Choiceless Computation and Logic

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
einer Doktorin der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Svenja Schalthöfer, M.Sc.
aus Meerbusch

Berichter: Universitätsprofessor Dr. Erich Grädel
Professor Dr. Anuj Dawar

Tag der mündlichen Prüfung: 24. Januar 2019

Abstract

Choiceless computation emerged as an approach to the fundamental open question in descriptive complexity theory: Is there a logic capturing polynomial time? The main characteristic distinguishing logic from classical computation is isomorphism-invariance. Consequently, choiceless computation was developed as a notion of isomorphism-invariant computation that operates directly on mathematical structures, independently of their encoding. In particular, these computation models are choiceless in the sense that they cannot make arbitrary choices from a set of indistinguishable elements.

Choiceless computation was originally introduced by Blass, Gurevich and Shelah in the form of Choiceless Polynomial Time (CPT), a model of computation using hereditarily finite sets of polynomial size as data structures. We study the structure and expressive power of Choiceless Polynomial Time, and expand the landscape of choiceless computation by a notion of Choiceless Logarithmic Space.

The unconventional definition of CPT makes it less accessible to established methods. Therefore, we examine the technical aspects of the definition in Chapter 3. An alternative characterisation of CPT is polynomial-time interpretation logic (PIL), a computation model based on iterated first-order interpretations. In Chapter 4, we study the consequences of that characterisation in terms of naturally arising fragments and extensions. In particular, we characterise PIL as an extension of fixed-point logic by higher-order objects, as well as a deterministic fragment of existential second-order logic. Furthermore, we define a fragment of PIL that limits the ability to construct sets to simple sets of tuples, and prove that this fragment is strictly less expressive than full PIL.

Towards a deeper understanding of the expressive power of CPT, we apply and extend known methods for its analysis in Chapter 5. The foundation of our investigation is the Cai-Fürer-Immerman query, a graph property that separates fixed-point logic from polynomial time and thus serves as a benchmark for the expressibility of logics within Ptime. As shown by Dawar, Richerby and Rossman, the Cai-Fürer-Immerman (CFI) query is definable in CPT if it is defined starting from linearly ordered graphs, but not while using only sets of bounded rank. We generalise their definability result to preordered graphs with colour classes of logarithmic size, as well as graph classes where the CFI construction yields particularly large graphs. The latter case is definable with sets of bounded rank. Using a novel formalisation of tuple-like objects, we prove that this is, however, not possible using the tuple-based fragment of PIL.

Finally, Chapter 6 is dedicated to our model of choiceless computation for logarithmic space. Our logic, called CLogspace, is based on the observation that the size of objects in logarithmic space is sensitive to their encoding. The semantics thus depends on an annotation of sets with varying sizes. We verify that our formalism can be evaluated in logarithmic space, and can be regarded as a model of choiceless computation, as it is included in Choiceless Polynomial Time. Moreover, it is strictly more expressive than previously known logics for logarithmic space.

Zusammenfassung

Auswahlfreie Berechnungsmodelle (*choiceless computation*) entwickelten sich aus der bedeutendsten offenen Frage der deskriptiven Komplexitätstheorie: Gibt es eine Logik für Polynomialzeit? Logiken unterscheiden sich von klassischen Berechnungsmodellen hauptsächlich durch Isomorphieinvarianz. Daraus ergibt sich das Konzept auswahlfreier Maschinen, also isomorphieinvarianter Algorithmen, die, ohne Umweg über eine bestimmte Kodierung, direkt auf mathematischen Strukturen arbeiten. Insbesondere bedeutet Auswahlfreiheit, dass diese Algorithmen aus einer Menge nicht unterscheidbarer Elemente kein beliebiges auswählen können.

Auswahlfreie Maschinen wurden erstmals durch Blass, Gurevich und Shelah in Form von Choiceless Polynomial Time (CPT) eingeführt. Dieses Modell basiert auf hereditär endlichen Mengen polynomieller Größe als Datenstrukturen. Wir untersuchen die Struktur und Ausdrucksstärke von Choiceless Polynomial Time, und entwickeln zudem ein neues auswahlfreies Berechnungsmodell für die Komplexitätsklasse Logspace.

Seine unkonventionelle Definition erschwert die Analyse von CPT mittels bewährter Methoden. Daher untersuchen wir in Kapitel 3 zunächst die technischen Aspekte der Definition. Eine alternative Darstellung für CPT ist polynomielle Interpretationslogik (PIL), welche auf iterierten FO-Interpretationen beruht. In Kapitel 4 betrachten wir Konsequenzen dieser Darstellung, genauer gesagt einige natürliche Fragmente und Erweiterungen. Insbesondere kann PIL sowohl als Erweiterung höherer Ordnung von Fixpunktlogiken, als auch als deterministische Einschränkung von existentieller Logik zweiter Stufe betrachtet werden. Zudem definieren wir eine Einschränkung von PIL auf die Erzeugung einfacher Mengen von Tupeln, und beweisen dass dieses Fragment in PIL strikt enthalten ist.

In Kapitel 5 analysieren wir die Ausdrucksstärke von CPT, indem wir bekannte Methoden anwenden und diese verallgemeinern. Dabei betrachten wir das Cai-Fürer-Immerman-Problem, ein Graphproblem, das Fixpunktlogik von Polynomialzeit trennt und deshalb zur Einordnung von Logiken innerhalb von Ptime verwendet wird. Das Cai-Fürer-Immerman- oder CFI-Problem ist, wie Dawar, Richerby und Rossman zeigten, zwar CPT-definierbar, sofern die Konstruktion von linear geordneten Graphen ausgeht, nicht aber unter Verwendung von Mengen mit beschränktem Rang. Wir verallgemeinern das Definierbarkeitsergebnis auf Präordnungen mit Farbklassen logarithmischer Größe, sowie auf Klassen, für die die CFI-Konstruktion besonders große Graphen ergibt. Im zweiten Fall sind bereits Mengen von beschränktem Rang ausreichend. Eine neuartige Charakterisierung von tupel-artigen Objekten ergibt, dass dies mit dem tupelbasierten PIL-Fragment nicht möglich ist.

Schließlich widmen wir uns in Kapitel 6 unserem auswahlfreien Berechnungsmodell für Logspace. Unsere Logik, abgekürzt CLogspace, fußt auf der Beobachtung, dass die Größe von Objekten in Logspace stark von ihrer Kodierung abhängt. Wir definieren daher eine Semantik basierend auf Mengen mit variablen Größenzuweisungen. Wir weisen nach, dass unser Formalismus in Logspace auswertbar ist, und als Fragment von CPT als auswahlfrei angesehen werden kann. Weiterhin ist die Ausdrucksstärke von CLogspace echt größer als die zuvor bekannter Logiken für Logspace.

Contents

1 Introduction

Choiceless computation emerged as an approach to the fundamental open question in descriptive complexity: Is there a logic capturing PTIME?

The result that set the field in motion is Fagin's Theorem [25], stating that existential second-order logic can define exactly those classes of finite structures that are decidable in NP. In light of Fagin's result, an answer to the question whether there is also a logic capturing PTIME would have remarkable complexity-theoretic consequences. If there is a logic capturing PTIME, then techniques from finite model theory could be applied to the problem whether P = NP. More importantly, if no such logic exists, then P $\neq$ NP.

Even though the question is undoubtedly of theoretical relevance, its original presentation by Chandra and Harel [18] has a more practical background: Their question focuses on database queries. Queries in that sense are the kind of transformations defined by real-world database query languages, mapping a relational database to a new relation over that database. The crucial property of a query is that it is independent of the encoding of the database. More formally, the output relation is invariant under isomorphisms between input databases. The connection to logic can be made through a recursive enumeration of all NP-computable queries using certain formulae in existential second-order logic. Chandra and Harel enquire about an analogous characterisation of the queries computable in polynomial time.

The corresponding question in finite model theory was made precise by Gurevich [35]: Is there a logic capturing polynomial time? Intuitively, a logic captures PTIME if it can express exactly those properties of finite structures that are computable in PTIME. But a formal statement of the question requires a definition of what constitutes a logic. One of the defining features of a logic is, again, isomorphism-invariance. In other words, the evaluation of logical formulae depends only on the structure instead of a specific encoding. Moreover, practical applications require an effective translation from formulae to algorithms. The precise definition of a logic capturing PTIME according to Gurevich is given in Section 2.3.

A typical starting point for the search for a logic capturing PTIME is fixed-point logic (FP). As shown independently by Immerman [41] and Vardi [59], FP captures

PTIME on the class of linearly ordered structures. However, the question about an isomorphism-invariant formalism characterising efficient computation without a linear order remains unsolved. Indeed, FP is too weak to express all PTIME-properties of unordered structures: It cannot define whether a structure is of even cardinality. The extension FP+C of FP by counting operators introduced by Immerman [42] already captures PTIME on many interesting classes of finite structures. Early capturing results for planar graphs [30] and graphs of bounded tree-width [33] have over the years evolved to more general classes of structures, finally leading up to Grohe's [31] theorem subsuming the classes studied previously: FP+C captures PTIME on every graph class excluding a fixed graph as a minor. Other influential expressibility and classification results, which we review in Section 3.1, have been obtained investigating stronger logics. For a more detailed exposition of recent results about FP+C, see Dawar's survey [21].

In spite of its expressive power on certain classes of structures, FP+C cannot express all PTIME-properties. This was first witnessed by a query developed by Cai, Fürer and Immerman [17], who defined a family of graphs for which the isomorphism query is not FP+C-definable, but computable in polynomial time, and in fact even in logarithmic space. The so-called Cai-Fürer-Immerman graphs, or CFI graphs, form rather simple instances of several problems. On the one hand, the graph isomorphism problem is feasible on graph classes of bounded degree and of bounded colour class size. In the original variant separating FP+C from PTIME, the CFI graphs exhibit both properties. On the other hand, it is known that the query can be reduced to solvability of linear equation systems over $\mathbb{Z}_2$. Altogether, the CFI graphs occur as a special case of several important PTIME-problems, for which they serve as a valuable benchmark.

In analogy to the extension by counting terms, fixed-point logic can be augmented with new operators that allow to define the Cai-Fürer-Immerman query. Simply adding an operator for that specific query, however, can hardly be considered natural. Using the connection to linear algebra instead, Dawar, Grohe, Holm and Laubner developed rank logic [22], an extension of fixed-point logic with the ability to solve linear equation systems over finite fields. A variant of rank logic [28] is currently one of the two main candidates for a logic capturing PTIME.

Choiceless computation Here we study the second candidate, converging on PTIME from a different perspective. Approaches based on fixed-point logics try to narrow the gap between the expressive power of known logics and PTIME. Alternatively, one can adjust the definition of PTIME machines to satisfy the

requirements for a logic. Consider the "logic" whose sentences are the Gödel numbers of Turing machines with polynomial running time. The syntax of this formalism is not decidable, violating Gurevich's definition of a logic. This can easily be solved by explicitly adding a polynomial to every Gödel number and considering only those computations of the corresponding machine that stay within the polynomial bound. But this is not isomorphism-invariant, because a Turing machine can behave differently on different encodings of the same structure.

So Turing machines are an inadequate computation model for that idea. The problem is that they operate on string representations of structures. But a logic has to be oblivious to encoding—it operates directly on mathematical structures. So in order to use a computation model as a logic in the way described above, that computation model has to differ fundamentally from classical computation. A more appropriate model would be characterised by isomorphism-invariant—*choiceless*—computation on structures.

Clearly, isomorphism-invariance means that isomorphic structures generate the same output. But for a formalism to be truly invariant under different encodings, it has to preserve the symmetries of the input structure *in every computation step*. So a choiceless algorithm can only make use of those properties of the structure that are preserved by automorphisms. Therefore, any two elements of the structure that are mapped to each other by an automorphism are treated equally. But this entails that operations that break symmetries are not permitted in choiceless computation. In particular, a choiceless algorithm cannot pick an arbitrary element out of a set of indistinguishable elements. It is choiceless in the sense that it cannot make these arbitrary choices. Choices from indistinguishable elements, though, are essential to many classical algorithms, including the augmenting path algorithm and Gaussian elimination. This limitation is compensated by the ability of choiceless algorithms to process whole *sets* of indistinguishable elements *in parallel*. In other words, sets serve as the data structures of choiceless computation.

The aim is to characterise polynomial time, or even smaller complexity classes, via choiceless computation. This is where the limitations of parallel computation become apparent. Whenever a classical algorithm chooses an arbitrary element of a set, its choiceless counterpart has to process the whole orbit of the chosen element. A sequence of such choices can quickly lead to exponentially many parallel computations. So a choiceless computation model for PTIME has to restrict the amount of parallelism permitted. But then it is not clear whether the remaining expressive power suffices to define all PTIME-computable queries.

An appropriate model of choiceless computation expressing all PTIME-queries would constitute a logic for PTIME. But it would also entail that polynomial time is the same as isomorphism-invariant polynomial time—isomorphism-invariance could then be attained at no computational cost.

Choiceless Polynomial Time Choiceless computation was originally introduced by Blass, Gurevich and Shelah [13] in the form of the logic Choiceless Polynomial Time (CPT). In its original presentation, it is a variant of abstract state machines, a formalism for computation on structures. CPT-*programs* extend a logical formalism for specifying hereditarily finite sets by certain algorithmic operations. More recent formalisations of CPT use the versatile set-building capabilities of the underlying logic to avoid most of these operations, reducing the formalism to *recursive construction of sets*. Since the sets are defined via a logic, isomorphism-invariance is maintained. Parallel computations are carried out by applying operations to all elements of a definable set at once. So the amount of parallelism is measured in terms of the complexity of the sets, more precisely the size of their transitive closure. To make the polynomial bound part of a logic with decidable syntax, every formula is explicitly augmented by a polynomial. Altogether, a sentence of Choiceless Polynomial Time consists of a formula specifying a choiceless algorithm and a polynomial restricting the complexity of the computation.

Intuitively, Choiceless Polynomial Time is as close as one can get to PTIME computation within the limitations of isomorphism-invariance (while maintaining the defining characteristics of a logic). Nevertheless, when Blass, Gurevich and Shelah [13] introduced the logic in 1999, they already conjectured that it cannot define all PTIME-queries. Moreover, they suggested multiple queries [14, 12] as possible witnesses of the separation. By now, however, most of these queries have been shown to be CPT-definable. Some of them are even expressible in FP+C, including perfect matching [7] and defining the sum of a given subset of an Abelian semigroup [3].

The surprising expressive power of Choiceless Polynomial Time is demonstrated by two classes of definability results in particular. The results in the first class are direct consequences of the explicit polynomial bound. Because of that bound, the complexity of permitted computations depends on the size of the input structure. This can be exploited by padding structures with irrelevant elements. More precisely, the original structure occurs as a small, definable substructure of the input structure whose size determines the complexity bound. In particular, if the input structure is of size $n!$, the set of all linear orders on a substructure of size n is of polynomial size.

Together with the fact that FP+C, and thus CPT, captures PTIME on the class of ordered structures, every PTIME-property of these linear orders can be defined. Logics in the classical sense, such as FP+C, do not benefit from padding. This is the case because the additional elements of the input structure do not influence the winning strategies in the pebble games for FP+C. Accordingly, Blass, Gurevich and Shelah [14] used padding arguments of that kind to show the first separation of CPT from FP+C.

With a more complex construction, Laubner [47] extended the padding technique to a capturing result on substructures of logarithmic size. Nevertheless, the method itself is based on a technical detail of the definition. In contrast, the second class of definability results directly uses the power of CPT to create sets. These results also separate CPT from FP+C. Queries that are definable in CPT but not in FP+C include isomorphism of the multipedes defined in [36] (shown in [14, 4]) and the CFI query over linearly ordered graphs [23]. The generalisation of these queries to so-called cyclic linear equation systems is CPT-definable as well, which plays an important role in the proof that CPT captures PTIME on certain structures of bounded colour class size [4].

These positive results suggest that CPT may indeed capture PTIME—or, at least, an interesting fragment thereof. In case it does not capture PTIME, verifying this would require proof techniques for inexpressibility results. Blass, Gurevich and Shelah [13] introduced a technique which seems flexible enough to accommodate different fragments of CPT—exemplified by the extensions proposed by Dawar, Richerby and Rossman [54, 23]. The basic idea is to show inexpressibility in CPT using pebble games for FP+C. To handle the sets defined by certain CPT-formulae, the games are played on structures extended by these sets. So the method presupposes these structures to be characterised in a meaningful way. Therefore it is not obvious whether the technique can be applied to full CPT. The current state of both positive and negative results is reviewed in detail in Section 3.1.

Choiceless Logarithmic Space Following the success of Choiceless Polynomial Time, we aim to apply the concept of choiceless computation to another major complexity class: logarithmic space. While polynomial time is a widely accepted notion of efficient computation, logarithmic space describes computations on huge data sets with significantly smaller working memory. In other words, a logic for LOGSPACE would characterise efficient computation for big data. In fact, when Chandra and Harel [18] first posed the question about a characterisation of PTIME, they enquired about LOGSPACE as well, claiming that most queries occurring in

practice are in fact Logspace-computable. From a complexity theoretic perspective, a logic for Logspace could improve understanding of the relationship between time and space complexity.

Indeed, a logic capturing Logspace has been pursued almost as long as a logic for Ptime. Early results for Logspace and Ptime bear interesting similarities: On ordered structures, both classes are captured by extensions of FO by some kind of iteration mechanism. Immerman [43] introduced three flavours of transitive closure logics and proved the respective capturing results: On ordered structures, deterministic transitive closure captures Logspace, symmetric transitive closure captures symmetric Logspace, and transitive closure captures NLogspace. Because of Reingold's [53] Logspace-algorithm computing symmetric transitive closures, the first two classes coincide. But, like FP, transitive closure logics cannot define the size of a structure. Moreover, augmenting these logics by counting operators does not suffice to capture the whole complexity class.

A natural and important query separating transitive closure logics from Logspace is tree isomorphism. In other words, the recursion mechanism of these logics is too weak to admit tree-like recursion. The limiting factor for recursion in Logspace is the memory necessary to store return addresses and intermediate results. Recursion over trees is possible because every vertex has a unique parent. This can also be generalised to directed acyclic graphs where the number of parents to be stored on each path is sufficiently small. The logic LREC (short for Logspace-recursion) developed by Grohe, Grußien, Hernich and Laubner [32] exhibits an operator characterising that kind of recursion. Again, known logics are strengthened by adding operators generalising previously inexpressible queries. But, as a fragment of FP+C, LREC cannot express the CFI query, which is computable in Logspace.

In fact, there are (to our knowledge) no stronger candidates for a logic capturing Logspace. There is, however, an early model of choiceless computation for logarithmic space, proposed by Grädel and Spielmann [29]. That formalism, which we denote BDTC for deterministic closure over bounded sets, is based on the idea that a Logspace-algorithm stores every element of the input structure as a number with logarithmically many bits. But Spielmann [57] already proved that BDTC does not capture Logspace. So it remains open how choiceless computation can be used to construct a stronger candidate for a logic capturing Logspace.

Contributions A huge obstacle when analysing Choiceless Polynomial Time is its unconventional definition. Hence Chapter 3 focuses on the specifics of the definition. In addition to the more common definition via creation of sets, we recapitulate the alternative characterisation in terms of iterated first-order interpretations developed in [55]. Using this characterisation, called polynomial-time interpretation logic or PIL, we can already infer an important property of choiceless computation: The essential mechanism is *recursive* creation of sets. Furthermore, we provide normal forms for CPT and PIL. One of these normal forms simplifies the standard technique for inexpressibility proofs on fragments of CPT without counting. The other two normal forms conceal the explicit polynomial bound, which turns out to be more natural starting from PIL.

The primary motivation behind interpretation logic is to ease the analysis of Choiceless Polynomial Time. In Chapter 4, we investigate how it helps to understand the structure of CPT. In particular, the definition of PIL suggests several syntactic fragments. One of the motivations for investigating these fragments is the hope that PIL collapses to some easier fragment. On the one hand, certain ways of reducing the dimension of the interpretations or restricting their signature preserve the expressive power of PIL. On the other hand, interpretations can easily be modified to define only tuples instead of arbitrary sets. We show that limiting the power to create sets in that way yields a strictly weaker logic.

An additional benefit of analysing fragments and extensions of PIL is evidence that CPT is a natural candidate for a logic capturing Ptime. Firstly, we show that fixed-point logic occurs as a natural fragment of PIL. Secondly, we define a hierarchy of extensions of PIL corresponding to the quantifier alternation hierarchy of second-order logic. Altogether, CPT can be viewed as an extension of fixed-point logic by higher-order objects, or as a deterministic restriction of existential second-order logic. The classification of fragments appeared as part of [27].

The central question about CPT concerns its expressive power: Can it express all queries computable in Ptime? In Chapter 5, we apply and extend established methods for expressibility and inexpressibility results. Recall that the Cai-Fürer-Immerman query is the prototypical query that is computable in Ptime, but not definable in FP+C. We identify new classes of graphs over which the CFI query is CPT-definable. Our first contribution generalises the result by Dawar, Richerby and Rossman [23] for the CFI query over linearly ordered graphs to preordered graphs with small, but not constant, colour classes. Secondly, we show that, whenever the CFI graphs are sufficiently large compared to the underlying graphs, CPT can define the CFI query using only sets of bounded rank. This second result relies on the

power of CPT to construct sets instead of ordered, i. e. sequence-like, objects. We formalise the notion of sequence-like objects and prove that they do not suffice to express the CFI query in CPT. The results presented in Chapter 5 were published in [51, 52].

So far, our results contribute to the already strong evidence that choiceless computation is a valuable foundation for logics in PTIME. In Chapter 6, we attend to choiceless computation as a basis for a logic for LOGSPACE. Choiceless Polynomial Time restricts the complexity of computations in terms of the size of the sets defined. The notion of a set of polynomial size is rather robust. But whether a set can be represented in logarithmic space strongly depends on the encoding of its elements. We introduce a novel formalism, called Choiceless Logspace or CLogspace, that accounts for flexible encodings by annotating sets with varying sizes. The semantics of our formulae depends on these size annotations.

First of all, this formalism is choiceless in the sense that it is included in CPT. Further, we show that it can be evaluated in logarithmic space. So, analogously to CPT, the big open question is whether CLogspace can express all queries in LOGSPACE. As a first step towards classifying its expressive power, we show that CLogspace includes known logics for LOGSPACE. This is the case for both LREC (a variant of which is already included syntactically) and BDTC.

BDTC can be easily separated from our formalism due to counting. To justify that it would not suffice to simply add counting to that logic, we show that it can define neither the tree isomorphism query nor symmetric transitive closures. Note that our logic can define that query because it includes LREC. The proof that LREC is strictly included in CLogspace provides evidence that CLogspace inherits the strengths of CPT: CLogspace captures LOGSPACE on structures with sufficiently large padding. We conclude that our formalism is a reasonable notion of Choiceless Logarithmic Space.

Acknowledgements

First of all, I would like to thank my advisor, Erich Grädel, for providing me with the right impulses at the right times, if, and only if, that was what I wished to happen. Moreover, I am grateful to Anuj Dawar for promptly agreeing to act as a co-examiner of my thesis while accommodating my tight schedule.

