abstract Structures with Logical Interpretations

Masterarbeit

Vorgelegt von:
Svenja Schalthöfer
Matrikelnummer 287652

Erstgutachter: Prof. Dr. Erich Grädel
Zweitgutachter: Prof. Dr. Martin Grohe

Dezember 2013

Contents

Chapter 1

Introduction

The work in this thesis is motivated by the central question of descriptive complexity theory: The question of whether there is a logic capturing PTIME. *Descriptive complexity theory* evolved from Fagin's result that existential second-order logic captures NP ([Fag74]), which gave rise to the general idea of capturing complexity classes with logics. Research on a logic capturing PTIME, which is recapitulated, for instance, in [Gro08], emerged from database theory, more precisely from the observation made by Aho and Ullmann ([AU79]) that SQL cannot express all queries decidable in polynomial time. As a result, Chandra and Harel asked for a recursive enumeration of the class of all polynomial-time decidable queries ([CH82]).

The slightly different question of whether there is a logic capturing PTIME was made precise by Gurevich in [Gur85]. In particular, he gave a very general definition of a logic, where the main difference between a logic and a machine model is that a logic works directly on structures, whereas a machine such as a Turing machine relies on a specific encoding. More precisely, a logic has to produce the same output on isomorphic structures. Additionally, a logic in the sense defined by Gurevich is a decidable set of objects called sentences. Therefore, the class of all Turing machines with polynomial time constraints does not constitue a logic for PTIME, as isomorphism invariance is an undecidable property of Turing machines because of Rice's theorem.

On proposing the question, Gurevich also conjectured that a logic capturing PTIME does not exist. However, the conjecture would be hard to prove: If there is no logic capturing PTIME, then PTIME $\neq$ NP, since, as shown by Fagin, there is a logic capturing NP. On the other hand, if there is a logic capturing PTIME, then the important question whether PTIME $\neq$ NP can be attacked by trying to separate existential second-order logic from that logic. So both results would have substantial consequences for complexity theory.

Assuming the conjecture that there is no logic capturing PTIME, it remains an interesting question which fragments of PTIME can be captured by logics. This may lead to a better understanding of both the complexity class PTIME and the logics analysed in the process.

The obvious reason why first-order logic is too weak to capture PTIME is that it lacks a mechanism for recursion or iteration. So fixed-point logic became a candidate for a logic for PTIME, since it enriches first-order logic by recursive computation in the form of a fixed-point operator. Indeed, fixed-point logic captures PTIME on ordered structures, which was shown independently by Immerman and Vardi ([Imm86], [Var82]).

Now one might say that this result suffices for the application in databases, since databases are ordered because of their physical representation. However, relational databases should abstract from the physical representation, so a logic for PTIME queries is still desirable. For this and the reasons given above, the search for a logic capturing PTIME continued.

Fixed-point logic itself does not capture PTIME because it fails to express already simple queries like the parity of a set. This query is based on a counting property, so Immerman ([Imm87]) introduced fixed-point logic with counting $(\mathrm{FP}+\mathrm{C})$ as a candidate for a logic capturing PTIME. But also $\mathrm{FP}+\mathrm{C}$ does not capture PTIME, as was shown by Cai, Fürer and Immerman, who, in [CFI92], defined for each natural number k a family of pairs of graphs that can be distinguished in PTIME but not by the k-variable fragment of infinitary logic with counting, and hence not in $\mathrm{FP}+\mathrm{C}$.

The candidate for a logic for PTIME that we are concerned with in this thesis is Choiceless Polynomial Time, which was introduced by Blass, Gurevich and Shelah in [BGS99]. Choiceless Polynomial Time (CPT) is a restriction of a logic referred to as BGS logic. Like fixed-point logic it has a mechanism for iteration. In addition to first-order expressions, the logic has the ability to construct hereditarily finite sets over the domain of the input structure. As a restriction, it lacks arbitrary choice (as in the pseudo code instruction "pick an arbitrary vertex"), which is replaced by parallel computation on isomorphic objects.

As the notion of computation suggests, BGS logic reminds of functional programming languages (in fact, it is based on abstract state machines). More precisely, the syntax originally introduced by Blass, Gurevich and Shelah consists of rules for updating functions and relations using the construction of sets. These rules are iterated until a given condition holds. Nevertheless, BGS logic is a logic in the general sense defined by Gurevich.

To obtain Choiceless Polynomial Time (CPT), BGS logic is restricted with polynomial bounds on the length of the computation and the amount of parallelism, which is measured in terms of the hereditarily finite sets involved in the computation.

Blass, Gurevich and Shelah showed ([BGS99]) that CPT is strictly stronger than the so-called relational machines ([AV91]), implying that CPT is also strictly stronger than fixed-point logic (which is subsumed by relational machines). But, like FP, CPT cannot express the parity of a set, as shown in [BGS99] (simplifications of parts of the proof were given by Rossman in [Ros10]). So CPT lacks the ability to count, and therefore it is reasonable to study the extension of CPT by a counting operation (resulting in $\mathrm{CPT}+\mathrm{C}$).

In [BGS02], it is shown that the existence of a perfect matching on bipartite graphs is expressable in $\mathrm{CPT}+\mathrm{C}$, after the converse was earlier conjectured by the same authors as an attempt to seperate $\mathrm{CPT}+\mathrm{C}$ from PTIME (in fact, it was shown that even $\mathrm{FP}+\mathrm{C}$ can define a perfect matching on bipartite graphs). The perfect matching problem on general graphs was then proposed as a candidate for separating $\mathrm{CPT}+\mathrm{C}$ and PTIME, but, as shown by Anderson, Dawar and Holm in [ADH13], the existence of a perfect matching on general graphs is also definable in $\mathrm{FP}+\mathrm{C}$.

Furthermore, the Cai-Fürer-Immerman query that separates $\mathrm{FP}+\mathrm{C}$ from PTIME is definable in CPT without counting ([BGS02],[DRR08]).

This suggests that $\mathrm{CPT}+\mathrm{C}$ captures at least a large fragment of PTIME. The only indication that $\mathrm{CPT}+\mathrm{C}$ does not capture PTIME is that Rossman proved in [Ros10] that there is a polynomial-time function problem not definable in $\mathrm{CPT}+\mathrm{C}$.

Another indication that $\mathrm{CPT}+\mathrm{C}$ is an interesting candidate for a logic for PTIME is the fact that, like Turing machines, it benefits from padding of the input, in [BGS99], Blass, Gurevich and Shelah described an algorithm that constructs all linear orders on a sufficiently small subset in order to decide all polynomial-time queries on this subset. In [Lau11], Laubner shows that $\mathrm{CPT}+\mathrm{C}$ captures PTIME on substructures of logarithmic size by constructing a canonisation of these substructures. The canonisation algorithm only takes singly exponential time in the size of the substructure.

Although various results have been proven about the expressive power of $\mathrm{CPT}+\mathrm{C}$, it remains open whether it can be separated from PTIME.

Since BGS logic is defined through a machine model, it cannot be analysed with the model theoretic techniques that are available for conventional logics. Therefore an alternative characterisation of BGS logic and CPT is desirable.

In this thesis, we formalise an idea proposed by Łukasz Kaiser, resulting in a representation of BGS based on first-order interpretations. With polynomial time and space restrictions, this logic, which we call interpretation logic, is equivalent to CPT.

The main idea of interpretation logic is to iterate the application of a first-order interpretation to the input structure until a given first-order sentence is true. Since interpretation logic is based on first-order logic, its definition remains relatively concise, in contrast to the definition of BGS logic. More importantly, this may make interpretation logic, and hence $\mathrm{CPT}+\mathrm{C}$, accessible to the tools of finite model theory.

The structure of interpretation logic also makes it possible to define some natural fragments, for instance via fragments of first-order logic, or by restricting the interpretations involved in the computation.

Since CPT without counting has already been separated from PTIME, we define an extension of interpretation logic that is equivalent to $\mathrm{CPT}+\mathrm{C}$. A counting operation would require interpretation logic to work on two-sorted structures like $\mathrm{FP}+\mathrm{C}$. To avoid this, we show that it suffices to extend interpretation logic (and CPT) by a suitable equicardinality operation.

Before going into the details of the proofs, we recapitulate important definitions and fix the notation in Chapter 2. In Chapter 3, we then give definitions of Choiceless Polynomial Time as introduced in [BGS99] and [Ros10] as a basis for later analysis. Then, in Section 3.2, we show that the extension of CPT by an equicardinality operation is as powerful as $\mathrm{CPT}+\mathrm{C}$, using the property that CPT works on hereditarily finite sets and therefore the finite ordinals are accessible.

This makes it possible to define a meaningful extension of interpretation logic by an equicardinality operation when we formalise interpretation logic in Chapter 4. In Section 4.1, we give some examples of interpretation logic, and in Section 4.2, we show that CPT and interpretation logic are equivalent, and that the same holds for their extensions by the respective equicardinality operation. Chapter 5 gives a conclusion and points out some questions for possible future work.

Chapter 2

Preliminaries

In this chapter, we review basic concepts of finite model theory and descriptive complexity theory and fix the notation used in the following chapters.

For a detailed introduction to finite model theory, the reader is referred to [Lib04]. Furthermore, the concept of a logic for PTIME and the corresponding generalised definition of logics is summarised in [Gro08].

2.1 Logics and Interpretations

A *signature* is a set $\tau = \{f_1, \ldots, f_k, R_1, \ldots, R_\ell\}$ of function symbols $f_1, \ldots, f_k$ and relation symbols $R_1, \ldots, R_\ell$ where each f_i is of arity s_i and each R_i is of arity r_i. τ is a *relational* signature if all elements of τ are relation symbols. A τ*-structure* $\mathfrak{A}$ is a tuple $(A, f_1^{\mathfrak{A}}, \ldots, f_k^{\mathfrak{A}}, R_1^{\mathfrak{A}}, \ldots, R_\ell^{\mathfrak{A}})$, where A is a non-empty set called the *domain* or *universe* of $\mathfrak{A}$, each $f_i^{\mathfrak{A}}$ is a function $f_i^{\mathfrak{A}} : A^{s_i} \mapsto A$ and each $R_i^{\mathfrak{A}}$ is a relation in A^{r_i}. We also write $\mathfrak{A} = (A, \tau)$. $\mathfrak{A}$ is *finite* if A is a finite set, and $\mathfrak{A}$ is *relational* if τ is relational.

Unless stated otherwise, we only consider structures that are finite and relational.

We denote by $\mathfrak{A} \upharpoonright \tau'$ the reduct of $\mathfrak{A}$ to the signature $\tau' \subseteq \tau$.

In this thesis, we introduce a logic that is based on first-order logic. *First-order logic* over the signature τ is denoted by FO$[\tau]$. For ease of notation, we allow nullary relation symbols and the formulae True and False.

To add an equicardinality operation to a logic based on FO, 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 freely in φ and y occurs freely in ψ, is a formula. The semantics is defined by $\mathfrak{A} = (A, \tau) \models \mathrm{H}\,xy\varphi\psi$ if and only if $|\{a \in A \mid \mathfrak{A} \models \varphi(a)\}| = |\{b \in A \mid \mathfrak{A} \models \psi(b)\}|$.

We denote the resulting logic by $\mathrm{FO} + \mathrm{H}$.

Since this thesis focuses on logics that are defined rather unconventionally, we review the general definition of a logic as it was introduced by Gurevich in [Gur85].

A *logic* $\mathcal{L}$ consists of a decidable set $\mathcal{L}[\tau]$ of *sentences* for each signature τ and a relation $\models$ between τ-structures and $\mathcal{L}[\tau]$-sentences such that for any structures $\mathfrak{A} \cong \mathfrak{B}$ and any sentence φ, $\mathfrak{A} \models \varphi$ if and only if $\mathfrak{B} \models \varphi$.

The logics we are concerned with in the following chapters are defined similarly to machine models. Since the output of a machine can be undefined, the resulting logics are three-valued (a logic satisfying the previous definition is called two-valued). $\mathcal{L}$ is a *three-valued logic* if $\mathcal{L}$ satisfies the conditions for a logic and $\mathfrak{A} \models \varphi$ is either true, false or undefined. If $\mathfrak{A} \cong \mathfrak{B}$ then $\mathfrak{A} \models \varphi$ is undefined if and only if $\mathfrak{B} \models \varphi$ is undefined.

In analogy to the languages decidable by a machine model, descriptive complexity theory is concerned with queries definable in a logic. A *Boolean query* is a class of finite structures that is closed under isomorphism. For a sentence $\varphi \in \mathcal{L}[\tau]$ in a two-valued logic $\mathcal{L}$, the *query* $\mathrm{Mod}(\varphi)$ *defined by* φ is the set of all τ-structures $\mathfrak{A}$ such that $\mathfrak{A} \models \varphi$. If $\mathcal{L}$ is three-valued, we additionally require that there is no structure $\mathfrak{B}$ such that the value of $\mathfrak{B} \models \varphi$ is undefined.

A query Q is definable in $\mathcal{L}$ if there is an $\mathcal{L}$-sentence defining Q.

One could alternatively classify the expressive power of a logic by the pairs of queries that it can separate. However, in the context of the search for a logic capturing PTIME, we are only interested in computations that halt in polynomial time on every input structure and therefore do not return undefined.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be (two-valued or three-valued) logics. Then $\mathcal{L}_1$ is at least as expressive as $\mathcal{L}_2$ ($\mathcal{L}_2 \leq \mathcal{L}_1$) if every Boolean query definable in $\mathcal{L}_2$ is also definable in $\mathcal{L}_1$. $\mathcal{L}_1$ and $\mathcal{L}_2$ are *expressively equivalent* ($\mathcal{L}_1 \equiv \mathcal{L}_2$) if and only if $\mathcal{L}_1 \leq \mathcal{L}_2$ and $\mathcal{L}_2 \leq \mathcal{L}_1$.

To properly embed our studies in the context of the search for a logic for PTIME, we specify the conditions for a logic for PTIME. A logic $\mathcal{L}$ *captures* PTIME if every Boolean query that is decidable in PTIME is definable in $\mathcal{L}$ and there is an algorithm that associates with each $\mathcal{L}[\tau]$-sentence φ a polynomial p and an algorithm that decides the query $\mathrm{Mod}(\varphi)$ in time $p(|A|)$

for every input structure $\mathfrak{A} = (A, \tau)$. In particular, it does not suffice that each Boolean query definable in $\mathcal{L}$ is decidable in PTIME.

The logic introduced in this thesis is based on logical interpretations, which we recapitulate in the remainder of this section.

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 $I = \big(\varphi_{\text{dom}}, \varphi_{\approx}, (\varphi_R)_{R\in\sigma}\big)$ of $\mathcal{L}[\sigma]$-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).

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 $\mathfrak{B} = I(\mathfrak{A})$ if there exists a mapping $h : \varphi_{\text{dom}}^{\mathfrak{A}} \mapsto B$ 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 $R \in \sigma$ and all $\overline{a_1}, \dots, \overline{a_r} \in \varphi_{\text{dom}}^{\mathfrak{A}}$, $(h(\overline{a_1}, \dots, \overline{a_r})) \in R^{\mathfrak{B}}$ if and only if $\mathfrak{A} \models \varphi_R(\overline{a_1}, \dots, \overline{a_r})$.

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

An interesting property of interpretations is that they define not only a mapping between structures, but an FO$[\tau, \sigma]$-interpretation also defines a mapping from τ-formulae to σ-formulae. An FO$[\tau, \sigma]$-interpretation I maps each τ-formula φ to the σ-formula φ^I, which is obtained by relativising all quantifiers in φ to φ_{dom}, replacing each formula $t_1 = t_2$ by $\varphi_{\approx}(t_1, t_2)$, and replacing every formula $R\bar{t}$ by the formula $\varphi_R(\bar{t})$. Then the following holds:

Lemma 1 (Interpretation Lemma)**.** *For every* FO$[\tau, \sigma]$*-interpretation I and every τ-structure $\mathfrak{A}$, it holds that*

$$\mathfrak{A} \models \varphi^I \textit{ if and only if } I(\mathfrak{A}) \models \varphi \ .$$

2.2 Set Theory

Next, we briefly introduce the set-theoretic concepts that form the basis for BGS logic, as well as some notation for set-theoretic encoding that we use in our BGS programs. These basic concepts that are used in the context of BGS logic are summarised in [BGS99].

We denote by $[n]$ the *von Neumann ordinal* associated with the natural number n, i.e. $[0] = \emptyset$ and $[n+1] = \{[i] \mid i \leq n\}$, and by ω the first infinite ordinal. The notation $[n]$ is used to make the set-theoretic encoding explicit, especially when defining BGS programs. We identify the cardinality $|A|$ of a finite set A with the respective von Neumann ordinal.

The sets used in BGS computations are built from *atoms*. We assume that the domain of any structure that is used as the input structure of a BGS program only consists of atoms. An *object* is either an atom or a set.

A set A is *transitive* if $x \subseteq A$ for every set $x \in A$. The *transitive closure* $\mathrm{TC}(A)$ of an object A is the least transitive set B with $A \in B$.

BGS programs operate on the hereditarily finite sets over the domain of the input structure. An object is *hereditarily finite* if its transitive closure is finite. The collection $\mathrm{HF}(A)$ of hereditarily finite sets 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.

In the BGS programs constructed in the following chapters, we frequently represent data in the form of ordered pairs and tuples of finite length. As BGS works on sets, we encode ordered pairs and tuples as follows: The pair $\langle a, b\rangle$ of objects a, b is identified with the set $\{a, \{a, b\}\}$. Tuples $\langle a_1, \ldots, a_k\rangle$ of arbitrary finite length are encoded inductively: $\langle a_1\rangle = a_1$, and $\langle a_1, \ldots, a_{k+1}\rangle = \langle\langle a_1, \ldots, a_k\rangle, a_{k+1}\rangle$. These encodings are later transformed to terms in BGS logic.

Chapter 3

Choiceless Polynomial Time

In this chapter, we give two definitions of Choiceless Polynomial Time and the underlying logic, which is denoted by BGS logic. Choiceless Polynomial Time (CPT) was originally defined by Blass, Gurevich and Shelah in [BGS99]. A more concise definition was presented, for instance, by Rossman in [Ros10]. We will denote the latter by BGS, respectively CPT, and the logic originally defined by Blass, Gurevich and Shelah by BGS_{orig}, respectively CPT_{orig}.

The core of the logic in both versions is to combine an iteration mechanism with the ability to construct arbitrary hereditarily finite sets (within constraints given by polynomial bounds). The construction of sets is realised by the terms of the logic, a fragment that is similar to first-order logic, where, instead of quantification, constructs of the form $\{s(x) : x \in t : \varphi(x)\}$ for terms s, t, φ are possible. The concept that constitutes a sentence in other logics is called a program in BGS and BGS_{orig}, reflecting the property that the logic is defined like a machine model.

In BGS_{orig}, a program consists of a set of rules that modify dynamic functions and predicates, and that are iterated until a given condition is true. According to the newer definition, the computation of a BGS program is simply the application of a single term until the halting condition is true, which shortens the definition.

CPT and CPT_{orig} are the fragments of BGS and BGS_{orig} defined by restricting the computations with certain polynomial bounds. The main idea is to bound the number of sets that are involved in the computation, which are called active objects.

We start with a few general definitions that form the basis of both BGS and BGS_{orig}.

As BGS and $\mathrm{BGS}_{\mathrm{orig}}$ work on hereditarily finite sets, the terms are constructed over a signature extended by set-theoretic function and relation symbols:

Definition 2. *Let σ be a relational signature. We denote by σ^{HF} the signature σ extended by the following function and relation symbols:*

- *A binary relation symbol* In*,*
- *nullary function symbols* Empty *and* Atoms*,*
- *unary function symbols* Union *and* TheUnique*,*
- *a binary function symbol* Pair*.*

Futhermore, we denote by $\sigma^{\mathrm{HF}}_{\mathrm{EqCard}}$ the signature $\sigma^{\mathrm{HF}} \cup \{\mathrm{EqCard}\}$, where EqCard *is a 2-ary relation symbol, and by $\sigma^{\mathrm{HF}}_{\mathrm{Card}}$ the signature $\sigma^{\mathrm{HF}} \cup \{\mathrm{Card}\}$, where* Card *is a unary function symbol.*

The predicates in σ are called input predicates*.*

The constant symbol Empty is sometimes abbreviated to $\emptyset$.

The corresponding functions and relations are defined by the hereditarily finite expansion of each structure:

Definition 3. *Let $\mathfrak{A} = (A, \sigma)$. Then the* hereditarily finite expansion $\mathrm{HF}(\mathfrak{A})$ *of $\mathfrak{A}$ is the structure with universe* $\mathrm{HF}(A)$ *and*

- $(a, b) \in \mathrm{In}^{\mathrm{HF}(\mathfrak{A})}$ *if and only if $a \in b$,*
- $\mathrm{Empty}^{\mathrm{HF}(\mathfrak{A})} = \emptyset$*,*
- $\mathrm{Atoms}^{\mathrm{HF}(\mathfrak{A})} = A$*,*
- $\mathrm{Union}^{\mathrm{HF}(\mathfrak{A})}(a) = \bigcup_{b \in a} b$*,*
- $\mathrm{TheUnique}^{\mathrm{HF}(\mathfrak{A})}(\{b\}) = b$ *and* $\mathrm{TheUnique}^{\mathrm{HF}(\mathfrak{A})}(a) = \emptyset$ *if a is not a singleton, and*
- $\mathrm{Pair}^{\mathrm{HF}(\mathfrak{A})}(a, b) = \{a, b\}$*.*

Let $\mathrm{HF}_{\mathrm{EqCard}}(\mathfrak{A})$ *be the* $\sigma^{\mathrm{HF}}_{\mathrm{EqCard}}$*-expansion of* $\mathrm{HF}(\mathfrak{A})$ *with*

$$\mathrm{EqCard}^{\mathrm{HF}_{\mathrm{EqCard}}(\mathfrak{A})} = \{(a,b) \mid a, b \textit{ are sets and } |a| = |b|\} \ ,$$

and let $\mathrm{HF}_{\mathrm{Card}}(\mathfrak{A})$ *be the* $\sigma^{\mathrm{HF}}_{\mathrm{Card}}$*-expansion of* $\mathrm{HF}(\mathfrak{A})$ *with*

$$\mathrm{Card}^{\mathrm{HF}_{\mathrm{Card}}(\mathfrak{A})}(a) = |a| \ .$$

Both BGS and $\mathrm{BGS}_{\mathrm{orig}}$ are based on the concept of terms. Since the definition of terms in both versions of the logic differs only by small variations of the syntax, we give the same definition for both BGS and $\mathrm{BGS}_{\mathrm{orig}}$ terms.

Definition 4. *The set of terms over a signature* $\sigma^{\mathrm{HF}}_{ext} \supseteq \sigma^{\mathrm{HF}}$ *is defined inductively as follows:*

Induction Base:

- *A variable is a term,*
- *each constant symbol* $c \in \sigma^{\mathrm{HF}}_{ext}$ *is a term.*

Induction Step:

- *If* $t_1, \dots, t_r$ *are terms, then* $f(t_1, \dots, t_r)$ *is a term for each* r*-ary function symbol* $f \in \sigma^{\mathrm{HF}}_{ext}$*,*
- *if* $t_1, \dots, t_r$ *are terms, then* $R(t_1, \dots, t_r)$ *is a Boolean term,*
- *if* t_1, t_2 *are terms, then* $t_1 = t_2$ *is a Boolean term,*
- *if* t_1, t_2 *are Boolean terms, then* $t_1 \wedge t_2$*,* $t_1 \vee t_2$ *and* $\neg t_1$ *are Boolean terms,*
- *if* s *and* t *are terms,* φ *is a Boolean term and* x *is a variable that does not occur freely in* t*, then* $\{s(x) : x \in t : \varphi(x)\}$ *is a term. This term binds the variable* x*. We say that* $\{s(x) : x \in t : \varphi(x)\}$ *is a comprehension term. Instead of* $\{s(x) : x \in t : \varphi(x)\}$ *we also write* $\{s(x) : x \in t\}$ *if* φ *is a tautology.*

When constructing terms for e.g. set-theoretic operations, we want to replace arbitrary terms for the free variables. Therefore, if $s(x_1, \dots, x_k)$ and $t_1, \dots, t_k$ are terms, then we denote by $s(t_1, \dots, t_k)$ the modified version of s where each free variable x_i is replaced with the term t_i. The same notation is applied for replacing the free variables by values $a_1, \dots, a_k$ from $\mathrm{HF}(A)$.