The contributions presented in this thesis would not have been possible without my colleagues and collaborators. Interpretation logic (which already appeared in my Master's thesis [55]) and its fragments presented in Chapter 4 are the results of joint work with Erich Grädel, Łukasz Kaiser and Wied Pakusa and appeared in [27]. The contributions presented in Chapter 5 were developed together with Wied Pakusa, who came up with the notion of strong supports, and Erkal Selman, who insisted on studying tuple-based formalisms in the first place. The results appeared as [51] and [52]. Beyond these collaborations, I would like to thank Erich Grädel for his helpful suggestions regarding Choiceless Logarithmic Space (presented in Chapter 6), as well as Faried Abu-Zaid, Wied Pakusa and Erkal Selman for thought-provoking discussions about Choiceless Polynomial Time.

Further, I am grateful to Katrin Dannert, Matthias Hoelzel, Konstantin Kotenko, Benedikt Pago, Wied Pakusa, Theophil Trippe, Daniel Wiebking, Richard Wilke and Hinrikus Wolf for proofreading parts of this thesis during various stages of completion. I thank my colleagues at MGI and i7, especially Faried, Martin L., Simon and Wied, for making my time there enjoyable through discussions of research, teaching and the quirks of everyday life.

Last but not least, I am grateful to my loved ones for supporting me during the ups and downs of this important phase of my life. I am especially grateful to my mother for years and years of encouragement.

2 Preliminaries

This thesis covers rather specialised issues in descriptive complexity. In consequence, we assume that the reader is familiar with basic concepts of finite model theory, as well as commonly used complexity classes. Concerning these topics, this chapter mainly introduces the necessary notation.

Choiceless Polynomial Time, the logic that forms the basis of our research, is presented in detail in Chapter 3. Some other less common logics, which will serve to classify the expressive power of Choiceless Polynomial Time itself, are defined in Section 2.2.

2.1 Structures and sets

Set inclusion is denoted by $\subseteq$, and strict inclusion by $\subsetneq$. $\mathcal{P}(A)$ denotes the power set of a set A, and $|A|$ its cardinality. For sets A, B, the disjoint union $A \uplus B$ is $\{\emptyset\} \times A \cup \{\{\emptyset\}\} \times B$. For convenience, we also write $C = A \uplus B$ to specify that $A \cup B = C$ for sets A and B that are already disjoint. This notation is used throughout Chapter 5. The symmetric difference $A \triangle B$ of sets A and B is $(A \setminus B) \cup (B \setminus A)$.

A signature $\sigma = (R_1, \ldots, R_k)$ consists of relation symbols $R_1, \ldots, R_k$ with associated arities $r_1, \ldots, r_k$. Note that we only consider finite, relational signatures. A σ-structure $\mathbf{A}$ is of the form $(A, R_1^{\mathbf{A}}, \ldots, R_k^{\mathbf{A}})$. We denote structures by $\mathbf{A}, \mathbf{B}, \ldots$ and refer to the domain of $\mathbf{A}$ as A. A graph with domain V and a binary edge relation E may also be referred to as $G = (V, E)$. Unless stated otherwise, all structures used throughout this thesis are finite. The class of finite σ-structures is denoted by $\mathrm{fin}(\sigma)$.

If $\sim$ is a congruence relation on a structure $\mathbf{A}$, we denote the equivalence class of an element $a \in A$ by $[a]_\sim$, and the factor structure by $\mathbf{A}_{/\sim}$. The subscript $\sim$ may be omitted if no ambiguities arise.

Elements of structures are usually assumed to be *atoms*, which means that they are not sets. If the context requires that a structure contains sets this will be stated

explicitly. An *object* is a set or an atom. A set a is transitive if $b \subseteq a$ for every set $b \in a$. The transitive closure $\mathrm{tc}(a)$ of an object a is the least (w.r.t. $\subseteq$) transitive set b with $a \in b$. Requiring that $a \in b$ instead of $a \subseteq b$ makes sure that the transitive closure of an atom is a set.

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 and all atoms in $\mathrm{tc}(x)$ are elements of A. For a σ-structure $\mathbf{A}$, we define its hereditarily finite expansion $\mathrm{HF}(\mathbf{A})$ as the (infinite) structure over $\mathrm{HF}(A)$ with relations $\sigma \cup \{\emptyset, A, \in\}$ intepreted in the obvious way. The rank $\mathrm{rk}(a)$ of a finite object is 0 if a is an atom or the empty set, and $1 + \max_{b \in a} \mathrm{rk}(b)$ otherwise.

We denote the first infinite ordinal by ω, and write $[n]$ for the ordinal associated with the natural number n whenever we intend to make the set-theoretic representation explicit. For the Kuratowski representation $\langle a, b \rangle$ of an ordered pair, we use the short form $\{a, \{a, b\}\}$. Extensions of Kuratowski pairs to arbitrary tuples, both in the recursive variant and as sets of pairs $\langle i, a_i \rangle$ are written $\langle a_1, \ldots, a_k \rangle$. We will make the encoding precise whenever that is necessary. Like in the case of ordinals, we write $\langle a_1, \ldots, a_k \rangle$ instead of $(a_1, \ldots, a_k)$ to emphasise the set representation.

2.2 Logics

If L is a logic and σ is a signature, $\mathrm{L}[\sigma]$ is the set of L-formulae over σ. Note that we give the precise definition of a logic in the sense of descriptive complexity in Section 2.3.

We denote the set of free variables of a formula φ by $\mathrm{free}(\varphi)$. Most logics we consider are evaluated over a structure $\mathbf{A}$ together with a variable assignment $\beta \colon X \supseteq \mathrm{free}(\varphi) \to A$ as usual. The extension $\beta[x \mapsto a]$ of an assignment β is the function mapping x to a and y to $\beta(y)$ for $y \neq x$.

For a formula φ with free variables $x_1, \ldots, x_k$, the expression $\mathbf{A} \models \varphi(a_1, \ldots, a_k)$ abbreviates $\mathbf{A}, \beta \colon x_1 \mapsto a_1, \ldots, x_k \mapsto a_k \models \varphi$. The formula resulting from replacing every occurrence of the free variable x by a term t is written as $\varphi[x/t]$ or $\varphi(t)$ if this does not lead to ambiguities. Analogously, we write $\varphi(R)$ if φ has a free relation variable. The set of tuples $a_1, \ldots, a_k \in A^k$ satisfying $\mathbf{A} \models \varphi(a_1, \ldots, a_k)$ is referred to as $\varphi^{\mathbf{A}}$.

The L-type of a tuple $a_1, \ldots, a_k$ is the set of all L-formulae φ with k free variables such that $\mathbf{A} \models \varphi(a_1, \ldots, a_k)$. Note that we additionally introduce a non-standard

notion of types in Chapter 5. The equality type of $a_1, \ldots, a_k$ is the restriction of its type to atomic formulae of the form $x_i = x_j$.

In what follows, we introduce some logics whose expressive power we later compare to that of (fragments and extensions of) Choiceless Polynomial Time. We assume the reader is familiar with second-order logic SO. The existential and universal fragments of SO are denoted by ∃SO and ∀SO, respectively. The fragment with k quantifier alternations is Σ_k if the first quantifier block is existential, and Π_k if it is universal.

Fixed-point logics By fixed-point logic FP, we denote both the variant with least and inflationary fixed-point operators, since these logics have the same expressive power [45].

A common generalisation of least and inflationary fixed-point logic is partial fixed-point logic PFP. The formulae of PFP[σ] are constructed like FP[σ]-formulae, where the operator ifp is replaced by pfp (in particular, negated occurrences of the respective relation symbol are allowed). To define the semantics of a formula $[\mathsf{pfp}R\overline{x}\varphi(R,\overline{x})](\overline{t})$, let $\mathbf{A}$ be a σ-structure, and let $F_\varphi \colon R \mapsto \{\overline{a} \in A^k : \mathbf{A} \models \varphi(R,\overline{a})\}$ be the update operator defined by φ. For $i \in \mathbb{N}$, define the stages R^i inductively as $R^0 = \emptyset$ and $R^{i+1} = F_\varphi(R^i)$. The *partial fixed point* of F_φ is a stage R^i with $R^i = R^{i+1}$ if such a stage exists, and the empty relation otherwise. Finally $\mathbf{A} \models [\mathsf{pfp}R\overline{x}\varphi(R,\overline{x})](\overline{t})$ if, and only if, the value of $\overline{t}$ is contained in the partial fixed point of F_φ.

The partial fixed point can always be determined in polynomial space. Since we are mainly interested in logics within Ptime, we consider the variant of PFP where the fixed point has to be reached after polynomially many iterations.

For any logic L, we define the Ptime-restriction L$\restriction$Ptime of L as the set of formulae (φ, p) where $p \colon \mathbb{N} \to \mathbb{N}$ is a polynomial. We will later also consider Logspace-restrictions of logics, where the set of formulae is the set of pairs (φ, f) with $\varphi \in$ L and $f \colon n \mapsto c \log n$ for some $c \in \mathbb{N}$. The semantics of a formula (φ, p), which will be defined for every logic L individually, usually restricts the number of iterations of some recursive formalism.

In PFP$\restriction$Ptime, the semantics of (φ, p) is defined like the semantics of φ except for subformulae of the form $[\mathsf{pfp}R\overline{x}\varphi(R,\overline{x})](\overline{t})$. In that case, the partial fixed point is redefined as a stage R^i with $R^i = R^{i+1}$ if such a stage with $i \leq p(|A|)$ exists, and $\emptyset$ otherwise.

Both FP and PFP can be augmented with simultaneous fixed-point inductions preserving expressive power. Let $\overline{\varphi} = \varphi_1(R_1, \ldots, R_k, \overline{x_1}), \ldots, \varphi_k(R_1, \ldots, R_k, \overline{x_k})$ be a system of formulae over $\sigma \cup \{R_1, \ldots, R_k\}$ (where each $\overline{x_i}$ is a tuple whose

length is the arity of R_i). With the system $\overline{\varphi}$, we associate the update operator $F_{\overline{\varphi}}\colon (R_1, \ldots, R_k) \mapsto (\{\overline{a} : \mathbf{A} \models \varphi_i(R_1, \ldots, R_k, \overline{a})\})_{1 \leq i \leq k}$. Then least, inflationary and partial fixed points of such an operator can be defined as before.

Logics with counting and equicardinality Many logics with counting operations are evaluated over two-sorted structures. For a σ-structure $\mathbf{A}$ with domain A, let $\mathbf{A}^*$ be the $(\sigma \uplus \{<\})$-structure over the disjoint union of A and $N(A) = \{0, \ldots, |A|\}$ where $N(A)$ is linearly ordered by $<$.

For a logic L, we denote by L+C its extension by counting terms. Formulae in L+C use two kinds of variables: *domain variables* which range over A, and *number variables* which range over $N(A)$. We use the Latin alphabet to denote domain variables, and the Greek one for number variables.

L+C extends the syntax of L by the following rule for constructing terms: If x is a domain variable and ψ is a formula, then $\#x.\psi$ is a term. For a structure $\mathbf{A}$ and a variable assignment β, the value $[\![\#x.\psi]\!]^{\mathbf{A},\beta}$ of $\#x.\psi$ is the number $n \in N(A)$ of elements a of A satisfying $\mathbf{A}, \beta[x \mapsto a] \models \psi$.

An alternative way to add counting to a logic is via the *Härtig quantifier* H. For any logic L, its extension L+H by the Härtig quantifier arises from adding to the definition of L the following rule for constructing formulae: If φ and ψ are formulae, then $\mathsf{H}x.\varphi(x).\psi(x)$ is a formula. A structure $\mathbf{A}$ together with an assignment β satisfies this formula if, and only if,

$$|\{a \in A : \mathbf{A}, \beta \models \varphi(a)\}| = |\{a \in A : \mathbf{A}, \beta \models \psi(a)\}|.$$

Analogously, we define the extension L+R by the *Rescher quantifier* via the rule $\mathsf{R}x.\varphi(x).\psi(x)$, where $\mathbf{A}, \beta \models \mathsf{R}x.\varphi(x).\psi(x)$ if, and only if,

$$|\{a \in A : \mathbf{A}, \beta \models \varphi(a)\}| \leq |\{a \in A : \mathbf{A}, \beta \models \psi(a)\}|.$$

When evaluated over structures augmented by a linearly ordered number sort, logics with the Härtig quantifier gain the same expressive power as their counterparts with counting terms. We denote by L^* the logic L that is evaluated over structures of the form $\mathbf{A}^*$ with an additional relation N to identify the number sort without distinguished number variables. For $\mathrm{L} \in \{\mathrm{FO}, \mathrm{FP}, \mathrm{PFP}, \mathrm{PFP}{\restriction}\textsc{Ptime}\}$, it is easy to see that $(\mathrm{L+H})^*$ has the same expressive power as L+C.

The Ptime-restriction of PFP+C and PFP+H is defined like the Ptime-restriction of PFP. As shown by Abiteboul and Vianu [2], PFP+C$\restriction$Ptime is equivalent to

FP+C in terms of expressive power. It follows that PFP+H↑PTIME is equivalent to FP+C as well. Surprisingly, their result for PFP↑PTIME without counting is of a different form: PFP↑PTIME is equivalent to FP if, and only if, PTIME = PSPACE.

To obtain closure under interpretations (see Section 2.4), logics with counting also need to be able to count tuples and equivalence classes of definable congruences. Otto [49] shows that both operations can be defined in FP+C and PFP+C. Since the number sort only contains numbers up to $|A|$, the number of k-tuples satisfying a formula is represented using k-tuples of numbers.

Counting of tuples or equivalence classes is, however, not possible for FO+C or (FO+H)*. Therefore we define the variant FO+H$_\sim$ with equicardinality quantifiers for equivalence classes of k-tuples. For every $k \in \mathbb{N}$, FO+H$_\sim$ allows for defining formulae of the form $\mathsf{H}^k_\sim \overline{x}\overline{y}.\varphi_\sim(\overline{x},\overline{y}).\psi(\overline{x}).\vartheta(\overline{x})$, where $\overline{x}$ and $\overline{y}$ are k-tuples of variables. Such a formula is satisfied in a structure $\mathbf{A}$ if, and only if, the relation $\sim$ defined by $\varphi_\sim$ is a congruence and the number of $\sim$-classes of tuples satisfying ψ is the same as the number of $\sim$-classes satisfying ϑ.

To define an extension of FO+C incorporating counting of tuples requires some encoding of numbers greater than the size of the structure. Following the definition used for the logic LREC in [32], we define a function interpreting k-tuples of numbers as single numbers. Formally, $\mathrm{enc}_A\colon \{0,\dots,|A|\}^k \to \{0,\dots,|A|^k\}$ is defined as $\mathrm{enc}_A(n_1,\dots,n_k) = \sum_{i=1}^k n_i \cdot (|A|+1)^{i-1}$ for any structure $\mathbf{A}$. We omit the subscript A when the structure $\mathbf{A}$ is fixed.

The logic FO+C$_{\mathrm{tup}}$ with counting of tuples is the extension of FO by the following rules for constructing formulae:

- If λ and μ are number variables, then $\lambda < \mu$ is a formula.
- If $\overline{\lambda}$ is a tuple of number variables, $\overline{u}$ is an arbitrary tuple of variables and φ is a formula, then $\#\overline{u}\varphi = \overline{\lambda}$ is a formula.

Let U be the set of variables occurring in $\overline{u}$, and L the set of variables occurring in $\overline{\lambda}$. Then $\mathrm{free}(\#\overline{u}\varphi = \overline{\lambda}) = (\mathrm{free}(\varphi) \setminus U) \cup L$.

Formulae of the form $\lambda < \mu$ are evaluated as before. For the semantics of $\psi = \#\overline{u}\varphi = \overline{\lambda}$, let $\mathbf{A}$ be a structure and $\beta\colon \mathrm{free}(\psi) \to A \cup N(A)$ a variable assignment. $\mathbf{A}, \beta$ satisfies ψ if, and only if, the number of extensions of β by values for $\overline{u}$ satisfying φ is $\mathrm{enc}(\beta(\overline{\lambda}))$.

Database query languages PFP and some of the stronger logics we introduce later can also be characterised as database query languages. These languages are based on iterated modification of relations. The stronger versions additionally

permit creation of new elements that are not in the input structure (or, to use the corresponding terminology, the database). In the spirit of database queries, the syntactic objects in these languages are programs instead of formulae. The basis for the languages we define here is *while*, originally introduced by Chandra and Harel [15]. The language *while* can be seen as the database variant of fixed-point logic: It is another way of extending first-order logic by iteration.

The syntax of *while*$[\sigma]$ for a signature σ is defined inductively as follows:

- If $X \in \sigma$ is a k-ary relation symbol and φ is an FO$[\sigma]$-formula with free variables $x_1, \ldots, x_k$, then $X \coloneqq \{(x_1, \ldots, x_k) : \varphi\}$ is a program.
- If Π_1 and Π_2 are programs, then their composition $\Pi_1; \Pi_2$ is a program.
- If Π is a program and φ is an FO-sentence, then `while` φ `do` Π `od` is a program.

A *while*$[\sigma]$-program Π defines a transformation between σ-structures. So let $\mathbf{A}$ be a σ-structure.

- If $\Pi = X \coloneqq \{(x_1, \ldots, x_k) : \varphi\}$, then $\Pi(\mathbf{A})$ is obtained from $\mathbf{A}$ by replacing the relation $X^{\mathbf{A}}$ with $X^{\Pi(\mathbf{A})} = \varphi^{\mathbf{A}}$.
- If $\Pi = \Pi_1; \Pi_2$ then $\Pi(\mathbf{A}) = \Pi_2(\Pi_1(\mathbf{A}))$.
- If $\Pi =$ `while` φ `do` Π' `od`, then $\Pi(\mathbf{A}) = \Pi^\ell(\mathbf{A})$ where ℓ is minimal such that $\Pi^\ell(\mathbf{A}) \not\models \varphi$, and $\Pi(\mathbf{A}) = \mathbf{A}$ if no such ℓ exists.

If $\mathbf{A}$ is a τ-structure for some $\tau \neq \sigma$, then the input structure is the $\sigma \cup \tau$-expansion of $\mathbf{A}$ where every relation in $\sigma \setminus \tau$ is empty. A *while*-program Π can be viewed as a formula by specifying a distinguished output relation X_{Out}. If Π is considered as a sentence, then $\mathbf{A} \models \Pi$ if, and only if, $X_{\text{Out}}^{\Pi(\mathbf{A})}$ is non-empty. As shown by Abiteboul and Vianu [1], *while* is equivalent to PFP in that sense.

The language while_{new} adds to *while* the capability to extend the structure by new elements. Formally, this is achieved by the following rule for constructing programs: If φ is an FO$[\sigma]$-formula with free variables $x_1, \ldots, x_k$ and $Y \in \sigma$ is a $(k+1)$-ary relation symbol, then $Y \coloneqq$ `tup-new`$\{(x_1, \ldots, x_k) : \varphi\}$ is a program.

If Π is of that form, then the domain of $\Pi(\mathbf{A})$ is the disjoint union of A and $\varphi^{\mathbf{A}}$. All relations except for Y are interpreted as in $\mathbf{A}$, and Y maps every tuple in $\varphi^{\mathbf{A}}$ to its associated new element, i. e.

$$Y^{\Pi(\mathbf{A})} = \{(a_1, \ldots, a_k, (a_1, \ldots, a_k)) : \mathbf{A} \models \varphi(a_1, \ldots, a_k)\}.$$

These programs, called *tuple-new statements*, can be seen as tuple-based invention of elements. Blass, Gurevich and van den Bussche [15] further extend the formalism

to allow for set-based inventions. The result is the language $\mathit{while}^{\mathit{sets}}_{\mathit{new}}$, which is obtained from $\mathit{while}_{\mathit{new}}$ by adding the following type of atomic programs:

If Y is a binary relation and φ is an FO[σ]-formula with free variables x, y, then $Y \coloneqq \texttt{set-new}\{(x, y) : \varphi\}$ is a program.

For a σ-structure $\mathbf{A}$, the formula φ induces the function $S\colon A \to \mathcal{P}(A)$ with $S(a) = \{b : \mathbf{A} \models \varphi(a, b)\}$ for $a \in A$. Then the domain of $\Pi(\mathbf{A})$ is $A \uplus \operatorname{Im}(S)$. The relation Y associates with every $a \in A$ the set of all $b \in A$ satisfying $\mathbf{A} \models \varphi(a, b)$.

Consider, for example, the program $Y \coloneqq \texttt{set-new}\{(x, y) : x = x\}$. The program creates a single new element corresponding to the whole domain of the input structure. If $\mathbf{A}$ is a complete bipartite graph, then $Y \coloneqq \texttt{set-new}\{(x, y) : Exy\}$ creates an element for each of the partitions.

As usual, the PTIME-restrictions of *while*, $\mathit{while}_{\mathit{new}}$ and $\mathit{while}^{\mathit{sets}}_{\mathit{new}}$ are the sets of formulae (Π, p) where $p\colon \mathbb{N} \to \mathbb{N}$ is a polynomial. The semantics is only changed for while-loops, i. e. programs of the form `while` φ `do` Π `od`. Then $(\Pi, p)(\mathbf{A}) = \Pi^{\ell}(\mathbf{A})$ for the minimal $\ell \leq p(|A|)$ such that $\Pi^{\ell+1}(\mathbf{A})$ has more than $p(|A|)$ elements or $\Pi^{\ell}(\mathbf{A}) \not\models \varphi$.

2.3 Descriptive complexity

The central notion of descriptive complexity is that of a logic capturing a complexity class. In the following, we provide the definition of such a logic as first coined by Gurevich [35]. A logic L in the sense of Gurevich associates with every signature σ a *decidable* set of sentences together with an *isomorphism-invariant* relation $\models_{\mathrm{L}}$ between σ-structures and sentences.

The expressive power of a logic is compared to a complexity class in terms of the queries it can define. Note that, in general, a query is a mapping associating with every structure a relation over that structures. However, we only consider Boolean queries i. e. the isomorphism-invariant version of decision problems. A Boolean query (or simply a query) over a signature σ is an isomorphism-closed class of finite σ-structures. An example of a query that is commonly used to separate logics without counting from PTIME is EVEN, defined as $\{\mathbf{A} \in \operatorname{fin}(\emptyset) : |A| \text{ is even}\}$. A sentence φ in a logic L defines a query $\mathcal{Q}$ if $\mathcal{Q} = \{\mathbf{A} \in \operatorname{fin}(\sigma) : \mathbf{A} \models \varphi\}$, and $\mathcal{Q}$ is L-definable if it is defined by some L-sentence φ.

A logic L captures a complexity class C if the following conditions hold for every signature σ:

1. There is an algorithm that, given a sentence φ in $\mathrm{L}[\sigma]$, produces a Turing machine in C, together with the corresponding resource bound, that decides the query defined by φ.
2. For every Boolean query $\mathcal{Q}$ over σ that is decidable in C, there is an L-sentence defining $\mathcal{Q}$.

To make the second condition precise, we say that a query $\mathcal{Q}$ is decidable in C if there is a Turing machine in C that accepts exactly those encodings of σ-structures that are in $\mathcal{Q}$ (for some fixed encoding of finite σ-structures). For the logics considered here, verifying the first condition is usually rather straightforward.

The second condition can often be established when restricted to certain classes of structures (which are themselves closed under isomorphisms). Let $\mathcal{C}$ be a class of finite structures. If the first condition is satisfied, and the second one holds for all queries $\mathcal{Q} \subseteq \mathcal{C}$, then L captures the complexity class C on the class $\mathcal{C}$.

The notion of a query also allows for comparing the expressive power of different logics. A logic L_2 is at least as expressive as L_1 ($\mathrm{L}_1 \leq \mathrm{L}_2$) if every Boolean query definable in L_1 is definable in L_2. If, additionally, there is a query that is expressible in L_2, but not in L_1, then $\mathrm{L}_1 < \mathrm{L}_2$. We say that L_1 and L_2 are equivalent ($\mathrm{L}_1 \equiv \mathrm{L}_2$) if $\mathrm{L}_1 \leq \mathrm{L}_2$ and $\mathrm{L}_2 \leq \mathrm{L}_1$.

For Ptime-restrictions of logics, the size of the input structure can influence evaluation of formulae. So padding a structure with irrelevant elements can, for instance, allow more iteration steps in fixed-point iterations. If that padding is large enough, it can be possible to define the set of all linear orders on the orginal structure, which is now only a small substructure of the input structure. Together with capturing results on the class of ordered structures, this yields capturing results for *padded structures*.

A padded σ-structure is a $(\sigma \cup \{U\})$-structure $\mathbf{A}$, where U is a unary relation. The underlying structure of $\mathbf{A}$ is the σ-reduct of the substructure induced by $U^{\mathbf{A}}$. We denote the size $|U^{\mathbf{A}}|$ of the underlying structure by u, and $|A|$ by n. Usually $u \ll n$, for example $2^u < n$.

Let $\mathcal{C}$ be a class of padded structures (typically defined through some inequality relating the parameters u and n). A formula φ defines a query $\mathcal{Q}$ on $\mathcal{C}$ if, for all $\mathbf{A} \in \mathcal{C}$ with underlying structure $\mathbf{U}$, $\mathbf{A} \models \varphi$ if, and only if, $\mathbf{U} \in \mathcal{Q}$. The query $\mathcal{Q}$ is L-definable on $\mathcal{C}$ if there is an L-sentence φ defining $\mathcal{Q}$ on $\mathcal{C}$. This induces the following definition of capturing on classes of padded structures: The first condition for capturing a complexity class remains unchanged, and the second condition requires instead that there is an L-sentence defining $\mathcal{Q}$ on the class $\mathcal{C}$.

2.4 Interpretations

Interpretations are a way to logically define transformations of structures. More precisely, an interpretation defines a transformation $\mathcal{I}$ from σ-structures to τ-structures where, for every σ-structure $\mathbf{A}$, the interpreted structure $\mathcal{I}(\mathbf{A})$ consists of equivalence classes of tuples over A.

Let L be a logic, let σ, τ be signatures, and let $d \geq 1$. A d-dimensional $\mathrm{L}[\sigma, \tau]$-interpretation with parameters $\overline{y}$ is a sequence

$$\mathcal{I} = (\varphi_\delta(\overline{x}, \overline{y}), \varphi_=(\overline{x_1}, \overline{x_2}, \overline{y}), (\varphi_R(\overline{x_1}, \ldots, \overline{x_r}, \overline{y}))_{R \in \tau})$$

of $\mathrm{L}[\sigma]$-formulae, where $\overline{x}$ as well as every $\overline{x_i}$ is a d-tuple of variables and r is the arity of $R \in \tau$. We call φ_δ the domain formula and $\varphi_=$ the equality formula. For every σ-structure $\mathbf{A}$ and assignment $\overline{y} \mapsto \overline{a}$, $\mathcal{I}$ defines a structure $\mathcal{I}(\mathbf{A}, \overline{a})$ whenever the relation $\sim$ defined by $\varphi_=[\overline{y}/\overline{a}]$ is a congruence. First we define the τ-structure $\mathbf{B}$ over the domain $\{\overline{b} \in A^d : \mathbf{A} \models \varphi_\delta(\overline{b}, \overline{a})\}$ where each relation R is defined as $R^{\mathbf{B}} = \{(\overline{b_1}, \ldots, \overline{b_r}) \in A^{dr} : \mathbf{A} \models \varphi_R(\overline{b_1}, \ldots, \overline{b_r}, \overline{a})\}$. Let $\sim$ be the relation defined by $\varphi_=$, i. e. $\sim = \{(\overline{b_1}, \overline{b_2}) \in B^2 : \mathbf{A} \models \varphi_=(\overline{b_1}, \overline{b_2}, \overline{a})\}$. If $\sim$ is a congruence, then $\mathcal{I}(\mathbf{A}, \overline{a})$ is the factor structure $\mathbf{B}_{/\sim}$.

The length k of the tuple $\overline{y}$ is the number of parameters of $\mathcal{I}$. If $k = 0$, then $\mathcal{I}$ is parameter-free and we write $\mathcal{I}(\mathbf{A})$ instead of $\mathcal{I}(\mathbf{A}, \overline{a})$. If $\mathcal{I}$ is a one-dimensional interpretation and φ_δ is a tautology, then $\mathcal{I}$ preserves the domain. If $\varphi_=$ is trivial, i. e. if $\varphi_=(\overline{x_1}, \overline{x_2})$ is equivalent to $\overline{x_1} = \overline{x_2}$, then $\mathcal{I}$ is congruence-free.

For certain logics, an interpretation also defines a transformation from τ-formulae to σ-formulae. We say that a logic L_1 is weakly closed under L_2-interpretations if every $\mathrm{L}_2[\sigma, \tau]$-interpretation $\mathcal{I}$ defines a mapping ${}^{\mathcal{I}} : \mathrm{L}_1[\tau] \to \mathrm{L}_2[\sigma]$ such that, for every sentence $\varphi \in \mathrm{L}_1[\tau]$ and every σ-structure $\mathbf{A}$,

$$\mathbf{A} \models \varphi^{\mathcal{I}} \text{ if, and only if, } \mathcal{I}(\mathbf{A}) \models \varphi.$$

If that condition holds for $\mathrm{L}_1 = \mathrm{L}_2$, then the logic is closed under interpretations.

Lemma 2.1 (Interpretation lemma)**.** FO *is closed under interpretations.*

The interpretation lemma can be proven by replacing every equality in φ by $\varphi_=$ and every relation symbol by the formula defining it, and relativising all quantifiers to the set defined by φ_δ.

In the light of the interpretation lemma, an FO$[\sigma, \tau]$-interpretation can also be seen as the logical counterpart of a compexity-theoretic reduction. We also refer to L-interpretations as L-reductions in that sense.

If $\mathcal{I}_1$ is an L$[\sigma, \sigma']$-interpretation and $\mathcal{I}_2$ is an L$[\sigma', \tau]$-interpretation, their concatenation $\mathcal{I}_1 \circ \mathcal{I}_2$ maps every σ-structure $\mathbf{A}$ to $\mathcal{I}_2(\mathcal{I}_1(\mathbf{A}))$ if both transformations are defined. For logics L that are closed under interpretations, the formulae in $\mathcal{I}_2$ can be transformed using $\mathcal{I}_1$. Then $\mathcal{I}_1 \circ \mathcal{I}_2$ is again defined by an interpretation, which we also denote by $\mathcal{I}_1 \circ \mathcal{I}_2$.

As shown by Krynicki [46], FO+H is not closed under interpretations. But in some special cases we can still obtain transformations of formulae.

Lemma 2.2 (Weak interpretation lemma for FO+H)**.**

1. FO *is weakly closed under* FO+H*-interpretations.*
2. FO+H *is closed under one-dimensional, congruence-free interpretations.*

Proof.

1. The translation is exactly the same as for FO-interpretations. Since the interpretation consists of FO+H-formulae, the result is also an FO+H-formula.
2. In addition to the translation steps for FO-formulae, relativise every Härtig quantifier to the domain formula. Since $\mathcal{I}$ is one-dimensional and congruence-free, the quantifier then indeed counts elements of the interpreted structure. □

2.5 Infinitary logic and pebble games

For $k \in \mathbb{N}$, we denote by C^k the k-variable fragment of infinitary logic with counting quantifiers, and by L^k the variant without counting quantifiers. Since every FP+C-sentence can be translated to C^k (see, for example, [49]), inexpressibility results for C^k carry over to FP+C. Lower bounds for C^k can be shown with the corresponding Ehrenfeucht-Fraïssé-style pebble games due to Immerman and Lander [44]. More precisely, structures $\mathbf{A}$ and $\mathbf{B}$ cannot be distinguished in C^k (which we express as $\mathbf{A} \equiv^{\mathrm{C}}_k \mathbf{B}$) if, and only if, Duplicator has a winning strategy in the k-pebble bijection game, which is defined as follows:

The game is played between two players called Spoiler and Duplicator. The positions of the game on structures $\mathbf{A}, \mathbf{B}$ are pairs of ℓ-tuples $(\overline{a}, \overline{b})$, for $\ell \leq k$, with $\overline{a} \in A^\ell$ and $\overline{b} \in B^\ell$. A position is interpreted as a placement of k pairs of pebbles on elements of A and B. The game starts with two empty tuples.

At a position $((a_1, \dots, a_\ell), (b_1, \dots, b_\ell))$, Spoiler picks up a pair of pebbles, so he chooses $i \leq k$. Duplicator's move is a bijection respecting the remaining pebbles. Formally, she plays a bijection g such that $g(a_j) = b_j$ for all $1 \leq j \leq \ell$ with $j \neq i$. Spoiler then places the ith pair of pebbles according to Duplicator's bijection, that is, he chooses $a \in A$ and the new position becomes the pair of tuples where a_i is replaced with a and b_i with $g(a)$.

Spoiler wins the game in a position $(\overline{a}, \overline{b})$ if it does not induce a partial isomorphism, whereas Duplicator wins if the play continues indefinitely.

2.6 Graphs and orders

Let $G = (V, E)$ be an undirected graph. For a vertex $v \in V$, $E(v)$ denotes the set of edges incident to v. If $V', V'' \subseteq V$, $E[V']$ is the set of edges in the subgraph induced by V', and (V', V'') denotes the (V', V'')-cut, i. e. all edges $\{v, w\}$ with $v \in V'$ and $w \in V''$. We also write (V', v) to denote the cut $(V', \{v\})$.

If $G = (V, E)$ is a directed graph, vE is the set of successors of v, and Ev is the set of its predecessors.

A preorder is a relation $\preceq$ that is reflexive and transitive. In contrast, a strict preorder is irreflexive and transitive. A total preorder is a preorder $\preceq$ such that every pair of elements is $\preceq$-comparable. A preordered graph is a structure $G^{\preceq} = (V, E, \preceq)$ where $\preceq$ is a total preorder of the vertices.

Let $\mathbf{A}$ be a structure with a total preorder $\preceq$. A subset $C \subseteq A$ where $\preceq$ is symmetric is called a colour class. Note that $\preceq$ induces a linear order on the colour classes. We denote the ith colour class with respect to that linear order by C_i. Let $f \colon \mathbb{N} \to \mathbb{N}$. A preordered structure $\mathbf{A}$ is f-bounded if every colour class is of size $f(|A|)$, and q-bounded if f is the constant q.

2.7 Automorphism groups

For a structure $\mathbf{A}$ with domain A, let $\mathrm{Sym}(A)$ be the symmetric group of A, and $\mathrm{Aut}(\mathbf{A}) \leq \mathrm{Sym}(A)$ the automorphism group of $\mathbf{A}$. Every automorphism $\pi \in \mathrm{Aut}(\mathbf{A})$ can be extended to an automorphism of $\mathrm{HF}(\mathbf{A})$ by setting $\pi(a) = \{\pi(b) : b \in a\}$ for every set $a \in \mathrm{HF}(A)$. Thus we can identify $\mathrm{Aut}(\mathbf{A})$ and $\mathrm{Aut}(\mathrm{HF}(\mathbf{A}))$.

Now let $\Gamma \leq \mathrm{Aut}(\mathbf{A})$ for a structure $\mathbf{A}$. The orbit of an object $a \in \mathrm{HF}(\mathbf{A})$ is the set $\mathrm{Orbit}_\Gamma(a) = \{\pi(a) : \pi \in \Gamma\}$. The stabiliser $\mathrm{Stab}_\Gamma(a)$ is the set of automorphisms $\pi \in \Gamma$ with $\pi(a) = a$, and its point-wise stabiliser $\mathrm{Stab}^\bullet_\Gamma(a)$ is the set

$\{\pi \in \Gamma : \pi(b) = b \text{ for every } b \in a\}$. In all cases, we omit the subscript if the group Γ can be uniquely identified. To reason about the sizes of orbits and stabilisers, we employ the orbit-stabiliser theorem: $|\Gamma| = |\mathrm{Orbit}_\Gamma(a)| \cdot |\mathrm{Stab}_\Gamma(a)|$.

Hereditarily finite objects can often be described in terms of their supports. A subset $\sigma \subseteq A$ is a support for $a \in \mathrm{HF}(A)$ if $\mathrm{Stab}^\bullet_\Gamma(\sigma) \subseteq \mathrm{Stab}_\Gamma(a)$.

3 Choiceless Polynomial Time

The origin of choiceless computation as it is examined in this thesis is the logic Choiceless Polynomial Time (CPT for short). We first aim to provide a basis for the deeper investigations presented subsequently.

Choiceless Polynomial Time has been a candidate for a logic capturing PTIME for almost 20 years. Why is this logic still believed to capture at least a significant fragment of PTIME, and how can its expressiveness be explained?

Some of CPT's distinguishing features can be observed in its very definition. As a formalism of choiceless *computation*, CPT can be seen as a class of algorithms that, instead of encodings, process structures directly. The data structures of these algorithms are higher-order objects—hereditarily finite sets over the input structure.

Formulae in CPT can define hereditarily finite sets by means of standard set-theoretic operations, most importantly comprehension of the form "the set of all x satisfying φ". The logic underlying CPT (called BGS after Blass, Gurevich and Shelah) gains its strength from *recursively* applying such operations, which in particular allows for defining arbitrarily nested sets.

Recursive creation of sets, however, is too strong to remain within polynomial time: The size of such sets can easily grow exponentially. This is prevented by imposing explicit polynomial bounds on the size of the sets created, as well as the number of computation steps. In other words, CPT is the PTIME-restriction of its underlying logic BGS.

The unconventional definition of this logic has some notable consequences regarding the question whether CPT captures PTIME. On the one hand, the explicit polynomial bound alone allows for a separation from fixed-point logic. But also the ability to create higher-order objects has been used in innovative ways over the years to obtain promising expressibility results. On the other hand, conventional methods for showing *in*expressibility results seem to fail for Choiceless Polynomial Time. However, several techniques have been developed by now that make it possible to show lower bounds for fragments of CPT.

Before we formally define CPT, we give an overview of known results and the underlying proof techniques, some of which are applied and built upon in this

thesis. The definition of CPT is then presented in two different variants. The first definition, based on the original formalism introduced by Blass, Gurevich and Shelah [13], implements the idea of computation over hereditarily finite sets. The second one, first introduced in [55], characterises CPT in terms of iterated first-order interpretations. This second characterisation, called interpretation logic, constitutes the foundation of our investigation of fragments and extensions of CPT in Chapter 4.

As part of the technical foundation for later results, we further present normal forms for both CPT and interpretation logic. Most importantly, two of these normal forms provide a way to hide the explicit polynomial bound. Not only does this simplify notation later on, it also constitutes an (albeit partial) answer to the question whether CPT can be defined without the polynomial bounds.

3.1 Previous work

The original definition of Choiceless Polynomial Time is due to Blass, Gurevich and Shelah [13]. Following Gurevich's formalisation of the question whether there is a logic capturing PTIME, CPT is specifically designed to be as close to a polynomial time machine model as possible without violating the properties defining a logic. Nevertheless, they conjectured that even this model of computation on structures is too weak to capture PTIME.

The first version of CPT, which we denote by CPT^{-}, did not possess a counting operation, and was immediately separated from PTIME using the query EVEN [13]. Thus we mainly consider the variant with counting defined in [14].

One of the two conditions for a logic capturing PTIME was verified immediately [13]: There is a machine computing for every CPT-sentence an equivalent PTIME-algorithm. So the vital question about Choiceless Polynomial Time is if it can express all queries within PTIME. From the beginning, various queries have been suggested to separate CPT from PTIME. So far, a separation has not been possible, instead, many of those queries turned out to be expressible. Before we turn to known results about the expressive power of CPT, as well as the difficulties in showing negative results, we summarise how CPT relates to other logics.

Characterisations As an isomorphism-invariant computation model, CPT was originally [13] defined in terms of abstract state machines. While this definition supports the view of formulae as algorithms operating directly on structures, the original formalisation is rather complexis rather complex.

A more concise definition due to Rossman [54] describes a CPT-sentence as a single logically defined transformation mapping hereditarily finite sets to hereditarily finite sets. That transformation is applied recursively until some halting condition is satisfied, at which point a logical formula defines whether the whole sentence is satisfied. In both definitions, the polynomial bound affects the number of sets *activated* during the iteration, that is, the size of the sets that occur somewhere in the computation. Based on Rossman's formalism, Grädel and Grohe [26] define a version of CPT whose syntactic objects resemble logical formulae more than algorithms.

All in all, the definition of CPT has evolved considerably compared to the original variant of abstract state machines. But it turns out to be even more robust, in the sense that it allows some less obvious characterisations via other known logics.

Blass, Gurevich and van den Bussche [15] investigate the relationship between CPT and certain database query languages. They introduce the extension $\mathit{while}_{new}^{sets}\uparrow\textsc{Ptime}$ of the established language $\mathit{while}_{new}\uparrow\textsc{Ptime}$ and prove that this extension is equivalent to CPT^-, the fragment of CPT without counting. Both while_{new} and $\mathit{while}_{new}^{sets}$ extend *while* by creation of new elements. But, whereas while_{new} defines a set of new elements based on a definable set of *tuples* from the previous structure, $\mathit{while}_{new}^{sets}$ supports *set*-based invention. It is shown that $\mathit{while}_{new}^{sets}\uparrow\textsc{Ptime}$ is indeed strictly more expressive than $\mathit{while}_{new}\uparrow\textsc{Ptime}$. This result forms the basis of our analysis of CPT over tuples in Chapters 4 and 5.

As shown in [55, 27], full CPT can be characterised by polynomial-time interpretation logic PIL, which is based on iterated first-order interpretations. We define this formalism in detail in Section 3.3 and study some consequences of the characterisation in Chapter 4.

Whereas those equivalent formalisms offer insights into the structure of CPT, the contribution of these results towards answering the question whether CPT captures Ptime is not obvious.

An idea with the potential to gain results about CPT's expressive power is to approach it from the point of view of circuit complexity. Since Ptime can be captured on ordered structures, it seems evident that the circuits to be studied have to be invariant under different encodings of the input structure. Although that notion of *symmetric circuits* led to a characterisation of FP and FP+C (shown by Anderson and Dawar [5]), it is still open whether there is a circuit model equivalent to CPT.

The final characterisation of CPT we consider yields a statement about computability over certain infinite sets that would directly imply that CPT captures Ptime.

Indeed CPT can be embedded in a model of polynomial time computation for a certain class of infinite sets, introduced by Bojańczyk and Toruńczyk [16]. In addition to the computation model, they define a complexity class, called fixed-dimension polynomial time, which serves as a notion of PTIME over these infinite sets. Every function computable by their computation model is also in the complexity class. Moreover, they show that if the converse also holds, then CPT captures PTIME.

Padded structures CPT is a PTIME-restriction, where the number of iterations and the size of the sets created is bounded by a polynomial. In consequence, larger structures allow for creating larger sets. So if sufficiently many irrelevant elements are added to a structure, larger sets over the original, smaller structure can be defined. This leads to the notion of padded structures.

The first definability result for the weaker logic CPT^- was obtained by Blass, Gurevich and Shelah [13] using padded structures. Results of this kind use the fact that FP, and thus CPT^-, captures PTIME on ordered structures. It follows that if a linear order is CPT-definable on some class of structures, then CPT captures PTIME on that class. But for structures with sufficiently large padding (i. e. $u! < n$), already CPT^- can define *all* linear orders on the underlying structure.

Blass, Gurevich and Shelah first used this idea to show that CPT^- cannot define EVEN on padded structures, which implies that the counting operation is indeed necessary. Later they could also prove that padding of exponential size suffices to define the Cai-Fürer-Immerman query [14].

Both the capturing result and definability of the padded Cai-Fürer-Immerman query imply that CPT is strictly more expressive than FP+C. The reason why padding results can separate CPT from FP+C is that the pebble games used to show lower bounds for FP+C are not affected by padding. Note that the separation from $\mathit{while}_{new}\uparrow\mathrm{PTIME}$ [15] also uses EVEN on padded structures. So the polynomial bounds, which on the one hand complicate the definition and analysis of Choiceless Polynomial Time, contribute to its strength on the other hand.

The result about the Cai-Fürer-Immerman query with padding of exponential size makes use of the particular structure of the query. However, Laubner [47] obtained a more general capturing result for padded structures with padding of exponential size: For every $c \in \mathbb{N}$, CPT captures PTIME on the class of padded structures where $u \leq c \log n$ and the underlying structure is a graph with edge colourings (that is, a structure with multiple binary relations).

In order to apply the capturing result for ordered structures, a CPT-formula does not have to define a linear order on the input structure. It suffices to define

a canonical copy instead. To that end Laubner implements an extension of a canonisation algorithm by Corneil and Goldberg [19] in CPT. Since that algorithm runs in time $2^{\mathcal{O}(n)}$, exponential padding suffices. The question whether the same is possible for general structures is open. Grädel and Grohe [26] raise the question whether the size of the padding can be improved further, for instance to the case where the underlying structure is of polylogarithmic size. In Section 4.1.1, we elaborate on the connection between these two open questions.

Queries that are already FP+C-definable Ever since Blass, Gurevich and Shelah [13] conjectured that Choiceless Polynomial Time does not capture PTIME, various queries have been suggested for separating CPT from PTIME. Studying these queries also led to new definability results for FP+C.

An extension of the proof that EVEN is not expressible in CPT^- yields that the existence of a perfect matching on bipartite graphs is not CPT^--definable either (this is also shown in [13]). In their followup paper [14], the same authors show that existence of a perfect matching, as well as the size of the largest matching, is FP+C-definable on bipartite graphs.

The question whether the query is still definable on general graphs was answered positively by Anderson, Dawar and Holm [6, 7]. They show not only that the maximum matching query is definable in FP+C, they also express certain cases of linear programming in the logic. This further implies that minimum cut and maximum flow are FP+C-definable.

As the standard algorithms for these queries rely on arbitrary choice, this result can be seen as a way of eliminating the need for arbitrary choice—the FP+C-definitions correspond to *choiceless* algorithms.

After several queries conjectured to separate CPT from PTIME had already been shown to be CPT-definable, Rossman (quoted in [12]) inquired about a problem of surprising simplicity: Can CPT define the sum over a subset of a given semigroup? In contrast to the setting in standard group-theoretic algorithms, the semigroup can be given by a full multiplication table instead of a generating set. For the sum of a subset to be well-defined, the semigroup is required to be Abelian.