The evaluation of a term yields a value in $\mathrm{HF}(A)$, using the functions and relations of the respective expansion of $\mathrm{HF}(\mathfrak{A})$. For Boolean terms, the value is either $\{\emptyset\}$ or $\emptyset$, where $\{\emptyset\}$ encodes True and $\emptyset$ encodes False.

Definition 5. *Let $\sigma^{\mathrm{HF}}_{ext} \supseteq \sigma^{\mathrm{HF}}$, and let $\mathrm{HF}_{ext}(\mathfrak{A})$ be a $\sigma^{\mathrm{HF}}_{ext}$-expansion of $\mathrm{HF}(\mathfrak{A})$ for a σ-structure $\mathfrak{A}$. The semantic operator $[\![\cdot]\!]^{\mathfrak{A}}$ assigns to each term over $\sigma^{\mathrm{HF}}_{ext}$ with an assignment of free variables $\overline{x} = x_1, \dots, x_k$ to values $\overline{a} = a_1, \dots, a_k$ in $\mathrm{HF}(A)$ its evaluation for the input structure $\mathfrak{A}$. The operator is defined inductively as follows:*

Induction Base

- *If $t(x) = x$, then $[\![t(a)]\!]^{\mathfrak{A}} = a$.*
- *If $t(\overline{x}) = c$, then $[\![t]\!]^{\mathfrak{A}} = c^{\mathrm{HF}_{ext}(\mathfrak{A})}$.*

Induction Step

- $[\![f(t_1(\overline{a}), \dots, t_r(\overline{a}))]\!]^{\mathfrak{A}} = f^{\mathrm{HF}_{ext}(\mathfrak{A})}([\![t_1(\overline{a})]\!]^{\mathfrak{A}}, \dots, [\![t_r(\overline{a}))]\!]^{\mathfrak{A}})$.
- $[\![R(t_1(\overline{a}), \dots, t_r(\overline{a}))]\!]^{\mathfrak{A}} = \{\emptyset\}$ *if* $([\![t_1(\overline{a})]\!]^{\mathfrak{A}}, \dots, [\![t_r(\overline{a})]\!]^{\mathfrak{A}}) \in R^{\mathrm{HF}_{ext}(\mathfrak{A})}$, *and* $[\![R(t_1(\overline{a}), \dots, t_r(\overline{a}))]\!]^{\mathfrak{A}} = \emptyset$ *otherwise.*
- *Analogous for $[\![t_1(\overline{a}) = t_2(\overline{a})]\!]^{\mathfrak{A}}$ and Boolean connectives, with the obvious semantics.*
- $[\![\{s(x, \overline{a}) : x \in t(\overline{a}) : \varphi(x, \overline{a})\}]\!]^{\mathfrak{A}}$ *is the set of all* $[\![s(b, \overline{a})]\!]^{\mathfrak{A}}$ *such that* $b \in [\![t(\overline{a})]\!]^{\mathfrak{A}}$ *and* $[\![\varphi(b, \overline{a})]\!]^{\mathfrak{A}}$ *is true.*

Both BGS and $\mathrm{BGS}_{\mathrm{orig}}$ heavily rely on terms to construct sets in $\mathrm{HF}(A)$. Apart from that, the approach of these logics differs in several aspects, which will be defined in the following.

The definition of $\mathrm{BGS}_{\mathbf{orig}}$ The computations of $\mathrm{BGS}_{\mathrm{orig}}$ programs are based on updating dynamic functions. Therefore, we further expand the signature σ^{HF} by a finite set of dynamic function symbols which contains at least the nullary predicates Halt and Out. Note that dynamic predicates are encoded as dynamic functions which can only take the values $\{\emptyset\}$ (True) and $\emptyset$ (False). So, for the definition of $\mathrm{BGS}_{\mathrm{orig}}$, we only consider signatures $\sigma^{\mathrm{HF}}_{\mathrm{ext}}$ containing dynamic functions, specifically Halt and Out.

$\mathrm{BGS}_{\mathrm{orig}}$ programs are constructed from rules that perform certain updates on the states of the computation, so before introducing rules and their semantics, we define $\mathrm{BGS}_{\mathrm{orig}}$ states:

A $\mathrm{BGS}_{\mathrm{orig}}$ *state* $\mathfrak{B}$ is a $\sigma^{\mathrm{HF}}_{\mathrm{ext}}$-expansion of $\mathrm{HF}(\mathfrak{A})$ (respectively $\mathrm{HF}_{\mathrm{EqCard}}(\mathfrak{A})$ if EqCard $\in \sigma^{\mathrm{HF}}_{\mathrm{ext}}$ and $\mathrm{HF}_{\mathrm{Card}}(\mathfrak{A})$ if Card $\in \sigma^{\mathrm{HF}}_{\mathrm{ext}}$), where, for every dynamic function symbol $f \in \sigma^{\mathrm{HF}}_{\mathrm{ext}}$, the set $\{(a_0, \dots, a_r) : f^{\mathfrak{B}}(a_0, \dots, a_{r-1} = a_r \neq \emptyset\}$

is finite. $\mathfrak{B}$ is an *initial state* if each dynamic function takes the value $\emptyset$ for each tuple $a_0, \ldots, a_{r-1}$.

States are used to define the semantics of rules. In the following, we first define rules syntactically:

Definition 6. *The set of* BGS_{orig} *transition rules over a signature* $\sigma_{ext}^{\mathrm{HF}}$ *is defined inductively as follows (where all terms are terms over* $\sigma_{ext}^{\mathrm{HF}}$*):*

Induction Base

- Skip *is a rule.*
- Update rules: *If f is an r-ary dynamic function symbol and $t_0, \ldots, t_r$ are terms, then $f(t_1, \ldots, t_r) := t_0$ is a rule.*

Induction Step

- Conditional rules: *If φ is a Boolean term and R_1, R_2 are rules, then* ***`if`*** *φ* ***`then`*** *R_1* ***`else`*** *R_2* ***`endif`*** *is a rule.*
- Do-forall rules: *If x is a variable, t is a term where x does not occur freely, and $R_0(x)$ is a rule, then* ***`do forall`*** *$x \in t$, $R_0(x)$* ***`enddo`*** *is a rule.*

The semantics of $\mathrm{BGS}_{\mathrm{orig}}$ rules is defined in terms of actions, i.e. each rule assigns an action to each state. This action will then define the sequel of a state, i.e. that state obtained from the current state by using the rule, which is later on used to define the computation of a program. To define actions, and, finally, the denotation of a rule, we need the following prerequisites.

Actions are defined on structures with variable assignments, formally: Let $\mathfrak{A}$ be a state and let β be a variable assignment. Then $(\mathfrak{A}, \beta)$ is an *expanded state.* Let $(\mathfrak{A}, \beta)$ be an expanded state. Then $(\mathfrak{A}, \beta)(x \mapsto a)$ is the expanded state obtained from $(\mathfrak{A}, \beta)$ by extending β such that a is assigned to x.

Actions consist of updates. An *update* of $(\mathfrak{A}, \beta)$ is a tuple $(f, \overline{a}, b)$, where f is an r-ary dynamic function, $\overline{a} \in A^r$ and $b \in A$. To *fire* an update $(f, \overline{a}, b)$ at $(\mathfrak{A}, \beta)$, redefine f in $\mathfrak{A}$ such that $f(\overline{a}) = b$ and f remains unchanged on all other tuples. The resulting state is called the *sequel* of $(\mathfrak{A}, \beta)$ with respect to $(f, \overline{a}, b)$.

The updates $(f, \overline{a}, b)$ and $(f, \overline{a}, c)$ *clash* if $b \neq c$.

An *action* of an expanded state $(\mathfrak{A}, \beta)$ is a set of updates of $(\mathfrak{A}, \beta)$. An action α is *performed* by firing all updates if α contains no clashing updates, and by doing nothing otherwise. Again, the resulting expanded state is called the *sequel* of $(\mathfrak{A}, \beta)$ with respect to α.

The *denotation* $\mathrm{Den}(R)$ of a rule R is a function mapping each expanded state $(\mathfrak{A}, \beta)$ that is *appropriate* for R (i.e. where R contains only function symbols in the signature of $\mathfrak{A}$, and β assigns all free variables in R) to an action. We write $\mathrm{Den}(R, (\mathfrak{A}, \beta))$ instead of $\mathrm{Den}(R)((\mathfrak{A}, \beta))$.

To *fire* R at $(\mathfrak{A}, \beta)$, perform the action $\mathrm{Den}(R, (\mathfrak{A}, \beta))$ at $\mathfrak{A}$. The *sequel* of $(\mathfrak{A}, \beta)$ with respect to R is the sequel of $(\mathfrak{A}, \beta)$ with respect to $\mathrm{Den}(R, (\mathfrak{A}, \beta))$.

The denotation of rules is defined inductively:

- $\mathrm{Den}(\mathrm{Skip}, (\mathfrak{A}, \beta)) = \emptyset$, so Skip is always mapped to no action.
- For $R = f(\overline{s}) := t$, $\mathrm{Den}(R, (\mathfrak{A}, \beta)) = \{(f, [\![\overline{s}]\!]^{\mathfrak{A}}, [\![t]\!]^{\mathfrak{A}})\}$, i.e. the denotation is the set containing only the update that redefines f as specified by R.
- If $R =$ `if` φ `then` R_1 `else` R_2 `endif`, then

$$\mathrm{Den}(R, (\mathfrak{A}, \beta)) = \begin{cases} \mathrm{Den}(R_1, (\mathfrak{A}, \beta)) & \text{if } [\![\varphi]\!]^{\mathfrak{A}} \text{ is true,} \\ \mathrm{Den}(R_2, (\mathfrak{A}, \beta)) & \text{otherwise.} \end{cases}$$

- If $R =$`do forall` $x \in t$, $R_0(x)$ `enddo`, then

$$\mathrm{Den}(R, (\mathfrak{A}, \beta)) = \bigcup \{\mathrm{Den}(R_0(x), (\mathfrak{A}, \beta)(x \mapsto a)) : a \in [\![t]\!]^{\mathfrak{A}}\} \ ,$$

  so the action assigned to a do-forall rule is the set of all updates defined by the subcomputations.

The notion of the denotation of a rule makes it possible to define $\mathrm{BGS}_{\mathrm{orig}}$ programs and their computations:

Definition 7. *A* $\mathrm{BGS}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$ program Π *is a rule without free variables over* $\sigma^{\mathrm{HF}}_{ext}$.

States of Π *are states over the signature* $\sigma^{\mathrm{HF}}_{ext}$.

Let $\mathfrak{A}$ *be a* σ*-structure. Without loss of generality, we assume that the domain of* $\mathfrak{A}$ *only consists of atoms. The* run *of a* $\mathrm{BGS}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$ *program* Π *on* $\mathfrak{A}$ *is a sequence* $(\mathfrak{A}_i)_{i<\kappa}$*, where* κ *is a natural number or* ω*, such that*

- $\mathfrak{A}_0$ *is an initial state with domain* $\mathrm{HF}(A)$*,*

- *$\mathfrak{A}_{i+1}$ is a sequel of $\mathfrak{A}_i$ with respect to Π for $i+1 < \kappa$,*
- *$\mathrm{Halt}^{\mathfrak{A}_i}$ is false for $i < \kappa - 1$,*
- *if κ is finite, then $\mathrm{Halt}^{\mathfrak{A}_{\kappa-1}}$ is true.*

If κ is finite, then Π accepts *$\mathfrak{A}$ if and only if $\mathrm{Out}^{\mathfrak{A}_{\kappa-1}}$ is true. If κ is infinite, then the output of Π is $\bot$.*

The polynomial-time restriction of $\mathrm{BGS}_{\mathrm{orig}}$, which we denote by $\mathrm{CPT}_{\mathrm{orig}}$, is defined in terms of the objects that are involved in a computation:

Definition 8. *Let $\mathfrak{B}$ be a BGS_{orig} state with universe $\mathrm{HF}(A)$ for some structure $\mathfrak{A} = (A, \sigma)$.*

An object $a \in \mathrm{HF}(A)$ is critical *at $\mathfrak{B}$ if*

- *a is an atom,*
- *$a \in \{\emptyset, \{\emptyset\}\}$,*
- *a is the value of a dynamic function, or*
- *there is a dynamic function f and a tuple $\bar{b}$ such that a is a component of $\bar{b}$ and $f^{\mathfrak{B}}(\bar{b}) \neq \emptyset$.*

An object $a \in \mathrm{HF}(A)$ if active *at $\mathfrak{A}$ if it is in $\mathrm{TC}(b)$ for some critical object b. If b is a value of the dynamic function f, or b is a component of a tuple where f takes a value different from $\emptyset$, we say that a is* activated by *f.*

An object is active in a run *ρ is it is active at some state of ρ.*

The polynomial-time restriction $\mathrm{CPT}_{\mathrm{orig}}$ of $\mathrm{BGS}_{\mathrm{orig}}$ is now defined by restricting the length and the number of active objects of a run of a program.

Definition 9. *A $\mathrm{CPT}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$* program *is a tuple $\overline{\Pi} = (\Pi, p, q)$, where Π is a $\mathrm{BGS}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$ program and $p : \mathbb{N} \mapsto \mathbb{N}$ and $q : \mathbb{N} \mapsto \mathbb{N}$ are polynomials.*

The run *of $\overline{\Pi}$ on a structure $\mathfrak{A} = (A, \sigma)$ is the maximal initial segment $\rho = (\mathfrak{A}_i)_{i<n}$ of the run of Π on $\mathfrak{A}$ such that $n < p(|A|)$ and at most $q(|A|)$ many objects are active in ρ.*

If $\mathrm{Halt}^{\mathfrak{A}_{n-1}}$ is true, then $\overline{\Pi}$ accepts $\mathfrak{A}$ if and only if Π accepts $\mathfrak{A}$. Otherwise, the output of $\overline{\Pi}$ is $\bot$.

A program Π is in $\text{BGS}_{\text{orig}} + \text{EqCard}$ (resp. $\text{CPT}_{\text{orig}} + \text{EqCard}$) if it is in $\text{BGS}_{\text{orig}}[\sigma^{\text{HF}}_{\text{ext}}]$ (resp. $\text{CPT}_{\text{orig}}[\sigma^{\text{HF}}_{\text{ext}}]$) for some signature where $\text{EqCard} \in \sigma^{\text{HF}}_{\text{ext}}$, and Π is in $\text{BGS}_{\text{orig}} + \text{C}$ (resp. $\text{CPT}_{\text{orig}} + \text{C}$) if Π is a BGS_{orig} (resp. CPT_{orig}) program over a signature $\sigma^{\text{HF}}_{\text{ext}}$ where $\text{Card} \in \sigma^{\text{HF}}_{\text{ext}}$.

Note that both BGS_{orig} and CPT_{orig} are three-valued logics, since the set of valid programs is defined syntactically and is therefore decidable.

The definition of BGS In the definition of BGS logic given by Rossman in [Ros10], the rules are omitted and programs are constructed directly from terms.

Definition 10. *Let σ^{HF}_{ext} be a signature with $\sigma^{\text{HF}}_{ext} \supseteq \sigma^{\text{HF}}$. A* $\text{BGS}[\sigma^{\text{HF}}_{ext}]$ program *is a tuple $\Pi = (\Pi_{step}, \Pi_{halt}, \Pi_{out})$, where $\Pi_{step}(x)$ is a non-Boolean* BGS *term over σ^{HF}_{ext} with one free variable and Π_{halt} and Π_{out} are Boolean* BGS *terms over σ^{HF}_{ext} without free variables.*

Let $\mathfrak{A} = (A, \sigma)$ be a σ-structure. The run *of Π on $\mathfrak{A}$ is the sequence $(a_i)_{i<\kappa}$ of elements of $\text{HF}(A)$, where κ is a natural number or ω, and*

- $a_0 = \emptyset$,
- $a_{i+1} = [\![\Pi_{step}(a_i)]\!]$ *for* $0 < i + 1 < \kappa$,
- $[\![\Pi_{halt}(a_i)]\!]$ *is false for all* $i < \kappa - 1$,
- *if κ is finite, then $[\![\Pi_{halt}(a_{\kappa-1})]\!]$ is true.*

If κ is finite, then the output of Π is $\Pi(\mathfrak{A}) = [\![\Pi_{out}(a_{\kappa-1})]\!]$, and otherwise, $\Pi(\mathfrak{A}) = \bot$.

The polynomial-time restriction of BGS only depends on the number of objects involved in the computation. Since a computation of a BGS program consists of evaluations of terms, the set of active objects is now defined as a property of a term.

Definition 11. *Let $\sigma^{\text{HF}}_{ext} \supseteq \sigma^{\text{HF}}$, and let $\text{HF}_{ext}(\mathfrak{A})$ be the σ^{HF}_{ext}-expansion of $\text{HF}(\mathfrak{A})$ for a σ-structure $\mathfrak{A} = (A, \sigma)$. The operator $\langle\!\langle \cdot \rangle\!\rangle^{\mathfrak{A}}$ denotes the set of active objects of a term evaluated in the structure $\mathfrak{A}$. It is defined inductively for every term $t(x_1, \ldots, x_k)$ and assignment of variables $\overline{x} = x_1, \ldots, x_k$ to values $\overline{a} = a_1, \ldots, a_k$ in $\text{HF}(A)$:*

Induction Base:

- *If* $t(x_i) = x_i$, *then* $\langle\!\langle t(a) \rangle\!\rangle^{\mathfrak{A}} = a$.
- *If* $t = c$ *for a constant symbol* $c \in \sigma_{ext}^{\mathrm{HF}}$, *then* $\langle\!\langle t \rangle\!\rangle = \{c^{\mathrm{HF}_{ext}(\mathfrak{A})}\}$.

Induction Step:

- $\langle\!\langle f(t_1(\overline{a}), \ldots, t_r(\overline{a})) \rangle\!\rangle^{\mathfrak{A}} = \{[\![f(t_1(\overline{a}), \ldots, t_r(\overline{a}))]\!]^{\mathfrak{A}}\} \cup \bigcup_{i=1}^{r} \langle\!\langle t_i(\overline{a}) \rangle\!\rangle^{\mathfrak{A}}$.
- $\langle\!\langle R(t_1(\overline{a}), \ldots, t_r(\overline{a})) \rangle\!\rangle^{\mathfrak{A}} = \bigcup_{i=1}^{r} \langle\!\langle t_i(\overline{a}) \rangle\!\rangle^{\mathfrak{A}}$.
- $\langle\!\langle \neg\varphi(\overline{a}) \rangle\!\rangle^{\mathfrak{A}} = \langle\!\langle \varphi(\overline{a}) \rangle\!\rangle^{\mathfrak{A}}$.
- $\langle\!\langle \varphi(\overline{a}) \wedge \psi(\overline{a}) \rangle\!\rangle^{\mathfrak{A}} = \langle\!\langle \varphi(\overline{a}) \vee \psi(\overline{a}) \rangle\!\rangle^{\mathfrak{A}} = \langle\!\langle \varphi(\overline{a}) \rangle\!\rangle^{\mathfrak{A}} \cup \langle\!\langle \psi(\overline{a}) \rangle\!\rangle^{\mathfrak{A}}$.
- $\langle\!\langle t_1(\overline{a}) = t_2(\overline{a}) \rangle\!\rangle^{\mathfrak{A}} = \langle\!\langle t_1(\overline{a}) \rangle\!\rangle^{\mathfrak{A}} \cup \langle\!\langle t_2(\overline{a}) \rangle\!\rangle^{\mathfrak{A}}$.
- $\langle\!\langle \{s(x, \overline{a}) : x \in t(\overline{a}) : \varphi(x, \overline{a})\} \rangle\!\rangle^{\mathfrak{A}} = \langle\!\langle t(\overline{a}) \rangle\!\rangle^{\mathfrak{A}} \cup \bigcup_{b \in [\![t(\overline{a})]\!]^{\mathfrak{A}}} \langle\!\langle \varphi(b, \overline{a}) \rangle\!\rangle^{\mathfrak{A}} \cup$ $\{[\![\{s(x, \overline{a}) : x \in t(\overline{a}) : \varphi(x, \overline{a})\}]\!]^{\mathfrak{A}}\}$.

The logic CPT is defined as the restriction of BGS to programs with runs of finite length and a polynomially bounded number of active objects on every structure.

Definition 12. *A* CPT$[\sigma_{ext}^{\mathrm{HF}}]$ program *is a tuple* $\overline{\Pi} = (\Pi, q)$, *where* Π *is a* BGS$[\sigma_{ext}^{\mathrm{HF}}]$ *program and* $q : \mathbb{N} \mapsto \mathbb{N}$ *is a polynomial.*

The run *of* $\overline{\Pi}$ *on a structure* $\mathfrak{A} = (A, \sigma)$ *is the maximal initial segment* $\rho = (a_i)_{i<n}$ *of the run of* Π *on* $\mathfrak{A}$ *such that no state occurs twice and the cardinality of the set*

$$\mathrm{Active}(\overline{\Pi}, \mathfrak{A}) = \langle\!\langle \Pi_{halt}(a_{n-1}) \rangle\!\rangle \cup \langle\!\langle \Pi_{out}(a_{n-1}) \rangle\!\rangle \cup \bigcup_{i=0}^{n-2} (\langle\!\langle \Pi_{step}(a_i) \rangle\!\rangle \cup \langle\!\langle \Pi_{halt}(a_i) \rangle\!\rangle)$$

is at most $q(|A|)$.

If $[\![\Pi_{halt}(a_{n-1})]\!]$ *is true, then* $\overline{\Pi}(\mathfrak{A}) = \Pi(\mathfrak{A})$. *Otherwise,* $\overline{\Pi}(\mathfrak{A}) = \bot$.

Note that, according to Rossman's definition, a CPT program does not explicitly incorporate the polynomial q. However, the above definition ensures that CPT is still a three-valued logic, in contrast to the language defined by the property that there exists a polynomial q that bounds the number of active objects, where the set of programs is not decidable. Using the above definition, both BGS and CPT are three-valued logics.

As for $\mathrm{BGS}_{\mathrm{orig}}$, we denote by $\mathrm{BGS}+\mathrm{EqCard}$ (resp. $\mathrm{CPT}+\mathrm{EqCard}$) the logic $\mathrm{BGS}[\sigma^{\mathrm{HF}} \cup \{\mathrm{EqCard}\}]$ (resp. $\mathrm{CPT}[\sigma^{\mathrm{HF}} \cup \{\mathrm{EqCard}\}]$), and by $\mathrm{BGS}+\mathrm{C}$ the logic $\mathrm{BGS}[\sigma^{\mathrm{HF}} \cup \{\mathrm{Card}\}]$ (resp. $\mathrm{CPT}[\sigma^{\mathrm{HF}} \cup \{\mathrm{Card}\}]$).

Note that this definition does not bound the length of a run. Therefore we show that the bound on the number of active objects already implies a bound on the length of the run.

Lemma 13. *Let $\overline{\Pi} = (\Pi, q)$ be a* CPT *program, and let $\mathfrak{A} = (A, \tau)$ be a finite structure. Then the length of the run of $\overline{\Pi}$ on $\mathfrak{A}$ is bounded by $q(|A|)$.*

Proof. For any state a_i of the run ρ of Π on $\mathfrak{A}$, $a_i = \emptyset$ or $a_i = [\![\Pi_{\mathrm{step}}(a)]\!]$ for some object a, so a_i is in $\mathrm{Active}(\overline{\Pi}, \mathfrak{A})$. By definition, no state occurs twice. So there are at most $q(|A|)$ states in ρ. □

3.1 Basic Set-Theoretic Operations

To illustrate the use of BGS logic and the computation of active objects and to simplify the BGS programs defined later, we demonstrate how to express some set-theoretic operations with BGS (and thus $\mathrm{BGS}_{\mathrm{orig}}$) terms.

First, we define a term defining the set $\{a\}$ for any object a in the hereditarily finite expansion of the input structure.