The question whether the query COMMUTATIVE SEMIGROUP SUBSET SUM is CPT-definable was finally settled by Abu Zaid, Dawar, Grädel and Pakusa [3] after it had been open for twelve years. They construct FP+C-formulae defining the query and show that counting is indeed necessary: Not even CPT can define the query without counting.

Symmetry and linear algebra Towards the goal of revealing whether CPT can express all PTIME-queries, the queries that are known *not* to be in FP+C are of particular interest. Prominent examples include the Cai-Fürer-Immerman graphs as well as the isomorphism problem for *multipedes*, a class of rigid structures defined by Gurevich and Shelah [36].

These queries, however, are specifically designed to obtain inexpressibility results and may thus seem rather artificial. This raises two questions: Are there natural queries that are still not definable in FP+C? And, can the queries that are not FP+C-definable be characterised by some common property?

In their discussion of multipedes, Blass, Gurevich and Shelah [14] already suggest a connection to linear algebra. Let us remark that multipedes come in different variants, called 3-multipedes and 4-multipedes, the difference being that 4-multipedes are an expansion of 3-multipedes with more information on their structure. The result that the isomorphism problem for 4-multipedes is CPT^--definable constitutes the first separation of CPT from FP+C without padding. For 3-multipedes, the question whether the isomorphism problem is CPT-definable remained open until recently. Remarkably, the PTIME-algorithm for that problem given by Blass, Gurevich and Shelah [14] is based on linear algebra. More precisely, the problem can be reduced to solvability of linear equation systems over $\mathbb{Z}_2$.

This observation led to a deeper investigation of problems from linear algebra over finite fields, more precisely definability of determinants, also published in [14]. Note that a precise definition of the query requires a meaningful encoding of matrices as structures. The encodings considered in [14] use the elements of an unordered set to index the rows and columns of a matrix. This formalisation is suitable for the goal of analysing the expressive power of CPT on *unordered* structures.

It is shown that for matrices coded that way the property that the determinant is zero (i. e. singularity) is definable in FP+C, but not in CPT^-. The subsequent question whether the value of the determinant is CPT-definable was answered positively in [12]. The definition presented there is an implementation of Csanky's [20] algorithm for determinants over finite fields.

The idea that linear algebra is the unifying theme of queries beyond FP+C was pursued again following a result by Atserias, Bulatov and Dawar [8]: FP+C cannot define solvability of linear equation systems over any (non-trivial) finite Abelian group. Rank logic FP+rk, introduced by Dawar, Grohe, Holm and Laubner [22], explicitly extends fixed-point logic by that capability. Both the CFI query and isomorphism of multipedes are definable in FP+rk. The proofs are based on first-order reductions to solvability of linear equation systems, which is an additional

indication that linear equation systems characterise the queries that are hard for FP+C. For an in-depth analysis of the connection between CPT and linear algebra, we refer to Pakusa's PhD thesis [50].

Recall that the first definability result for the Cai-Fürer-Immerman query from [14] requires padded structures. The version without padding was studied by Dawar, Richerby and Rossman [23], who established CPT^{-}-definability in case the CFI graphs are defined starting from linearly ordered graphs. As this variant is already inexpressible in FP+C, the result constitutes another separation from FP+C without padding. Moreover, their techniques provide new insight into the strengths of CPT. One of the core ideas is to define sets that respect the many symmetries of CFI graphs in a certain way. Algorithmically speaking, this makes it possible to perform a large number of parallel computations on these sets without violating the polynomial bound. Details on this CPT-algorithm are provided in Section 5.2.1.

Both multipedes and CFI graphs over ordered graphs are 2-bounded structures. So these results give rise to the question whether CPT can capture PTIME on classes of structures of bounded colour class size. A positive answer is provided by Abu Zaid, Grädel, Grohe and Pakusa [4] for the case that the colour classes additionally have Abelian automorphism groups. They show that such structures with *Abelian colours* can be canonised in CPT.

A central ingredient of the canonisation procedure is solvability of *cyclic linear equation systems* over finite rings. The equations in such systems are partitioned into blocks where every pair of variables within a block is explicitly related by an equation. This means that whenever the value of any variable in a block is fixed, the values of all other variables in that block are also uniquely determined. In CFI graphs, these blocks, or colour classes, correspond to the edge gadgets. Indeed, cyclic linear equation systems generalise the linear equation systems describing the CFI query. The sets used to solve cyclic linear equation systems in CPT generalise the objects created in the algorithm for the CFI query in [23]. Intuitively, these sets (called *hyperterms*) inherit the property that multiple operations can be applied in parallel within the polynomial bound.

It is easy to see that 2-bounded structures have Abelian colours. Since 3-multipedes are 2-bounded, the canonisation result implies that the isomorphism problem for 3-multipedes is CPT-definable, answering the question from [14].

Solvability of linear equation systems is of particular importance for the, as yet unknown, connection between CPT and rank logic. If all linear equation systems could be solved in CPT (as enquired in [26]), this would imply inclusion of FP+rk in CPT.

Furthermore, certain fragments of rank logic are separated from each other by a generalisation of the CFI query [40]. As observed in [4], this generalisation consists of structures with Abelian colours. So the capturing result also separates CPT from these fragments.

Lower bounds As can be inferred from the existence of padding arguments, as well as the more elaborate definability results for the CFI query and structures with Abelian colours, the unconventional definition of Choiceless Polynomial Time seems to contribute significantly to its strengths. However, in case CPT does not capture PTIME, exactly these particularities make it difficult to prove that inexpressibility. Nevertheless, the proof techniques developed since CPT has been defined already serve to show meaningful lower bounds for fragments of CPT.

The technique that has been studied most extensively is based on pebble games for infinitary logic. Formulae in Choiceless Polynomial Time basically consist of iterated terms creating hereditarily finite sets. Thus, if all the sets that are to be created are already present in the input structure, the CPT-formula can be simulated by a fixed-point iteration. Indeed, the fixed-point theorems shown by Blass, Gurevich and Shelah [13] yield a translation from CPT^- to FP over a structure enriched by the so-called *active* objects. The same technique transforms CPT-formulae (with counting) to FP+C.

Of course the set of objects used by a CPT-formula depends on the input structure itself. Thus, to show inexpressibility results, one has to show that any such structures over active objects are indistinguishable in fixed-point logic. In some cases, however, CPT-formulae can only define hereditarily finite sets with certain properties, which can be used to over-approximate the structure over the active objects. In the first inexpressibility proof by Blass, Gurevich and Shelah [13], this property is defined via the action of the automorphism group of the input structure: They show that CPT^--formulae over the empty signature can only define sets with a support of constant size.

Further, hereditarily finite sets are uniquely determined by their *form-matter-decomposition*: The *form* is a syntactic object that defines the set structure in infinitary logic, and the *matter* consists of the atoms in the support. This allows to transfer winning strategies for Duplicator in the bijective k-pebble game on the atoms to the game on the structures extended by the hereditarily finite sets supported by a small number of atoms. Thus it is possible to find structures whose extensions by sets with small support can be shown to be L^m-equivalent for suitable

m (depending on the size of the supports). For a more formal description of this proof technique, see Section 5.2.

Since, for sufficiently large structures over the empty signature, the structures over objects with small supports are L^m-equivalent, CPT^- cannot define Even. Blass, Gurevich and Shelah [13] also generalise their result to structures with unary relations. By a suitable reduction, this implies that existence of a perfect matching is not definable in CPT^-.

One of the most challenging parts of the proof is called the Support Theorem, stating that every object defined by a CPT^--program has a small support. Rossman [54] replaces combinatoric arguments by group-theoretic ones to obtain a simpler proof of that theorem. Further, he applies his method to show that there is a function problem in Ptime that cannot be defined in CPT, even with counting: Given a finite vector space, CPT can construct neither its dual nor the set of hyperplanes. The proof is based on the fact that the elements of the set to be constructed have no small support and thus cannot be defined by any CPT-program over a vector space. But note that the argument cannot be applied to decision problems, i. e. Boolean queries, because in that case, the objects in question would not necessarily have to be constructed by the program.

Dawar, Richerby and Rossman [23] further extend the original proof technique to show that, even though the CFI query is definable in CPT^-, this requires sets of arbitrary nesting depth. More precisely, CPT-formulae using only sets of bounded rank can define only sets with constant support. This again makes it possible to transfer winning strategies from the pebble game (now for C^k) over the atoms to the game on sets with bounded support. In addition to an extension of the proof technique, this constitutes one of the first results about a natural fragment of CPT other than CPT^-. Analysing such fragments provides a deeper understanding of which operations are necessary to obtain the full expressive power of CPT, ruling out simplifications of the formalism that are too weak to be themselves candidates for a logic for Ptime.

A completely different approach to lower bounds is Shelah's [56] zero-one-law for CPT^-. His proof uses *inductive systems* similar to Ehrenfeucht-Fraïssé-games, providing a game-theoretic characterisation of CPT^- itself instead of taking the detour via fixed-point logic. The approach is clarified by Blass and Gurevich [9, 11], who also provide additional proof details in [10].

For the precise statement of the zero-one-law, it is important to note that CPT was originally defined as a three-valued logic. A formula (or program, as specified in the original setting) may produce the value "true", "false" or "undecided", where the

latter occurs if no result can be defined within the polynomial bounds. Accordingly, the zero-one-law is phrased as follows: For every CPT$^-$-program, either almost all graphs produce the output true or undecided, or almost all graphs produce the output false or undecided.

The first inexpressibility results for CPT concerned structures with many automorphisms (like sets or coloured sets), whose difficulty for choiceless computation seems rather obvious—whenever a CPT-formula defines an object, it has to define its whole orbit. But the zero-one-law can be viewed as a result about rigid structures, because almost all finite graphs have no non-trivial automorphisms.

The zero-one-law was applied by Blass, Gurevich and Shelah [14] to show that counting is necessary to define determinants in CPT. The fact that the property that a matrix has determinant 1 does not have asymptotic probability 0 or 1 immediately implies the result. Further, since determinants are CPT-definable [12], it follows that the zero-one-law does not hold for full CPT with counting. However, we do not know of any attempts to adapt the inductive systems to full CPT.

3.2 Definition of Choiceless Polynomial Time

Even though the basic idea of set-theoretic computation was preserved, the definition of Choiceless Polynomial Time has evolved from the original formalism in Blass-Gurevich-Shelah's paper [13]. Their original definition, based on abstract state machines, still bears the closest resemblance to a machine model. This model combines set-theoretic operations into *programs* with control structures such as assigning sets to variables, and, most importantly, parallel execution of a program for all members of a set as well as iteration of programs. Parallel execution aims to cure the syntactic deficiency of choiceless computation: whereas CPT cannot make arbitrary choices, such as selecting a single, non-definable vertex from a graph, it can process all of them at once.

Later versions of CPT can simulate parallel computation by recursive manipulations of a single set. Still, the amount of parallelism, or equivalently the size of the sets defined, has to remain within the polynomial bounds. Having to process all members of a definable set at once poses the main difficulty in constructing CPT-formulae.

Defining the semantics of CPT in terms of abstract state machines requires some technical overhead. Rossman [54], however, provides a more concise definition, reducing the formalism of programs to iteration of a single operation on hereditarily

finite sets. The more recent definition by Grädel and Grohe [26] we present here moves the syntax of CPT even closer to that of a logic in the classical sense.

As mentioned above, the number of objects that can be processed simultaneously has to be kept small in order to remain in PTIME. Thus, CPT imposes a polynomial bound on the number and size of sets created. Recall that, according to Gurevich's definition of a logic, the syntax has to be decidable. This justifies the definition of CPT as a PTIME-restriction—a logic where a polynomial bound is part of every sentence. Note, however, that the logic underlying CPT is already strong enough to define whether a set satisfies the bound (see Section 3.4), so the implicit evaluation of the bound does not add to the expressive power.

Definition 3.1 (Syntax of Choiceless Polynomial Time)**.** CPT is the set of pairs (t, p) (respectively (φ, p)), where t is a BGS-term, φ is a BGS-formula and $p\colon \mathbb{N} \to \mathbb{N}$ is a polynomial.

For a vocabulary σ, BGS$[\sigma]$ is defined inductively as follows:

- Every variable x is a term.
- $\emptyset$ and Atoms are terms.
- If $s_1, \ldots, s_k$ are terms and $R \in \sigma$ is a k-ary relation symbol, then $Rs_1 \ldots s_k$ is a formula.
- Boolean combinations of formulae are again formulae.
- If r, s are terms, then Unique(s), Pair(r, s), Union(s) and Card(s) are terms.
- If s, t are terms, then $s = t$ and $s \in t$ are formulae.
- If r and s are terms, x is a variable that is not free in r, and φ is a formula, then $\{s : x \in r : \varphi\}$ is a *comprehension term*.
- If s is a term with only one free variable, then s^* is an *iteration term*.

Terms and formulae that do not contain iteration terms as subterms are also called *ordinary* terms and formulae. Free variables are defined as usual, where the variable x is bound in the comprehension term $\{s : x \in r : \varphi\}$, and an iteration term s^* binds the free variable of s.

As usual, a sentence is a formula without free variables. In the spirit of the original CPT-definition, we sometimes refer to sentences as programs or algorithms and describe their mechanics accordingly. The fragments of BGS and CPT without counting, that is, without terms of the form Card(s), are denoted BGS$^-$ and CPT$^-$, respectively.

The semantics of formulae with and without polynomial bounds only differs in the evaluation of iteration terms: The polynomial bound restricts the number of

iterations and the size of the sets occurring as stages. For the sake of simplicity, we first define the semantics of BGS, i. e. of terms and formulae without bounds, and then state the differences for iteration terms with bounds.

Definition 3.2 (Semantics of BGS)**.** Let $\mathbf{A}$ be a σ-structure, let $\overline{a} = (a_1, \ldots, a_k)$ be a tuple in $\mathrm{HF}(A)^k$ and $\beta \colon X \supseteq \{x_1, \ldots, x_k\} \to A$ with $\beta(x_i) = a_i$. For a term t with free variables $x_1, \ldots, x_k$, we define the value $[\![t]\!]^{\mathbf{A},\beta} \in \mathrm{HF}(A)$. We also interpret $[\![t]\!]^{\mathbf{A}}$ as a function from $\mathrm{HF}(A)^k$ to $\mathrm{HF}(A)$ and write $[\![t]\!]^{\mathbf{A}}(\overline{a})$ to denote $[\![t]\!]^{\mathbf{A},\beta}$.

- $[\![x_i]\!]^{\mathbf{A}}(a_i) = a_i$.
- $[\![\emptyset]\!]^{\mathbf{A}} = \emptyset$ and $[\![\mathrm{Atoms}]\!]^{\mathbf{A}} = A$.
- $\mathbf{A}, \beta \models Rs_1 \ldots s_k$ if, and only if, $([\![s_1]\!]^{\mathbf{A},\beta}, \ldots, [\![s_k]\!]^{\mathbf{A},\beta}) \in R^{\mathbf{A}}$.
- Boolean combinations of formulae are interpreted as usual.
- $[\![\mathrm{Unique}(s)]\!]^{\mathbf{A}}(\overline{a}) = \begin{cases} a, & \text{if } [\![s]\!]^{\mathbf{A}}(\overline{a}) = \{a\}, \\ \emptyset, & \text{otherwise.} \end{cases}$
- $[\![\mathrm{Pair}(r, s)]\!]^{\mathbf{A}}(\overline{a}) = \{[\![r]\!]^{\mathbf{A}}(\overline{a}), [\![s]\!]^{\mathbf{A}}(\overline{a})\}$.
- $[\![\mathrm{Union}(s)]\!]^{\mathbf{A}}(\overline{a}) = \bigcup_{b \in [\![s]\!]^{\mathbf{A}}(\overline{a})} b$.
- $[\![\mathrm{Card}(s)]\!]^{\mathbf{A}}(\overline{a}) = [|[\![s]\!]^{\mathbf{A}}(\overline{a})|]$.
- $\mathbf{A}, \beta \models s = t$ if, and only if, $[\![s]\!]^{\mathbf{A},\beta} = [\![t]\!]^{\mathbf{A},\beta}$.
- $\mathbf{A}, \beta \models s \in t$ if, and only if, $[\![s]\!]^{\mathbf{A},\beta} \in [\![t]\!]^{\mathbf{A},\beta}$.
- $[\![\{s : x \in r : \varphi\}]\!]^{\mathbf{A}}(\overline{a}) = \{[\![s]\!]^{\mathbf{A}}(\overline{a}, b) : b \in [\![r]\!]^{\mathbf{A}}(\overline{a}) \text{ with } \mathbf{A}, \beta[x \mapsto b] \models \varphi\}$.
- For the semantics of an iteration term t^*, define the sequence $(t^i)_{i \in \mathbb{N}}$ as $t^0 = \emptyset$ and $t^{i+1} = t\,[x/t^i]$, where x is the free variable of t. Then

$$[\![t]\!]^{\mathbf{A}} = \begin{cases} [\![t^\ell]\!]^{\mathbf{A}}, & \text{for the least } \ell \text{ with } [\![t^\ell]\!]^{\mathbf{A}} = [\![t^{\ell+1}]\!]^{\mathbf{A}} \\ & \text{if such an } \ell \text{ exists,} \\ \emptyset, & \text{otherwise.} \end{cases}$$

  The intermediate values $[\![t^i]\!]^{\mathbf{A}}$ are the *stages* of the iteration, and the value ℓ for which the fixed point is reached is its *length* $\mathrm{len}(t^*, \mathbf{A})$.

Whenever the structure $\mathbf{A}$ can be inferred from the context, we omit the superscript and write $[\![t]\!]$ instead of $[\![t]\!]^{\mathbf{A}}$. Note that formulae can equivalently be defined as *Boolean terms* taking the values $\emptyset$ and $\{\emptyset\}$. This eliminates the need to distinguish between terms and formulae when reasoning about them inductively.

The polynomial bound in terms and formulae of CPT guarantees that evaluation in Ptime is possible. To achieve this, both the number of computation steps and the space required for representing the occurring sets has to be small enough.

Any straightforward encoding of a set encompasses encodings of all elements of its transitive closure. Thus the polynomial bound restricts the transitive closure of every stage of the iteration.

Note that it suffices to apply the bound to iteration terms, since the set operations defined by other terms only lead to polynomial growth.

Definition 3.3 (Semantics of CPT)**.** With the exception of iteration terms, the semantics of every CPT-term (t, p) is defined by replacing t by (t, p) in the definition. If t^* is an iteration term, we define

$$[\![(t^*, p)]\!]^{\mathbf{A}} = \begin{cases} [\![t^\ell]\!]^{\mathbf{A}}, & \text{for the least } \ell \leq p(|A|) \text{ with } [\![t^\ell]\!]^{\mathbf{A}} = [\![t^{\ell+1}]\!]^{\mathbf{A}} \\ & \text{and } \left|\mathrm{tc}\left([\![t^i]\!]^{\mathbf{A}}\right)\right| \leq p(|A|) \text{ for all } i \leq \ell \text{ if such an } \ell \text{ exists,} \\ \emptyset, & \text{otherwise.} \end{cases}$$

The *length of the iteration* $\mathrm{len}(t^*, p, \mathbf{A})$ is the least ℓ with $[\![t^*]\!]^{\mathbf{A}} = [\![t^\ell]\!]^{\mathbf{A}}$. In particular, $\ell = 0$ if the second case applies.

In earlier definitions of CPT [13, 54], the polynomial bound restricts the number of objects *activated* by a term or formula. Intuitively, this is a set of hereditarily finite objects that are constructed when evaluating the term or formula. Formalising the set of objects necessary for evaluating a CPT-formula also proves useful beyond the definition of the polynomial bound:

The ability to recursively build up new sets becomes superfluous if these sets are already part of the structure over which a formula is evaluated. So, over such a structure, every CPT-formula can be translated to FP+C. This translation is the first step of the proof technique for inexpressibility results established by Blass, Gurevich and Shelah [13]. The notion of active objects describes which objects have to be present in the structure for the translation to be possible.

Definition 3.4 (Active objects)**.** Let (t, p) be a (possibly Boolean) CPT$[\sigma]$-term, let $\mathbf{A}$ be a σ-structure, and let S be the set of terms s such that the iteration term s^* is a subterm of t. The *active objects* of (t, p) over $\mathbf{A}$ are

$$\mathrm{act}(t, p, \mathbf{A}) = A \cup \bigcup_{s \in S} \bigcup_{i=0}^{\mathrm{len}(s^*, p, \mathbf{A})} \mathrm{tc}\left([\![s^i]\!]^{\mathbf{A}}\right).$$

Further, let $\mathbf{act}(t, p, \mathbf{A})$ be the substructure of $\mathrm{HF}(\mathbf{A})$ over the domain $\mathrm{act}(t, p, \mathbf{A})$.

Note that this definition of active objects differs from the one usually given in the literature. The standard definition (see, for instance, [54]) is more detailed in

the sense that it encompasses all objects that play any part in the computation. For example, a term is said to activate all objects that are activated by any of its subterms, regardless of whether these objects occur in the value of the term or in an iteration.

Our definition, however, turns out to be sufficient for the translation of CPT to weaker logics, as shown at the end of Section 3.4.

3.2.1 Examples and frequently used terms

In later chapters, CPT-formulae will mostly be described in an abstract way instead of explicitly written down in the correct syntax. To convey an intuition of what is possible in CPT, we provide some example formulae defining properties that can already be expressed in weaker logics.

The BGS-formula $\{x : x \in \text{Atoms} : Px\} \neq \emptyset$ is equivalent to the FO-formula $\exists x Px$. Similarly, $\exists x \exists y Exy$ can be translated to

$$\{x : x \in \text{Atoms} : \{y : y \in \text{Atoms} : Exy\} \neq \emptyset\} \neq \emptyset.$$

The following BGS-formula expresses that the vertex x in a graph has less outgoing than incoming edges:

$$\text{Card}(\{y : y \in \text{Atoms} : Exy\}) \in \text{Card}(\{y : y \in \text{Atoms} : Eyx\}).$$

Since these terms do not have iteration subterms, the polynomial bound can be chosen arbitrarily; it will not affect the semantics.

To illustrate the use of iteration terms, we construct a term that defines the set of all vertices of a graph reachable from some element in the relation S. Evaluation of iteration terms always starts with the empty set, so S will be added to the current set in every stage. Any non-empty set will be extended by all vertices that are reachable from some vertex in the current set via a single edge.

The case that the current value of x is a non-empty set of vertices is covered by the following term:

$$t_{\text{extend}}(x) = \text{Union}(\text{Pair}(\{x, \{y : y \in \text{Atoms} : \{z : z \in x : Ezy\} \neq \emptyset\}\})).$$

Then the iteration t^* of

$$t(x) = \text{Union}(\text{Pair}(\text{Union}(\text{Pair}(x, \{y : y \in \text{Atoms} : Sy\})), t_{\text{extend}}))$$

defines the desired set. This can be written as the CPT-term $(t^*, n \mapsto n+1)$.

The set of active objects of this term in a structure $\mathbf{A}$ consists of all atoms reachable from a vertex in S, and, for every $i < |A|$, the set of vertices that are reachable from S in i many steps.

As can be seen from the previous term, simple operations (in this case, the union of three sets) can quickly become difficult to read. In the remainder of this section, we define several terms that we later use as shorthand for common operations. Most of these operations were first defined in [13].

Let us start with the basic set-theoretic operations. The set $\{x\}$ is defined as $\mathrm{Pair}(x, x)$. The union $x \cup y$ of sets x, y is expressed by the term $\mathrm{Union}(\mathrm{Pair}(x, y))$. Their intersection is defined by $\{z : z \in x : z \in y\}$, and the term $\{z : z \in x : z \notin y\}$ defines $x \setminus y$. We use the abbreviations $\neq$ and $\notin$ in the obvious way.

The Kuratowski pair $\langle x, y\rangle = \{x, \{x, y\}\}$ is expressed by $\mathrm{Pair}(x, \mathrm{Pair}(x, y))$. If x is an ordered pair of that form, then

$$\pi_1(x) = \mathrm{Unique}(\{y : y \in \mathrm{Union}(x) : y \in x\})$$

defines the first, and

$$\begin{aligned}\pi_2(x) = &\,\mathrm{Unique}(\{y : y \in \mathrm{Union}(x) : y \neq \pi_1(x) \wedge y \notin \pi_1(x)\}) \\ &\cup \mathrm{Unique}(\{y : y \in \mathrm{Union}(x) : y \notin \pi_1(x)\})\end{aligned}$$

defines the second entry of x. Depending on whether the entries of the pair are distinct, one of the Unique-statements in π_2 always has the value $\emptyset$.

The cartesian product $x \times y$ as a set of Kuratowski pairs is then defined by

$$\mathrm{Union}(\{\{\langle x', y'\rangle : y' \in y\} : x' \in x\}).$$

Note that $\{s : x \in r\}$ is used as a shorthand for the comprehension term $\{s : x \in r : \varphi_{\mathrm{true}}\}$ where φ_{true} is a tautology. The term for the cartesian product can be nested to define comprehension of the form

$$\{s(x_1, \ldots, x_k) : \langle x_1, \ldots, x_k\rangle \in y_1 \times \cdots \times y_k : \varphi(x_1, \ldots, x_k)\}.$$

We also write $\{s(x_1, \ldots, x_k) : x_1 \in y_1, \ldots, x_k \in y_k : \varphi(x_1, \ldots, x_k)\}$ instead.

Further, the term defining Kuratowski pairs can be nested to obtain longer tuples of the form $\langle\langle a_0, \ldots, a_{k-1}\rangle, a_k\rangle$ for fixed k. Projections π^k_i for $1 \leq i \leq k$ can then be defined inductively as $\pi^k_k(x) = \pi_2(x)$ and $\pi^k_i(x) = \pi^{k-1}_i(\pi_1(x))$ for $i < k$. Note

that the representation $\{\langle 1, a_1\rangle, \ldots, \langle k, a_k\rangle\}$ can be defined just as easily. For fixed k both representations need polynomial space, so they can be used interchangeably.

Finally, to express certain "algorithms" in CPT, we introduce a term $\text{if}_\varphi(t_1, t_2)$ whose value is t_1 if φ holds and t_2 otherwise. So

$$\text{if}_\varphi(t_1, t_2) = \{t_1 : x \in \{\emptyset\} : \varphi\} \cup \{t_2 : x \in \{\emptyset\} : \neg\varphi\},$$

where $x \notin \text{free}(t_1) \cup \text{free}(t_2)$.

To ease notation, we use the operations defined here as if they were part of the syntax.

3.3 Interpretation logic

The unconventional syntax, especially in the original definition, makes CPT harder to understand and analyse than other logics. This provoked the question whether there is a way to characterise CPT in a more accessible way. Indeed it turned out [55, 27] that CPT-formulae are just iterated first-order interpretations (with polynomial bounds). This characterisation as *polynomial-time interpretation logic* PIL not only makes CPT easier to define, it also enables new ways to study the structure of the logic.

In particular, it gives rise to a number of natural fragments and extensions of CPT, which we explore further in Chapter 4 as well as Section 5.5. This allows, for instance, to embed FP+C into PIL in a natural way.

When defining PIL, we proceed as with CPT and start with the underlying logic without polynomial bounds.

The core of a PIL-program is an FO+H-interpretation $\mathcal{I}_{\text{step}}$ that is applied inductively. This interpretation may use a different signature than the input structure, so an initial interpretation $\mathcal{I}_{\text{init}}$ is applied to the input structure first. Both the halting condition for the iteration of $\mathcal{I}_{\text{step}}$ and the truth value of the program are defined via FO+H-formulae that are evaluated over the structures constructed during the iteration.

Definition 3.5. Let σ, τ be relational signatures. An IL$[\sigma, \tau]$-sentence (or program) is a tuple $\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}})$, where $\mathcal{I}_{\text{init}}$ is an FO+H$[\sigma, \tau]$-interpretation, $\mathcal{I}_{\text{step}}$ is an FO+H$[\tau, \tau]$-interpretation, both $\mathcal{I}_{\text{init}}$ and $\mathcal{I}_{\text{step}}$ are parameter-free, and ψ_{halt} and ψ_{out} are FO+H$[\tau]$-sentences.

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

We call σ the *input signature*, and τ the *output signature* of Π. If $\sigma \subseteq \tau$, we can omit $\mathcal{I}_{\text{init}}$ and consider programs of the form $(\mathcal{I}, \psi_{\text{halt}}, \psi_{\text{out}})$ instead. Then the initial state $\mathbf{A}_0$ of the run of $(\mathcal{I}, \psi_{\text{halt}}, \psi_{\text{out}})$ on $\mathbf{A}$ is the τ-expansion of $\mathbf{A}$ where all relations in $\tau \setminus \sigma$ are empty. Though we will use this simpler syntax as well, some of the results shown in this chapter explicitly concern restricted output signatures.

Similarly to CPT, the polynomial bounds guarantee that formulae can be evaluated in PTIME. Thus, we place restrictions on the number of computation steps, that is, the length of the run and the space required for the intermediate objects, which are now the structures $\mathbf{A}_i$ occurring as states. If the time and space bounds are not satisfied anymore, the computation halts. Together with the fact that first-order interpretations can be evaluated in PTIME, this ensures that our logic is contained in PTIME.

Definition 3.6. *Polynomial-time interpretation logic* PIL is the PTIME-restriction of IL with the following semantics: Let $\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}})$. The run of (Π, p) on $\mathbf{A}$ is the maximal initial segment $(\mathbf{A}_i)_{i\leq n}$ of the run $(\mathbf{A}_i)_{i<\kappa}$ of Π on $\mathbf{A}$ with $n \leq p(|A|)$ and $|A_i| \leq p(|A|)$ for all $i \leq n$. Then $\Pi_p(\mathbf{A}) = \mathbf{A}_n$, and $\mathbf{A} \models (\Pi, p)$ if, and only if, $\Pi_p(\mathbf{A}) \models \psi_{\text{out}}$.

We also say that (Π, p) *accepts* $\mathbf{A}$ if $\mathbf{A} \models (\Pi, p)$. Otherwise, (Π, p) *rejects* $\mathbf{A}$.

For any logic L, the variant of PIL where FO+H is replaced by L is denoted PIL(L). Analogously to CPT$^-$, PIL$^-$ is the variant of PIL without counting, i. e. PIL$^-$ = PIL(FO).

Before we discuss equivalence between PIL and CPT and some consequences of the proof, we provide two simple example programs. A straightforward example requiring some kind of iteration is reachability in a graph. Since PIL operates on relational structures, we consider graphs with unary relations S and T, and define the property that there is a path from a vertex in S to a vertex in T.

Example 3.7. The program $(\mathcal{I}, \psi_{\text{halt}}, \psi_{\text{out}})$ successively enlarges the relation S by all elements that are reachable from S via a single edge. Then the formula $\psi_{\text{out}} = \exists x(Sx \wedge Tx)$ checks whether an element of T has been reached. Now let $\mathcal{I} = (\varphi_\delta, \varphi_=, \varphi_E, \varphi_S, \varphi_T)$, where all formulae except for φ_S are trivial. The relation S is updated by the formula $\varphi_S = Sx \vee \exists y(Sy \wedge Eyx)$. The program halts if no

more elements can be added to S, so $\psi_{\text{halt}} = \forall x(\varphi_S(x) \to Sx)$. Together with the identity as the polynomial bound, this program behaves correctly.

Note that reachability can already be defined with one-dimensional interpretations, that is, without creating any new elements. To illustrate how creation of equivalence classes of tuples can be used, we present a program for EVEN. Because of the Härtig quantifier, counting in general requires a number sort. So we use the ability of PIL to create new objects to successively build up a second sort, tracking the parity of the size of that sort.

Example 3.8. Our program is of the form $((\mathcal{I} = (\varphi_\delta, \varphi_=, \varphi_{C_0}, \varphi_{C_1}), \psi_{\text{halt}}, \psi_{\text{out}}), p)$. In every iteration a single element is added to the domain. This can be achieved by encoding every element a of the current structure as the tuple (a, a), and the new element by the equivalence class $\{(a, b) : a \neq b\}$. So the domain formula is $\varphi_\delta(x, y) = (x = x)$, and the equality formula is

$$\varphi_=(x_1, y_1, x_2, y_2) = (x_1 = y_1 \wedge x_2 = y_2 \wedge x_1 = x_2) \vee (x_1 \neq y_1 \wedge x_2 \neq y_2).$$

The relations C_0 and C_1 serve two purposes: They mark the new elements, and they keep track of whether the number of new elements is even. When the first new element is created, it is added to C_1. In subsequent iterations, all new elements are added to C_0 if they were in C_1 before, and vice versa. Formally, the relations are defined as follows:

$$\varphi_{C_0}(x, y) = (x = y \wedge C_1 x) \vee (x \neq y \wedge \exists x C_1 x).$$

and

$$\varphi_{C_1}(x, y) = (x = y \wedge C_0 x) \vee (x \neq y \wedge \forall x \neg C_1 x),$$

The program halts if as many elements have been created as there are elements of the input structure. So $\psi_{\text{halt}} = \mathsf{H}x.(C_0 x \vee C_1 x).\neg(C_0 x \vee C_1 x)$. Finally, ψ_{out} checks whether the number of elements created is even, so $\psi_{\text{out}} = \exists x C_0 x$.

In the final state, the number of old and new elements coincides, so the polynomial $n \mapsto 2n$ suffices as a bound.

Interpretations and ordinary BGS**-terms** Now that we have seen some examples of how PIL is used, we take a closer look at how it relates to CPT. Most importantly, as shown already in [55], PIL is indeed a characterisation of CPT, which holds for the variants with and without counting.

Theorem 3.9 ([55, 27])**.**

1. PIL $\equiv$ CPT.
2. PIL$^-$ $\equiv$ CPT$^-$.

The proof, especially in the way it is presented in [27], yields first insights into which mechanisms of CPT are responsible for its expressive power. The basic components of CPT's syntax are set-theoretic operations (i. e. ordinary terms), iteration and polynomial bounds. The simulation of CPT-formulae by PIL-programs shows that creation of sets is of little value without the ability to do so recursively. More precisely, the logic over ordinary BGS-terms collapses to a variant of FO+H.

To see this, we analyse the key lemma from the equivalence proof in [27]. Stated informally, every ordinary term can be simulated by a sequence of FO+H-interpretations.

Note that the values of ordinary terms, their subterms and their free variables can be arbitrary hereditarily finite sets. But a term can always access the input structure using the constant Atoms. So a structure permitting simulation of an ordinary term has to contain a copy of the input structure, as well as some ordinals for the counting operation. Further requirements are necessary to guarantee the existence of polynomial bounds when simulating iteration terms, but we omit these here.

Definition 3.10. Let $\mathbf{A}$ be a σ-structure. A structure $\mathbf{S} = (S, \tau_{\text{state}})$, where $\sigma \subseteq \tau_{\text{state}}$, is an $\mathbf{A}$-*state* if it is isomorphic to the τ_{state}-expansion of a transitive substructure of HF(A) containing the cardinality of each of its elements.

Simulation of ordinary terms by sequences of interpretations is now defined with respect to $\mathbf{A}$-states. We treat formulae as Boolean terms to avoid the case distinction in the definition.

Definition 3.11. Let t be an ordinary BGS$[\sigma]$-term with free variables $\overline{x} = x_1, \ldots, x_k$ and let $\tau_{\text{state}} \supseteq \sigma \cup \{A, \in, \emptyset, R_t\}$, where R_t is a $(k+1)$-ary relation symbol. A sequence $\mathcal{I}_1, \ldots, \mathcal{I}_m$ of $[\tau_{\text{state}}, \tau_{\text{state}}]$-interpretations *simulates* t if for all σ-structures $\mathbf{A}$ and all $\mathbf{A}$-states $\mathbf{S}$ of vocabulary τ_{state}, the structure $\mathbf{S}_{\text{out}} \coloneqq \mathcal{I}_1 \circ \cdots \circ \mathcal{I}_m(\mathbf{S})$ has the following properties (up to isomorphism):

- $\mathbf{S}_{\text{out}}$ is an $(\mathbf{A}, t)$-state,
- $\mathbf{S} \subseteq \mathbf{S}_{\text{out}}$,
- $R_t^{\mathbf{S}_{\text{out}}} = \{(a_1, \ldots, a_k, [\![t]\!]^{\mathbf{A}}(a_1, \ldots, a_k)) : (a_1, \ldots, a_k) \in S\}$.

Ordinary BGS-terms can then be translated to FO+H-interpretations by induction. It should be noted that omitting the additional requirements from [27] only simplifies the proof.

Lemma 3.12 ([27])**.** *For every ordinary* BGS*-term t, there is a sequence of* FO+H*-interpretations simulating t.*

In contrast to what the original formalisation of the simulation (Definition 10 and Lemma 11 in [27]) suggests, a single FO+H-interpretation does in general not suffice for this simulation. To see this, recall that FO+H is not closed under interpretations. So the concatenation of interpretations is not guaranteed to be expressible as an interpretation again. Since concatenation of interpretations can clearly be simulated in PIL, this does not change the main result.

For the result about ordinary terms, however, we have to choose a logic that permits concatenation of interpretations. Since the interpretation lemma holds for FO, only the translation of the Härtig quantifier fails. In the interpreted structure, formulae with Härtig quantifiers would have to count the number of equivalence classes of tuples of the original structure. Augmenting FO+H with extended Härtig quantifiers that have exactly that ability results in FO+H$_\sim$, a logic that is closed under interpretations.

Further, PIL needs iteration to create the number sort. But FO-interpretations can create a number sort of polynomial length (and hence a sufficient number of ordinals) from a linear order of the size of the input structure. So (FO+H$_\sim$)* suffices to create states and simulate sequences of interpretations with a single interpretation.

Corollary 3.13. *The fragment of* BGS *containing only ordinary terms is equivalent to* (FO+H$_\sim$)**, and ordinary* BGS$^-$*-terms are equivalent to* FO*.*

We can conclude from Corollary 3.13 that iteration is a crucial ingredient to the expressive power of CPT: Creation of hereditarily finite sets as it can be done with ordinary terms is still included in (FO+H$_\sim$)*, and thus in, for instance, FP+C.

Interpretation logic over stronger logics When specifying programs in interpretation logic, it is sometimes convenient to use interpretations in a stronger logic than FO+H. Therefore we show that PIL is robust under exchanging its underlying logic, as long as that logic is not stronger than PIL itself.

Simulating a logic stronger than FO+H, however, requires the additional capabilities of interpretation logic, i. e. iterated interpretations. But to translate

interpretations in that stronger logic to regular PIL, one has to take into account that interpretations consist of formulae with free variables. So we state the result in a way that the PIL-formulae can handle free variables.

Lemma 3.14. *Let* $\mathrm{L} \geq \mathrm{FO{+}H}$ *be a logic such that for every formula* $\varphi(x_1, \dots, x_k) \in \mathrm{L}[\sigma]$*, there is a* $\mathrm{PIL}[\sigma, \tau]$*-program* Π *for some signature* τ *such that, for every* σ*-structure* $\mathbf{A}$*,* $\Pi(\mathbf{A})$ *is isomorphic to the expansion* $\mathbf{B}$ *of* $\mathbf{A}$ *by a* k*-ary relation* R_φ *with* $R_\varphi^{\mathbf{B}} = \varphi^{\mathbf{B}}$*. Then* $\mathrm{PIL}(\mathrm{L}) \equiv \mathrm{PIL}$*.*

Proof. Since, by assumption, $\mathrm{L} \geq \mathrm{FO{+}H}$, $\mathrm{PIL} \leq \mathrm{PIL}(\mathrm{L})$.

For the other direction, let $\Pi = (\mathcal{I}, \psi_{\text{halt}}, \psi_{\text{out}})$ be a $\mathrm{PIL}(\mathrm{L})$-program (w. l. o. g. we omit the initial interpretation). We construct a PIL-program $\Pi' = (\mathcal{I}', \psi'_{\text{halt}}, \psi'_{\text{out}})$ equivalent to Π.

To evaluate a formula φ occurring in $\mathcal{I}$, Π' creates a copy of the current state and runs the program for φ that exists by assumption, relativised to that copy. When all subprograms have been run, Π' can apply $\mathcal{I}$ to the original copy using the relations defined by the subprograms. The resulting state is a state of the run of Π. The formulae ψ_{halt} and ψ_{out} can be simulated in a similar way. □

3.4 Normal forms

In addition to creation of higher-order objects, CPT differs from more conventional logics because of the explicit polynomial bound. As indicated by the results about padded structures, the polynomial bound significantly contributes to the strength of CPT as a candidate for capturing Ptime. But it also complicates notation, and, more importantly, poses another difficulty in showing inexpressibility results.

A question asked frequently about CPT is whether the explicit polynomial bound can be avoided. One may wish for a formalism that guarantees polynomial-time evaluation by some natural syntactic criterion, such as the restriction in least fixed-point logic that relations defined by a fixed-point formula may occur only positively. Although a characterisation of CPT that is natural in that sense currently seems out of reach, it has frequently been suggested that the polynomial bound can at least be hidden in other components of the formula.

The basic idea is to check the polynomial bound using a BGS-formula instead of adding a bound from the outside. To do this, the formula has to define the value of the polynomial, track the number of stages of every iteration and determine the size of the transitive closure in every stage. Our first normal form implements this

well-known idea, extending the time-explicit programs from [13] and the mechanism for counting computation steps from [26].

It turns out that the actual construction is rather involved. Using interpretation logic instead of BGS, however, a similar effect can be achieved more easily. Furthermore, the normal form for BGS converts the external bound to a polynomial that is defined by a formula, obscuring the syntax even more. In contrast, there is a natural way to extract a polynomial from an IL-program: A k-dimensional initial interpretation can create up to n^k many elements from an input structure of size n. Indeed, the polynomial bound of PIL-programs can be hidden in the dimension of the initial interpretation.

Of course this still means that the program has to explicitly check whether the polynomial bound is satisfied in every iteration step. But checking the bounds requires less effort in IL than in BGS. One can even infer simple syntactic criteria that guarantee that an IL-program is in the required form.

Our final normal form concerns the translation of CPT to FP+C evaluated over the active objects, which is a crucial part of the standard technique for showing inexpressibility results explained in Section 3.1 and, in more detail, in Section 5.2. In that context, the normal form ensures that all objects required for simulating the original CPT-formula are active, i. e. occur in some stage of an iteration term.

On the one hand, we show that our definition of the active objects suffices for the translation. On the other hand, the result can be slightly improved: Instead of fixed-point logic, we show that CPT can be translated to $(\mathrm{FO+H}_\sim)^*$ and CPT^- to FO. Whereas the former would require accessible games for $(\mathrm{FO+H}_\sim)^*$ to prove inexpressibility results, the latter may lead to simpler proofs of results about CPT^-.

Since many known definability results for CPT are rather complex, the actual formulae are rarely written down. Therefore we additionally use the normal forms as examples of how concrete formulae in CPT and PIL can be constructed.

Implicit CPT The goal is to create BGS-formulae that ensure that every iteration conforms to a given polynomial bound. So the formula has to keep track of the number of iterations as well as the size of the transitive closure of every stage. These values are compared to the value of the polynomial. As the purpose of evaluating the polynomial is to check the bounds of iteration terms, it is most convenient to express the value of the polynomial by an ordinary BGS-term.

Lemma 3.15. *For every polynomial p with coefficients from $\mathbb{N}$, there is an ordinary* BGS*-term t_p such that, for every structure $\mathbf{A}$, $[\![t_p]\!]^{\mathbf{A}} = [p(|A|)]$.*

Proof. It suffices to show that we can evaluate every monomial $c \cdot n^k$ and define the sum of the results, i. e. the sum of two ordinals. For $p\colon n \mapsto c \cdot n^k$, let $t_p = \mathrm{Card}([c] \times \mathrm{Atoms}^k)$. The sum of ordinals x, y can be defined by the term $\mathrm{Card}(\{\langle z, \emptyset\rangle : z \in x\} \cup \{\langle z, \{\emptyset\}\rangle : z \in y\})$. □

Further, we need to define the transitive closure of every stage to check whether its size satisfies the bound. We again aim to define the transitive closure without using another iteration term. Later in the proof we will only consider iteration terms t^* where t is an ordinary term. So it suffices to find, for every ordinary term t, a term defining the transitive closure of its value.

For a comprehension term $t = \{s : x \in r : \varphi\}$, the transitive closure of $[\![t]\!]$ contains the transitive closure of $[\![s]\!](a)$ for every $a \in [\![r]\!]$. To use the term for $\mathrm{tc}([\![s]\!](a))$ (which will be obtained from the induction hypothesis) it is, however, necessary to encode information about the transitive closure of a in the value of the free variable (consider, for instance, the case $s(x) = x$).

So there has to be a way to define the transitive closure of every element of the value $[\![r]\!]$. This can, of course, easily be defined using iteration terms. But, since we aim to use only ordinary terms, we instead define a mapping that associates with every element of the transitive closure the transitive closure of that element. Therefore, for a structure $\mathbf{A}$, we define the function $F_{\mathrm{tc}}\colon \mathrm{HF}(A) \to \mathrm{HF}(A)$ as $F_{\mathrm{tc}}(a) = \{\langle b, \mathrm{tc}(b)\rangle : b \in \mathrm{tc}(a)\}$.

Lemma 3.16. *For every ordinary* BGS[σ]*-term t with free variables $x_1, \dots, x_k$, there is an ordinary* BGS[σ]*-term t_F such that, for every σ-structure $\mathbf{A}$ and all $a_1, \dots, a_k \in \mathrm{HF}(A)$, $[\![t_F]\!]^{\mathbf{A}}(\langle a_1, F_{\mathrm{tc}}(a_1)\rangle, \dots, \langle a_k, F_{\mathrm{tc}}(a_k)\rangle) = F_{\mathrm{tc}}([\![t]\!]^{\mathbf{A}}(a_1, \dots, a_k))$.*

Proof. First note that, given the term t_F, the transitive closure of the value of t can be extracted using the term $t_{\mathrm{tc}} = \pi_2(\mathrm{Unique}(\{x : x \in t_F : \pi_1(x) = t\}))$, if the values of the free variables are of the correct form.