Lemma 14. *There is a* BGS *term $t_{sing}(x)$ such that for any term t, $[\![t_{sing}(t)]\!] = \{[\![t]\!]\}$ and $|\langle\!\langle t_{sing}(t)\rangle\!\rangle| = |\langle\!\langle t\rangle\!\rangle| + 2$.*

Proof. Let $t_{\mathrm{sing}}(x) = \mathrm{Pair}(x, x)$. Clearly, the term has the required semantics, and $|\langle\!\langle t_{\mathrm{sing}}\rangle\!\rangle| = |\{[\![\mathrm{Pair}(t,t)]\!]\} \cup \langle\!\langle t\rangle\!\rangle| = |\langle\!\langle t\rangle\!\rangle| + 1$. □

For ease of notation, we write $\{t\}$ instead of $t_{\mathrm{sing}}(t)$.

The Union function in BGS is only defined as the union over a set. Next, we define the union of two sets as a special case of that function.

Lemma 15. *There is a* BGS *term $t_{\cup}(x, y)$ such that for any terms t_1, t_2, $[\![t_{\cup}(t_1, t_2)]\!] = [\![t_1]\!] \cup [\![t_2]\!]$ and $|\langle\!\langle t_{\cup}(t_1, t_2)\rangle\!\rangle| \leq |\langle\!\langle t_1\rangle\!\rangle \cup \langle\!\langle t_2\rangle\!\rangle| + 2$.*

Proof. Let $t_{\cup}(x, y) = \mathrm{Union}\,(\mathrm{Pair}\,(x, y))$. Then by definition $[\![t_{\cup}(t_1, t_2)]\!] = [\![t_1]\!] \cup [\![t_2]\!]$, and

$$\begin{aligned}\langle\!\langle t_{\cup}(t_1, t_2)\rangle\!\rangle &= \{[\![t_{\cup}(t_1, t_2)]\!]\} \cup \langle\!\langle \mathrm{Pair}(t_1, t_2)\rangle\!\rangle \\ &= \{[\![t_1]\!] \cup [\![t_2]\!]\} \cup \{\{[\![t_1]\!], [\![t_2]\!]\}\} \cup \langle\!\langle t_1\rangle\!\rangle \cup \langle\!\langle t_2\rangle\!\rangle \ ,\end{aligned}$$

so $|\langle\!\langle t_{\cup}(t_1, t_2)\rangle\!\rangle| = |\langle\!\langle t_1\rangle\!\rangle \cup \langle\!\langle t_2\rangle\!\rangle| + 2$. □

Instead of $t_\cup(x, y)$, we use the standard notation $x \cup y$.

The BGS programs formulated later encode data in ordered pairs and tuples. In the following, we present BGS terms defining ordered pairs and tuples using the set-theoretic encoding specified in Section 2.2. Whenever a tuple or pair occurs in a BGS term, it will denote an application of the respective term.

Lemma 16. *There is a* BGS *term* $\mathrm{OrderedPair}(x, y)$ *such that, for any terms* s, t, $[\![\mathrm{OrderedPair}(s, t)]\!] = \langle [\![s]\!], [\![t]\!] \rangle$ *and* $|\langle\!\langle \mathrm{OrderedPair}(s, t) \rangle\!\rangle| = |\langle\!\langle s \rangle\!\rangle \cup \langle\!\langle t \rangle\!\rangle| + 2$.

Proof. Let $\mathrm{OrderedPair}(x, y) = \mathrm{Pair}(x, \mathrm{Pair}(x, y))$. Then, by definition, the term constructs an ordered pair, and, for any terms s, t,

$$\begin{aligned}\langle\!\langle \mathrm{OrderedPair}(s, t) \rangle\!\rangle &= \langle\!\langle s \rangle\!\rangle \cup \langle\!\langle \mathrm{Pair}(s, t) \rangle\!\rangle \cup \{[\![\mathrm{OrderedPair(s, t)}]\!]\} \\ &= \langle\!\langle s \rangle\!\rangle \cup \langle\!\langle t \rangle\!\rangle \cup \{\{s, t\}\} \cup \{\{s, \{s, t\}\}\} \ ,\end{aligned}$$

so $|\langle\!\langle \mathrm{OrderedPair}(s, t) \rangle\!\rangle| = |\langle\!\langle s \rangle\!\rangle \cup \langle\!\langle t \rangle\!\rangle| + 2$. □

To handle ordered pairs correctly, we also need a way to project a set encoding an ordered pair to the first and second element of the pair:

Lemma 17. *There are* BGS *terms* proj_1 *and* proj_2 *such that, for any term* t *with* $[\![t]\!] = \langle a, b \rangle$ *for some* $a, b \in \mathrm{HF}(A)$ *(where* A *is the domain of the input structure),* $[\![\mathrm{proj}_1(t)]\!] = a$, $[\![\mathrm{proj}_2(t)]\!] = b$, $|\langle\!\langle \mathrm{proj}_1(t) \rangle\!\rangle| = |\langle\!\langle t \rangle\!\rangle| + |a| + 5$ *and* $|\langle\!\langle \mathrm{proj}_2(t) \rangle\!\rangle| = |\langle\!\langle t \rangle\!\rangle| + |a| + 8$.

Proof.

$$\begin{aligned}\mathrm{proj}_1(x) &= \mathrm{TheUnique}\left(\{x_1 : x_1 \in \mathrm{Union}(x) : \mathrm{In}(x_1, x)\}\right) \\ \mathrm{proj}_2(x) &= \mathrm{TheUnique}(\{x_2 : x_2 \in \mathrm{Union}(x) : \\ &\qquad x_2 \neq \mathrm{proj}_1(x) \wedge \neg\, \mathrm{In}(x_2, \mathrm{proj}_1(x))\}) \ .\end{aligned}$$

Let t be a term with $[\![t]\!] = \langle a, b\rangle$. Then

$$\begin{aligned}
|\langle\!\langle \mathrm{proj}_1(t)\rangle\!\rangle| = \Bigg| &\{[\![\{x_1 : x_1 \in \mathrm{Union}(t) : \mathrm{In}(x_1, \mathrm{proj}_1(t)\}]\!]\} \cup \{[\![\mathrm{proj}_1(t)]\!]\} \cup \\
&\{[\![\mathrm{Union}(t)]\!]\} \cup \langle\!\langle t\rangle\!\rangle \cup \bigcup_{a_1 \in [\![\mathrm{Union}(t)]\!]} \langle\!\langle \mathrm{In}(a_1, t)\rangle\!\rangle \cup \\
&\bigcup_{a_1 \in [\![\mathrm{Union}(t)]\!] : [\![\mathrm{In}(a_1,t)\}]\!] = \mathrm{True}} \langle\!\langle a_1\rangle\!\rangle \Bigg| \\
&\le 3 + \left| \langle\!\langle t\rangle\!\rangle \cup \bigcup_{a_1 \in [\![a]\!] \cup \{[\![a]\!], [\![b]\!]\}} (\langle\!\langle a_1\rangle\!\rangle \cup \langle\!\langle t\rangle\!\rangle) \right| \\
&= 3 + |\langle\!\langle t\rangle\!\rangle \cup [\![a]\!] \cup \{[\![a]\!], [\![b]\!]\}| \\
&\le |\langle\!\langle t\rangle\!\rangle| + |a| + 5 \ .
\end{aligned}$$

Given the number of active objects of the subterm proj_1, it is now possible to compute that number for proj_2. Let t be a term such that $[\![t]\!] = \langle a, b\rangle$.

$$\begin{aligned}
|\langle\!\langle \mathrm{proj}_2(t)\rangle\!\rangle| = \Bigg| &\{[\![\{x_2 : x_2 \in \mathrm{Union}(t) : x_2 \neq \mathrm{proj}_1(t) \wedge \neg\, \mathrm{In}(x_2, \mathrm{proj}_1(t))\}]\!]\} \cup \\
&\{[\![\mathrm{proj}_2(t)]\!]\} \cup \langle\!\langle \mathrm{Union}(t)\rangle\!\rangle \cup \\
&\bigcup_{a_2 \in [\![\mathrm{Union}(t)]\!]} \langle\!\langle a_2 \neq \mathrm{proj}_1(t) \wedge \neg\, \mathrm{In}(a_2, \mathrm{proj}_1(t))\rangle\!\rangle \cup \\
&\bigcup_{\substack{a_2 \in [\![\mathrm{Union}(t)]\!] : \\ [\![a_2 \neq \mathrm{proj}_1(t) \wedge \neg\, \mathrm{In}(a_2, \mathrm{proj}_1(t))]\!] = \mathrm{True}}} \langle\!\langle a_2\rangle\!\rangle \Bigg| \\
=& 3 + \left| \bigcup_{a_2 \in [\![\mathrm{Union}(t)]\!]} \langle\!\langle a_2\rangle\!\rangle \cup \langle\!\langle \mathrm{proj}_1(t)\rangle\!\rangle \right| \\
\le& 3 + |\langle\!\langle \mathrm{proj}_1(t)\rangle\!\rangle| = |\langle\!\langle t\rangle\!\rangle| + |a| + 8 \ ,
\end{aligned}$$

which shows the bound given in the lemma. □

The terms for ordered pairs can now be used to define tuples of arbitrary fixed length. Note that we do not compute the number of active objects of these terms here, since we only use them in $\mathrm{BGS}_{\mathrm{orig}}$ programs (recall that an object is active in an $\mathrm{BGS}_{\mathrm{orig}}$ program if it is involved in changing a dynamic function).

Lemma 18. *For each $k \in \mathbb{N}$, there is a* BGS *term* Tup^k *such that for any terms $t_1, \dots, t_k$, $[\![\mathrm{Tup}^k(t_1, \dots, t_k)]\!] = \langle [\![t_1]\!], \dots [\![t_k]\!] \rangle$.*

Proof. Induction on k.

Induction Base: $\mathrm{Tup}^1(x_1) = x_1$.

Induction Step:

$$\mathrm{Tup}^{k+1}(x_1, \dots, x_{k+1}) = \mathrm{OrderedPair}\left(\mathrm{Tup}^k(x_1, \dots, x_k), x_{k+1}\right) \ .$$

□

As for ordered pairs, we also define terms that project tuples to each of their components. One such term is defined for each combination of the length k of a tuple and the number i of the component.

Lemma 19. *For each $i, k \in \mathbb{N}$ with $i \leq k$, there is a* BGS *term* Tup^k_i, *such that for any term t with $[\![t]\!] = \langle a_1, \dots, a_k \rangle$, $[\![\mathrm{Tup}^k_i(t) = a_i]\!]$.*

Proof. Induction on k.

Induction Base: $\mathrm{Tup}^1_1(x) = x$.

Induction Step:

$$\mathrm{Tup}^k_k(x) = \mathrm{proj}_2(x), \ \mathrm{Tup}^k_i(x) = \mathrm{Tup}^{k-1}_i(\mathrm{proj}_1(x)) \text{ for } i < k \ .$$

□

Terms for ordered pairs also make it possible to define the Cartesian product of two sets as a set of ordered pairs.

Lemma 20. *There is a* BGS *term* $t_\times$ *such that* $[\![t_\times(t_1, t_2)]\!] = [\![t_1]\!] \times [\![t_2]\!]$ *and* $|\langle\!\langle t_\times(t_1, t_2)\rangle\!\rangle| \leq |\langle\!\langle t_1\rangle\!\rangle| + |\langle\!\langle t_2\rangle\!\rangle| + |[\![t_2]\!]| \cdot (3\,|[\![t_1]\!]| + 2) + 2$ *for any* BGS *terms* t_1, t_2.

Proof. Let $t_\times(x,y) = \text{Union}\left(\{\{\langle z_1, z_2\rangle : z_1 \in x\} : z_2 \in y\}\right)$. By definition, the semantics of the term is as required. Let t_1, t_2 be BGS terms. Then the bound for the number of active objects is obtained as follows:

$$\begin{aligned}
|\langle\!\langle t_\times(t_1,t_2)\rangle\!\rangle| &= |\langle\!\langle \text{Union}\left(\{\{\langle z_1, z_2\rangle : z_1 \in t_1 :\} : x_2 \in t_2\}\right)\rangle\!\rangle| \\
&= \Bigg| \{[\![t_\times(t_1,t_2)]\!]\} \cup \{[\![\{\{\langle z_1, z_2\rangle : z_1 \in t_1\} : z_2 \in t_2\}]\!]\} \cup \\
&\qquad \langle\!\langle t_2\rangle\!\rangle \cup \bigcup_{a_2 \in [\![t_2]\!]} \Bigg(\{[\![\{\langle z_1, t_2\rangle : z_1 \in t_1\}]\!]\} \cup \\
&\qquad \langle\!\langle t_1\rangle\!\rangle \cup \bigcup_{a_1 \in [\![t_1]\!]} \langle\!\langle \text{OrderedPair}(a_1, a_2)\rangle\!\rangle \Bigg) \Bigg| \\
&= \Bigg| \{[\![t_\times(t_1,t_2)]\!]\} \cup \{[\![\{\{\langle z_1, z_2\rangle : z_1 \in t_1\} : z_2 \in t_2\}]\!]\} \cup \\
&\qquad \langle\!\langle t_1\rangle\!\rangle \cup \langle\!\langle t_2\rangle\!\rangle \cup \bigcup_{a_2 \in [\![t_2]\!]} \Bigg(\{[\![\{\langle z_1, t_2\rangle : z_1 \in t_1\}]\!]\} \cup \\
&\qquad \bigcup_{a_1 \in [\![t_1]\!]} \left(\{\{[\![a_1]\!], \{[\![a_1]\!], [\![a_2]\!]\}\}\} \cup \langle\!\langle a_1\rangle\!\rangle \cup \{\{[\![x_1]\!], [\![x_2]\!]\}\}\right) \Bigg) \Bigg| \\
&\leq 2 + |\langle\!\langle t_1\rangle\!\rangle| + |\langle\!\langle t_2\rangle\!\rangle| + |[\![t_2]\!]| \cdot (3\,|[\![t_1]\!]| + 2)
\end{aligned}$$

□

We also write $t_1 \times t_2$ instead of $t_\times(t_1, t_2)$.

With these operations, we have defined the necessary tools for constructing the BGS and BGS_{orig} programs used in the proofs in the remainder of this thesis.

3.2 Counting and Equicardinality

In the following, we show that the cardinality operation in CPT can be replaced by an equicardinality operation without loss of expressive power.

Theorem 21. $\text{CPT} + \text{EqCard} \equiv \text{CPT} + \text{C}$

It is easy to see that there is an equivalent $\mathrm{CPT}+\mathrm{C}$ program for each $\mathrm{CPT}+\mathrm{EqCard}$ program (express EqCard using the function Card). For the other direction, we use the fact that CPT programs work on hereditarily finite sets, which makes it possible to define finite ordinals. So the value of $\mathrm{Card}(x)$ will be defined as the unique ordinal that has the cardinality $|x|$ for any set x.

Recall that a CPT program applies some BGS term Π_{step} to a set x_i in each step i. The $\mathrm{CPT}+\mathrm{EqCard}$ program simulating a CPT program will use a term that works on the ordered pair $\langle [n], x_i\rangle$ instead of the set x_i, where $[n]$ contains all ordinals necessary during the computation, and simulates the application of Π_{step} to x_i. Since the number of active objects of each run, and thus the cardinality of the occuring sets, is bounded by a given polynomial, it is possible to determine $[n]$ in an initialisation step before the computation of the original program is simulated.

To initialise such an ordinal $[n]$, the program starts with the empty set and constructs the successor ordinal until there is an ordinal with cardinality $q(|A|)$ (where q is the bound on the number of active objects and A the domain of the input structure). So we first show how to construct successor ordinals in BGS logic.

Lemma 22. *There is a* BGS *term* Suc *such that for each term* t_n *with* $[\![t_n]\!] = [n]$ *it holds that* $[\![\mathrm{Suc}(t_n)]\!] = [n+1]$ *and* $|\langle\!\langle \mathrm{Suc}(t_n)\rangle\!\rangle| \leq |\langle\!\langle t_n\rangle\!\rangle| + 3$.

Proof. Let $\mathrm{Suc}(x) = x \cup \{x\}$. By definition, Suc computes the successor of a finite ordinal, and

$$\begin{aligned} |\langle\!\langle \mathrm{Suc}(t)\rangle\!\rangle| &= |\langle\!\langle \mathrm{Union}\left(\mathrm{Pair}\left(t, \mathrm{Pair}(t,t)\right)\right)\rangle\!\rangle| \\ &= |\{t \cup \{t\}\} \cup \{\{t,\{t\}\}\} \cup \{t\} \cup \langle\!\langle t\rangle\!\rangle| \\ &\leq |\langle\!\langle t\rangle\!\rangle| + 3 \ . \end{aligned}$$

□

In order to represent the cardinality of any set occuring in the computation, the ordinal in the first component of the pair should be $[q(|A|)+1]$, with q and A defined as above. Therefore the program initialises the ordinals until one with cardinality $q(|A|)$ has been constructed. This is determined using a term that outputs a set with cardinality $q(|A|)$.

Lemma 23. *For every polynomial q over the natural numbers, there is a* BGS *term $t_q(x)$ and a polynomial* active_q *such that for any term t with no free variables where $[\![t]\!]$ is a set, $|[\![t_q(t)]\!]| = q(|[\![t]\!]|)$ and $|\langle\!\langle t_q(t)\rangle\!\rangle| \leq \text{active}_q(\langle\!\langle t\rangle\!\rangle)$.*

Proof by induction on polynomials.

Induction Base:

- $q = x$, where x is a variable: $t_q(x) = x$. Clearly, $|[\![t_q(t)]\!]| = |[\![t]\!]| = q(|[\![t]\!]|)$ and $|\langle\!\langle t_q(t)\rangle\!\rangle| = |\langle\!\langle t\rangle\!\rangle|$, which is a polynomial in $|\langle\!\langle t\rangle\!\rangle|$.
- $q = c$, where c is a constant: $t_q = \text{Suc}^c(\emptyset)$. By Lemma 22, which specifies the term Suc, the number of active objects is bounded by a polynomial.

Induction Step:

- $q = q_1 + q_2$: By induction hypothesis, there are terms t_{q_1} and t_{q_2} for q_1 and q_2 and polynomials active_{q_1} and active_{q_2} bounding $|\langle\!\langle t_{q_1}\rangle\!\rangle|$ and $|\langle\!\langle t_{q_2}\rangle\!\rangle|$. Let
$$t_q(x) = (\{\emptyset\} \times t_{q_1}(x)) \cup (\{\{\emptyset\}\} \times t_{q_2}(x)\ )\ .$$
Let t be a term without free variables that evaluates to a set. By definition, $|[\![t_q(t)]\!]| = |[\![t_{q_1}(t)]\!]| + |[\![t_{q_2}(t)]\!]| = q(|[\![t]\!]|)$ (using the induction hypothesis). Furthermore,
$$\begin{aligned}
|\langle\!\langle t_q(t)\rangle\!\rangle| &= |\langle\!\langle \{\emptyset\} \times t_{q_1}(t) \cup \{\{\emptyset\}\} \times t_{q_2}(t)\rangle\!\rangle| \\
&\leq |\langle\!\langle \{\emptyset\} \times t_{q_1}(t)\rangle\!\rangle| + |\langle\!\langle \{\{\emptyset\}\} \times t_{q_2}(t)\rangle\!\rangle| + 2 && (3.1)\\
&\leq (|\langle\!\langle \{\emptyset\}\rangle\!\rangle| + |\langle\!\langle t_{q_1}(t)\rangle\!\rangle| + |[\![t_{q_1}(t)]\!]| \cdot (3\,|[\![\{\emptyset\}]\!]| + 2) + 2) + \\
&\quad (|\langle\!\langle \{\{\emptyset\}\}\rangle\!\rangle| + |\langle\!\langle t_{q_2}(t)\rangle\!\rangle| + |[\![t_{q_2}(t)]\!]| \cdot (3\,|[\![\{\{\emptyset\}\}]\!]| + 2) + 2) + 2 && (3.2)\\
&= (|\langle\!\langle t_{q_1}(t)\rangle\!\rangle| + 5\,|[\![t_{q_1}(t)]\!]| + 4) + (|\langle\!\langle t_{q_2}(t)\rangle\!\rangle| + 5\,|[\![t_{q_2}(t)]\!]| + 5) + 2 && (3.3)\\
&\leq \text{active}_{q_1}(|[\![t]\!]|) + \text{active}_{q_2}(|[\![t]\!]|) + 5q_1(|[\![t]\!]|) + 5q_2(|[\![t]\!]|) + 11\ . && (3.4)
\end{aligned}$$

Inequality (3.1) holds because of the bound for the number of active objects of $t_\cup$ obtained from Lemma 15. Similarly, Step (3.2) follows from the bound for $t_\times$ in Lemma 20. In Step (3.3), we use that, by

Lemma 14, the number of active objects of the terms for $\{\emptyset\}$ and $\{\{\emptyset\}\}$ is constant.

Finally, Inequality (3.4) follows from the fact that the polynomials active_{q_1} and active_{q_2} bound the number of active objects of t_{q_1} and t_{q_2}, respectively. Furthermore, as observed above, the cardinality of $[\![t_{q_1}(t)]\!]$ (resp. $[\![t_{q_2}(t)]\!]$) is exactly $q_1(|[\![t]\!]|)$ (resp. $q_2(|[\![t]\!]|)$).

So there is also a polynomial active_q bounding the number of active objects of t_q.

- $q = q_1 \cdot q_2$: By induction hypothesis, there are terms t_{q_1} and t_{q_2} for q_1 and q_2 and polynomials active_{q_1} and active_{q_2} that bound $|\langle\!\langle q_1\rangle\!\rangle|$ and $|\langle\!\langle q_2\rangle\!\rangle|$. Let
$$t_q = t_{q_1} \times t_{q_2} \ .$$
Let t be a term without free variables such that $[\![t]\!]$ is a set. Clearly, $|[\![t_q(t)]\!]| = |[\![t_{q_1}(t)]\!]| \cdot |[\![t_{q_2}(t)]\!]| = q_1(|[\![t]\!]|) \cdot q_2(|[\![t]\!]|) = q(|[\![t]\!]|)$. The number of active objects is bounded by the following polynomial:
$$\begin{aligned} |\langle\!\langle t_q(c)\rangle\!\rangle| &= |\langle\!\langle t_{q_1} \times t_{q_2}\rangle\!\rangle| \\ &\le |\langle\!\langle t_{q_1}\rangle\!\rangle| + |\langle\!\langle t_{q_2}\rangle\!\rangle| + |[\![t_{q_2}]\!]| \cdot (3\,|[\![t_{q_1}]\!]| + 2) + 2 \qquad (3.5)\\ &\le \mathrm{active}_{q_1}(|[\![t]\!]|) + \mathrm{active}_{q_2}(|[\![t]\!]|) + 3q_1(|[\![t]\!]|) + 2q_2(|[\![t]\!]|) + 2 \ . \qquad (3.6)\end{aligned}$$
Note that Inequality (3.5) again follows from the bound for the number of active objects of $t_\times$ (Lemma 20), and Inequality (3.6) follows from the induction hypothesis.

By induction, there is a term t_q for every polynomial q that computes a set with cardinality $q(|[\![t]\!]|)$ for any input set defined by a term t. □

With these terms, it is possible to initialise the necessary ordinals. For the actual simulation of the given CPT program, we modify Π_{step} to obtain a term that assumes as input both an ordinal $[n]$ and a set x and where each occurence of the function symbol Card is replaced by a term that defines the cardinality using EqCard and the input ordinal.

Lemma 24. *Let $t(\overline{x})$ be a* BGS + C *term with free variables $\overline{x} = x_1, \ldots, x_k$. Then there is a* BGS + EqCard *term $t^{\mathrm{EqCard}}(x_0, \overline{x})$ with $k + 1$ free variables* *such that, if $\left|\left\langle\!\left\langle t(\bar{t})\right\rangle\!\right\rangle\right|$ is bounded by $q(|A|)$ and $[\![t_0]\!] = [q(|A|+1]$ for some input structure $\mathfrak{A} = (A, \tau)$ and terms t_0 and $\bar{t} = t_1, \ldots, t_k$ that substitute the free variables, $\left[\!\left[t^{\mathrm{EqCard}}(t_0, \bar{t})\right]\!\right] = \left[\!\left[t(\bar{t})\right]\!\right]$. Moreover, there is a constant c such that for all such terms $t_0, \bar{t}$, $\left|\left\langle\!\left\langle t^{\mathrm{EqCard}}(t_0, \bar{t})\right\rangle\!\right\rangle\right| \leq |\langle\!\langle t_0\rangle\!\rangle| + 2q(|A|) + c$.*

Proof. We show by induction on t that there is a term t^{EqCard} where the semantics is as required by the lemma and

$$\left\langle\!\left\langle t^{\mathrm{EqCard}}(t_0, \bar{t})\right\rangle\!\right\rangle \subseteq \left\langle\!\left\langle t(\bar{t})\right\rangle\!\right\rangle \cup \langle\!\langle t_0\rangle\!\rangle \cup [q(|A|)+1] \cup \left\{\left\{\left|t'(\bar{t})\right|\right\} \mid t' \text{ is a subterm of } t\right\}$$

if $\left|\left[\!\left[t(\bar{t})\right]\!\right]\right|$ is bounded by $q(|A|)$. Since $\left\langle\!\left\langle t(\bar{t})\right\rangle\!\right\rangle$ is bounded by $q(|A|)$ and the number of subterms is constant, this implies the required bound of $|\langle\!\langle t_0\rangle\!\rangle| + 2q(|A|) + c$, where c depends only on t.

For atomic terms, it is clear that $t^{\mathrm{EqCard}}(x_0, \overline{x}) = t(\overline{x})$ fulfills the requirements.

For $t(\overline{x}) = f(s_1(\overline{x}), \ldots, s_r(\overline{x}))$, where f is a function symbol, define $t^{\mathrm{EqCard}}(\overline{x}) = f(s_1^{\mathrm{EqCard}}(\overline{x}), \ldots, s_r^{\mathrm{EqCard}}(\overline{x}))$, where s_i^{EqCard} is the BGS + EqCard term for s_i. By induction hypothesis, t^{EqCard} satisfies the requirements of the lemma. The cases of relation symbols, Boolean connectives and comprehension terms are analogous.

For $t(\overline{x}) = \mathrm{Card}(s(\overline{x}))$, let

$$t^{\mathrm{EqCard}}(x_0, \overline{x}) = \mathrm{TheUnique}\left(\left\{z : z \in x_0 : \mathrm{EqCard}\left(z, s^{\mathrm{EqCard}}\left(x_0, \overline{x}\right)\right)\right\}\right) \quad ,$$

where s^{EqCard} is the BGS + EqCard term for s that exists by induction hypothesis.

Let $\mathfrak{A} = (A, \tau)$ and $\bar{t} = t_1, \ldots, t_k$ be such that $\left|\left\langle\!\left\langle t(\bar{t})\right\rangle\!\right\rangle\right|$ is bounded by the polynomial $q(|A|)$. Then $\left|\left[\!\left[t(\bar{t})\right]\!\right]\right| \in [q(|A|) + 1]$, and thus $t^{\mathrm{EqCard}}(t_0, \bar{t})$ for $[\![t_0 = [q(|A|) + 1]]\!]$ defines the correct ordinal.

Now consider the number of active objects of $t^{\mathrm{EqCard}}\left(t_0, \bar{t}\right)$.

$$\begin{aligned}
&\left\langle\!\left\langle \mathrm{TheUnique}\left(\left\{z : z \in t_0 : \mathrm{EqCard}\left(z, s^{\mathrm{EqCard}}\left(t_0, \bar{t}\right)\right)\right\}\right)\right\rangle\!\right\rangle \\
={}& \left\{\left|\left[\!\left[s\left(\bar{t}\right)\right]\!\right]\right|\right\} \cup \left\langle\!\left\langle \left\{z : z \in t_0 : \mathrm{EqCard}\left(z, s^{\mathrm{EqCard}}\left(t_0, \bar{t}\right)\right)\right\}\right\rangle\!\right\rangle \\
={}& \left\{\left|\left[\!\left[s\left(\bar{t}\right)\right]\!\right]\right|\right\} \cup \left\{\left\{\left|\left[\!\left[s\left(\bar{t}\right)\right]\!\right]\right|\right\}\right\} \cup \langle\!\langle t_0\rangle\!\rangle \cup \bigcup_{a \in [\![t_0]\!]} \left\langle\!\left\langle \mathrm{EqCard}\left(a, s^{\mathrm{EqCard}}\left(t_0, \bar{t}\right)\right)\right\rangle\!\right\rangle \cup \\
& \qquad \bigcup_{a \in [\![t_0]\!] : \left[\!\left[\mathrm{EqCard}\left(a, s^{\mathrm{EqCard}}\left(t_0, \bar{t}\right)\right)\right]\!\right] = \mathrm{True}} \langle\!\langle a \rangle\!\rangle \\
={}& \left\{\left|\left[\!\left[s\left(\bar{t}\right)\right]\!\right]\right|\right\} \cup \left\{\left\{\left|\left[\!\left[s\left(\bar{t}\right)\right]\!\right]\right|\right\}\right\} \cup \langle\!\langle t_0\rangle\!\rangle \cup \bigcup_{a \in [\![t_0]\!]} \langle\!\langle a \rangle\!\rangle \cup \left\langle\!\left\langle s^{\mathrm{EqCard}}\left(t_0, \bar{t}\right)\right\rangle\!\right\rangle \\
\subseteq{}& \left\langle\!\left\langle t\left(\bar{t}\right)\right\rangle\!\right\rangle \cup \langle\!\langle t_0\rangle\!\rangle \cup [q(|A|)+1] \cup \left\{\left\{\left|\left[\!\left[t'\left(\bar{t}\right)\right]\!\right]\right|\right\} \mid t' \text{ is a subterm of } t\right\} \qquad (3.7)
\end{aligned}$$

Note that, by induction hypothesis,

$$\begin{aligned}
\left\langle\!\left\langle s^{\mathrm{EqCard}}\left(t_0, \bar{t}\right)\right\rangle\!\right\rangle \subseteq \left\langle\!\left\langle s\left(\bar{t}\right)\right\rangle\!\right\rangle \cup \langle\!\langle t_0\rangle\!\rangle \cup [q(|A|)+1] \\
\cup \left\{\left\{\left|s'\left(\bar{t}\right)\right|\right\} \mid s' \text{ is a subterm of } s\right\}.
\end{aligned}$$

So, since $\langle\!\langle s(\bar{t})\rangle\!\rangle \subseteq \langle\!\langle t(\bar{t})\rangle\!\rangle$ and every subterm of s is also a subterm of t, the subset relation in Equation (3.7) holds, which completes the induction. □

Now it remains to specify, for a given CPT program Π, a CPT + EqCard program that uses these terms to simulate Π in the way explained above.

Lemma 25. *Let $\overline{\Pi} = (\Pi, q)$ be a* CPT + C *program for a* BGS + C *program $\Pi = (\Pi_{step}, \Pi_{halt}, \Pi_{out})$ and a polynomial q. There is* CPT + EqCard *program $\overline{\Pi}' = (\Pi', q')$ such that, for each finite structure $\mathfrak{A}$, the output of the runs of Π and Π' on $\mathfrak{A}$ is the same.*

Proof. We denote by $\Pi' = (\Pi'_{\mathrm{step}}, \Pi'_{\mathrm{halt}}, \Pi'_{\mathrm{out}})$ the BGS + EqCard program constructed in this proof. Π' will work as follows:

1. On $\emptyset$, construct the ordered pair $\langle \emptyset, \emptyset \rangle$.

2. Construct successor ordinals in the first component of the pair until $[q(|A|)]$ is in that set (this is determined using t_q).

3. Simulate the computation of Π_{step} in the second component using $\Pi_{\mathrm{step}}^{\mathrm{EqCard}}$.

To indicate whether the program is currently in Step 2, we use the following formula φ_{init} which is true if and only if $[q(|A|)]$ has not been constructed yet:

$$\varphi_{\text{init}}(y) = \{z : z \in \text{proj}_1(y) : \text{EqCard}(z, t_q(\text{Atoms}))\} = \emptyset \ ,$$

where t_q is the term obtained from Lemma 23 and proj_1 returns the first component of an ordered pair as shown in Lemma 17. The formula ensures that the first component of a given ordered pair contains an element of cardinality $q(|\text{Atoms}|)$, which suffices because the program will ensure that the input is always an ordered pair whose first component is an ordinal.

Π'_{step} uses that formula to perform the three steps defined above sequentially:

$$\begin{aligned}\Pi'_{\text{step}}(x) =& \{\text{OrderedPair}(y, y) : y \in \{\emptyset\} : x = \emptyset\} && (3.8)\\ & \cup\{\text{OrderedPair}(\text{Suc}(\text{proj}_1(y)), \text{proj}_2(y)) : \\ & \qquad\qquad y \in \{x\} : x \neq \emptyset \wedge \varphi_{\text{init}}(y)\} && (3.9)\\ & \cup\{\text{OrderedPair}(\text{proj}_1(y), \Pi^{\text{EqCard}}_{\text{step}}(\text{proj}_1(y), \text{proj}_2(y))) : \\ & \qquad\qquad y \in \{x\} : x \neq \emptyset \wedge \neg\varphi_{\text{init}}(y)\} \ , && (3.10)\end{aligned}$$

where $\Pi^{\text{EqCard}}_{\text{step}}$ is the term simulating Π_{step} which is obtained from Lemma 24.

Each of the three subterms returns a singleton containing exactly one ordered pair if the condition is fulfilled, and the empty set otherwise. So, since the conditions are mutually exclusive, Π'_{step} returns an ordered pair updated as required. Subterm (3.8) initialises the pair $\langle\emptyset, \emptyset\rangle$ at the beginning of the computation. Subterm (3.9) constructs the next ordinal in the first component if the ordinal $[q(|A|)]$ does not exist yet, and subterm (3.10) computes the next state of the original program in the second component using $\Pi^{\text{EqCard}}_{\text{step}}$. So Π'_{step} indeed simulates Π_{step} in the desired way.

Π'_{halt} and Π'_{out} are also modified versions of Π_{halt} and Π_{out} that work on ordered pairs:

$$\begin{aligned}\Pi'_{\text{halt}}(x) &= \neg\varphi_{\text{init}} \wedge \Pi^{\text{EqCard}}_{\text{halt}}(\text{proj}_1(x), \text{proj}_2(x)) \ ,\\ \Pi'_{\text{out}}(x) &= \Pi^{\text{EqCard}}_{\text{out}}(\text{proj}_1(x), \text{proj}_2(x)) \ ,\end{aligned}$$

where $\Pi^{\text{EqCard}}_{\text{halt}}$ and $\Pi^{\text{EqCard}}_{\text{out}}$ are the BGS + EqCard terms equivalent to Π_{halt} and Π_{out} obtained from Lemma 24.

By definition of Π'_{step}, $[\![\Pi'_{\text{step}}(\langle[q(|A|)], x\rangle)]\!] = \langle[q(|A|)], [\![\Pi_{\text{step}}(x)]\!]\rangle$ for all x, so, by Lemma 24, $\Pi'_{\text{halt}}(x)$ is true if and only if the initialisation of the first component of x is completed and $\Pi_{\text{halt}}(\text{proj}_2(x))$ is true. Applying the same argument for Π'_{out} yields that $(\Pi'_{\text{step}}, \Pi'_{\text{halt}}, \Pi'_{\text{out}})$ is equivalent to $(\Pi_{\text{step}}, \Pi_{\text{halt}}, \Pi_{\text{out}})$.

It remains to show that there is a CPT + EqCard program $\overline{\Pi}' = (\Pi', q')$ equivalent to Π', i.e. that $\text{Active}(\overline{\Pi}', \mathfrak{A}) \leq q'(|A|)$ for every finite structure $\mathfrak{A} = (A, \tau)$.

First we compute the number of active objects of the formula φ_{init}, which is a subformula of several of the previously defined terms. Note that this formula only occurs in cases where the free variable is replaced by a pair $\langle[q(|A|) + 1], a'\rangle$, so let a be such a pair.

$$\begin{aligned}
|\langle\!\langle\varphi_{\text{init}}(a)\rangle\!\rangle| &= |\langle\!\langle\{z : z \in \text{proj}_1(a) : \text{EqCard}(z, t_q(\text{Atoms}))\} = \emptyset\rangle\!\rangle| \\
&\leq \Bigg|\{[\![\{z : z \in \text{proj}_1(a) : \text{EqCard}(z, t_q(\text{Atoms}))\}]\!]\} \cup \langle\!\langle\text{proj}_1(a)\rangle\!\rangle \cup \\
&\qquad \bigcup_{b \in [\![\text{proj}_1(a)]\!]} (\langle\!\langle b\rangle\!\rangle \cup \langle\!\langle\text{EqCard}(b, t_q(\text{Atoms}))\rangle\!\rangle) \cup \langle\!\langle\emptyset\rangle\!\rangle\Bigg| \\
&= \Bigg|\{[\![\{z : z \in \text{proj}_1(a) : \text{EqCard}(z, t_q(\text{Atoms}))\}]\!]\} \cup \langle\!\langle\text{proj}_1(a)\rangle\!\rangle \cup \\
&\qquad \langle\!\langle t_q(\text{Atoms})\rangle\!\rangle \cup \bigcup_{b \in [\![\text{proj}_1(a)]\!]} \langle\!\langle b\rangle\!\rangle \cup \langle\!\langle\emptyset\rangle\!\rangle\Bigg| \\
&\leq 1 + |\langle\!\langle a\rangle\!\rangle| + (q(|A|) + 1) + 5 + \text{active}_q(|\langle\!\langle\text{Atoms}\rangle\!\rangle|) + \\
&\qquad (q(|A| + 1) + 1 \qquad (3.11) \\
&= 2q(|A|) + \text{active}_q(1) + 10 \ ,
\end{aligned}$$

where active_q is the polynomial bounding the number of active objects of the term t_q according to Lemma 23. In addition to this bound, the inequality in Equation (3.11) follows from the bound for proj_1 (Lemma 17) and the fact that $|[\![\text{proj}_1(a)]\!]| = q(|A|) + 1$.

The term Π'_{step} consists of several applications of the term $t_\cup$. Since $t_\cup$ only changes the number of active objects by a constant, we analyse the three main subterms separately.

$$\begin{aligned}
&|\langle\!\langle\{\text{OrderedPair}(y,y) : y \in \{\emptyset\} : x = \emptyset\}\rangle\!\rangle| \\
=&\big|\{[\![\{\text{OrderedPair}(y,y) : y \in \{\emptyset\} : x = \emptyset\}]\!]\}\cup \\
&\langle\!\langle\{\emptyset\}\rangle\!\rangle \cup \langle\!\langle x = \emptyset\rangle\!\rangle \cup \langle\!\langle \text{OrderedPair}(\emptyset,\emptyset)\rangle\!\rangle\big| ,
\end{aligned}$$

so the number of active objects of the first subterm is constant. For the second subterm, we again assume that the free variable is replaced by a pair $a = \langle [n], a'\rangle$ for a natural number n (the only other case is that the free variable is replaced by $\emptyset$, which activates less objects).

$$\begin{aligned}
&|\langle\!\langle\{\text{OrderedPair}(\text{Suc}(\text{proj}_1(y)), \text{proj}_2(y)) : y \in \{a\} : a \neq \emptyset \wedge \varphi_{\text{init}}(y)\}\rangle\!\rangle| \\
\leq&1 + |\langle\!\langle\{a\}\rangle\!\rangle \cup \langle\!\langle a \neq \emptyset\rangle\!\rangle \cup \langle\!\langle \varphi_{\text{init}}(a)\rangle\!\rangle \cup \langle\!\langle \text{OrderedPair}(\text{Suc}(\text{proj}_1(a)), \text{proj}_2(a))\rangle\!\rangle| \\
\leq&5 + (2q(|A|) + \text{active}_q(1) + 10) + |\langle\!\langle \text{Suc}(\text{proj}_1(a))\rangle\!\rangle| + |\langle\!\langle \text{proj}_2(a)\rangle\!\rangle| + 2 \qquad (3.12)\\
\leq&2q(|A|) + \text{active}_q(1) + (|\langle\!\langle a\rangle\!\rangle| + (q(|A|)+1) + 5) + 3 + \\
&\qquad (|\langle\!\langle a\rangle\!\rangle| + (q(|A|)+1) + 8) + 17 \qquad (3.13)\\
=&4q(|A|) + \text{active}_q(1) + 35
\end{aligned}$$

The inequality in Equation (3.12) is implied by the bound for φ_{init} computed above, and Equation (3.13) uses the bounds for proj_1 and proj_2 from Lemma 17.

For the last subterm of Π'_{step}, we again assume that the value a assigned to the free variable is of the form $\langle [n], a'\rangle$ for $n \leq q(|A|) + 1$. Additionally, we assume that a' is a state of the run of Π on the given structure $\mathfrak{A}$. By construction, this assumption is true in any run of Π'.

$$\left|\left\langle\!\!\left\langle\left\{\mathrm{OrderedPair}\left(\mathrm{proj}_1(y),\Pi^{\mathrm{EqCard}}_{\mathrm{step}}\left(\mathrm{proj}_1(y),\mathrm{proj}_2(y)\right)\right) : y\in\{a\} : a\neq\emptyset\wedge\neg\varphi_{\mathrm{init}}(y)\right\}\right\rangle\!\!\right\rangle\right| \tag{3.14}$$

$$\begin{aligned}
=&\Bigg|\left\{\left[\!\!\left[\left\{\mathrm{OrderedPair}\left(\mathrm{proj}_1(y),\Pi^{\mathrm{EqCard}}_{\mathrm{step}}\left(\mathrm{proj}_1(y),\mathrm{proj}_2(y)\right)\right) : y\in\{a\} : a\neq\emptyset\wedge\neg\varphi_{\mathrm{init}}(y)\right\}\right]\!\!\right]\right\}\cup \\
&\langle\!\langle\{a\}\rangle\!\rangle\cup\langle\!\langle a\neq\emptyset\rangle\!\rangle\cup\langle\!\langle\neg\varphi_{\mathrm{init}}(a)\rangle\!\rangle\cup \\
&\bigcup_{\substack{b\in\{a\}:\\ [\![a\neq\emptyset\wedge\neg\varphi_{\mathrm{init}}(b)]\!]=\mathrm{True}}}\left\langle\!\!\left\langle\mathrm{OrderedPair}\left(\mathrm{proj}_1(b),\Pi^{\mathrm{EqCard}}_{\mathrm{step}}\left(\mathrm{proj}_1(b),\mathrm{proj}_2(b)\right)\right)\right\rangle\!\!\right\rangle\Bigg| \\
\leq&1+2+2+(2q(|A|)+\mathrm{active}_q(1)+10)+ \qquad (3.15)\\
&\left|\left\langle\!\!\left\langle\mathrm{OrderedPair}\left(\mathrm{proj}_1(a),\Pi^{\mathrm{EqCard}}_{\mathrm{step}}\left(\mathrm{proj}_1(a),\mathrm{proj}_2(a)\right)\right)\right\rangle\!\!\right\rangle\right| \\
\leq&2q(|A|)+\mathrm{active}_q(1)+|\langle\!\langle a\rangle\!\rangle|+(q(|A|)+1)+(2q(|A|)+c)+20 \\
&\text{for some constant } c \qquad (3.16)\\
=&5q(|A|)+\mathrm{active}_q(1)+c' \text{ for some constant } c'\ .
\end{aligned}$$

The inequality in Equation (3.15) again uses the bound for φ_{init}. Furthermore, the range of the union contains at most one element. If it is nonempty, $[\![\mathrm{proj}_1(a)]\!]$ and $[\![\mathrm{proj}_2(a)]\!]$ satisfy the conditions for the bound on $\left|\left\langle\!\!\left\langle\Pi^{\mathrm{EqCard}}_{\mathrm{step}}\right\rangle\!\!\right\rangle\right|$ in Lemma 24, which justifies Equation (3.16).

Since the number of active objects of these main subterms of Π'_{step} is polynomially bounded, the number of active objects of their union Π'_{step} is polynomially bounded (by Lemma 15). It remains to show that the union over all states of the sets of active objects of Π'_{step} is still bounded by a polynomial. Therefore consider the length of a run of Π' on a finite structure $\mathfrak{A}$ with domain A. During the initialisation, each state is a pair $\langle[n],\emptyset\rangle$ for $n\leq q(|A|)+1$, so there are $q(|A|)+1$ such states. In the simulation step, each state is a pair $\langle[q(|A|)+1],x_i\rangle$, where x_i is a state of the run of Π on $\mathfrak{A}$.

So, since by Lemma 13 the number of states of the corresponding run of

Π is bounded by q, the number of states of the run of Π' is also bounded by a polynomial.

Similar arguments show that $|\langle\!\langle \Pi'_{\text{halt}}(a) \rangle\!\rangle|$ for such pairs a is bounded by a polynomial. The same holds for $|\langle\!\langle \Pi'_{\text{out}}(a) \rangle\!\rangle|$ in case $[\![\Pi'_{\text{halt}}(a)]\!] = \text{True}$.

It follows that there is a polynomial q' such that the number of active objects of each run of Π' is bounded by $q'(|A|)$, since there are polynomially many states in which Π'_{step} activates polynomially many objects each.

$\square$

Together with the observation that $\text{CPT} + \text{EqCard} \leq \text{CPT} + \text{C}$, Lemma 25 implies Theorem 21. In the following chapter, we will use that result to define an alternative characterisation of $\text{CPT} + \text{C}$ that uses only an equicardinality quantifier instead of a counting operation.

Chapter 4

Interpretation Logic

In the following, we introduce intepretation logic, give some examples of how interpretations are used in computations, and classify polynomial-time interpretation logic with respect to CPT. The main result of this chapter is that the polynomial-time restriction of interpretation logic is equivalent to CPT, and the extension by the Härtig quantifier (again restricted to polynomial time) is equivalent to $\mathrm{CPT} + \mathrm{EqCard}$ and thus $\mathrm{CPT} + \mathrm{C}$.

As mentioned in the introduction, interpretation logic is based on the idea of iterating logical interpretations. Like BGS, interpretation logic can be viewed as a machine model, therefore the "formulae" of interpretation logic are called programs and we refer to the evaluation as the computation of a program.

A program in interpretation logic consists of an initial interpretation called I_{init}, a main interpretation called I_{step}, and formulae φ_{halt} and φ_{out}. On a given input structure, a program first initialises the necessary relations using I_{init}. This usually means that the signature is extended by new relation symbols that store additional data throughout the computation.

Then the program modifies the domain and relations of the current structure according to I_{step} until the formula φ_{halt} is true. At the end, the program accepts the input structure if the structure obtained by the computation satisfies φ_{out}.

In analogy to the bounds on $\mathrm{CPT}_{\mathrm{orig}}$ programs, we restrict both the length of a run and the number of objects used in the computation. More precisely, a program in polynomial-time interpretation logic is bounded by polynomials p and q, where p restricts the length of each run of the program, and q bounds the size of the structures occuring in the computation.

The formal definition of this model is given in the following.

Let τ, σ be relational signatures and let $\mathcal{L} \in \{\mathrm{FO}, \mathrm{FO}+\mathrm{H}\}$. An $\mathrm{IL}[\tau, \sigma]$ *program* over $\mathcal{L}$ is a tuple $\Pi = \big(I_{\text{init}}, I_{\text{step}}, \varphi_{\text{halt}}, \varphi_{\text{out}}\big)$ where I_{init} is an $\mathcal{L}[\tau, \sigma]$ interpretation, I_{step} is an $\mathcal{L}[\sigma, \sigma]$ interpretation, and φ_{halt} and φ_{out} are $\mathcal{L}[\sigma]$ sentences. We also say that Π is an $\mathrm{IL}[\tau]$ program.

Let $\mathfrak{A}$ be a τ-structure, and let $\mathfrak{A}_0 = I_{\text{init}}(\mathfrak{A})$ and $\mathfrak{A}_{i+1} = I_{\text{step}}(\mathfrak{A}_i)$ for $0 \leq i < \omega$. If there is a structure $\mathfrak{A}_n$ such that $\mathfrak{A}_n \models \varphi_{\text{halt}}$ and $\mathfrak{A}_i \not\models \varphi_{\text{halt}}$ for all $i < n$, then the *run* of Π on $\mathfrak{A}$ (of length $n-1$) is the sequence $(\mathfrak{A}_i)_{i \leq n}$, and the *output* $\Pi(\mathfrak{A})$ is True if $\mathfrak{A}_n \models \varphi_{\text{out}}$ and False otherwise.