We construct the terms t_F by induction on t. The cases $t = x$, $t = \emptyset$ and $t = \mathrm{Atoms}$ are trivial. For $t = \mathrm{Pair}(r, s)$, let $t_F = r_F \cup s_F \cup \{\langle t, r_{\mathrm{tc}} \cup s_{\mathrm{tc}} \cup \{t\}\rangle\}$. For $t = \mathrm{Union}(s)$, we use the fact that the transitive closure of all elements of $[\![s]\!]$ can already be defined, and let $t_F = s_F \setminus \{\langle s, s_{\mathrm{tc}}\rangle\}$. Similarly, for $t = \mathrm{Unique}(s)$, we define $t_F = \mathrm{if}_{t=\emptyset}(\{\langle \emptyset, \{\emptyset\}\rangle\}, s_F \setminus \{\langle s, s_{\mathrm{tc}}\rangle\})$.

Finally, for the case $t = \{s : x \in r : \varphi\}$, we use the term s_F to define $F_{\mathrm{tc}}(a)$ for every $a \in [\![t]\!]$. Since the value of the free variable x of s_F has to be of the correct form, we extract $F_{\mathrm{tc}}(a)$ for every $a \in [\![r]\!]$ from the term r_F as follows:

$$r_{F(x)}(x) = \{y : y \in r_F : \pi_1(y) \in \pi_2(\mathrm{Unique}(\{z : z \in r_F : \pi_1(z) = x\}))\}.$$

Then

$$t_F = \{s_F(\langle x, r_{F(x)}(x)\rangle) : x \in r : \varphi(x)\}$$
$$\cup\, \{\langle t, \{t\} \cup \mathrm{Union}(\{s_{\mathrm{tc}}(\langle x, r_{F(x)}(x)\rangle) : x \in r : \varphi(x)\})\rangle\}. \qquad \square$$

It follows that, with terms of the form t_{tc} as defined in the proof of Lemma 3.16, the transitive closure of the value of an ordinary term can be defined. Next, we aim to translate CPT-terms into a form where, at every stage of the iteration terms, the transitive closure of the current value is compared to the polynomial bound. Since t_{tc} is only defined for ordinary terms t, we define a translation reducing the nesting depth of iteration terms. Note that alternative definitions of CPT only allow a single iteration term. So our proof also demonstrates how to show equivalence between these definitions.

Lemma 3.17. *For every* CPT$[\sigma]$*-term* (t, p) *(including formulae as Boolean terms), there is an equivalent* BGS*-term* t' *and a polynomial* p' *such that at most one iteration term* s^* *occurs in* t' *and, for every* σ*-structure* $\mathbf{A}$*,* $\mathrm{len}(s, \mathbf{A}) \leq p'(|A|)$ *and* $|\,\mathrm{tc}([\![s^j]\!]^{\mathbf{A}})| \leq p'(|A|)$ *for* $1 \leq j \leq \mathrm{len}(s, \mathbf{A})$*.*

Proof. The most important steps of the translation are validation of the polynomial bounds and elimination of nested iteration terms. We only show the translation explicitly for iteration terms. The term that remains after inductively reducing the nesting depth of iteration terms does not have to be an iteration term itself. In case it is not an iteration term, it can still have multiple (unnested) iteration subterms. But combining those into a single iteration term only requires a slight modification of the construction we give explicitly.

Let (t^*, p) be a CPT-term where t^* is an iteration term, and let $s_1^*, \ldots, s_k^*$ be the iteration terms occurring as subterms of t. As terms can be translated inductively, suppose that $s_1^*, \ldots, s_k^*$ have no iteration subterms, i. e. $s_1, \ldots, s_k$ are ordinary terms. Our translated term will be an iteration term r^* that simulates the iteration of $s_1, \ldots, s_k$ simultaneously and uses the results to simulate the iteration of t. The iteration is thus pulled out to r^* such that r becomes an ordinary term. The value of t^* can then be extracted from the value of r^*.

The stages of r^* will be tuples whose components represent the stages of $s_1^*, \ldots, s_k^*$ and t^* as well as additional sets that serve to maintain the transitive closure. Only the components corresponding to $s_1, \ldots, s_k$ are modified until they have all reached a fixed point. The outer term t^* is evaluated afterwards. Since the values of $s_1^*, \ldots, s_k^*$

are already present as entries of the tuple at that point, every occurrence of s_i^* in t can be replaced by the respective entry.

The term $r = \langle r_1, \ldots, r_{2k+3} \rangle$ defines a $(2k+3)$-tuple where the intended meaning of the entries is as follows: For $1 \leq i \leq k$, the ith entry represents the current stage of s_i, and the $(k+i)$th entry represents the transitive closure in terms of the function F_{tc}. Let u be the free variable of r whose value is assumed to be a $(2k+3)$-tuple at every stage. We denote by $\mathrm{val}_i(u)$ the term $\pi_i(u)$ defining the current value of s_i, and by $F_i(u)$ the term $\pi_{k+i}(u)$ extracting the function F_{tc} for that value.

The entries $2k+1$ and $2k+2$ serve the same purpose for t instead of s_i. The projections $\pi_{2k+1}(u)$ and $\pi_{2k+2}(u)$ are denoted by $\mathrm{val}_t(u)$ and $F_t(u)$, respectively. The last entry with index $2k+3$ counts the number of iterations, first for the s_i, then for t. Accordingly, the projection $\pi_{2k+3}(u)$ is called count.

We define the term r component-wise, providing terms val_i^+, val_t^+, F_i^+, F_t^+ and count^+ transforming the respective entries to their value in the next stage. For all these terms, u is the unique free variable, that is, they depend on the whole tuple from the previous stage.

Terms of the form val_i^+ for $1 \leq i \leq k$ are intended to define the next stage of s_i depending on whether the polynomial bound is satisfied, and the terms F_i^+ extend the respective function F_{tc}. If the polynomial bound could be ignored, the next stage of s_i would be defined by $s_i(\mathrm{val}_i(u))$. The corresponding function F_{tc} would be defined by

$$F_i^{\mathrm{pre}} := (s_i)_F(\langle \mathrm{val}_i(u), F_i(u) \rangle),$$

where $(s_i)_F$ is the term from Lemma 3.16.

From F_i^{pre}, we can extract the transitive closure of the next stage and check whether it satisfies the bound defined by t_p:

$$\varphi_i^{\mathrm{tc}} = t_p \notin \mathrm{Card}(\pi_2(\mathrm{Unique}(\{x : x \in F_i^{\mathrm{pre}} : \pi_1(x) = \mathrm{val}_i\}))).$$

Then the formula $\varphi_i^{\leq p} := \varphi_i^{\mathrm{tc}} \wedge t_p \notin \mathrm{count} \cup \{\mathrm{count}\}$ verifies that both the size of the next stage and its index satisfy the polynomial bound.

The actual next stage and its corresponding transitive closure function are then expressed by the terms

$$\mathrm{val}_i^+ = \mathrm{if}_{\varphi_i^{\leq p}(u)}(s_i(\mathrm{val}_i(u)), t_{p+1})$$

and

$$F_i^+ = \mathrm{if}_{\varphi_i^{\leq p}(u)}(F_i^{\mathrm{pre}}, t_{p+1}).$$

Note that setting the next stage of s_i to $\emptyset$ instead of t_{p+1} in case the bound is not satisfied would have unwanted consequences. Since the initial stage is also $\emptyset$, the iteration would just start anew. The value t_{p+1} is used because it cannot occur as a stage in any valid iteration.

The definition of the terms val_t^+ and F_t^+ is almost analogous, but now we have to take into account that the respective entries of the tuple are not modified until all previous entries have reached a fixed point. This can be checked with terms of the form if_φ for suitable φ. Similarly, the counter defined by the term count^+ is reset when the iteration of t starts. Further, every occurrence of some s_i^* in t is replaced by $\mathrm{if}_{\mathrm{val}_i(u)\neq t_{p+1}}(\mathrm{val}_i(u), \emptyset)$.

Then r is an ordinary term, and r^* defines a tuple with the value of t^* or t_{p+1} in its $(2k+1)$st component. The actual value of t^*, where the placeholder t_{p+1} is replaced by $\emptyset$ again, can be extracted from r^*. Note that doing this in the obvious way with a term of the form $\mathrm{if}_\varphi(t_1, t_2)$ would violate the condition that only one iteration term occurs as a subterm, because r^* would occur both in φ and as t_1.

But we can modify r such that, whenever a fixed point would be reached (i. e. the computation of t^* terminates), the whole tuple is replaced by $\{\mathrm{val}_t\}$ or $\{\emptyset\}$, respectively. In both cases, the additional pair of braces prevents the value $\emptyset$, which would be indistinguishable from the initial stage. Then the value of t^* can be extracted with the function Unique, where r occurs only once as a subterm.

The polynomial p' can be computed from p and the nesting depth of iteration subterms of t. In case $s_1, \ldots, s_k$ are ordinary terms, for instance, the length of the iteration of r^* on a structure $\mathbf{A}$ is always bounded by $2p(|A|)$, since the s_i and t need at most $p(|A|)$ steps each. Further, every entry of the tuple contains either a stage of some iteration term or a function representing its transitive closure. So the transitive closure of every stage of r^* is of size polynomial in $p(|A|)$. □

Terms of the form defined above can be used to avoid treating the polynomial bound explicitly. That is, instead of CPT-terms of the form (t, p), we can consider BGS-terms for which some polynomial bound is guaranteed to exist by construction.

Implicit PIL When transforming CPT-formulae to BGS-formulae with implicit bounds, the majority of the effort is spent eliminating nested iteration terms and storing the transitive closure. Whereas other CPT-definitions allow nested iteration terms, some variation of the second difficulty will occur independently of the specific definition. In PIL-programs, however, iteration is not nested, and space is measured

in terms of the size of structures. Both factors lead to simplifications in hiding the polynomial bound.

Furthermore, the number of tuples defined by the initial interpretation is a natural way to determine a bound that is polynomial in the size of the input structure. Consequently, we define a fragment of IL where, in every step, the number of iteration steps and the size of the current structure are compared to the number of elements created initially. This leads to several requirements for this IL-fragment, which we denote by *implicit* PIL.

In order to compare values to the polynomial bound, the set of elements representing that bound has to be preserved throughout the computation. This is done with a unary relation Count whose size remains unchanged by the step-interpretation. An additional unary relation Steps serves to represent the number of steps executed so far. That is, the step interpretation increases the size of Steps by one. Finally, the program halts whenever the bound is exceeded. So the formula for the halting condition compares both the size of Steps and the size of the rest of the structure to Count.

Formally, an IL$[\sigma, \tau]$-program $(\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}})$ is in *implicit* PIL if:

- $\tau \supseteq \{\text{Count}, \text{Steps}\}$ for unary relations Count, Steps,
- $|\,\text{Steps}\,|^{\mathcal{I}_{\text{step}}(\mathbf{A})} = |\,\text{Steps}\,|^{\mathbf{A}} + 1$ and $|\,\text{Count}\,|^{\mathcal{I}_{\text{step}}(\mathbf{A})} = |\,\text{Count}\,|^{\mathbf{A}}$ for every structure $\mathbf{A}$ over τ,
- If $\mathbf{A} \not\models \mathsf{R}x.\,\text{Steps}\,x.\,\text{Count}\,x \wedge \mathsf{R}x.(\neg\,\text{Steps}\,x \wedge \neg\,\text{Count}\,x).\,\text{Count}\,x$, then $\mathbf{A} \models \psi_{\text{halt}}$.

Note that, by Lemma 3.14, PIL over FO with the Rescher quantifier is not stronger than PIL itself, so the Rescher quantifier can be considered syntactic sugar.

We now show that these conditions guarantee the existence of a polynomial bound, and, conversely, that every PIL-program can be translated to implicit PIL.

Lemma 3.18. *implicit* PIL $\leq$ PIL

Proof. Let $\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}})$ be in implicit PIL, and let k be the maximum of the dimensions of $\mathcal{I}_{\text{init}}$ and $\mathcal{I}_{\text{step}}$. We show that $(\Pi, (3n^k)^k)$ is a PIL-program equivalent to Π, i. e. that Π never exceeds that bound.

Consider the run ρ of Π on a σ-structure $\mathbf{A}$. Since the dimension of $\mathcal{I}_{\text{init}}$ is $\leq k$, and $|\,\text{Count}\,|$ does not change after $\mathcal{I}_{\text{init}}$, $|\,\text{Count}\,|^{\mathbf{A}_i} \leq |A|^k$ in every state $\mathbf{A}_i$ of ρ. Since $|\,\text{Steps}\,|$ is increased by one in every step, $\mathbf{A}_i$, $|\,\text{Steps}\,|^{\mathbf{A}_i} \geq i - 1$ in each state $\mathbf{A}_i$, $i > 1$. So, by the halting condition, $|\,\text{Steps}\,| \leq |\,\text{Count}\,| \leq n^k$, so the number of states in ρ is bounded by $n^k + 1$.

For the bound on the size of the states, note that, if $|A_i \setminus (\mathrm{Count} \cup \mathrm{Steps})| > n^k \geq |\mathrm{Count}|$ in state $\mathbf{A}_i$, the program halts. If $\mathbf{A}_i$ is the first state exceeding the bound, then $|A_{i-1} \setminus (\mathrm{Count}^{\mathbf{A}_{i-1}} \cup \mathrm{Steps}^{\mathbf{A}_{i-1}})| \leq n^k$. Since both Count and Steps contain at most n^k elements it follows that $|A_{i-1}| \leq 3n^k$. Then, as $\mathcal{I}_{\mathrm{step}}$ is of dimension $\leq k$, $|A_i| \leq |A_{i-1}|^k \leq (3n^k)^k$. So Π respects the PIL time and space bounds. □

It remains to show that every PIL-program can be translated to an implicit one.

Lemma 3.19. PIL $\leq$ *implicit* PIL.

Proof. Let (Π, p) be a PIL$[\sigma, \tau]$-program, with $\Pi = (\mathcal{I}_{\mathrm{init}}, \mathcal{I}_{\mathrm{step}}, \psi_{\mathrm{halt}}, \psi_{\mathrm{out}})$ and a polynomial $p\colon \mathbb{N} \to \mathbb{N}$. First consider the case that $p\colon n \mapsto n^k$ for some $k \in \mathbb{N}$. We define an equivalent program $\Pi' = (\mathcal{I}'_{\mathrm{init}}, \mathcal{I}'_{\mathrm{step}}, \psi'_{\mathrm{halt}}, \psi'_{\mathrm{out}})$ in implicit PIL.

$\mathcal{I}'_{\mathrm{init}}$ is a $[\sigma, \tau \uplus \{\mathrm{Count}, \mathrm{Steps}\}]$-interpretation where, for every structure $\mathbf{A}$, $\mathcal{I}'_{\mathrm{init}}(\mathbf{A})$ is isomorphic to the disjoint union of $\mathcal{I}_{\mathrm{init}}(\mathbf{A})$ and a set of size n^k that makes up the relation Count. Let ℓ be the dimension of $\mathcal{I}_{\mathrm{init}}$. Then $\mathcal{I}'_{\mathrm{init}}$ is of dimension $\ell + k$. The elements of $\mathcal{I}_{\mathrm{init}}(\mathbf{A})$ are encoded in the first ℓ components of each tuple, identified by the property that the entries x_ℓ and $x_{\ell+1}$ coincide. Every tuple $(x_1, \ldots, x_{\ell+k})$ with $x_\ell \neq x_{\ell+1}$ is a representative of a k-tuple (identified by the last k entries) in Count. This is realised with the following interpretation $\mathcal{I}'_{\mathrm{init}} = (\varphi'_\delta, \varphi'_=, (\varphi'_R)_{R \in \tau'})$, where $(\varphi_\delta, \varphi_=, (\varphi_R)_{R\in\tau}) = \mathcal{I}_{\mathrm{init}}$ and $\tau' = \tau \uplus \{\mathrm{Count}, \mathrm{Steps}\}$:

$$\varphi'_\delta(x_1, \ldots, x_{\ell+k}) = \; x_\ell = x_{\ell+1} \to \varphi_\delta(x_1, \ldots, x_\ell),$$

$$\begin{aligned}\varphi'_=(x_1, \ldots, x_{\ell+k}, y_1, \ldots, y_{\ell+k}) =& (x_\ell = x_{\ell+1} \wedge y_\ell = y_{\ell+1} \wedge \varphi_=(x_1, \ldots, x_\ell, y_1, \ldots, y_\ell)) \\ & \vee (x_\ell \neq x_{\ell+1} \wedge y_\ell \neq y_{\ell+1} \wedge \bigwedge_{1\leq i\leq k} x_{\ell+i} = y_{\ell+i}),\end{aligned}$$

$$\begin{aligned}\varphi'_{\mathrm{Steps}}(x_1, \ldots, x_{\ell+k}) &= \; x_1 \neq x_1, \\ \varphi'_{\mathrm{Count}}(x_1, \ldots, x_{\ell+k}) &= \; x_\ell \neq x_{\ell+1},\end{aligned}$$

$$\begin{aligned}\varphi'_R(x^1_1, \ldots, x^1_{\ell+k}, \ldots, x^r_1, \ldots, x^r_{\ell+k}) =& \varphi_R(x^1_1, \ldots, x^1_\ell, x^2_1, \ldots, x^2_\ell, \ldots, x^r_1, \ldots, x^r_\ell) \wedge \\ & \bigwedge_{1\leq i\leq r} x^i_\ell = x^i_{\ell+1} \quad \text{for } R \in \tau.\end{aligned}$$

Next we describe $\mathcal{I}'_{\text{step}}$. This interpretation ensures that the ith state in the run of Π' is isomorphic to the disjoint union of the ith state in the run of Π with a set Count of size n^k and a set Steps of size $i-1$. Every tuple $(a_1, \dots, a_\ell)$ in the next state in the run of Π can then be represented by a tuple $(a'_1, \dots, a'_\ell)$ of elements that are in neither Count nor Steps. The elements of Count in the state $\mathbf{A}_{i+1}$ are the tuples $(a, \dots, a)$ for $a \in \text{Count}^{\mathbf{A}_i}$, and the elements of Steps are the tuples $(a, \dots, a)$ for $a \in \text{Steps}^{\mathbf{A}_i}$ supplemented by the equivalence class of all tuples $(a_1, \dots, a_\ell)$ from Count where $a_1 \neq a_2$.

The formula ψ'_{halt} is the conjunction of ψ_{halt} and the halting condition from the definition of implicit PIL, and ψ'_{out} is obtained by relativising ψ_{out} to the complement of $\text{Count} \cup \text{Steps}$.

By construction, every state in any run of Π' is of the desired form. Since Π is in PIL, every state satisfies the size bounds stated in ψ'_{halt}, so Π' halts if, and only if, Π does. So Π' is equivalent to Π.

At the beginning, we assumed that p is of the form $n \mapsto n^k$ for some k. If p is not of that form, it remains to modify the program such that the original bound is verified explicitly in every step. This can be achieved by adding an additional relation Count' of size exactly $p(|A|)$ that is initialised by the step-interpretation. This initialisation is a sequence of interpretations creating elements for each monomial in p. For each monomial $n \mapsto cn^k$, an interpretation creating $|A|^k$ new elements is applied c times. After Count' is constructed, the size of Steps and of the remaining structure in every step is compared not only to $|\,\text{Count}\,|$ (as required by the definition of implicit PIL), but to $|\,\text{Count}'\,|$ as well.

Note that the number of elements created by the initial interpretation has to be an upper bound on $p(|A|) + |\,\text{Count}'\,|$. This completes the translation of arbitrary PIL-programs to implicit PIL. □

The main motivation behind implicit PIL was to show that, indeed, the polynomial bound can be hidden in a natural property of IL-programs, and that the artificial truncation of runs in PIL does not add any expressive power. As it is equivalent to PIL, implicit PIL serves the intended purpose. Further, it is a means to abbreviate notation, because the polynomial bound does not have to be handled explicitly anymore.

Translation to $(\text{FO+H}_\sim)^*$ over active objects We conclude this chapter with a normal form for CPT-formulae that helps translate them to formulae in weaker logics, which are then evaluated over the active objects. In particular, we show

that $(\mathrm{FO+H}_\sim)^*$ actually suffices to simulate CPT-formulae over the active objects, strengthening previous results [13, 23].

Lemma 3.20.

1. *For every* CPT*-sentence* (φ, p) *over* σ*, there is an* $(\mathrm{FO+H}_\sim)^*$*-sentence* φ_{FO} *such that* $\mathbf{A} \models (\varphi, p)$ *if, and only if,* $\mathbf{act}(\varphi, p, \mathbf{A}) \models \varphi_{\mathrm{FO}}$ *for every* σ*-structure* $\mathbf{A}$*.*
2. *For every* CPT^-*-sentence* (φ, p) *over* σ*, there is an* FO*-sentence* φ_{FO} *such that* $\mathbf{A} \models (\varphi, p)$ *if, and only if,* $\mathbf{act}(\varphi, p, \mathbf{A}) \models \varphi_{\mathrm{FO}}$ *for every* σ*-structure* $\mathbf{A}$*.*

Proof. Since the number sort and the modified Härtig quantifiers are only used for the Card-operator, the second statement will follow directly from the proof for the first one.

The primary purpose of the normal form in our proof is to simulate iteration terms in a logic that cannot iterate. As the desired formula in the weaker logic can access all objects activated by the given CPT-formula, we make sure that not only every stage, but also the history of all previous stages is available as an active object. Hence we modify iteration terms such that every stage already encodes all previous stages. More precisely, we define for every iteration term t^* an iteration term t^*_{stages} such that, for every structure $\mathbf{A}$ and every $i \in \mathbb{N}$,

$$(t^i_{\mathrm{stages}})^{\mathbf{A}} = \left\{ \langle j, (t^j)^{\mathbf{A}} \rangle : j \leq i \right\}.$$

Note that, if u is a stage in that form, there are terms $t_{\max}(u)$ and $t_{\mathrm{cur}}(u)$ that, respectively, extract the maximal i such that $\langle i, a_i \rangle$ occurs in the stage, and its value a_i. Then the term t_{stages} adds the pair $\langle t_{\max}(u) + 1, t(t_{\mathrm{cur}}(u)) \rangle$ to the current set u unless a fixed point is reached. Using t_{cur}, the value of the original term t^* can be extracted again.

In addition to the active objects of t^*, the stages of t^*_{stages} contain ordinals up to the length of the iteration, and the sets defining ordered pairs. So a polynomial bound for t^* implies the existence of a polynomial bound for t^*_{stages}.

A sentence φ in that normal form can be translated to an $(\mathrm{FO+H}_\sim)^*$-sentence over the active objects as follows. Note that $\mathbf{act}(\varphi, p, \mathbf{A})$ is an $\mathbf{A}$-state. So, by Lemma 3.12, every ordinary term can be simulated by an $(\mathrm{FO+H}_\sim)^*$-interpretation. By the interpretation lemma, we can also define the value of every term in the input structure $\mathbf{act}(\varphi, p, \mathbf{A})$.