If there is no $\mathfrak{A}_n$ with $\mathfrak{A}_n \models \varphi_{\text{halt}}$, then the run of Π on $\mathfrak{A}$ is $(\mathfrak{A}_i)_{i<\omega}$ and $\Pi(\mathfrak{A}) = \bot$.

The structures $\mathfrak{A}_i$ (for $i \leq n$ if n is minimal such that $\mathfrak{A}_n \models \varphi_{\text{halt}}$ and $i < \omega$ if no such n exists), are the *states* of the run, $\mathfrak{A}_0$ is the *initial state*, and $\mathfrak{A}_n$, if it exists, is the *final state*.

If $\Pi(\mathfrak{A}) = \text{True}$, then Π *accepts* $\mathfrak{A}$, if $\Pi(\mathfrak{A}) = \text{False}$, then Π *rejects* $\mathfrak{A}$.

Clearly, IL over a logic $\mathcal{L} \in \{\mathrm{FO}, \mathrm{FO}+\mathrm{H}\}$ is a three-valued logic in the sense defined in Section 2.1: The set of sentences over a signature τ is the set of all $\mathrm{IL}[\tau]$ programs, which is decidable because $\mathcal{L}$ is a logic, and for a τ-structure $\mathfrak{A}$ and an $\mathrm{IL}[\tau]$ program Π, we say that $\mathfrak{A} \models \Pi$ if $\Pi(\mathfrak{A}) = \text{True}$, and $\mathfrak{A} \not\models \Pi$ if $\Pi(\mathfrak{A}) = \text{False}$.

To obtain a logic that only defines queries in PTIME, we add the following polynomial bounds to IL programs.

Let Π be an $\mathrm{IL}[\tau, \sigma]$ program over $\mathcal{L}$, and let p and q be polynomials. Then $\overline{\Pi} = (\Pi, p, q)$ is a $\mathrm{PIL}[\tau, \sigma]$ program over $\mathcal{L}$.

Let $\mathfrak{A} = (A, \tau)$ be a τ-structure and let $\mathfrak{A}_i$ for $i < \omega$ be defined as above. The *run* of $\overline{\Pi}$ on $\mathfrak{A}$ is the sequence $(\mathfrak{A}_n)_{i \leq n}$, where n is minimal such that

- $\mathfrak{A}_i \not\models \varphi_{\text{halt}}$ for all $i < n$,
- $n < p(|A|)$, and
- the size of all $\mathfrak{A}_i$ for $i \leq n$ is at least $q(|A|)$.

If the run of $\overline{\Pi}$ on $\mathfrak{A}$ coincides with the run of Π on $\mathfrak{A}$, i.e. if the run of Π on $\mathfrak{A}$ already satisfies the polynomial bounds, then $\overline{\Pi}(\mathfrak{A}) = \Pi(\mathfrak{A})$. Otherwise, $\overline{\Pi}(\mathfrak{A}) = \bot$.

$\overline{\Pi} = (\Pi, p, q)$ is a *polynomial-time version* of Π if $\overline{\Pi}(\mathfrak{A}) = \Pi(\mathfrak{A})$ for all τ-structures $\mathfrak{A}$.

Like IL, PIL is a three-valued logic. Note that, as we will show later, PIL over $\mathrm{FO}+\mathrm{H}$ is equivalent to $\mathrm{CPT}+\mathrm{C}$, so PIL over $\mathrm{FO}+\mathrm{H}$ could also be defined as a two-valued logic with the additional requirement that each program maintains a counter and halts as soon as it exceeds the polynomial bounds. But, since this technical detail will not be important for any of the programs considered in this thesis, we define PIL as a three-valued logic uniformly for both FO and $\mathrm{FO}+\mathrm{H}$.

In the following, we denote by IL (resp. PIL) interpretation logic over FO, and by $\mathrm{IL}+\mathrm{H}$ (resp. $\mathrm{PIL}+\mathrm{H}$) interpretation logic over $\mathrm{FO}+\mathrm{H}$.

4.1 Example IL programs

In this section, we present some example IL programs to give an intuition of what is possible in IL and PIL and to illustrate some techniques that seem generally useful for defining queries in IL. Some of the techniques introduced here are also used extensively in the main proof of this chapter.

First, we give examples of operations that can be implemented using only a single interpretation, i.e. that do not need the iteration mechanism of IL. In order to use these interpretations in PIL programs, we then show that for each interpretation, there is a PIL program that only applies that interpretation.

An important feature of logical interpretations is their ability to extend the domain by additional elements. Therefore we introduce a method of adding a definable object to the domain, and then generalise that method to be able to create several new elements corresponding to an FO-definable set of existing elements. This approach is often used for constructing more complex IL programs.

Lemma 26. *There is an* $\mathrm{FO}[\tau, \tau \cup \{R_a\}]$ *interpretation* I *such that for each structure* $\mathfrak{A} = (A, \tau)$*, the domain of* $I(\mathfrak{A})$ *is isomorphic to a copy of* A *enriched by a new element* a*, where* $R_a^{I(\mathfrak{A})} = \{a\}$ *and the relations in* τ *remain unchanged on the copy of* A*.*

Proof. In $I(\mathfrak{A})$, each element $a \in A$ is represented by the pair (a, a), and the new element is represented by the equivalence class containing all pairs (a, b)

such that $a \neq b$. So the interpretation $\left(\varphi_{\text{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \tau \cup \{R_a\}}\right)$ with

$$\begin{aligned}
\varphi_{\text{dom}}(x,y) &= \text{True} \ , \\
\varphi_{\approx}(x,y,x',y') &= x \neq y \wedge x' \neq y' \ , \\
\varphi_{R_a}(x,y) &= x \neq y \ , \\
\varphi_R(x_1,y_1,\dots,x_r,y_r) &= \bigwedge_{1 \le i \le r} x_i = y_i \wedge Rx_1 \dots x_r \text{ for each } r\text{-ary } R \in \tau
\end{aligned}$$

has the desired property. □

The same idea is now used to define an interpretation that creates a new element for each element satisfying a given FO-formula, and that defines a relation symbol that identifies the correspondence between old and new elements.

Lemma 27. *Let φ be an* FO$[\tau]$*-formula with k free variables. There is an* FO$[\tau, \tau \dot\cup \{R\}]$*-interpretation I for a $k+1$-ary predicate R such that, for each $\mathfrak{A} = (A, \tau)$, $I(A)$ is isomorphic to a copy of A enriched by exactly one element $a_{\overline{b}}$ for each $\overline{b} \in A^k$ with $\mathfrak{A} \models \varphi(\overline{b})$, where $R^{I(\mathfrak{A})} = \left\{\left(\overline{b}, a_{\overline{b}}\right) \mid \mathfrak{A} \models \varphi\left(\overline{b}\right)\right\}$ and the relations in τ remain unchanged on the copy of A.*

Proof. Take a $k+1$-dimensional interpretation I. Each object $a \in A$ is represented by the equivalence class containing all tuples $(b_1, \dots, b_k, a, a)$, and the new object for the tuple $(b_1, \dots, b_k)$ is represented by the equivalence class containing all tuples $(b_1, \dots, b_k, a_1, a_2)$ for $a_1 \neq a_2$. This is realised by the FO-interpretation $\left(\varphi_{\text{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \tau \cup \{R\}}\right)$ with

$$\begin{aligned}
\varphi_{\text{dom}}(x_1,\dots,x_k,y,z) &= y = z \vee (y \neq z \wedge \varphi(x_1,\dots,x_k)) \, , \\
\varphi_{\approx}(\overline{x},y,z,\overline{x'},y',z') &= (y = z \wedge y = y' \wedge z = z') \\
&\quad \vee \left(y \neq z \wedge y' \neq z' \wedge \bigwedge_{1 \le i \le k} x_i = x'_i \right) , \\
\varphi_{R_V}(\overline{x^1},y^1,z^1,\dots,\overline{x^{k+1}},y^{k+1},z^{k+1}) &= \bigwedge_{1 \le i \le k} y^i = z^i \wedge \varphi(y^1,\dots,y^k) \wedge \\
&\quad y^{k+1} \neq z^{k+1} \wedge \bigwedge_{1 \le i \le k} y^i = x_i^{k+1} \ .
\end{aligned}$$

□

As a first example of a PIL program, we show how interpretations can be transformed to equivalent PIL programs.

Remark 28. *Let $I = \big(\varphi_{dom}, \varphi_{\approx}, (\varphi_R)_{R\in\sigma}\big)$ be a k-dimensional* $\mathrm{FO}[\tau,\sigma]$ *(resp.* $\mathrm{FO}+\mathrm{H}[\tau,\sigma]$*)-interpretation. There is a* $\mathrm{PIL}[\tau,\sigma]$ *(resp.* $\mathrm{PIL}+\mathrm{H}[\tau,\sigma]$*) program $\overline{\Pi_I} = (\Pi_I, p, q)$ such that the final state of the run of Π_I on each τ-structure $\mathfrak{A}$ is $I(\mathfrak{A})$.*

Proof. Take $\Pi_I = (I, I_{\mathrm{id}}, \mathrm{True}, \mathrm{True})$, where I_{id} is the identity. $\overline{\Pi_I} = (\Pi_I, 1, n^k)$ is a polynomial-time version of Π_I. □

Logics in a classical sense incorporate Boolean connectives, so they are naturally closed under Boolean operations. Since PIL does not have this syntactic property, we show explicitly that Boolean connectives are definable in PIL.

Lemma 29. *The class of* PIL*-definable queries is closed under intersection.*

Proof. Let $\overline{\Pi_1} = (\Pi_1, p_1, q_1)$ and $\overline{\Pi_2} = (\Pi_2, p_2, q_2)$ be $\mathrm{PIL}[\tau,\sigma]$ programs with $\Pi_1 = \big(I^1_{\mathrm{init}}, I^1_{\mathrm{step}}, \varphi^1_{\mathrm{halt}}, \varphi^1_{\mathrm{out}}\big)$ and $\Pi_2 = \big(I^2_{\mathrm{init}}, I^2_{\mathrm{step}}, \varphi^2_{\mathrm{halt}}, \varphi^2_{\mathrm{out}}\big)$. Assume that $I^1_{\mathrm{init}}, I^2_{\mathrm{init}}, I^1_{\mathrm{step}}$ and I^2_{step} are k-dimensional interpretations (for $\ell < k$, modify ℓ-dimensional intepretations in a way that tuples that coincide on the first ℓ components represent the same element of the interpreted structure). We construct an IL program $\Pi_1 \cap \Pi_2 = \big(I_{\mathrm{init}}, I_{\mathrm{step}}, \varphi_{\mathrm{halt}}, \varphi_{\mathrm{out}}\big)$ that accepts exactly those structures accepted by both Π_1 and Π_2. During initialisation, $\Pi_1 \cap \Pi_2$ creates two copies of the input structure. Then $\Pi_1 \cap \Pi_2$ runs Π_1 and Π_2 simultaneously on one of these copies each, using nullary predicates $\mathrm{Halt}_1, \mathrm{Halt}_2, \mathrm{Out}_1, \mathrm{Out}_2$ to store the result of the program that halts first.

To create these copies, let I be an interpretation that, for each existing object, adds another object to the domain (use Lemma 27 with the formula $\varphi = \mathrm{True}$), and let R be the relation symbol introduced by I. Then I_{init} is I extended by the formulae $\varphi_{\mathrm{Halt}_1} = \varphi_{\mathrm{Halt}_2} = \varphi_{\mathrm{Out}_1} = \varphi_{\mathrm{Out}_2} = \mathrm{False}$, $\varphi_{\mathrm{Init}} = \mathrm{True}$ and a formula φ_P for a unary predicate P that labels the new copy of the domain:

$$\varphi_P(x,y) = \exists x' \exists y' \varphi_R(x', y', x, y) \quad .$$

The interpretation I_{step} applies the original interpretations: I^1_{init} and I^2_{init} to P and its complement, respectively, in the first step (which is marked by Init), and afterwards applies I^1_{step} and I^2_{step} until the simulated programs halt.

Let $I^1_{\text{init}} = \left(\varphi^{\text{init},1}_{\text{dom}}, \varphi^{\text{init},1}_{\approx}, \left(\varphi^{\text{init},1}_R\right)_{R\in\sigma}\right)$, $I^2_{\text{init}} = \left(\varphi^{\text{init},2}_{\text{dom}}, \varphi^{\text{init},2}_{\approx}, \left(\varphi^{\text{init},2}_R\right)_{R\in\sigma}\right)$, $I^1_{\text{step}} = \left(\varphi^1_{\text{dom}}, \varphi^1_{\approx}, (\varphi^1_R)_{R\in\sigma}\right)$ and $I^2_{\text{step}} = \left(\varphi^2_{\text{dom}}, \varphi^2_{\approx}, (\varphi^2_R)_{R\in\sigma}\right)$, and define the interpretation $I_{\text{step}} = \left(\varphi_{\text{dom}}, \varphi_{\approx}, (\varphi_R)_{R\in\sigma\dot\cup\{P,\text{Init},\text{Halt}_1,\text{Halt}_2,\text{Out}_1,\text{Out}_2\}}\right)$ as follows (let $\overline{x} = (x_1,\dots,x_k), \overline{y} = (y_1,\dots,y_k)$, and $\overline{x^i} = (x^i_1,\dots,x^i_k)$):

$$\begin{aligned}
\varphi_{\text{dom}}(\overline{x}) = & \bigwedge_{1\le i\le k} Px_i \wedge \text{Init} \wedge \varphi^{\text{init},1}_{\text{dom}}(\overline{x}) \vee \bigwedge_{1\le i\le k} \neg Px_i \wedge \text{Init} \wedge \varphi^{\text{init},2}_{\text{dom}}(\overline{x}) \vee \quad , \\
& \bigwedge_{1\le i\le k} Px_i \wedge \neg\,\text{Init} \wedge \neg\,\text{Halt}_1 \wedge \varphi^1_{\text{dom}}(\overline{x}) \vee \\
& \bigwedge_{1\le i\le k} \neg Px_i \wedge \neg\,\text{Init} \wedge \neg\,\text{Halt}_2 \wedge \varphi^2_{\text{dom}}(\overline{x}) \\
\varphi_{\approx}(\overline{x},\overline{y}) = & \bigwedge_{1\le i\le k} (Px_i \wedge Py_i) \wedge \text{Init} \wedge \varphi^{\text{init},1}_{\approx}(\overline{x},\overline{y}) \vee \\
& \bigwedge_{1\le i\le k} (\neg Px_i \wedge \neg Py_i) \wedge \text{Init} \wedge \varphi^{\text{init},1}_{\approx}(\overline{x},\overline{y}) \vee \\
& \bigwedge_{1\le i\le k} (Px_i \wedge Py_i) \wedge \neg\,\text{Init} \wedge \neg\,\text{Halt}_1 \wedge \varphi^1_{\approx}(\overline{x},\overline{y}) \vee \\
& \bigwedge_{1\le i\le k} (\neg Px_i \wedge \neg Py_i) \wedge \neg\,\text{Init} \wedge \neg\,\text{Halt}_2 \wedge \varphi^2_{\approx}(\overline{x},\overline{y}) \\
\varphi_R(\overline{x^1},\dots,\overline{x^r}) = & \bigwedge_{1\le i\le k} Px^j_i \wedge \text{Init} \wedge \varphi^{\text{init},1}_R(\overline{x^1},\dots,\overline{x^r}) \vee \\
& \bigwedge_{1\le i\le k} \neg Px^j_i \wedge \text{Init} \wedge \varphi^{\text{init},2}_R(\overline{x^1},\dots,\overline{x^r}) \vee \\
& \bigwedge_{1\le i\le k} Px^j_i \wedge \neg\,\text{Init} \wedge \neg\,\text{Halt}_1 \wedge \varphi^1_R(\overline{x^1},\dots,\overline{x^r}) \vee \\
& \bigwedge_{1\le i\le k} \neg Px^j_i \wedge \neg\,\text{Init} \wedge \neg\,\text{Halt}_2 \wedge \varphi^2_R(\overline{x^1},\dots,\overline{x^r}) \text{ for all } R\in\sigma \\
\varphi_P(\overline{x}) = & \bigwedge_{1\le i\le k} Px_i \\
\varphi_{\text{Init}} = & \,\text{False} \\
\varphi_{\text{Halt}_1} = & \,\text{Halt}_1 \vee {\varphi^1_{\text{halt}}}' \\
\varphi_{\text{Out}_1} = & \,\text{Out}_1 \vee ({\varphi^1_{\text{halt}}}' \wedge \neg\,\text{Halt}_1) \quad ,
\end{aligned}$$

analogously for $\varphi_{\mathrm{Halt}_2}$ and φ_{Out_2}, where ${\varphi^i_{\mathrm{halt}}}'$ is obtained from $\varphi^i_{\mathrm{halt}}$ by relativising all quantifiers to P (resp. the complement of P). Then I_{step} simulates Π_1 on P and Π_2 on the complement of P, and stores the result in $\mathrm{Halt}_1, \mathrm{Halt}_2, \mathrm{Out}_1, \mathrm{Out}_2$. So let $\varphi_{\mathrm{halt}} = \mathrm{Halt}_1 \wedge \mathrm{Halt}_2$ and $\varphi_{\mathrm{out}} = \mathrm{Out}_1 \wedge \mathrm{Out}_2$. Then $\Pi_1 \cap \Pi_2$ computes the desired output.

Thus there is an IL program that accepts exactly the intersection of the Boolean queries accepted by Π_1 and Π_2. It remains to show that there is a PIL program with the same property. We show that there is a polynomial-time version of $\Pi_1 \cap \Pi_2$. Let $\mathfrak{A} = (A, \tau)$ be a structure, and let ρ be the run of $\Pi_1 \cap \Pi_2$ on $\mathfrak{A}$. The length of the run of Π_1 on $\mathfrak{A}$ is bounded by p_1, and the length of the run of Π_2 on $\mathfrak{A}$ is bounded by p_2. So, since ρ halts when both simulated programs halt, and each state of ρ except for the initial state corresponds to a state of the simulated programs, the length of ρ is bounded by $\max(p_1, p_2) + 1$.

Furthermore, each state in ρ except for the initial state is isomorphic to the disjoint union of states of the runs of Π_1 and Π_2 on $\mathfrak{A}$, so the size of each state of ρ is bounded by $q_1 + q_2$.

So $(\Pi_1 \cap \Pi_2, \max(p_1, p_2) + 1, q_1 + q_2)$ is a polynomial-time version of $\Pi_1 \cap \Pi_2$ for any PIL programs Π_1, Π_2, so PIL is closed under intersection.

$\square$

Showing that PIL is closed under complementation is a lot less involved, since it can directly use negation of FO-formulae.

Remark 30. *The class of* PIL*-definable queries is closed under complementation.*

Proof. Let $\Pi = \big(I_{\mathrm{init}}, I_{\mathrm{step}}, \varphi_{\mathrm{halt}}, \varphi_{\mathrm{out}}\big)$ be an IL program. Then, clearly, $\Pi' = (I_{\mathrm{init}}, I_{\mathrm{step}}, \varphi_{\mathrm{halt}}, \neg\varphi_{\mathrm{out}})$ accepts a structure $\mathfrak{A}$ if Π rejects $\mathfrak{A}$ and Π' rejects $\mathfrak{A}$ if Π accepts $\mathfrak{A}$.

If (Π, p, q) is a polynomial-time version of Π, then (Π', p, q) is a polynomial-time version of Π'. So PIL is closed under complementation. $\square$

Like CPT, PIL can be viewed as a machine model. Since sequential execution of different programs is a natural operation in many machine models, we define the concatenation of PIL programs. As we will see later, concatenation is a valuable tool for constructing PIL programs.

Definition 31. *Let* $\Pi_1 = \big(I^1_{init}, I^1_{step}, \varphi^1_{halt}, \varphi^1_{out}\big)$ *be an* $\mathrm{IL}[\tau, \tau']$ *program and let* $\Pi_2 = \big(I^2_{init}, I^2_{step}, \varphi^2_{halt}, \varphi^2_{out}\big)$ *be an* $\mathrm{IL}[\tau', \sigma]$ *program over* FO *or* FO + H.

Furthermore, let

- $I^1_{\mathit{init}} = \Big(\varphi^{\mathit{init},1}_{\mathit{dom}}, \varphi^{\mathit{init},1}_{\approx}, \big(\varphi^{\mathit{init},1}_{R}\big)_{R\in\tau'}\Big)$,
- $I^2_{\mathit{init}} = \Big(\varphi^{\mathit{init},2}_{\mathit{dom}}, \varphi^{\mathit{init},2}_{\approx}, \big(\varphi^{\mathit{init},2}_{R}\big)_{R\in\sigma}\Big)$,
- $I^1_{\mathit{step}} = \Big(\varphi^{\mathit{step},1}_{\mathit{dom}}, \varphi^{\mathit{step},1}_{\approx}, \big(\varphi^{\mathit{step},1}_{R}\big)_{R\in\tau'}\Big)$ *and*
- $I^2_{\mathit{step}} = \Big(\varphi^{\mathit{step},2}_{\mathit{dom}}, \varphi^{\mathit{step},2}_{\approx}, \big(\varphi^{\mathit{step},2}_{R}\big)_{R\in\sigma}\Big)$.

Then the concatenation $\Pi_1 \circ \Pi_2 = \big(I_{\mathit{init}}, I_{\mathit{step}}, \varphi_{\mathit{halt}}, \varphi_{\mathit{out}}\big)$ *of* Π_1 *and* Π_2 *is the* $\mathrm{IL}[\tau,\sigma']$ *program defined as follows:*

- $\sigma' = \sigma \cup \{P\}$, *where* P *is a new nullary predicate,*
- $I_{\mathit{init}} = \big(\varphi^{\mathit{init}}_{\mathit{dom}}, \varphi^{\mathit{init}}_{\approx}, (\varphi^{\mathit{init}}_{R})_{R\in\sigma'}\big)$ *and* $I_{\mathit{step}} = \Big(\varphi^{\mathit{step}}_{\mathit{dom}}, \varphi^{\mathit{step}}_{\approx}, \big(\varphi^{\mathit{step}}_{R}\big)_{R\in\sigma'}\Big)$,
- P *is true as soon as* $\Pi_1 \circ \Pi_2$ *has simulated* Π_1, *i.e.* $\varphi^{\mathit{init}}_{P} = \mathrm{False}$ *and* $\varphi^{\mathit{step}}_{P} = P \vee \varphi^1_{\mathit{halt}}$
- *while* P *is false and* Π_1 *does not halt, apply* I^1_{step}, *after* Π_1 *halts apply* I^2_{init} *once, otherwise apply* I^2_{step}*:*

  $$\varphi^{\mathit{step}}_{\mathit{dom}} = \big(\neg\varphi^1_{\mathit{halt}} \wedge \neg P \wedge \varphi^{\mathit{step},1}_{\mathit{dom}}\big) \vee \big(\varphi^1_{\mathit{halt}} \wedge \neg P \wedge \varphi^{\mathit{init},2}_{\mathit{dom}}\big) \vee \big(P \wedge \varphi^{\mathit{step},2}_{\mathit{dom}}\big) \quad ,$$

  analogously for $\varphi_{\approx}$ *and* φ_R, $R \in \sigma$,
- $\Pi_1 \circ \Pi_2$ *halts when* Π_2 *halts and has the same output as* Π_2, *i.e.* $\varphi_{\mathit{halt}} = \varphi^2_{\mathit{halt}}$ *and* $\varphi_{\mathit{out}} = \varphi^2_{\mathit{out}}$.

Consider the run ρ_1 of Π_1 on a structure $\mathfrak{A}$ and the run ρ_2 of Π_2 on the final state of ρ_1. By definition, $\Pi_1 \circ \Pi_2$ acts like Π_1 until $\varphi^1_{\mathrm{halt}}$ is true, so the state at that step is the final state of ρ_1, and then, the program acts like Π_2, so the final state of the run of $\Pi_1 \circ \Pi_2$ on $\mathfrak{A}$ is the final state of ρ_2, and the program has the respective output. So running $\Pi_1 \circ \Pi_2$ on $\mathfrak{A}$ yields the same final state and output as running Π_1 and Π_2 sequentially.