Then every iteration term can be translated to the formula that states that there is a set containing pairs of the form $\langle j, a_j \rangle$ with $j \leq i$ for some i, such that $a_{i+1} = t(a_i)^{\mathbf{A}}$, and the maximal a_i is the least fixed point of t. □

In consequence, structures $\mathbf{A}, \mathbf{B}$ are indistinguishable in CPT if, and only if, the structures over the active objects for every CPT-sentence are indistinguishable in $(\text{FO+H}_\sim)^*$. The standard games for Härtig quantifiers from [46] do not account for counting of equivalence classes, so it is unclear how this statement of the result can be used to simplify inexpressibility proofs. Note, however, that counting of equivalence classes is only used as a prerequisite of Lemma 3.12. Using the standard definition of active objects [54] instead, one could eliminate the modified Härtig quantifiers and use the games for standard Härtig quantifiers. Furthermore, the statement for CPT^- yields that for that fragment it suffices to compare the structures over the active objects using Ehrenfeucht-Fraïssé games.

3.5 Conclusion

We inspected different ways to formalise Choiceless Polynomial Time in order to support the more intricate questions addressed in the following chapters. Accordingly, we reviewed two definitions of CPT. The proof that CPT in its standard form is equivalent to polynomial-time interpretation logic already yields insight into its structure: Without iteration, i. e. without recursive definitions of sets, CPT collapses to the extension of FO+H by counting of equivalence classes of tuples, which is included in FP+C.

Further, we explored the question if—and how—CPT can be defined without the explicit polynomial bound, providing normal forms of the underlying logics BGS and IL that simulate the polynomial bound implicitly. Even though these normal forms can be characterised syntactically, it remains open whether there are more natural syntactic criteria to ensure evaluation in polynomial time. An example of such a natural criterion is the condition that guarantees monotonicity in least fixed-point inductions. Note, however, that the polynomial bound for fixed-point logics is induced by the arity of the relations defined, which can be seen as a similarity to our definition of implicit PIL. Nevertheless, it seems plausible that a definition without—explicit or implicit—polynomial bounds significantly eases inexpressibility proofs. The question if such a definition exists remains open.

Exploration of different characterisations of CPT is motivated by the question how the definition can make it simpler to write formulae or prove negative results. Our final normal form, which allows to translate CPT^--formulae to FO, could be applied to simplify proofs that certain queries cannot be expressed without counting. Some implications of the definition via iterated first-order interpretations, most of

which concern the fragments this definition induces, are studied in the next chapter. Even though interpretation logic already eases the analysis of CPT, the development of other characterisations optimised for, e. g., inexpressibility proofs could be a useful subject for future research.

4 Fragments and extensions of Choiceless Polynomial Time

The known formalisations of Choiceless Polynomial Time discussed in the previous chapter still do not seem to admit the kind of comprehensive toolkit that may be desirable to analyse its expressive power. A possible approach to developing such a toolkit would be to further simplify the definition. Could there be a syntactic fragment of CPT that already yields the full expressive power? And which features of CPT are essential to its expressiveness? Whereas the original definition of CPT makes it hard to even determine syntactic fragments, interpretation logic allows different ways of defining natural fragments.

First we study two rather obvious restrictions of PIL-programs: Programs over interpretations of bounded dimension, and output signatures with bounded arity. Both parameters induce hierarchies that, as we show in Sections 4.1.1 and 4.1.2, collapse at level two. Indeed, two-dimensional interpretations and a single binary relation, respectively, already capture all of PIL. One-dimensional interpretations, however, lack the means to create new elements, whereas unary relations cannot store all the necessary information about the input structure. Consequently, neither the one-dimensional fragment nor the one where the interpretations may create only unary relations correspond to full PIL.

The result about binary relations additionally implies that it would suffice to capture Ptime on the class of graphs. However, we also provide an intuition as to why the same connection does not trivially follow for padded graphs.

Restrictions of interpretations also provide a way to limit the core feature of CPT: The ability to iteratively create hereditarily finite sets. As we have already seen, creation of sets without iteration can be simulated by sequences of FO+H-interpretations. But what happens if we limit the kind of sets constructed?

Interpretations create tuples over the input structure, which are combined into sets using the equality formula. So interpretations without equality formulae—or, in other words, without congruences—provide a formalisation of CPT "without sets". In Section 4.1.3, we show that PIL without congruences is strictly less expressive

than PIL, both with and without counting. A more detailed examination of PIL without congruences is presented in the next chapter.

In addition to complicating its analysis, the unusual definition of CPT raises the question whether it is a natural candidate for a logic capturing PTIME. While the notion of choiceless computation is certainly justified, a logic for PTIME can also be seen as either an extension of weaker logics or a restriction of stronger ones. Interpretation logic makes the connection to both fixed-point logic and second-order logic visible. The restriction of PIL^- to one-dimensional interpretations is, in fact, equivalent to the PTIME-restriction of partial fixed-point logic. For the variant with counting, this implies equivalence to FP+C.

Moreover, interpretation logic can easily be extended to permit non-determinism and, in a similar way, quantifier alternations. As a consequence, $\exists$SO as well as the whole quantifier alternation hierarchy can be defined as natural extensions of PIL.

4.1 Fragments of Choiceless Polynomial Time

We first study the syntactic fragments of CPT arising from its characterisation as PIL. Certain restrictions of the first two parameters, namely the output signature and the dimension of the interpretations, already correspond to full PIL. Though the proofs in these first two subsections are mainly simple simulations, the results already provide new insights into the structure of PIL and thus CPT. The main part of this section is the analysis of PIL without congruences, which, as argued above, serves as a model of PIL over tuples instead of arbitrary sets.

4.1.1 Arity of output relations

The main motivation behind studying restricted output signatures is the observation that, up to FO-interpretations, every structure is a graph. More formally, every structure can be interpreted in a graph in a way that the original structure can be recovered using another interpretation. A PIL-program can use these interpretations to transform the input structure into a graph and perform the necessary computations on that graph. Formalising this idea, we get

Lemma 4.1. $\mathrm{PIL}[\sigma, \{E\}] \equiv \mathrm{PIL}[\sigma, \tau]$ *for all signatures* σ, τ.

To transform arbitrary structures into graphs, we use the following well-known lemma, which is shown, for example, in [37]:

Lemma 4.2. *For every signature σ, there is an* FO$[\sigma, \{E\}]$*-interpretation* $\mathcal{I}_\sigma$ *and an* FO$[\{E\}, \sigma]$*-interpretation* $\mathcal{I}_\sigma^{-1}$ *such that, for every* σ*-structure* $\mathbf{A}$*,*

$$\mathcal{I}_\sigma^{-1}(\mathcal{I}_\sigma(\mathbf{A})) \cong \mathbf{A}.$$

Further, $\mathcal{I}_\sigma^{-1}$ *is a one-dimensional, congruence-free interpretation.*

The obvious way to apply this lemma to obtain a PIL$[\sigma, \{E\}]$-program is to translate the input structure to a graph in the initial interpretation. The step-interpretation has to emulate the behaviour of the original (w. l. o. g.) PIL$[\sigma, \sigma]$-program. So it translates every state back to a σ-structure, applies the original step-interpretation, and then translates the result back to a graph. But, since FO+H is not closed under interpretations, it is not obvious that combining the interpretations in that way again yields an FO+H-interpretation. This constitutes the main difficulty when proving Lemma 4.1.

Proof of Lemma 4.1. Let Π be a program in implicit PIL$[\sigma, \tau]$. For ease of notation, we translate Π to the form $(\mathcal{I}, \psi_{\text{halt}}, \psi_{\text{out}})$ such that $\mathcal{I}$ is an FO+H$[\sigma \cup \tau, \sigma \cup \tau]$-interpretation and ψ_{halt} and ψ_{out} are FO+H$[\sigma \cup \tau]$-formulae.

Then the IL$[\sigma, \{E\}]$-program $\Pi_E = (\mathcal{I}_\sigma, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}^{\mathcal{I}_\sigma^{-1}}, \psi_{\text{out}}^{\mathcal{I}_\sigma^{-1}})$ with $\mathcal{I}_{\text{step}} = \mathcal{I}_\sigma^{-1} \circ \mathcal{I} \circ \mathcal{I}_\sigma$ is equivalent to Π.

Since $\mathcal{I}$ is an FO+H-interpretation, and ψ_{halt} and ψ_{out} are FO+H-formulae, we first show that $\mathcal{I}_{\text{step}}$, $\psi_{\text{halt}}^{\mathcal{I}_\sigma^{-1}}$ and $\psi_{\text{out}}^{\mathcal{I}_\sigma^{-1}}$ are well-defined. By Lemma 4.2, $\mathcal{I}_\sigma^{-1}$ is one-dimensional and congruence-free. So, by the weak interpretation lemma for FO+H (Lemma 2.2), it defines a transformation of FO+H-formulae. This means that $\psi_{\text{halt}}^{\mathcal{I}_\sigma^{-1}}$ and $\psi_{\text{out}}^{\mathcal{I}_\sigma^{-1}}$ are well-defined. Further, applying this transformation to the formulae in $\mathcal{I}$ yields an FO+H-interpretation $\mathcal{I}_\sigma^{-1} \circ \mathcal{I}$.

Since, again by Lemma 2.2, FO is weakly closed under FO+H-interpretations, $\mathcal{I}_\sigma^{-1} \circ \mathcal{I}$ defines a translation from FO-formulae to FO+H-formulae. This translation can by applied to every formula in $\mathcal{I}_\sigma$.

We show by induction that $\mathcal{I}_\sigma^{-1}(\mathcal{I}_{\text{step}}^k(\mathcal{I}_\sigma(\mathbf{A}))) \cong \mathcal{I}^k(\mathbf{A})$ for every $\sigma \cup \tau$-structure $\mathbf{A}$ and every $k \in \mathbb{N}$. This implies that $\Pi(\mathbf{A}) \cong \mathcal{I}_\sigma^{-1}(\Pi_E(\mathbf{A}))$ and thus the correctness of the translation.

The induction base for $k = 0$ follows directly from Lemma 4.2. For the induction step, note that $\mathcal{I}_{\text{step}}^{k+1}(\mathcal{I}_\sigma(\mathbf{A})) = \mathcal{I}_\sigma(\mathcal{I}(\mathcal{I}_\sigma^{-1}(\mathcal{I}_{\text{step}}^k(\mathcal{I}_\sigma(\mathbf{A})))))$. Hence, by the induction hypothesis, $\mathcal{I}_\sigma^{-1}(\mathcal{I}_{\text{step}}^{k+1}(\mathcal{I}_\sigma(\mathbf{A})))$ is isomorphic to $\mathcal{I}_\sigma^{-1}(\mathcal{I}_\sigma(\mathcal{I}(\mathcal{I}^k(\mathbf{A})))) \cong \mathcal{I}^{k+1}(\mathbf{A})$.

It follows that Π_E is equivalent to Π. It is, however, not in implicit PIL anymore. But, since Π is in implicit PIL, there is a polynomial p that bounds the size of every

structure in the run of Π on a σ-structure $\mathbf{A}$. Then p^k, where k is the dimension of $\mathcal{I}_\sigma$, is a bound for the structures in the run of Π_E on $\mathbf{A}$. So (Π_E, p^k) is a PIL-program equivalent to Π_E. □

In a similar way, the interpretation from Lemma 4.2 can be used as a reduction. Thus we obtain another simple consequence:

Corollary 4.3. *If* PIL *captures polynomial time on the class of graphs, then* PIL *captures polynomial time.*

Proof. Assume PIL captures PTIME on the class of graphs. Then, for every query $\mathcal{Q} \subseteq \mathrm{fin}(\sigma)$ in PTIME, the query $\mathcal{Q}' = \{\mathcal{I}_\sigma(\mathbf{A}) : \mathbf{A} \in \mathcal{Q}\}$ is also in PTIME and thus definable by a PIL-program (Π, p). Then the program that first applies $\mathcal{I}_\sigma$ and then runs Π on the resulting structure defines $\mathcal{Q}$. A polynomial bound for that program is p^k, where k is the dimension of $\mathcal{I}_\sigma$. □

It seems surprising at first that the captuting result for *padded* graphs cannot be generalised that way. Recall that, as shown by Laubner [47], CPT captures PTIME on the class of padded structures with $u \leq c \log n$ over relations of arity at most two for every $c \in \mathbb{N}$. Now let σ be a signature, and let $\mathcal{I}$ be a d-dimensional interpretation that acts like $\mathcal{I}_\sigma$ on the underlying structure of every padded structure. Further, let $\mathbf{A}$ be a padded structure with $|A| = n$ and $|U^{\mathbf{A}}| = u$. The interpretation $\mathcal{I}_\sigma$ creates at least one element for every tuple occurring in one of the relations. So $U^{\mathcal{I}(\mathbf{A})}$ may have u^k elements, where k is the arity of some relation in σ. But $\mathcal{I}$ can only create $n^d \geq 2^{ud}$ elements in total, instead of the required 2^{u^k} elements.

If, however, $\mathcal{I}$ is a $(k+1)$-dimensional interpretation which applies $\mathcal{I}_\sigma$ to the underlying structure and preserves $A \setminus U^{\mathbf{A}}$, we get

$$|U^{\mathcal{I}(\mathbf{A})}| \leq u^k \leq (c \log n)^k \leq (c \log |\mathcal{I}(\mathbf{A})|)^k.$$

So the interpreted structure is a padded structure where the underlying structure is of polylogarithmic size. This connects the open questions about capturing on padded structures by Laubner [47] and Grädel and Grohe [26]:

Lemma 4.4. *If* CPT *captures* PTIME *on every class of padded graphs with* $u \leq (c \log n)^k$ *for* $c, k \in \mathbb{N}$*, then* CPT *captures* PTIME *on the class of padded structures with* $u \leq c \log n$ *for every* $c \in \mathbb{N}$*.*

In the remainder of this subsection, we analyse the expressive power of different variants of PIL over unary relations. Whereas a single binary output relation suffices

to gain the full expressive power of PIL, the fragment where the output relation can contain only unary relations is included in FP+C.

Lemma 4.5. *Let σ, τ be signatures such that τ only consists of monadic predicates. Then* $\mathrm{PIL}[\sigma, \tau] \leq \mathrm{FP{+}C}$.

Proof. Let $((\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}}), p)$ be a $\mathrm{PIL}[\sigma, \tau]$-program such that τ is monadic. Then for any input structure $\mathbf{A}$, $\mathcal{I}_{\text{step}}(\mathbf{A})$ is a structure over unary predicates. Note that structures over unary signatures can be transformed to, e. g., planar graphs by an FO-interpretation in the sense of Lemma 4.2. Since FP+C captures Ptime on the class of planar graphs, there is an FP+C-formula φ such that, for every τ-structure $\mathbf{B}$, $\mathbf{B} \models \varphi$ if, and only if, the program $((\mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}}), p)$ without the initial interpretation accepts $\mathbf{B}$. But FP+C is closed under interpretations, so there is a formula $\varphi^{\mathcal{I}_{\text{init}}}$ equivalent to the full PIL-program. □

It remains open whether the inclusion is strict, that is, whether the restriction of PIL to unary output relations is as expressive as FP+C. Even though these PIL-programs can still create new elements based on equivalence classes of tuples, their step-interpretations cannot access the relations from the input structure. To allow access to the input relations, however, the input structure has to be preserved. But expressing this as a simple syntactic restriction means restricting the programs to one-dimensional interpretations.

As we will show in the next subsection, one-dimensional interpretations do not suffice to capture the full expressive power of PIL. Nevertheless, the example programs from Section 3.3 demonstrate that PIL-programs with one-dimensional interpretations using only monadic relations as additional "data structures" can still express reachability and certain counting properties.

Remark 4.6. *Let $k \geq 2$ and $\tau = \{C_0, C_1, \ldots, C_{k-1}\}$, where $C_0, \ldots, C_{k-1}$ are unary relations. There is a* $\mathrm{PIL}[\emptyset, \tau]$*-program* Π *with* $\mathbf{A} \models \Pi$ *if, and only if,* $|\mathbf{A}|$ *is a multiple of* k.

Proof. The program Π can be constructed from the program in Example 3.8 with a small modification. Instead of relations C_0, C_1, use all k relations for modulo counting. More precisely, the newly created elements are moved from C_i to $C_{(i+1) \pmod k}$ in every iteration. □

Recall that the program from Example 3.7 defining reachability only modifies the unary relation S. However, the formula defining that relation uses the binary input relation E. We formalise that kind of "read-only" access to higher-arity relations as

follows: A PIL-program *modifies only unary relations* if it is one-dimensional and, for every relation R of arity $r \geq 2$, $\varphi_R(x_1, \ldots, x_r) = Rx_1 \ldots x_r$.

Then the program from Example 3.7 witnesses

Remark 4.7. *There is a* PIL *program* Π *that modifies only unary relations such that, for every* $\{E, S, T\}$*-structure* $\mathbf{A}$*, where* E *is a binary relation and* S *and* T *are unary,* $\mathbf{A} \models \Pi$ *if, and only if, there is an* E*-path in* $\mathbf{A}$ *from a vertex in* S *to a vertex in* T*.*

4.1.2 Dimension of interpretations

The situation for interpretations of restricted dimension is similar to the one for output relations of restricted arity: While two-dimensional interpretations already suffice to express all PIL-definable properties, one-dimensional interpretations induce a true fragment. Both with and without counting, this fragment is equivalent to (partial) fixed-point logic. We denote by k-dimensional PIL the set of all PIL-programs $((\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}}), p)$ where both $\mathcal{I}_{\text{init}}$ and $\mathcal{I}_{\text{step}}$ are at most k-dimensional.

Lemma 4.8. *two-dimensional* PIL $\equiv$ *two-dimensional* PIL* $\equiv$ PIL*.*

Proof. For $k \geq 2$, every $(k+1)$-dimensional interpretation can be simulated by the concatenation $\mathcal{I}_1 \circ \mathcal{I}_2$ of a k-dimensional and a two-dimensional interpretation as follows: $\mathcal{I}_1$ creates all k-tuples, and preserves a copy of the original structure over those tuples where all entries coincide (marked by a unary relation). Binary relations $R_1, \ldots, R_k$ associate the entry a_i of every k-tuple with its copy $(a_i, \ldots, a_i)$. Then $\mathcal{I}_2$ extends every tuple $(a_1, \ldots, a_k)$ by every possible entry a_{k+1}. The formulae for domain, equality and relations from the $(k+1)$-dimensional interpretation are translated such that they are evaluated over the copy of the input structure.

Applying this argument inductively, every interpretation can be reduced to a sequence of two-dimensional interpretations. An interpretation logic program can apply these interpretations sequentially. So two-dimensional PIL* $\equiv$ PIL.

The number sort of the input structure in two-dimensional PIL* can be constructed by subsequently adding single elements to a new unary relation. This is possible with two-dimensional interpretations, which proves the first equivalence. □

The key to the translation to interpretations of smaller dimension is that it is still possible to create new elements. One-dimensional interpretations, however, can only *restrict* the domain of the input structure. So iterated one-dimensional

interpretations can be seen as iterated updates to relations—in other words, fixed-point inductions. Therefore we can show that FP+C appears as the one-dimensional fragment of PIL.

This, however, is only the case if we equip the input structure with a number sort. Without the number sort, one-dimensional PIL would have to create a sufficiently large linear order. But, since it cannot create new elements, it would have to define that order on elements of the input structure, which would imply that one-dimensional PIL captures PTIME.

Lemma 4.9. *one-dimensional* $\text{PIL}^* \equiv \text{FP+C}$.

Proof. In contrast to inflationary or least fixed-point inductions, PIL-iterations can change the relations arbitrarily. Therefore we simulate one-dimensional PIL^* in $(\text{PFP+H})^* \uparrow \text{PTIME}$ instead. Since counting can be simulated by Härtig quantifiers in the presence of a number sort, and $\text{PFP+C} \uparrow \text{PTIME}$ is equivalent to FP+C, this will imply the result.

A PIL-program can be translated to $(\text{PFP+H})^*$ using simultaneous fixed points to modify all relations at once. Because a PFP-formula cannot modify the domain of a structure, we represent the domain of every state of the PIL-program by a unary relation. Instead of merging elements with the equality formula, the PFP formula maintains the corresponding congruence relation. Then the translation is straightforward, except for formulae with the Härtig quantifier, since these formulae have to count equivalence classes. But this is possible because counting of equivalence classes is definable in FP+C.

Since the polynomial bound carries over, one-dimensional PIL^*-programs can be translated to $(\text{PFP+H})^* \uparrow \text{PTIME}$-formulae.

For the other direction, we inductively translate $(\text{PFP+H})^*$-formulae to one-dimensional PIL^*. To translate formulae with free variables, we add a k-dimensional relation Out to the output signature, and show that, for every formula φ with free variables $x_1, \ldots, x_k$, there is a one-dimensional PIL^*-program Π such that $\mathbf{A} \models \varphi(a_1, \ldots, a_k)$ if, and only if, $([a_1]_\sim, \ldots, [a_k]_\sim) \in \text{Out}^{\Pi(\mathbf{A})}$. This condition is well-defined because Π is a one-dimensional program, so the elements of every state can be seen as equivalence classes of elements of the input structure. For reasons of uniformity, we translate formulae without free variables the same way, using a nullary relation Out. Then the output formula of the PIL-program for a sentence simply tests if that relation is true.

This makes it possible to translate $(\text{PFP+H})^*$-formulae inductively:

- The translation is trivial for atomic formulae.
- For Boolean combinations, consider the concatenation $\Pi_1 \circ \Pi_2$ of two one-dimensional PIL-programs, which can easily be defined using an auxiliary relation that indicates which interpretation is currently applied. A program for formulae $\varphi \wedge \psi$, $\varphi \vee \psi$ or $\neg\varphi$ is obtained by defining the Out-relation appropriately in the concatenation of the programs for the subformulae.
- For $\exists x\varphi(x)$, decrease the arity of Out and define it with $\varphi'_{\mathrm{Out}} = \exists x\varphi_{\mathrm{Out}}(x)$.
- For $\mathsf{H}x.\varphi(x).\psi(x)$, we define the Out-relation of the concatenation of the programs for φ and ψ using the Härtig quantifier.
- For $[\mathrm{PFP}_{X,\bar{x}}\varphi](\bar{x})$, we add a relation X_{prev} to the output signature and define it with $\varphi_{X_{\mathrm{prev}}}(\bar{y}) = X\bar{y}$. The program for φ is then iterated until $X = X_{\mathrm{prev}}$. □

The arguments in the previous lemma survive on one-sorted finite structures in the absence of counting. So it follows that one-dimensional $\mathrm{PIL}^- \equiv \mathrm{PFP}{\restriction}\textsc{Ptime}$. However, in this case the relationship between the polynomial-time restrictions of partial fixed-point logic and least (or inflationary) fixed-point logic is (probably) a different one, since $\mathrm{PFP}{\restriction}\textsc{Ptime} \equiv \mathrm{LFP}$ if, and only if, $\textsc{Ptime} = \textsc{Pspace}$.

Corollary 4.10. *one-dimensional* $\mathrm{PIL}^- \equiv \mathrm{LFP}$ *if, and only if,* $\textsc{Ptime} = \textsc{Pspace}$.

We see that both with and without counting, fixed-point logic appears as a one-dimensional fragment of PIL and thus CPT. Conversely, CPT can be viewed as the extension of fixed-point logic to higher-order objects.