Remark 32. *Let* Π_1 *be an* $\mathrm{IL}[\tau,\tau']$ *program, let* Π_2 *be an* $\mathrm{IL}[\tau',\sigma]$ *program, and let* (Π_1, p_1, q_1), (Π_2, p_2, q_2) *be polynomial-time versions of* Π_1 *and* Π_2. *Then there is a polynomial-time version of* $\Pi_1 \circ \Pi_2$.

Proof. $(\Pi_1 \circ \Pi_2, p, q)$ with $p(n) = p_1(n) + p_2(q_1(n))$ and $q(n) = q_2(q_1(n))$ for all $n \in \mathbb{N}$ is a polynomial-time version of $\Pi_1 \circ \Pi_2$. □

In particular, the definition of concatenation demonstrates that it is possible for an IL program to simulate another IL program. A similar technique can be used to iterate an IL program in a loop-like construct until a given FO- or $\mathrm{FO}+\mathrm{H}$-formula holds.

Definition 33. *Let* $\Pi = \left(I_{init}, I_{step}, \varphi_{halt}, \varphi_{out}\right)$ *be an* $\mathrm{IL}[\tau, \sigma]$ *program over* FO *or* $\mathrm{FO}+\mathrm{H}$ *with* $\tau \subseteq \sigma$*, where* $I_{init} = \left(\varphi^{init}_{dom}, \varphi^{init}_{\approx}, (\varphi^{init}_R)_{R\in\sigma}\right)$ *and* $I_{step} = \left(\varphi^{step}_{dom}, \varphi^{step}_{\approx}, \left(\varphi^{step}_R\right)_{R\in\sigma}\right)$*, and let* $\psi \in \mathrm{FO}+\mathrm{H}[\sigma]$*. Then the* iteration *of* Π *with respect to the formula* ψ *is the* $\mathrm{IL}[\tau, \sigma]$ *program* $\Pi^\psi = \left(I^\psi_{init}, I^\psi_{step}, \varphi^\psi_{halt}, \varphi^\psi_{out}\right)$ *where* $\varphi^\psi_{halt} = \varphi_{halt} \wedge \psi$*,* $\varphi^\psi_{out} = \varphi_{out}$*,* $I^\psi_{init} = I_{init}$ *and* $I^\psi_{step} = \left(\varphi^\psi_{dom}, \varphi^\psi_{\approx}, \left(\varphi^\psi_R\right)_{R\in\sigma}\right)$ *with*

- $\varphi^\psi_{dom} = \left(\varphi_{halt} \wedge \varphi^{init}_{dom}\right) \vee \left(\neg\varphi_{halt} \wedge \varphi^{step}_{dom}\right)$
- $\varphi^\psi_{\approx} = \left(\varphi_{halt} \wedge \varphi^{init}_{\approx}\right) \vee \left(\neg\varphi_{halt} \wedge \varphi^{step}_{\approx}\right)$
- $\varphi^\psi_R = \left(\varphi_{halt} \wedge \varphi^{init}_R\right) \vee \left(\neg\varphi_{halt} \wedge \varphi^{step}_R\right)$ *for* $R \in \sigma$*.*

The program Π^ψ halts if and only if both φ_{halt} and ψ are true, i.e. if and only if Π halts and ψ is true. Whenever φ_{halt} is true and Π^ψ does not halt, it applies I_{init} again, and otherwise, it applies I_{step}. So Π^ψ indeed simulates Π until ψ holds.

Note that if there is a polynomial-time version of Π, it is not guaranteed that there is also a polynomial-time version of Π^ψ, since ψ does not have to be true after a bounded number of runs of Π.

As mentioned in Chapter 1, $\mathrm{CPT}+\mathrm{C}$ programs benefit from padding of the input structure, because, as illustrated in [BGS99], a linear order can be constructed on a sufficiently small definable subset of the input structure. Then, since $\mathrm{FP}+\mathrm{C}$ already captures PTIME on ordered structures, any PTIME query can be defined in $\mathrm{CPT}+\mathrm{C}$ on the resulting ordered substructure.

In the following, we implement the naive approach given in [BGS99], which, in contrast to the canonisation algorithm proposed in [Lau11], simply constructs all linear orders on a subset, in interpretation logic.

Assume we have given an input structure $\mathfrak{A} = (A, \tau)$ with a predicate $P \in \tau$ such that $\left|P^{\mathfrak{A}}\right|! \leq |A|$. Our IL program outputs a structure where each element of the domain represents either an element of the input structure or a linear order on P. Moreover, it defines a relation $R_<$ that encodes the linear orders as follows: In the final state $\mathfrak{A}_{\text{fin}}$ of a run of the program, a tuple (ℓ, a, b) is in $R_<^{\mathfrak{A}_{\text{fin}}}$ if and only if $a <_\ell b$, where $<_\ell$ is the linear order represented by ℓ.

To construct these orders, the program first creates a new linear order $<_\ell$ for each pair (a, b) of elements of $P^{\mathfrak{A}}$ (assuming that $P^{\mathfrak{A}}$ has at least two elements, because otherwise the task is trivial) and defines $a <_\ell b$. In the following steps, a new linear order is created for each pair (ℓ, a) of a linear order ℓ and an element a of the input structure such that a is not in the order $<_\ell$ represented by ℓ. The new order is then isomorphic to $<_\ell$ extended by a as the new maximal element.

This is achieved by the IL program $\Pi = \left(I_{\text{init}}, I_{\text{step}}, \varphi_{\text{halt}}, \varphi_{\text{out}}\right)$ defined as follows. I_{init} creates the initial linear orders with a modified version of the interpretation obtained from Lemma 27. More precisely, consider the interpretation that adds an object for each pair of distinct elements in P according to Lemma 27. Add the following formula defining $R_<$ to obtain I_{init} from that interpretation:

$$\varphi_{R_<}(x, y, z) = \varphi_{R_{\text{new}}}(y, z, x) \ ,$$

where $\varphi_{R_{\text{new}}}$ is the formula defining the relation R_{new} thats maps each pair (a, b) of elements in P to the new element $\ell_{(a,b)}$ associated with (a, b). After I_{init} has been applied, the following formula defines the elements that represent a linear order:

$$\varphi_{\text{order}}(x) = \exists y \exists z R_< xyz \ .$$

I_{step} is again derived from an interpretation that adds new objects according to Lemma 27, this time for any pair of elements that satisfies the following formula:

$$\varphi(x, y) = \varphi_{\text{order}}(x) \land Py \land \forall z \left(\neg R_< xyz \land \neg R_< xzy\right) \ .$$

The interpretation is again extended by a formula defining $R_<$. Note that $R_< \ell ab$ should be true if either $a <_{\ell'} b$, or a is already ordered by $<_{\ell'}$ and b is the new maximal element, where ℓ encodes the order that extends $<_{\ell'}$.

$$\begin{aligned}\varphi_{R_<}(x, y, z) =& \exists x' \exists y' \, (\varphi_{R_{\text{new}}}(x', y', x) \wedge R_< x'yz) \vee \\ & \exists x' \, (\varphi_{R_{\text{new}}} x'zx \wedge \exists y' \, (R_< x'y'y \vee R_< x'yy')) \quad ,\end{aligned}$$

where $\varphi_{R_{\text{new}}}$ is defined as above.

Π should halt when all linear orders have been constructed, i.e. when all orders are total orders on P:

$$\varphi_{\text{halt}} = \forall x \forall y \, (\varphi_{\text{order}}(x) \wedge Py \rightarrow \exists z (R_< xyz \vee R_< xzy)) \quad .$$

The program Π constructs all linear orders on P, and, if $\left|P^{\mathfrak{A}}\right|! \leq |A|$, the length of the run on $\mathfrak{A}$ is linear in $|A|$, because in each state, at least one of the $|A|!$ orders is created. The size of the largest state differs from that of the input structure by the number of linear orders, so it is also linear in $|A|$.

Note that Π is not a PIL program. A polynomial-time program with the same functionality would have to check whether P is small enough, which is obviously possible in PIL + H, but presumably not in PIL without the equicardinality quantifier.

4.2 Equivalence of PIL + H and CPT + EqCard

In the following, the expressive power of polynomial-time interpretation logic is compared to that of Choiceless Polynomial Time. More precisely, it is shown that PIL and CPT are equivalent. Moreover, also the extensions by the respective equicardinality operators (the EqCard relation and the Härtig quantifier) remain equivalent:

Theorem 34.

1. CPT $\equiv$ PIL.
2. CPT + EqCard $\equiv$ PIL + H.

We start with the relatively straight-forward simulation of PIL programs in CPT, i.e. we show the following lemma:

Lemma 35.

1. $\mathrm{PIL} \leq \mathrm{CPT}$.
2. $\mathrm{PIL} + \mathrm{H} \leq \mathrm{CPT} + \mathrm{EqCard}$.

The proof uses the definition of CPT introduced by Blass, Gurevich and Shelah, which we refer to as $\mathrm{CPT}_{\mathrm{orig}}$. A program in interpretation logic consists of FO or $\mathrm{FO} + \mathrm{H}$ interpretations, which are sequences of FO or $\mathrm{FO} + \mathrm{H}$ formulae. Therefore we first establish a notion of equivalence between $\mathrm{BGS}_{\mathrm{orig}}$ and FO (resp. $\mathrm{BGS}_{\mathrm{orig}} + \mathrm{EqCard}$ and $\mathrm{FO} + \mathrm{H}$), and then show that for each FO formula, there is a $\mathrm{BGS}_{\mathrm{orig}}$ term (and for each $\mathrm{FO} + \mathrm{H}$ formula, there is a $\mathrm{BGS}_{\mathrm{orig}} + \mathrm{EqCard}$ term) that is equivalent in that sense.

Definition 36. *Let $\varphi(\overline{x})$ be an $\mathcal{L}[\sigma]$-formula, where $\mathcal{L}$ is an extension of* FO*, and let $\sigma^{\mathrm{HF}}_{ext} \supseteq \sigma^{\mathrm{HF}}$. A Boolean $\mathrm{BGS}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$ term $t_\varphi(\overline{x}, y)$ is equivalent to φ over $\mathrm{BGS}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$ terms if, for every $\mathrm{BGS}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$ term s and every state $\mathfrak{A}$ of a $\mathrm{BGS}_{orig}[\sigma^{\mathrm{HF}}_{ext}]$ program, $(\llbracket s \rrbracket^{\mathfrak{A}}, \sigma^{\mathrm{HF}}_{ext}) \models \varphi(\overline{a})$ if and only if $\llbracket t_\varphi(\overline{a}, s) \rrbracket^{\mathfrak{A}}$ is true.*

Lemma 37. *For every* FO *formula φ, there is a Boolean BGS_{orig} term t_φ that is equivalent to φ over BGS_{orig} terms.*

Proof. Proof by induction on φ.

Induction Base: Clear.

Induction Step: For Boolean connectives, there is a direct translation to $\mathrm{BGS}_{\mathrm{orig}}$ terms. So let $\varphi(\overline{x}) = \exists z \psi(\overline{x}, z)$. Then let

$$t_\varphi(\overline{x}, y) = \mathrm{In}(\emptyset, \{\emptyset : z \in y : t_\psi(\overline{x}, y, z)\}) \ .$$

$\square$

Lemma 38. *For every* $\mathrm{FO} + \mathrm{H}$ *formula φ, there is a $\mathrm{BGS}_{orig} + \mathrm{EqCard}$ term t_φ that is equivalent to φ over $\mathrm{BGS}_{orig} + \mathrm{EqCard}$ terms.*

Proof. Analogous to the proof of Lemma 37. If $\varphi(\overline{x}) = \mathrm{H}\, z_1 z_2 \psi_1(\overline{x}, z_1) \psi_2(\overline{x}, z_2)$, then let

$$t_\varphi(\overline{x}, y) = \mathrm{EqCard}\left(\{z_1 : z_1 \in y : t_{\psi_1}(\overline{x}, y, z_1)\}, \{z_2 : z_2 \in y : t_{\psi_2}(\overline{x}, y, z_2)\}\right) \ .$$

$\square$

Now we can show that each PIL program can be simulated in $\mathrm{CPT}_{\mathrm{orig}}$ and each PIL + H program can be simulated in $\mathrm{CPT}_{\mathrm{orig}} + \mathrm{EqCard}$. To cover both PIL and PIL + H in the proof, we show equivalence generically for all extensions of PIL and $\mathrm{CPT}_{\mathrm{orig}}$ that satisfy the condition shown in Lemmas 37 and 38.

Lemma 39. *Let $\mathcal{L}$ be a logic and let $\tau^{\mathrm{HF}}_{ext} \supseteq \tau^{\mathrm{HF}}$ such that for each $\mathcal{L}[\tau]$-formula φ, there is a Boolean term over τ^{HF}_{ext} that is equivalent to φ over $\mathrm{BGS}_{orig}[\tau^{\mathrm{HF}}_{ext}]$ terms. Then for each* PIL *program over $\mathcal{L}$, there is an equivalent $\mathrm{BGS}_{orig}[\tau^{\mathrm{HF}}_{ext}]$ program.*

Proof. Let $\mathcal{L}$ be a logic that fulfills the condition for the lemma with respect to $\mathrm{BGS}_{\mathrm{orig}}[\tau^{\mathrm{HF}}_{\mathrm{ext}}]$ for some signature $\tau^{\mathrm{HF}}_{\mathrm{ext}} \supseteq \tau^{\mathrm{HF}}$, and let $\overline{\Pi} = (\Pi, p, q)$ with $\Pi = \big(I_{\mathrm{init}}, I_{\mathrm{step}}, \varphi_{\mathrm{halt}}, \varphi_{\mathrm{out}}\big)$ be a $\mathrm{PIL}[\tau, \sigma]$ program over the logic $\mathcal{L}$, where $I_{\mathrm{init}} = \Big(\varphi^{\mathrm{init}}_{\mathrm{dom}}, \varphi^{\mathrm{init}}_{\approx}, \big(\varphi^{\mathrm{init}}_{R}\big)_{R\in\sigma}\Big)$ and $I_{\mathrm{step}} = \Big(\varphi^{\mathrm{step}}_{\mathrm{dom}}, \varphi^{\mathrm{step}}_{\approx}, \big(\varphi^{\mathrm{step}}_{R}\big)_{R\in\sigma}\Big)$.

We construct a $\mathrm{BGS}_{\mathrm{orig}}$ program Π_{CPT} equivalent to Π. To simulate the run ρ of Π on any given structure, the domain and predicates of the current state of ρ are representend as dynamic predicates and updated accordingly until φ_{halt} is true.

So Π_{CPT} uses the following predicates:

- An r-ary input predicate P for each r-ary $P \in \tau$ to represent the predicates of the input structure,
- a dynamic constant Dom that represents the set of objects that correspond to elements in the domain of the current state, and
- a dynamic constant R for each $R \in \sigma$ to represent the predicates of the states in ρ as sets of tuples,
- a nullary dynamic predicate Init that marks whether I_{init} or I_{step} is currently applied.

By assumption, there are Boolean $\mathrm{BGS}_{\mathrm{orig}}[\tau^{\mathrm{HF}}_{\mathrm{ext}}]$ terms $t^{\mathrm{init}}_{\mathrm{dom}}$, $t^{\mathrm{step}}_{\mathrm{dom}}$, $\{t^{\mathrm{init}}_{R}\}$, $\{t^{\mathrm{step}}_{R}\}$ that are equivalent to $\varphi^{\mathrm{init}}_{\mathrm{dom}}$, $\varphi^{\mathrm{step}}_{\mathrm{dom}}$, $\{\varphi^{\mathrm{init}}_{R}\}$, $\{\varphi^{\mathrm{step}}_{R}\}$, respectively, over $\mathrm{BGS}_{\mathrm{orig}}[\tau^{\mathrm{HF}}_{\mathrm{ext}}]$ terms. For ease of notation, we assume that these terms are modified in a way that instead of values $a_1, \ldots, a_k$ of a sequence of k free variables encoding an element of the interpreted structure, the terms take as input sets encoding k-tuples $\langle a_1, \ldots, a_k\rangle$.

Π_{CPT} first applies I_{init} to the input structure (where the domain is defined by the predicate Atoms) and then applies I_{step} to the resulting structure until φ_{halt} is true. The predicate Init is modified accordingly after that step. In each step, the dynamic constants Dom and $\{R\}_{R\in\sigma}$ have to be updated in a way that they indicate the domain and extension of the predicates of the current state of Π.

The domain of each state of Π consists of the equivalence classes of the relation $\approx$ defined by $\varphi_{\approx}^{\mathrm{step}}$ and $\varphi_{\approx}^{\mathrm{init}}$, respectively. Since Π_{CPT} works with hereditarily finite sets, these objects can be encoded naturally as the actual $\approx$-equivalence classes. So, to apply I_{init}, the domain is modified by the following rule:

$$\begin{aligned}\mathrm{Dom} \coloneqq \big\{\{y : y \in \mathrm{Atoms}^k : t_{\mathrm{dom}}^{\mathrm{init}}(y, \mathrm{Atoms}^k) \wedge t_{\approx}^{\mathrm{init}}(x, y, \mathrm{Atoms}^k)\}\\ : x \in \mathrm{Atoms}^k : t_{\mathrm{dom}}^{\mathrm{init}}(x, \mathrm{Atoms}^k)\big\} \quad ,\end{aligned}$$

where k is the dimension of I_{init}, and the last free variable of $t_{\mathrm{dom}}^{\mathrm{init}}$, $t_{\approx}^{\mathrm{init}}$ and $t_{\mathrm{dom}}^{\mathrm{init}}$ represents the term defining the domain over which the respective terms are evaluated, as specified in Definition 36. Note that the set Atoms^k is definable with the term $t_\times$ obtained from Lemma 20.

Each r-ary relation symbol R is now updated using the rule

$$R \coloneqq \big\{x : x \in \mathrm{Dom}^r : t_R^{\mathrm{init}}(x, \mathrm{Dom}^r)\big\} \quad .$$

Clearly, formulae for applying I_{step} can be defined analogously.

After these initial steps, the dynamic predicate Init is set to false. Whenever Init is false, the program checks whether φ_{halt} (again, there is a Boolean term that is equivalent to this formula over states) holds in $[\![\mathrm{Dom}]\!]$. If the formula is true, the dynamic predicates Halt and Out are updated to correspond to the values of φ_{halt} and φ_{out}, otherwise, Dom and the constants R for relations $R \in \sigma$ are updated according to I_{step}.

By definition, after each step, Dom is interpreted by a set containing exactly the elements in the universe of the corresponding state of the run of Π, and the value of R is exactly the extension of R in that state. So, as Halt and Out are defined accordingly, Π_{CPT} accepts $\mathfrak{A}$ if and only if Π accepts $\mathfrak{A}$, and rejects $\mathfrak{A}$ if and only if Π rejects $\mathfrak{A}$.

It remains to show that there is a polynomial-time version of Π_{CPT}. Since $\overline{\Pi} = (\Pi, p, q)$ is a polynomial-time version of Π, the length of a run of Π on a structure $\mathfrak{A} = (A, \tau)$ is bounded by $p(|A|)$, and the size of each state of the run is bounded by $q(|A|)$. So clearly the length of the run of Π_{CPT} on $\mathfrak{A}$ is bounded by $p(|A|)$.

Now consider the number of active objects. The non-trivial critical objects are the sets that are in Dom and in R for a predicate $R \in \sigma$ in some state. As an object can only be in R if it is a tuple of fixed length of objects in Dom, the number of objects activated by each dynamic predicate R is polynomial in the number of objects activated by Dom, so it suffices to show that this number is bounded polynomially.

Claim 40. *The number of objects activated by* Dom *in the nth state of a run is bounded by* $(n-1) \cdot c \cdot q(|A|)^\ell + n \cdot q(|A|) + |A|^k \cdot d$*, where k is the dimension of I_{init}, ℓ is the dimension of I_{step}, and c and d are constants.*

Proof of the claim. After the initialisation step, the value of Dom is a set of $|A_1|$ many sets of k-tuples of atoms, where $\mathfrak{A}_1 = (A_1, \sigma)$ is the successor of the initial state in the run of Π on $\mathfrak{A}$. So the transitive closure of that set consists of the elements of the set Atoms, all k-tuples of atoms, and the $|A_1|$ $\approx$-equivalence classes.

Let d be the number of sets necessary to encode a k-tuple. Then the number of objects activated by Dom is $|A_1| + |A|^k \cdot d \leq q(|A|) + |A|^k \cdot d$.

Assume that in the nth state, there are at most $(n-1) \cdot c \cdot q(|A|)^\ell + n \cdot q(|A|) + |A|^k \cdot d$ objects activated by Dom. Afer $n+1$ steps, Dom consists of $|A_{n+1}|$ many sets of ℓ-tuples of elements that were in Dom in the previous step. The objects in the transitive closure of Dom are these $|A_{n+1}|$ sets, and the sets encoding the $|A_n|^\ell$ many tuples. Note that the transitive closure of any element of these tuples consists of previously activated objects. So the number of objects activated in step $n+1$ is $|A_{n+1}| + |A_n|^\ell \cdot c$, where c is the number of sets necessary to encode an ℓ-tuple. Note that $|A_{n+1}| + |A_n|^\ell \cdot c \leq q(|A|) + q(|A|)^\ell \cdot c$

So the total number of objects activated by Dom after $n+1$ steps is bounded by

$$\begin{aligned} & \big(q(|A|) + q(|A|)^\ell \cdot c\big) + \Big((n-1) \cdot q(|A|)^\ell \cdot c + n \cdot q(|A|) + |A|^k \cdot d\Big) \\ = \; & n \cdot q(|A|)^\ell \cdot c + (n+1) \cdot q(|A|) + |A|^k \cdot d \quad . \end{aligned}$$

□

Since the number n of states is clearly bounded by $p(|A|)$, this shows that there is indeed a polynomial-time version of Π_{CPT}, which completes the proof of the lemma.

□

So there is an equivalent CPT program for each PIL program, and an equivalent $\mathrm{CPT} + \mathrm{EqCard}$ program for each $\mathrm{PIL} + \mathrm{H}$ program.

The other direction of Theorem 34 is also shown simultaneously for CPT and $\mathrm{CPT} + \mathrm{EqCard}$:

Lemma 41.

1. $\mathrm{CPT} \leq \mathrm{PIL}$

2. $\mathrm{CPT} + \mathrm{EqCard} \leq \mathrm{PIL} + \mathrm{H}$

Since the proofs for both parts of the lemma differ only in the simulation of the equicardinality function, the following lemmas cover the respective logic with or without an equicardinality operation whenever possible.

To show that every CPT program can be simulated by a PIL program, we will first construct subprograms for the terms that occur in the computation of the CPT program. As shown in Section 4.1, PIL programs can be combined using concatenation and iteration, therefore it is possible to construct the program from smaller subprograms. In the following, some of the subprograms of the programs simulating BGS terms are introduced. To motivate the following constructions, we briefly explain the idea behind the simulation.

The proof uses Rossman's definition of BGS logic and CPT. Recall that, according to that definition, a BGS program is a tuple $(\Pi_{\mathrm{step}}, \Pi_{\mathrm{halt}}, \Pi_{\mathrm{out}})$ of BGS terms. So an equivalent PIL program has to represent these terms in some way. We will show that each BGS (resp. $\mathrm{BGS} + \mathrm{EqCard}$) term can be simulated by a PIL (resp. $\mathrm{PIL} + \mathrm{H}$) program.

A PIL program simulating a term enriches the domain of the input structure $\mathfrak{A}$ by all objects in $\mathrm{HF}(\mathfrak{A})$ necessary to compute the term, i.e. its active objects. A predicate In will then be used to represent the set structure of these objects. So a program simulating a BGS term initialises this and other necessary predicates and then successively creates objects representing the sets defined by subterms (this will be achieved by constructing the program inductively). Therefore, the PIL programs adding definable objects to the domain that are introduced in Section 4.1 are an important part of this proof.

To make the simulation of terms possible, the first subprogram initialises the domain of a structure representing a substructure of $\mathrm{HF}(\mathfrak{A})$ for the input structure $\mathfrak{A} = (A, \tau)$. That substructure needs to exhibit objects representing the elements in A, as well as objects representing the set of all atoms and the empty set, and a predicate In that defines the set structure of the current subset of $\mathrm{HF}(A)$. This initialisation step is defined as follows:

Lemma 42. *For any relational signature τ, there is a* $\mathrm{PIL}[\tau, \sigma]$ *program for* $\sigma = \tau \dot\cup \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$ *such that, in the final state* $\mathfrak{A}_{\mathit{fin}}$ *of the run on any structure* $\mathfrak{A} = (A, \tau)$*, there are objects* a*,* $\emptyset$ *and* $\{\emptyset\}$ *such that* $\mathrm{Atoms}^{\mathfrak{A}_{\mathit{fin}}} = \{a\}$*,* $\mathrm{Empty}^{\mathfrak{A}_{\mathit{fin}}} = \{\emptyset\}$*, and* $\mathrm{In}^{\mathfrak{A}_{\mathit{fin}}} = \{(x, a) \mid x \in A\} \cup \{(\emptyset, \{\emptyset\})\}$*.*

Proof. Let Π_1 be an $\mathrm{IL}[\tau, \tau \dot\cup \{\mathrm{Atoms}, \mathrm{In}\}]$ program that creates a new object a defined by a relation Atoms as demonstrated in Lemma 26, extended by the formula $\varphi_{\mathrm{In}}(x, y) = \mathrm{Atoms}\, y \wedge \neg\, \mathrm{Atoms}\, x$, and let Π_2 be an $\mathrm{IL}[\tau', \tau' \dot\cup \{\mathrm{Empty}\}]$ program (where $\tau' = \tau \cup \{\mathrm{Atoms}, \mathrm{In}\}$) that creates an element $\emptyset$ defined by a relation Empty, again according to Lemma 26. Let Π_3 be an IL program that creates another element $\{\emptyset\}$ which is defined by the relation R, and that updates In with the formula

$$\varphi_{\mathrm{In}}(x, y) = \mathrm{In}\, xy \vee (\mathrm{Empty}\, x \wedge Ry) \ .$$

Then $\Pi_1 \circ \Pi_2 \circ \Pi_3$ is the desired program. By Remark 28, there are polynomial-time versions of Π_1, Π_2 and Π_3, so, by Remark 32, there is a polynomial-time version of $\Pi_1 \circ \Pi_2 \circ \Pi_3$. □

Since Atoms and Empty are defined by relations in our IL program, let $\mathrm{HF}_{\mathrm{rel}}(\mathfrak{A})$ be the structure that is defined like $\mathrm{HF}(\mathfrak{A})$ except for the following modification: In the signature of $\mathrm{HF}_{\mathrm{rel}}(\mathfrak{A})$, Empty and Atoms are unary relation symbols, and $\mathrm{Empty}^{\mathrm{HF}_{\mathrm{rel}}(\mathfrak{A})} = \{\emptyset\}$ and $\mathrm{Atoms}^{\mathrm{HF}_{\mathrm{rel}}(\mathfrak{A})} = \{A\}$. This definition will simplify the notation in the following definitions.

When a BGS term is computed within the run of a program, it is guaranteed that in all terms evaluated during that run, the number of active objects is polynomially bounded. Recall that the set of active objects depends on the values of the free variables of a term. So the bound holds for any values that may be assigned to the free variables during any run of the program. However, this does not imply any bound on the number of active objects for variable assignments that never occur during a run of the program. Therefore, the set of elements that can be replaced for the free variables during the computation will be defined by a new relation.

So an input structure for a program simulating a term has to exhibit such a relation restricting the values of the free variables in a way that the number of active objects is bounded. Furthermore, an input structure has to resemble the state of a run of a CPT or CPT + EqCard program. These properties are formalised in the following definition.

Definition 43. *Let t be a term over $\tau^{\mathrm{HF}}_{\mathrm{EqCard}}$ with k free variables, let $\mathfrak{A} = (A, \tau)$ be a structure and let $q : \mathbb{N} \mapsto \mathbb{N}$. Then the structure $\mathfrak{B} = (B, \tau')$ is a $(t, \mathfrak{A}, q)$-state if*

1. *$\tau \cup \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}, V\} \subseteq \tau'$, where* Atoms *and* Empty *are unary predicates,* In *is a 2-ary predicate and V is a k-ary predicate,*
2. *$\mathfrak{B} \restriction \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$ is isomorphic to a substructure $\mathfrak{H}$ of the reduct $\mathrm{HF}_{rel}(\mathfrak{A}) \restriction \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$ containing the element $\{\emptyset\}$,*
3. *the domain H of $\mathfrak{H}$ is a transitive set,*
4. *$\left|H \cup \bigcup\{\langle\!\langle t(b_1, \ldots, b_k)\rangle\!\rangle \mid (b_1, \ldots, b_k) \in V^{\mathfrak{B}}\}\right| \leq q(|A|)$.*

We also say that $\mathfrak{B}$ is a $(t, \mathfrak{A}, q)$-state with respect to V.

The notion of a $(t, \mathfrak{A}, q)$-state describes the structures that occur as final states of the computation of a BGS(+ EqCard) term (the non-final states may, for instance, not define the predicate In for all sets). However, the main property of such a final state is that the domain contains an object representing the set defined by the given term, and that there is a relation defining the output of a term for any possible assignment of the free variables. These properties are now formalised in the definition of simulation.

Definition 44. *Let t be a* BGS *term over $\tau^{\mathrm{HF}}_{ext} \subseteq \tau^{\mathrm{HF}} \cup \{\mathrm{EqCard}\}$ with k free variables for a relational signature τ, and let $q : \mathbb{N} \mapsto \mathbb{N}$ be a polynomial. Furthermore, let $\tau' \supseteq \tau \cup \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}, V\}$. An $\mathrm{IL}[\tau', \sigma]$ program Π* simulates *t if $\sigma \supseteq \tau' \cup \{R_t\}$ for a $(k+1)$-ary relation symbol $R_t \notin \tau'$, and in the final state $\mathfrak{A}_{fin} = (A_{fin}, \sigma)$ of the run of Π on any $(t, \mathfrak{A}, q)$-state $\mathfrak{B}$ for any finite τ-structure $\mathfrak{A}$ the following holds:*

- *$\mathfrak{A}_{fin}$ is a $(t, \mathfrak{A}, q)$-state, where π is the isomorphism from the reduct $\mathfrak{A}_{fin} \restriction \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$ to a substructure of the $\{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$-reduct of $\mathrm{HF}_{rel}(\mathfrak{A})$ (such an isomorphism exists by definition of $(t, \mathfrak{A}, q)$-states).*

- *Let B be the domain of $\mathfrak{B}$, and let $\pi_{\mathfrak{B}}$ be the isomorphism from $\mathfrak{B}$ to a substructure of $\mathrm{HF}_{rel}(\mathfrak{A}) \restriction \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$. Then for each $b \in B$ there is an object $a_b \in A_{fin}$ that encodes b, i.e. $\pi(a_b) = \pi_{\mathfrak{B}}(b)$, and $(b_1, \dots, b_k) \in V^{\mathfrak{B}}$ if and only if $(a_{b_1}, \dots, a_{b_k}) \in V^{\mathfrak{A}_{fin}}$.*

- *For each $\overline{v} = (v_1, \dots, v_k) \in V^{\mathfrak{B}}$, there is an object $a_{t,\overline{v}} \in A_{fin}$ encoding $[\![t(\overline{v})]\!]$, i.e. $\pi(a_{t,\overline{v}}) = \pi_{\mathfrak{B}}([\![t(\overline{v})]\!])$.*

- *All new objects are active objects of t: If $a \in \pi(A_{fin})$, then $a \in \pi_{\mathfrak{B}}(B)$ or $a \in \langle\!\langle t(\overline{v}) \rangle\!\rangle$ for some $\overline{v}$ encoded by a tuple in $V^{\mathfrak{B}}$.*

- *R_t associates each tuple $\overline{v}$ with the object representing $[\![t(\overline{v})]\!]$: $R_t^{\mathfrak{A}_{fin}} = \{(\pi^{-1}(\overline{v}), a_{t,\overline{v}}) \mid \overline{v} \in V^{\mathfrak{B}}\}$.*

A $\mathrm{PIL}[\tau', \sigma]$ *program $\overline{\Pi}$ simulates the term t if $\overline{\Pi}$ is a polynomial-time version of an* $\mathrm{IL}[\tau', \sigma]$ *program simulating t.*

Now that the notion of simulation is introduced, we can show that PIL programs can simulate BGS and BGS + EqCard terms.

Lemma 45. *For any* BGS *term t, there is a* PIL *program $\overline{\Pi}_t = (\Pi_t, q_t)$ that simulates t.*

Proof. The lemma is shown by induction on BGS terms.

Induction Base

1. $t = x$, where x is a variable: Let I be an interpretation that preserves the domain and the relations in τ', and that defines R_t with
$$\varphi_{R_t}(x, y) = x = y \wedge Vx \ .$$
Let $\overline{\Pi}_t$ be a polynomial-time version of the IL program Π_I applying I obtained from Remark 28. Clearly, $I(\mathfrak{B})$ is still a $(t, \mathfrak{A}, q)$-state with all required elements in the domain, and R_t fulfills the required property by definition.

2. $t = c$ for a constant symbol c: Since τ is a relational signature, $c \in \{\emptyset, \mathrm{Atoms}\}$. Since $\mathfrak{B}$ is a $(t, \mathfrak{A}, q)$-state, there are objects $\emptyset, a \in B$ and relation symbols $\mathrm{Empty}, \mathrm{Atoms} \in \tau'$ such that $\mathrm{Empty}^{\mathfrak{B}} = \{\emptyset\}$ and $\mathrm{Atoms}^{\mathfrak{B}} = \{a\}$. Choose Π_t as in Case 1, but set $\varphi_{R_t}(x) = \mathrm{Empty}\, x$ or $\varphi_{R_t}(x) = \mathrm{Atoms}\, x$, respectively.

Induction Step

1. $t(\overline{x}) = f(t_1(\overline{x}), \dots, t_r(\overline{x}))$ for terms $t_1, \dots, t_r$. Since $\langle\!\langle t_i(\overline{a}) \rangle\!\rangle \subseteq \langle\!\langle t(\overline{a}) \rangle\!\rangle$ for $1 \leq i \leq r$ and any assignment of variables $\overline{x}$ to values $\overline{a}$ in $\mathrm{HF}(A)$ for an input structure $\mathfrak{A} = (A, \tau)$, $|\langle\!\langle t(\overline{a}) \rangle\!\rangle| \leq q(|A|)$ implies that $|\langle\!\langle t_i(\overline{a}) \rangle\!\rangle| \leq q(|A|)$ for any polynomial q. So any $(t, \mathfrak{A}, q)$-state is also a $(t_i, \mathfrak{A}, q)$-state for $1 \leq i \leq r$. So, by the induction hypothesis, there are PIL programs $\overline{\Pi}_{t_1}, \dots, \overline{\Pi}_{t_r}$ that simulate $t_1, \dots, t_r$.

   τ is relational, so $f \in \{\mathrm{Pair}, \mathrm{Union}, \mathrm{TheUnique}\}$.

   (a) $t(\overline{x}) = \mathrm{Pair}(t_1(\overline{x}), t_2(\overline{x}))$: By the above observation, there is an $\mathrm{IL}[\tau', \tau'']$ program Π_{t_1} simulating t_1 and an $\mathrm{IL}[\tau'', \sigma]$ program Π_{t_2} simulating t_2. In particular, the final state of the run of Π_{t_1} or Π_{t_2} on any $(t, \mathfrak{A}, q)$-state is again a $(t, \mathfrak{A}, q)$-state, since V is not changed and the other properties of a $(t, \mathfrak{A}, q)$-state are independent of t. Thus the final state of the run of $\Pi_{t_1} \circ \Pi_{t_2}$ is again a $(t, \mathfrak{A}, q)$-state, and all objects added to the domain by $\Pi_{t_1} \circ \Pi_{t_2}$ represent active objects of t_1 or t_2.

   Now let $I = \Big(\varphi_{\mathrm{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \sigma \dot\cup \{R_t\}}\Big)$ be an interpretation that adds an element $a_{t,\overline{v}}$ for each tuple $\overline{v}$ in V (use Lemma 27 with the formula $Vx_1 \dots x_k$), where R_t is the relation symbol introduced by I. Let I' be an interpretation that preserves the domain and the relations except for In, which is defined by the formula

$$\begin{aligned}\varphi_{\mathrm{In}}(x, y) = \mathrm{In}\, xy \vee \exists x_1 \dots \exists x_k \,((R_{t_1} x_1 \dots x_k x \vee R_{t_2} x_1 \dots x_k x) \\ \wedge\, R_t x_1 \dots x_k y) \quad .\end{aligned}$$

   Let Π_I and $\Pi_{I'}$ be the IL programs that apply I and I', respectively. Then the final state of a run of $\Pi_{t_1} \circ \Pi_{t_2} \circ \Pi_I \circ \Pi_{I'}$ contains all elements required by the definition of simulation, all objects added to the domain represent objects in $\langle\!\langle t(\overline{v}) \rangle\!\rangle$ for values $\overline{v}$ in V of the free variables, and R_t and In are defined accordingly. However, the domain may still contain several objects of the form $a_{t,\overline{v}}$ that encode the same set.

   Let $I_{\mathrm{state}} = \Big(\varphi_{\mathrm{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \sigma \dot\cup \{R_t\}}\Big)$, where $\varphi_{\mathrm{dom}}(x)$ is a tautology, $\varphi_R(x_1, \dots, x_r) = Rx_1 \dots x_r$ for any r-ary relation symbol $R \in \sigma \dot\cup \{R_t\}$ and

$$\varphi_{\approx}(x, y) = \forall z\, (\mathrm{In}(z, x) \leftrightarrow \mathrm{In}(z, y)) \quad .$$

Let $\mathfrak{A}'_{\text{fin}}$ be the final state of the run of $\Pi_{t_1} \circ \Pi_{t_2}$ on some input structure, and let $\mathfrak{A}''_{\text{fin}}$ be the final state of the run of $\Pi_I \circ \Pi_{I'}$ on $\mathfrak{A}'_{\text{fin}}$. Note that any set encoded by several objects in $\mathfrak{A}''_{\text{fin}}$ has only elements that are already encoded by elements of the domain of $\mathfrak{A}'_{\text{fin}}$, which is isomorphic to a substructure of $\text{HF}_{\text{rel}}(\mathfrak{A}) \restriction \{\text{Atoms}, \text{Empty}, \text{In}\}$. Therefore I_{state} indeed suffices to ensure that all these sets are encoded by only a single element each. So $\mathfrak{A}''_{\text{fin}} \restriction \{\text{Atoms}, \text{Empty}, \text{In}\}$ is yet again isomorphic to a substructure of $\text{HF}_{\text{rel}}(\mathfrak{A}) \restriction \{\text{Atoms}, \text{Empty}, \text{In}\}$ which contains only elements that already occur in $\mathfrak{A}'_{\text{fin}}$ or that encode $[\![t(\overline{v})]\!]$ for some $\overline{v} \in V^{\mathfrak{A}'_{\text{fin}}}$. Thus, by Property 4 of $(t, \mathfrak{A}, q)$-states, the number of elements in the domain of $\mathfrak{A}''_{\text{fin}}$ is bounded by $q(|A|)$, and $\mathfrak{A}''_{\text{fin}}$ is a $(t, \mathfrak{A}, q)$-state.
So let $\Pi_t = \Pi_{t_1} \circ \Pi_{t_2} \circ \Pi_I \circ \Pi_{I'} \circ \Pi_{I_{\text{state}}}$.
By definition, Π_t simulates t. As the lemma requires a PIL program, it remains to show that there is a polynomial-time version of Π_t. By induction hypothesis, there are polynomial-time versions of Π_{t_1} and Π_{t_2}. By Remark 28, there are also polynomial-time versions of Π_I, $\Pi_{I'}$ and Π_{state}. So by Remark 32, there is a polynomial-time version of Π_t.

(b) $t(\overline{x}) = \text{Union}(t_1(\overline{x}))$: Let $\Pi_t = \Pi_{t_1} \circ \Pi_I \circ \Pi_{I'} \circ \Pi_{I_{\text{state}}}$, where Π_{t_1} is an $\text{IL}[\tau', \sigma]$ program for t_1 obtained from the induction hypothesis, I and I_{state} are defined as in Case 1a, and $I' = \left(\varphi'_{\text{dom}}, \varphi'_{\approx}, (\varphi'_R)_{R \in \sigma'}\right)$ is the interpretation with $\sigma' = \sigma \dot\cup \{R_t\}$ that preserves the domain and the relations in $\sigma' \setminus \{\text{In}\}$, and defines In with the formula

$$\varphi_{\text{In}}(x, y) = \text{In}\, xy \vee \exists x_1 \ldots \exists x_k \, (R_t x_1 \ldots x_k y \wedge \\ \exists x_{t_1} \exists z \, (R_{t_1} x_1 \ldots x_k x_{t_1} \wedge \text{In}\, xz \wedge \text{In}\, z x_{t_1})) \quad .$$

With the same reasoning as in Case 1a, Π_t simulates t and there is a polynomial-time version of Π_t.

(c) $t(\overline{x}) = \text{TheUnique}(t_1(\overline{x}))$: Let $\Pi_t = \Pi_{t_1} \circ \Pi_I$ where Π_{t_1} is an $\text{IL}[\tau', \sigma]$ program defined as in the previous cases and Π_I is the IL program applying the interpretation $I = \left(\varphi_{\text{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \sigma \dot\cup \{R_t\}}\right)$ that preserves the domain and any r-ary relation $R \in \sigma$, and R_t is defined with respect to the fact that $[\![t(\overline{v})]\!]$ is the empty set if $[\![t_1(\overline{v})]\!]$ is not a singleton and $[\![t(\overline{v})]\!] = a$ if $[\![t_1(\overline{v})]\!] = \{a\}$:

$$\begin{aligned}\varphi_{R_t}(x_1,\dots,x_k,y) =& \exists x_{t_1}\,(R_{t_1}x_1\dots x_k x_{t_1} \wedge (\text{Empty}\, y \wedge \\ &\neg\exists y'(\text{In}\, y'x \wedge \forall z(\text{In}\, z x_{t_1} \to z = y'))) \vee \\ &(\text{In}\, y x_{t_1} \wedge \forall z(\text{In}\, z x_{t_1} \to z = y))) \quad .\end{aligned}$$

Since the final state $\mathfrak{A}_{\text{fin}}$ of the run of Π_{t_1} on a $(t,\mathfrak{A},q)$-state is again a $(t,\mathfrak{A},q)$-state, the domain A_{fin} of $\mathfrak{A}_{\text{fin}}$ encodes a transitive subset of $\text{HF}(A)$, so there is an object encoding $[\![t(\overline{v})]\!]$ in A_{fin} for each $\overline{v}$ in V. So I suffices to define R_t as required. Again, there are polynomial-time versions of Π_{t_1} and Π_I, so there is a polynomial-time version of Π_t.

2. $t(\overline{x}) = R(t_1(\overline{x}),\dots,t_r(\overline{x}))$: As in Case 1, there are programs $\Pi_{t_1},\dots,\Pi_{t_r}$ simulating $t_1,\dots,t_r$, and the final state of the run of $\Pi_{t_1}\circ\dots\circ\Pi_{t_r}$ on a $(t,\mathfrak{A},q)$-state is again a $(t,\mathfrak{A},q)$-state. Let $\Pi_t = \Pi_{t_1}\circ\dots\circ\Pi_{t_r}\circ\Pi_I$, where $I = \left(\varphi_{\text{dom}},\varphi_{\approx},(\varphi_R)_{R\in\sigma\dot\cup R_\varphi}\right)$ (σ is chosen such that $\Pi_{t_1}\circ\dots\Pi_{t_r}$ is a $\text{IL}[\tau',\sigma]$ program) is an interpretation that preserves the domain and all relations in σ. For defining φ_{R_t}, let φ_t be the formula

$$\varphi_t(x_1,\dots,x_k) = \exists x_{t_1}\dots\exists x_{t_r}\left(\bigwedge_{1\le i\le r} R_{t_i}x_1\dots x_k x_{t_i} \wedge R x_{t_1}\dots x_{t_r}\right) \quad ,$$

which is true if and only if the value of t is true (which is encoded by the value $\{\emptyset\}$). Then let

$$\begin{aligned}\varphi_{R_t}(x_1,\dots,x_k,y) =& (\neg\varphi_t \wedge \text{Empty}\, y) \vee (\varphi_t \wedge \exists z(\text{Empty}\, z \wedge \\ &\text{In}(z,y) \wedge \forall z'(\text{In}(z',y) \to z' = z))) \quad .\end{aligned}$$

So φ_{R_t} sets the value of t to $\{\emptyset\}$ if it evaluates to true, and to $\emptyset$ otherwise.

3. $t(\overline{x}) = t_1(\overline{x}) = t_2(\overline{x})$: Let $\Pi_\varphi = \Pi_{t_1}\circ\Pi_{t_2}\circ\Pi_I$, where Π_{t_1} and Π_{t_2} are defined as in the previous cases, choose σ such that $\Pi_{t_1}\circ\Pi_{t_2}$ is an $\text{IL}[\tau',\sigma]$ program, and let $I = \left(\varphi_{\text{dom}},\varphi_{\approx},(\varphi_R)_{R\in\sigma}\right)$ be the interpretation defined as in the previous case, with the modification that

$$\varphi_t(x_1,\dots,x_k) = \forall y\,(R_{t_1}x_1\dots x_k y \leftrightarrow R_{t_2}x_1\dots x_k y) \quad .$$

4. $t = \neg t_1$ or $t = t_1 \wedge t_2$ or $t_1 \vee t_2$: Analogously.

5. $t(\overline{x}) = \{s(v,\overline{x}) : v \in t_1(\overline{x}) : \varphi(v,\overline{x})\}$: Since $\langle\!\langle t_1(\overline{a})\rangle\!\rangle \subseteq \langle\!\langle t(\overline{a})\rangle\!\rangle$ for any values $\overline{a}$, any $(t,\mathfrak{A},q)$-state is also a $(t_1,\mathfrak{A},q)$-state. So, by induction hypothesis, there is an $\mathrm{IL}[\tau',\tau_{t_1}]$ program Π_{t_1} that simulates t_1, especially for input structures that are $(t,\mathfrak{A},q)$-states.

   The relation R_{t_1} then defines the set of possible values for the free variables of φ during the computation of t, so the final state of the run of Π_{t_1} can be transformed into a $(\varphi,\mathfrak{A},q)$-state: Consider the interpretation $I_{V'} = \left(\varphi^{V'}_{\mathrm{dom}}, \varphi^{V'}_{\approx}, \left(\varphi^{V'}_{R}\right)_{R\in\tau_{t_1}\dot\cup\{V'\}}\right)$ where the domain and the relations in τ_{t_1} are preserved, and

$$\varphi^{V'}_{V'}(v,x_1,\dots,x_k) = \exists x_{t_1} R_{t_1} x_1 \dots x_k x_{t_1} \wedge \mathrm{In}\, v x_{t_1} \ ,$$

   and $\Pi_{V'}$ is the IL program that applies $I_{V'}$.

   Let $\mathfrak{A}^{t_1}_{\mathrm{fin}}$ be the final state of a run of Π_{t_1} on a $(t,\mathfrak{A},q)$-state. Then, since $\bigcup_{b\in[\![t_1(\overline{a})]\!]}\langle\!\langle \varphi(b,\overline{a})\rangle\!\rangle \subseteq \langle\!\langle t(\overline{a})\rangle\!\rangle$ for each $\overline{a}\in V^{\mathfrak{B}}$ where $\mathfrak{B}$ is a $(t,\mathfrak{A},q)$-state, $\mathfrak{A}^{t_1}_{\mathrm{fin}}$ is a $(\varphi,\mathfrak{A},q)$-state with respect to V'. Furthermore, it holds that $\bigcup_{b\in[\![t_1(\overline{a})]\!]}\langle\!\langle s(b,\overline{a})\rangle\!\rangle \subseteq \langle\!\langle t(\overline{a})\rangle\!\rangle$ for any tuple $\overline{a}$, so the final state of a run of $\Pi_{t_1}\circ\Pi_{V'}\circ\Pi_{\varphi}$ is also an $(s,\mathfrak{A},q)$-state.

   So by induction hypothesis, there are programs Π_φ and Π_s that simulate φ and s, respectively, also on any $(t,\mathfrak{A},q)$-state, and, since V is not changed, the final state of any run of Π_φ or Π_s on a $(t,\mathfrak{A},q)$-state is again a $(t,\mathfrak{A},q)$-state.

   Now let $I = \left(\varphi_{\mathrm{dom}}, \varphi_{\approx}, (\varphi_R)_{R\in\tau_s\dot\cup\{R_t\}}\right)$, where τ_s is the signature such that $\Pi_\varphi\circ\Pi_s$ is an $\mathrm{IL}[\tau_{t_1},\tau_s]$ program, be an interpretation that creates an object for each tuple $\overline{v}$ in V according to Lemma 27, and R_t is the predicate defining that object. Then let $I' = \left(\varphi'_{\mathrm{dom}}, \varphi'_{\approx}, (\varphi'_R)_{R\in\tau_s\dot\cup\{R_t\}}\right)$ be the interpretation that preserves the domain and all relations except for In, and updates In as follows:

$$\begin{aligned}\varphi_{\mathrm{In}}(x,y) = \mathrm{In}\, xy \ \vee\ \exists x_1\dots\exists x_k(&R_t x_1\dots x_k y \wedge \exists v(R_s v x_1\dots x_k x \wedge \\ &\exists x_{t_1} R_{t_1} x_1\dots x_k x_{t_1} \wedge \mathrm{In}\, v x_{t_1} \\ &\wedge R_\varphi v x_1\dots x_k)))\ .\end{aligned}$$

   So I' adds to In all pairs (x,y) where y represents $[\![t(\overline{a})]\!]$ for some $\overline{a}$ and x represents $[\![s(b,\overline{a})]\!]$ for some $b\in[\![t_1(\overline{a})]\!]$ such that $[\![\varphi(b,\overline{a})]\!]$ is true, i.e. $x\in[\![t(\overline{a})]\!]$.

Define $\Pi_t = \Pi_{t_1} \circ \Pi_{V'} \circ \Pi_\varphi \circ \Pi_s \circ \Pi_I \circ \Pi_{I'} \circ \Pi_{I_{\text{state}}}$, where I_{state} is defined as in Case 1a. As observed above, Π_{t_1} computes a predicate for t_1, $\Pi_{V'}$ defines the predicate V' such that the resulting structure is a $(\varphi, \mathfrak{A}, q)$-state and an $(s, \mathfrak{A}, q)$-state with respect to V', Π_φ and Π_s simulate φ and s, Π_I creates a definable object $a_{t,\overline{v}}$ for each $\overline{v}$ in V, $\Pi_{I'}$ defines the predicate In for the new objects and $\Pi_{I_{\text{state}}}$ ensures the property that a reduct of the resulting final state is isomorphic to a substructure of $\mathrm{HF}_{\mathrm{rel}}(\mathfrak{A}) \upharpoonright \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$. Since only active objects of $t(\overline{v})$ for $\overline{v}$ in V are added to the domain and V is not changed, the final state of any run of Π_t on a $(t, \mathfrak{A}, q)$-state is a $(t, \mathfrak{A}, q)$-state.

Again, there are polynomial-time versions of all IL programs used above, so there is also a polynomial-time version of Π_t.

By induction, for any BGS term t there is a PIL program simulating t. $\square$

For the second part of Lemma 41, terms over the signature extended by EqCard are simulated by $\mathrm{PIL}+\mathrm{H}$ programs.

Lemma 46. *For any* BGS *term t over $\sigma^{\mathrm{HF}}_{\mathrm{EqCard}}$, there is a* $\mathrm{PIL}+\mathrm{H}$ *program $\overline{\Pi}_t$ simulating t.*

Proof. Analogous to the proof of Lemma 45. In the induction step, we add the following case:

$t(\overline{x}) = \mathrm{EqCard}(t_1(\overline{x}), t_2(\overline{x}))$: As in the proof of Lemma 45, let Π_{t_1} be an $\mathrm{IL}[\tau', \tau'']$ program simulating t_1, and let Π_{t_2} be an $\mathrm{IL}[\tau'', \sigma]$ program simulating t_2. Let Π_I be the program that applies the interpretation $I = \big(\varphi_{\mathrm{dom}}, \varphi_{\approx}, (\varphi_R)_{R \in \sigma'}\big)$ that is defined as in Case 2 of the proof of Lemma 45, with the modification that

$$\begin{aligned}\varphi_t(x_1, \dots, x_k) = & \exists x_{t_1} \exists x_{t_2} \big(R_{t_1} x_1 \dots x_k x_{t_1} \land R_{t_2} x_1 \dots x_k x_{t_2} \land \\ & \mathrm{H}\, y_1 y_2\, \mathrm{In}\, y_1 x_{t_1}\, \mathrm{In}\, y_2 x_{t_2}\big) \ .\end{aligned}$$

Then $\Pi_t = \Pi_{t_1} \circ \Pi_{t_2} \circ \Pi_I$ simulates t. $\square$

Given programs in interpretation logic for BGS and $\mathrm{BGS}+\mathrm{EqCard}$ terms, it is now possible to define an equivalent PIL (resp. $\mathrm{PIL}+\mathrm{H}$) program for each CPT (resp. $\mathrm{CPT}+\mathrm{EqCard}$) program.

Proof of Lemma 41. We only show the lemma explicitly for CPT, because the proof for CPT + EqCard is completely analogous (note that, since every PIL program is also a PIL + H program, the proof for CPT + EqCard uses the same subprograms).

Let $\overline{\Pi}_{\mathrm{CPT}} = (\Pi_{\mathrm{CPT}}, q)$, where $\Pi_{\mathrm{CPT}} = (\Pi_{\mathrm{step}}, \Pi_{\mathrm{halt}}, \Pi_{\mathrm{out}})$ and q is a polynomial, be a CPT program. We construct an equivalent PIL program that works as follows:

By Lemma 42, there is an IL program that initialises objects representing the sets Atoms, $\emptyset$ and $\{\emptyset\}$. First run that program, and then run another IL program that initialises a unary predicate Cur with $\varphi_{\mathrm{Cur}}(x) = \mathrm{Empty}\, x$. Let Π_{init} be the concatenation of these IL programs. By Lemma 42 and Remark 28, there are polynomial-time versions of both programs, so there is a polynomial-time version of Π_{init}.

Let $\mathfrak{A}^{\mathrm{init}}_{\mathrm{fin}}$ be the final state of the run of Π_{init} on a structure $\mathfrak{A} = (A, \tau)$. By definition of Π_{init}, $\mathfrak{A}^{\mathrm{init}}_{\mathrm{fin}} \restriction \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$ is isomorphic to a substructure of $\mathrm{HF}_{\mathrm{rel}}(\mathfrak{A}) \restriction \{\mathrm{Atoms}, \mathrm{Empty}, \mathrm{In}\}$.

Since the number of active objects in the run of $(\Pi_{\mathrm{step}}, \Pi_{\mathrm{halt}}, \Pi_{\mathrm{out}})$ on $\mathfrak{A}$ is bounded by $q(|A|)$, the size of $\langle\!\langle \Pi_{\mathrm{step}}(\emptyset) \rangle\!\rangle$ and $\langle\!\langle \Pi_{\mathrm{halt}}(\emptyset) \rangle\!\rangle$ is bounded by $q(|A|)$. So $\mathfrak{A}^{\mathrm{init}}_{\mathrm{fin}}$ is a $(\Pi_{\mathrm{step}}, \mathfrak{A}, q)$-state and a $(\Pi_{\mathrm{halt}}, \mathfrak{A}, q)$-state with respect to the predicate Cur.

Then, by Lemma 45, there are IL programs $\Pi^{\mathrm{IL}}_{\mathrm{step}}$ and $\Pi^{\mathrm{IL}}_{\mathrm{halt}}$ that simulate Π_{step} and Π_{halt}, respectively, on the final state of any run of Π_{init}. Let R_{step} (resp. R_{Halt}) be the predicate for Π_{step} (resp. Π_{halt}) computed by $\Pi^{\mathrm{IL}}_{\mathrm{step}}$ (resp. $\Pi^{\mathrm{IL}}_{\mathrm{halt}}$).

Let Π_{cur} be an IL program that applies an interpretation that preserves the domain and the relations except for Cur, and that updates Cur by the formula

$$\varphi_{\mathrm{Cur}}(x) = \exists y\, (\mathrm{Cur}\, y \wedge R_{\mathrm{step}} yx) \quad .$$

Consider the program $\Pi = \Pi^{\mathrm{IL}}_{\mathrm{step}} \circ \Pi^{\mathrm{IL}}_{\mathrm{halt}} \circ \Pi_{\mathrm{cur}}$. Then the iteration Π^{ψ} for $\psi = \exists x (R_{\mathrm{Halt}} x \wedge \neg\, \mathrm{Empty}\, x)$ computes predicates for the terms Π_{step} and Π_{halt} and updates Cur accordingly until Π_{halt} is true.

By the definition of simulation, whenever $\Pi^{\mathrm{IL}}_{\mathrm{step}}$ is run on a $(\Pi_{\mathrm{step}}, \mathfrak{A}, q)$-state, the final state is again a $(\Pi_{\mathrm{step}}, \mathfrak{A}, q)$-state. Since Π_{cur} sets Cur to the set containing only the object representing $[\![\Pi_{\mathrm{step}}(a_i)]\!]$, where a_i is the object previously defined by Cur, the final state of the run of Π during the execution of $\Pi_{\mathrm{init}} \circ \Pi^{\psi}$ is again a $(\Pi_{\mathrm{step}}, \mathfrak{A}, q)$-state and a $(\Pi_{\mathrm{halt}}, \mathfrak{A}, q)$-state, given that $[\![\Pi_{\mathrm{halt}}(a_i)]\!]$ is false (note that the number of active objects of $\Pi_{\mathrm{step}}(\Pi_{\mathrm{step}}(a_i))$

and $\Pi_{\mathrm{halt}}(\Pi_{\mathrm{step}}(a_i))$ is bounded by $q(|A|))$.

Π simulates Π_{step}, and ψ states that Π_{halt} evaluates to true in the current state, so Π^{ψ} executes Π as many times as $\overline{\Pi}_{\mathrm{CPT}}$ computes Π_{step}. So, since by Lemma 13, the length of the run of $\overline{\Pi}_{\mathrm{CPT}}$ is bounded by q, Π is run at most $q(|A|)$ times. Let (Π, p', q') be a polynomial-time version of Π. Then the length of the run of Π^{ψ} on a structure $\mathfrak{B}$ with domain B that is a $(\Pi_{\mathrm{step}}, \mathfrak{A}, q)$-state and a $(\Pi_{\mathrm{halt}}, \mathfrak{A}, q)$-state is bounded by $q(|A|) \cdot p'(|B|)$.

Since the final state of the run of Π on such a structure $\mathfrak{B}$ is again a $(\Pi_{\mathrm{halt}}, \mathfrak{A}, q)$-state, the number of elements in the domain of the final state of each run of Π during the computation of Π^{ψ} is bounded by $q(|A|)$. So the number of elements of any state of the run of Π^{ψ} on the final state of a run of Π_{init} is bounded by $q'(q(|A|))$.

Thus there is a polynomial-time version of $\Pi_{\mathrm{init}} \circ \Pi^{\psi}$.

Consider the final state $\mathfrak{A}_{\mathrm{fin}}$ of a run of $\Pi_{\mathrm{init}} \circ \Pi^{\psi}$. The predicate Cur defines an object representing some a such that $\langle\!\langle \Pi_{\mathrm{out}}(a) \rangle\!\rangle$ is a subset of $\mathrm{Active}(\overline{\Pi}_{\mathrm{CPT}}, \mathfrak{A})$, and by Lemma 45 and the definition of simulation, each object in the domain of $\mathfrak{A}_{\mathrm{fin}}$ represents an object in $\mathrm{Active}(\overline{\Pi}_{\mathrm{CPT}}, \mathfrak{A})$. So $\mathfrak{A}_{\mathrm{fin}}$ is a $(\Pi_{\mathrm{out}}, \mathfrak{A}, q)$-state.

Let $\Pi^{\mathrm{IL}}_{\mathrm{out}} = \left(I^{\mathrm{out}}_{\mathrm{init}}, I^{\mathrm{out}}_{\mathrm{step}}, \varphi^{\mathrm{out}}_{\mathrm{halt}}, \varphi^{\mathrm{out}}_{\mathrm{out}}\right)$ be the IL program simulating Π_{out} obtained from Lemma 45. $\Pi^{\mathrm{IL}}_{\mathrm{out}}$ computes the output of Π_{CPT}, so we replace $\Pi^{\mathrm{IL}}_{\mathrm{out}}$ with ${\Pi^{\mathrm{IL}}_{\mathrm{out}}}' = (I^{\mathrm{out}}_{\mathrm{init}}, I^{\mathrm{out}}_{\mathrm{step}}, \varphi^{\mathrm{out}}_{\mathrm{halt}}, \varphi^{\mathrm{out}\prime}_{\mathrm{out}})$, where

$$\varphi^{\mathrm{out}\prime}_{\mathrm{out}} = \forall x \, (\mathrm{Cur}\, x \rightarrow \exists y (R_{\mathrm{out}} y \wedge \neg\, \mathrm{Empty}\, y)) \quad .$$

By Lemma 45, there is a polynomial-time version of $\Pi^{\mathrm{IL}}_{\mathrm{out}}$.

So, by definition, $\Pi_{\mathrm{init}} \circ \Pi^{\psi} \circ {\Pi^{\mathrm{IL}}_{\mathrm{out}}}'$ simulates the computation of the program $(\Pi_{\mathrm{step}}, \Pi_{\mathrm{halt}}, \Pi_{\mathrm{out}})$, and since there are polynomial-time versions of Π_{init}, Π^{ψ} and ${\Pi^{\mathrm{IL}}_{\mathrm{out}}}'$, there is a polynomial-time version of the concatenation.

Hence, for each CPT program, there is an equivalent PIL program, which shows Lemma 41. □

Lemmas 35 and 41 directly imply the equivalence of the logics PIL + H and CPT + EqCard (resp. PIL and CPT), so we have shown Theorem 34 which constitutes the main result of this thesis.

Corollary 47. $\mathrm{PIL} \not\equiv \mathrm{PIL} + \mathrm{H}$.

Proof. As shown in [BGS99], there is a query that is definable in CPT + C (and thus in CPT + EqCard) but not in CPT. By Theorem 34, this query is definable in PIL + H but not in PIL. □

So this chapter results in the classification of PIL and PIL + H with respect to the logics CPT and CPT + EqCard, where CPT + EqCard is equivalent to CPT + C and therefore an interesting candidate for a logic capturing PTIME.

Chapter 5

Conclusion and Future Work

This thesis extends the work on Choiceless Polynomial Time, which was introduced by Blass, Gurevich and Shelah ([BGS99]). Previous results suggest that CPT with counting is a reasonable candidate for a logic capturing large fragments of PTIME, or even PTIME itself. Queries that have previously been conjectured to separate $\mathrm{CPT}+\mathrm{C}$ from PTIME fail to do so, for instance the Cai-Fürer-Immermann query ([BGS02],[DRR08]), and perfect matching on bipartite ([BGS02]) and on general graphs ([ADH13]), and, like Turing machines, CPT benefits from padding of the input ([BGS99],[Lau11]). Therefore, the analysis of Choiceless Polynomial Time may lead to valuable insights concerning the logical characterisation of PTIME.

In this thesis, we introduced polynomial-time interpretation logic (PIL) as an alternative formulation of CPT based on iterated first-order interpretations. In Chapter 4, we showed that PIL is indeed equivalent to CPT.

To obtain an alternative definition of $\mathrm{CPT}+\mathrm{C}$ based on PIL, we first showed in Chapter 3 that, instead of counting operators, it suffices to extend CPT by an equicardinality quantifier. Analogously, also PIL can be equipped with an equicardinality operation in the form of the Härtig quantifier to accomplish equivalence to $\mathrm{CPT}+\mathrm{C}$ (as shown in Chapter 4).

These results are summarised in Figure 5.1.

Thus our work gives rise to a more concise definition of $\mathrm{CPT}+\mathrm{C}$ that is based on first-order logic and therefore has the potential to make $\mathrm{CPT}+\mathrm{C}$ more accessible for further investigation via classical methods of finite model theory.

Moreover, our work suggests various questions for future research. In order to develop a better understanding of the power of first-order interpre-

$$\begin{tikzcd}
\mathrm{PTIME} \arrow[d, phantom, "{\text{[BGS99],[BGS02]}}\ \supseteq"] & \\
\mathrm{CPT}+\mathrm{C} \arrow[d, phantom, "{\text{Thm. 21}}\ ="] & \\
\mathrm{CPT}+\mathrm{EqCard} \arrow[r, phantom, "\overset{\text{Thm. 34}}{=}"] \arrow[d, phantom, "{\text{[BGS99],[BGS02]}}\ \supsetneq"] & \mathrm{PIL}+\mathrm{H} \arrow[d, phantom, "{\text{Cor. 47}}\ \supsetneq"] \\
\mathrm{CPT} \arrow[r, phantom, "\overset{\text{Thm. 34}}{=}"] & \mathrm{PIL}
\end{tikzcd}$$

Figure 5.1: An overview of the classification of interpretation logic compared to CPT (each of the logics is identified with the class of queries definable in that logic).

tations, it is desirable to analyse how certain CPT-definable queries can be defined directly in PIL. The usage of the equicardinality operation, for instance for defining the parity of a set, without the explicit translation of a $\mathrm{CPT}+\mathrm{C}$ program, seems particularly interesting.

Furthermore, the characterisation of CPT via first-order interpretations suggests a stratification of this logic with respect to many natural parameters. Fragments of CPT induced by PIL may again coincide with other logics analysed in the context of CPT. Since PIL is based on FO, one could consider fragments of PIL defined by fragments of FO, for instance PIL with bounded quantifier rank (note that the quantifier rank of the formulae used for simulating BGS programs largely depends on the number of free variables of the subterms of the program).

Another way to define fragments of PIL is to restrict the structure of the interpretations by only allowing interpretations of fixed dimensions or by omitting the equality formula. Two very interesting fragments of PIL are *one-dimensional* PIL, i.e. the restriction of PIL to one-dimensional interpretations, and PIL without the equality formula, which we denote by $\mathrm{PIL}^{-\approx}$.

These fragments appear to bear similarities to the polynomial-time restriction of partial fixed-point logic and one of its extensions. Partial fixed-point logic is equivalent to **while**, an extension of first-order logic by while loops. Therefore **while**, which has also been used in [AHV95] to show that lfp = pfp if and only if PTIME = PSPACE, could serve as a tool to establish a connection between CPT and fixed-point logic.

For details about **while** and the extension $\mathbf{while}_{\mathrm{new}}$ (originally introduced in [AV91] as $\mathbf{while}^{\mathrm{invent}}$), which is based on invention of new elements from existing tuples, the reader is referred to [AHV95].

while is based on first-order logic with assignment and iteration, but does not allow for invention of new elements. This characterisation coincides with one-dimensional PIL, since one-dimensional interpretations can only restrict the domain of the input structure and update relations. So it is an interesting question for further research whether there is a correspondence between one-dimensional PIL and **while**.

The relation between CPT and extensions of **while** has first been studied by Blass, Gurevich and van den Bussche in [BGVdB02]. They show that $\mathbf{while}_{\mathrm{new}}$ does not suffice to achieve the same expressive power as CPT. Instead, $\mathbf{while}_{\mathrm{new}}$ has to be enhanced by an operation that makes it possible to create new elements corresponding to sets, instead of tuples only, resulting

$$\begin{tikzcd}
\text{PIL} & \overset{\text{[BGVdB02]}}{=} & \textbf{while}^{\text{sets}}_{\text{new}}|_{\text{PTIME}} & & \\
? & & \text{[BGVdB02]}\ \rotatebox{90}{$\supsetneq$} & & \\
\text{PIL}^{-\approx} & ? & \textbf{while}_{\text{new}}|_{\text{PTIME}} & & \\
? & & \rotatebox{90}{$\supseteq$} & & \\
\text{1-dim. PIL} & ? & \textbf{while}\,|_{\text{PTIME}} & \overset{\text{[AV95]}}{\underset{\text{iff PTIME=PSPACE}}{=}} & \text{LFP}
\end{tikzcd}$$

Figure 5.2: A classification of fragments of **while** and PIL and possible equivalences (where each logic is identified with the class of queries definable in that logic).

in $\textbf{while}^{\text{sets}}_{\text{new}}$. It is shown that the restriction of $\textbf{while}^{\text{sets}}_{\text{new}}$ to polynomial time is equivalent to CPT.

Like $\textbf{while}_{\text{new}}$, interpretations without an equality formula are restricted to tuple-based invention, which suggests that $\text{PIL}^{-\approx}$ could fill the gap that, in the hierarchy of extensions of **while**, is covered by the PTIME restriction of $\textbf{while}_{\text{new}}$. Therefore the fragment $\text{PIL}^{-\approx}$ and the question of how it compares to $\textbf{while}_{\text{new}}$ shall also be covered in future work.

The hierarchy of the previously described extensions of **while** and the resulting questions for further research on PIL are illustrated in Figure 5.2.

If a correspondence in this sense between fragments of PIL and other logics could be establish, this would be another indication that PIL is a natural formulation, and therefore an interesting alternative to the original definition of CPT.

Bibliography

[ADH13] Matthew Anderson, Anuj Dawar, and Bjarki Holm. Maximum matching and linear programming in fixed-point logic with counting. In *LICS*, pages 173–182. IEEE Computer Society, 2013.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[AU79] Alfred V Aho and Jeffrey D Ullman. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 110–119. ACM, 1979.

[AV91] Serge Abiteboul and Victor Vianu. Generic computation and its complexity. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 209–219, New York, NY, USA, 1991. ACM.

[AV95] Serge Abiteboul and Victor Vianu. Computing with first-order logic. *Journal of computer and System Sciences*, 50(2):309–335, 1995.

[BGS99] Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1):141–187, 1999.

[BGS02] Andreas Blass, Yuri Gurevich, and Saharon Shelah. On polynomial time computation over unordered structures. *Journal of Symbolic Logic*, pages 1093–1125, 2002.

[BGVdB02] Andreas Blass, Yuri Gurevich, and Jan Van den Bussche. Abstract state machines and computationally complete query languages. *Information and Computation*, 174(1):20–36, 2002.

[CFI92] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[CH82] Ashok Chandra and David Harel. Structure and complexity of relational queries. *Journal of Computer and system Sciences*, 25(1):99–128, 1982.

[DRR08] Anuj Dawar, David Richerby, and Benjamin Rossman. Choiceless polynomial time, counting and the Cai–Fürer–Immerman graphs. *Annals of Pure and Applied Logic*, 152(1):31–50, 2008.

[Fag74] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. 1974.

[Gro08] Martin Grohe. The quest for a logic capturing ptime. In *Logic in Computer Science, 2008. LICS'08. 23rd Annual IEEE Symposium on*, pages 267–271. IEEE, 2008.

[Gur85] Yuri Gurevich. *Logic and the challenge of computer science*. University of Michigan, Computing Research Laboratory, 1985.

[Imm86] Neil Immerman. Relational queries computable in polynomial time. *Information and control*, 68(1):86–104, 1986.

[Imm87] Neil Immerman. Expressibility as a complexity measure: results and directions. In *Structure in Complexity Theory Conference*, pages 194–202, 1987.

[Lau11] Bastian Laubner. *The Structure of Graphs and New Logics for the Characterization of Polynomial Time*. PhD thesis, Humboldt University of Berlin, 2011.

[Lib04] Leonid Libkin. *Elements of finite model theory*. Springer, 2004.

[Ros10] Benjamin Rossman. Choiceless computation and symmetry. In *Fields of logic and computation*, pages 565–580. Springer, 2010.

[Var82] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, Dezember 2013
Svenja Schalthöfer