4.1.3 PIL without congruences

In the previous sections we have already seen that the characterisation of CPT as PIL makes it easier to find natural syntactic fragments. In particular, there is a simple restriction on the power to create sets, which we study in this section.

Structures defined via interpretations consist of congruence classes of fixed-length tuples over the input structure. Intuitively, joining the tuples into congruence classes is what makes it possible to create sets out of these tuples. So omitting the congruence, i. e. the equality formula, turns PIL into a formalism that creates tuples instead of sets. We thus consider the following fragment of PIL:

Definition 4.11. An IL-program $\Pi = (\mathcal{I}_{\mathrm{init}}, \mathcal{I}_{\mathrm{step}}, \psi_{\mathrm{halt}}, \psi_{\mathrm{out}})$ is in congruence-free IL, or $\sim$-free IL, if $\mathcal{I}_{\mathrm{init}}$ and $\mathcal{I}_{\mathrm{step}}$ are congruence-free. (Π, p) is in $\sim$-free PIL if Π is congruence-free.

In the following, we show that both with and without counting, creation of sets is necessary to obtain the full expressive power of PIL.

Theorem 4.12.

1. $\sim$-free $\mathrm{PIL}^- < \mathrm{PIL}^-$
2. $\sim$-free $\mathrm{PIL} < \mathrm{PIL}$.

Moreover, in Section 5.5, we provide a more general characterisation of CPT "over tuples" that includes $\sim$-free PIL and allows for a more fine-grained classification of its expressive power.

Congruence-free PIL without counting Without counting, there is a known characterisation of CPT as an extension of a tuple-based formalism: As shown by Blass, Gurevich and van den Bussche [15], the PTIME-restriction of the database query language $\mathit{while}_{new}^{sets}$ is equivalent to CPT^-. The language $\mathit{while}_{new}^{sets}$ is an extension of while_{new} by creation of new elements based on sets, whereas while_{new} permits only tuple-based creation. Indeed, $\mathit{while}_{new}^{sets}{\restriction}\textsc{Ptime}$ is strictly more expressive than $\mathit{while}_{new}{\restriction}\textsc{Ptime}$ [15]. So the first part of Theorem 4.12 is a direct consequence of the following lemma:

Lemma 4.13. ($\sim$-free PIL^-) $\equiv \mathrm{while}_{\mathrm{new}}{\restriction}\textsc{Ptime}$.

Proof. Let (Π, p) be a $\mathrm{PIL}[\sigma,\tau]$-program. A while_{new}-program P_{sim} can simulate Π by modifying relations D and R_i for $R_i \in \tau$. D represents the domain of the current state of Π. For every application of an interpretation in Π, P_{sim} creates new elements for all tuples satisfying the domain formula, assigns these elements to D and redefines the relations in τ according to the interpretation. The halting condition is checked by placing these steps within a `while`-loop. After the loop, the program defines the output relation with the output-formula of Π.

Then each iteration of P_{sim} corresponds to a state in the run of Π, so there is a constant c such that each run executes at most $c \cdot p(|\mathbf{A}|)$ many atomic expressions. The size of each state in a run of Π on $\mathbf{A}$ is bounded by $p(|\mathbf{A}|)$. Since P_{sim} has to keep all elements created during the run, there may be $p(|\mathbf{A}|)^2$ many elements in the last state. So $(P_{\mathrm{sim}}, c \cdot p^2)$ is a $\mathit{while}_{new}{\restriction}\textsc{Ptime}$-program equivalent to (Π, p).

Conversely, programs of $\mathit{while}_{new}{\restriction}\textsc{Ptime}$ can be translated to $\sim$-free PIL inductively. Relations can of course be modified with PIL-programs. Sequential execution and loops of PIL-programs are again PIL-programs (with and without congruences). So the only interesting case is creation of new elements. In the PIL-program, each

element that is already in the domain is represented by the tuple $(a, \dots, a)$. The usual approach to creating a new element for every tuple $(a_1, \dots, a_j)$ would be to represent that element by an equivalence class, say, $\{(a_1, \dots, a_j, b) : b \neq a_1\}$. Without congruences, however, this could quickly lead to an exponential blowup.

Therefore, we create in an initial step a sufficiently small set of elements marked by a new predicate P and only create copies of the new elements for each element of P. The size of P can be polynomial in the size of the input structure, as long as it remains unchanged throughout the computation. Then a statement $Y :=$ `tup-new`$\{(x_1, \dots, x_j) \mid \varphi\}$ is simulated in a way that the new element $(a_1, \dots, a_j, b)$ is created if, and only if, b is in P, each a_i is either in the original domain or in the copy associated with b, and $(a_1, \dots, a_j)$ satisfies φ.

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

Together with the equivalence between $\mathit{while}\restriction\textsc{Ptime} \equiv \text{PFP}\restriction\textsc{Ptime}$ and one-dimensional PIL$^-$, we conclude that

$$\text{one-dimensional PIL}^- < (\sim\text{-free PIL}^-) < \text{PIL}^-.$$

Congruence-free PIL with Counting The congruence-free fragment of PIL$^-$ can simply be translated to another language which is already known to be strictly less expressive than CPT$^-$. For $\sim$-free PIL with counting, we do not obtain a precise characterisation. Nevertheless, over structures of bounded colour class size, that fragment is included in CPT over sets of bounded rank—another fragment that is strictly included in CPT. Although the characterisation of the tuple-based fragment of CPT we derive in Section 5.5 yields more accurate results, this inclusion clearly suffices for a separation. Additionally, we again examine the relation between congruence-free and one-dimensional PIL, and obtain a similar picture as for the case without counting.

Lemma 4.14. FP+C $<$ ($\sim$-free PIL) $<$ PIL.

Recall that, to simulate FP+C in one-dimensional PIL, we augmented the input structure with a number sort. So, to show the inclusion as it is stated in Lemma 4.14, we have to create that number sort in $\sim$-free PIL. The problem is again that, in general, an interpretation cannot create a single new element without congruences. We can, however, easily overcome this obstacle by creating not only *one*, but a

polynomial number of linear orders whose lengths correspond to the size of the input structure (by using techniques similar to the ones we used in the proof of Lemma 4.13).

Moreover, to see that the inclusion is strict we apply a standard padding argument. It is easy to see that also in the absence of congruences, an IL-program (without polynomial bounds) can create all linear orders on the domain of the input structure. With polynomial bounds, this is still possible on padded structures of appropriate size. Hence, $\sim$-free PIL captures PTIME on certain classes of padded structures. In particular, it can define the CFI query on padded structures, which is not possible in FP+C.

The second inclusion in Lemma 4.14 is trivial. To show that the inclusion is strict, we prove the following more general statement:

Lemma 4.15. *On every class of structures of bounded colour class size, every program in* $\sim$-free PIL *can be translated to a* CPT*-sentence activating only sets of bounded rank.*

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

The main idea is to represent the elements of the structures $\mathbf{A}_i$ in the run $(\mathbf{A}_i)$ of $\overline{\Pi}$ on a structure $\mathbf{A} \in \mathcal{K}$ where $\mathbf{A} = C_1 \preceq \cdots \preceq C_m$ by tuples in $(n^\ell \times A^{q \cdot m})$ where $n = |A|$. Since q is a constant and since we are given the preorder $\preceq$ on A of length m it is easy to see that the second component of such objects can be represented over $\mathbf{A}$ as sets of constant rank. Concerning the representation of the number n^ℓ, note that the ordinals up to n^ℓ have unbounded rank. However, every q-bounded structure of size n has at least $\frac{n}{q}$ colour classes, so these numbers can be represented as tuples of colour classes. Since tuples of bounded length suffice, these can be represented as sets of bounded rank.

Further, relations over sets of bounded rank are again sets of bounded rank. Hence, it suffices to show that such a representation for the elements of the structures $\mathbf{A}_i$ is CPT-definable.

Let us consider the step from $\mathbf{A}_i$ to $\mathbf{A}_{i+1}$ in the simulation of $\overline{\Pi}$ on $\mathbf{A}$. We assume that we have already represented the elements of $\mathbf{A}_i$ as elements in $(n^\ell \times A^{q \cdot m})$. First of all, since the interpretations in $\overline{\Pi}$ are of dimension k and since the domain of the structure $\mathbf{A}_{i+1}$ is of size $\leq n^\ell$ we can easily represent the elements of $\mathbf{A}_{i+1}$

as tuples in $(n^\ell \times A^{k \cdot q \cdot m})$. To again obtain elements in $(n^\ell \times A^{q \cdot m})$ we perform the following steps. For convenience let $\lambda \coloneqq k \cdot q \cdot m$.

(i) First we define the following strict preorder $\prec$ on the set A^λ: For $\overline{a}, \overline{b} \in A^\lambda$ we set $\overline{a} \prec \overline{b}$ if

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

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

(ii) From the definition of $\prec$ it follows that each set of $\prec$-incomparable elements $[\overline{a}]$ can be characterised by

- the list of colour classes to which the entries $\overline{a}(1), \ldots, \overline{a}(\lambda)$ belong, and
- the list of equality types $t_1, \ldots, t_m$ of the sub-tuples $(\overline{a} \restriction C_1), \ldots, (\overline{a} \restriction C_m)$ of $\overline{a}$ which arise by restricting to entries in $C_1, \ldots, C_m$, respectively.

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

Altogether, we can identify an element $(x, \overline{a}) \in (n^\ell \times A^\lambda)$ of $\mathbf{A}_{i+1}$ by a number $\leq n^\ell$ together with an element in $A^{q \cdot m}$: Since $\mathbf{A}_{i+1}$ has $\leq n^\ell$ elements, there are at most n^ℓ combinations of numbers x and $\prec$-classes, so we can order these combinations and encode their positions as numbers $\leq n^\ell$.

All required operations can be defined in CPT over sets of bounded rank. □

In [23], Dawar, Richerby and Rossman show that no CPT-formula activating only sets of bounded rank can define the Cai-Fürer-Immerman query over graphs of degree 3. But in that case, the CFI graphs have bounded colour class size. So $\sim$ -free PIL cannot define that version of the CFI query either.

The restriction of CPT to sets of bounded rank is of particular interest because it is one of the few naturally arising fragments of CPT examined before the introduction of interpretation logic. In Section 5.5, we continue to analyse the relation between congruence-free PIL and CPT over sets of bounded rank, both of which constitute a restriction of the power to create sets.

Let us conclude with the remark that the lack of congruences only poses a true restriction because of the space bound of PIL-programs. Indeed, if we allow arbitrarily large structures as states, already $\sim$-free IL^- can express all of SO.

Remark 4.16. *For every* $\mathrm{SO}[\sigma]$*-formula* φ*, there is a* $\sim$-free IL^-*-program* Π *and a polynomial* p *such that, for every* σ*-structure* $\mathbf{A}$*, the run of* Π *on* $\mathbf{A}$ *has length* $p(|A|)$ *and* Π *accepts* $\mathbf{A}$ *if, and only if,* $\mathbf{A} \models \varphi$*.*

Proof. Π can be constructed by induction on φ. The interesting case is $\varphi = \exists X\psi$, where X is a k-ary relation variable. In order to quantify over all relations, it is desirable to enrich the input structure $\mathbf{A}$ by a copy of A^k for each relation $X \subseteq A^k$. The difficulty with this approach is to index these copies: One would need, for each $X \subseteq A^k$, some distinguished element of the interpreted structure. It is not obvious how this can be done without a linear order.

But, since there is no space bound, we create a large number of copies of A^k instead. When creating these copies, relations S, T and X with the following semantics are maintained: An element of the current state is in X if, and only if, it represents a tuple that is in X in its respective copy. S contains all pairs of elements that are in the same copy, and T maps every element representing a tuple to its entries. Then $\mathbf{A} \models \varphi$ if, and only if, the final state contains a copy of A^k defining a relation X that satisfies ψ. More precisely, the program evaluates the formula $\exists x\psi'$ on the final state, where, in ψ', all occurrences of X are relativised to elements that are in the same copy as x.

We show that a $\sim$-free IL^--program can construct these copies of A^k in A^k steps. Since the goal is to obtain all subsets of A^k, the ith step of the program creates copies of all i-element subsets of A^k. The basic idea is to extend every $(i-1)$-element subset by all tuples that do not occur in that subset yet. But, since the subsets of the previous step cannot be indexed by single elements, the program creates a new copy for every combination of

- an element a representing a tuple in some copy,
- a tuple $\overline{b} \in A^k$,
- the cases $\overline{b} \in X$ and $\overline{b} \notin X$. □

4.2 Alternating PIL and second order logic

The characterisation of CPT as PIL makes evident some natural fragments of CPT, some of which correspond to known logics within PTIME. In this section, we show that it is comparably simple to extend PIL in natural ways to obtain known fragments of second order logic.

In contrast to other logics, PIL is rather algorithmic in nature. In particular, the satisfaction relation is defined via the *run* of a *program*. This definition suggests that PIL-programs can also be evaluated in a nondeterministic way, that is, by guessing parts of the run. The obvious choice would be to guess one out of several step-interpretations.

The approach we follow here, however, is more general and uses a natural extension of interpretations: We now consider interpretations with parameters. In a deterministic setting, it is unclear how to assign values from the current domain to the parameters. These values can, however, be guessed nondeterministically.

We will see that guessing tuples of elements in an iterative fashion is as expressive as guessing sets of tuples. In other words, the nondeterministic version of PIL is equivalent to existential second order logic, and thus captures NP. Universal set quantification and quantifier alternations can be obtained with obvious modifications: We can require that *all* choices of parameters induce an accepting run, and further generalise the resulting logic to alternations of universal and existential choices. These alternating variants of PIL indeed correspond to the quantifier alternation hierarchy of second order logic.

All these variants of PIL are simply based on permitting parameters in the interpretations. The existential and universal variant share the same syntax, whereas some additional syntactic adaptions are necessary to formalise alternation.

Definition 4.17. A PIL$[\sigma, \tau]$-*program with parameters* is a pair (Π, p), where $\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}})$ is defined like an IL-program, with the exception that $\mathcal{I}_{\text{step}}$ may have $k \geq 0$ parameters.

A *run* of Π on a σ-structure $\mathbf{A}$ is a sequence $(\mathbf{A}_i, \overline{a_i})_{0 \leq i \leq n}$, $n \in \mathbb{N}$, such that

- for each $i \leq n$, $\mathbf{A}_i$ is a τ-structure and $\overline{a_i}$ is a k-tuple over $\mathbf{A}_i$,
- $\mathbf{A}_0 = \mathcal{I}_{\text{init}}(\mathbf{A})$,
- $\mathbf{A}_i = \mathcal{I}_{\text{step}}(\mathbf{A}_i, \overline{a_i})$ and $\mathbf{A}_i \neq \mathbf{A}_{i-1}$ for all $0 < i \leq n$,
- $n \leq p(|\mathbf{A}|)$ and $|\mathbf{A}_i| \leq p(|\mathbf{A}|)$ for all $i \leq n$,
- $\mathbf{A}_n \models \psi_{\text{halt}}$ or $|\mathcal{I}_{\text{step}}(\mathbf{A}_n, \overline{a_n})| > p(|\mathbf{A}|)$ or $n = p(|\mathbf{A}|)$.

The run is accepting if $\mathbf{A}_n \models \varphi$.

Analogously to PIL without parameters, we may omit $\mathcal{I}_{\text{init}}$ and define $\tau \supseteq \sigma$ instead.

We first consider the nondeterministic variant of PIL and its dual with universal choices. Both variants can be defined in a straightforward way as different evaluations of PIL-programs with parameters. We then proceed to show that the existential and universal versions of PIL correspond to existential and universal second order logic, respectively. This sets the stage for the analysis of the generalisations with quantifier alternations, whose definition is slightly more involved.

Definition 4.18. A program in *existential* PIL (∃PIL) is a PIL-program Π with parameters with the following semantics:

Π accepts a structure $\mathbf{A}$ if, and only if, there exists an accepting run of Π on $\mathbf{A}$.

One may ask whether this is a suitable definition of nondeterminism, as it would also be possible to nondeterministically guess interpretations instead of parameters, in analogy to a nondeterministic Turing machine that guesses its next state. But note that programs of that form can easily be simulated in existential PIL by assigning to every possible step-interpretation an equality type of parameters. However, it is not obvious whether a translation in the other direction is possible.

Therefore, we consider existential PIL as the natural nondeterministic extension of PIL. The universal variant is defined analogously.

Definition 4.19. A program in *universal* PIL (∀PIL) is a PIL-program Π with parameters with the following semantics:

Π accepts a structure $\mathbf{A}$ if, and only if, all runs of Π on $\mathbf{A}$ are accepting.

Universal and existential PIL are obviously dual to each other: An ∃PIL-program $((\mathcal{I}, \psi_{\text{halt}}, \psi_{\text{out}}), p)$ rejects the structure $\mathbf{A}$ if, and only if, there is no run on $\mathbf{A}$ whose final state satisfies φ. But this holds if, and only if, the ∀PIL-program $((\mathcal{I}, \psi_{\text{halt}}, \neg\psi_{\text{out}}), p)$ accepts $\mathbf{A}$. So we get

Lemma 4.20. *A query is* ∃PIL*-definable if, and only if, its complement is* ∀PIL*-definable.*

In particular, this means that it suffices to compare ∃PIL to ∃SO; the equivalence between ∀PIL and ∀SO follows directly.

In the following we show that ∃SO reappears as the nondeterministic extension of PIL and thus CPT. Conversely, this means that CPT can be seen as a deterministic restriction of ∃SO.

Lemma 4.21. $\exists\mathrm{SO} \leq \exists\mathrm{PIL}$

Proof. Let $\vartheta = \exists X_1 \dots \exists X_k \varphi$ with $\varphi \in \mathrm{FO}$ be an $\exists$SO-formula. For $1 \leq i \leq k$, let r_i be the arity of X_i, and let $r_{\max}$ be the maximum of the arities. The $\exists$PIL-program $(\Pi = (\mathcal{I}, \psi_{\mathrm{halt}}, \psi_{\mathrm{out}}), p\colon n \mapsto kn^{r_{\max}})$ with $\psi_{\mathrm{halt}} = \psi_{\mathrm{out}} = \varphi$ and $\mathcal{I}$ defined as follows is equivalent to ϑ. $\mathcal{I}$ is a one-dimensional interpretation with $r_{\max} + k$ parameters preserving the domain. For parameters $a_1, \dots, a_{r_{\max}}, b_1, \dots, b_k$, it adds the tuple $a_1, \dots, a_{r_i}$ to the relation X_i if, and only if, $b_i = a_1$.

If $\mathbf{A} \models \vartheta$, then there are relations $X_1, \dots, X_k$ satisfying φ. So the run where the parameters are chosen to successively add the elements of $X_1, \dots, X_k$ to the respective relations is accepting. This run satisfies the polynomial bound because there are at most $|A|^{r_{\max}}$ many tuples that can be added to every relation.

If there exists an accepting run of Π on $\mathbf{A}$, then the final state of that run satisfies φ. Since the final state is just an expansion of $\mathbf{A}$ with relations $X_1, \dots, X_k$, this means that $\mathbf{A}$ satisfies ϑ. □

Lemma 4.22. $\exists\mathrm{PIL} \leq \exists\mathrm{SO}$

Proof. Since $\exists$SO captures NP, it suffices to show that $\exists$PIL-programs can be evaluated in NP. To simulate one application of the step-interpretation, nondeterministically guess the parameters and construct the interpreted structure. This can be done in polynomially many steps because of the polynomial bounds of the $\exists$PIL-program. □

So our nondeterministic extension of PIL indeed corresponds to existential second-order logic. By Lemma 4.20, the analogous result for the universal variant follows directly.

Corollary 4.23. $\forall\mathrm{PIL} \equiv \forall\mathrm{SO}$.

Note that the simulation in the proof of Lemma 4.21 only uses one-dimensional interpretations. So one-dimensional $\exists$PIL suffices to express $\exists$SO, and hence all of $\exists$PIL.

Corollary 4.24. $\exists\mathrm{PIL} \equiv$ *one-dimensional* $\exists\mathrm{PIL}$

In the remainder of this section, we consider the generalisation of this characterisation to arbitrary quantifier alternations. For this purpose we define an alternating version of parameterised PIL. Note that alternating Turing machines are defined via universal and existential states. The states of PIL-programs with parameters, however, only consist of a structure and a tuple of elements from that structure.

So we have to encode in every state of that form whether it is existential or universal. This is implemented by augmenting the output signature with k new relation symbols marking the k alternations between existential and universal states.

Definition 4.25 (Syntax of alternating PIL)**.** Let $(\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}}), p)$ be a PIL$[\sigma, \tau]$-program with parameters. (Π, p) is in Σ_kPIL if there are nullary relations $Y_0, \ldots, Y_{k-1} \in \tau$ such that every formula φ_{Y_i} in $\mathcal{I}_{\text{step}}$ is of the form $Y_i \vee \psi$ for some formula ψ.

(Π, p) with Π as above is in Π_kPIL if there exist nullary relations $Y_1, \ldots, Y_k \in \tau$ such that every formula φ_{Y_i} in $\mathcal{I}_{\text{step}}$ is of the form $Y_i \vee \psi$ for some formula ψ.

A program is in *alternating* PIL, or APIL, if it is in Σ_kPIL or Π_kPIL for some $k \in \mathbb{N}$.

In analogy to alternating Turing machines, the semantics of alternating PIL-programs is defined via a two-player game on the run tree of the program.

Definition 4.26 (Semantics of alternating PIL)**.** An alternating PIL$[\sigma, \tau]$-program $(\Pi = (\mathcal{I}_{\text{init}}, \mathcal{I}_{\text{step}}, \psi_{\text{halt}}, \psi_{\text{out}}), p)$ with ℓ parameters accepts a structure $\mathbf{A}$ if, and only if, Eve has a winning strategy in the following game between players Adam and Eve:

A position of the game is either a pair $(\mathbf{B}, n)$ such that $\mathbf{B}$ is the nth state of any run of (Π, p) on $\mathbf{A}$, or a triple $(\mathbf{B}, \bar{b}, n)$ where $\mathbf{B}, n$ are as in the other case and $\bar{b} \in B^\ell$.

The game starts in the position $(\mathcal{I}_{\text{init}}(\mathbf{A}), 1)$. A position $(\mathbf{B}, \bar{b}, n)$ is a terminal position if $\mathbf{B} \models \psi_{\text{halt}}$, $|\mathcal{I}_{\text{step}}(\mathbf{B}, \bar{b})| > p(|A|)$ or $n = p(|A|)$. Otherwise, its unique successor position is $(\mathcal{I}_{\text{step}}(\mathbf{B}, \bar{b})), n + 1)$.

From the position $(\mathbf{B}, n)$, the current player can move to all $(\mathbf{B}, \bar{b}, n)$ for $\bar{b} \in B^\ell$. The current player is determined by the maximal $i \leq k$ such that $\mathbf{B} \models A_i$. If i is even, Eve moves from position $(\mathbf{B}, n)$, otherwise Adam moves.

Eve wins in the terminal position $(\mathbf{B}, \bar{b}, n)$ if, and only if, $\mathbf{B} \models \psi_{\text{out}}$.

Note that the definition for $k = 1$ implies that Σ_1PIL $\equiv \exists$PIL and Π_1PIL $\equiv \forall$PIL. Furthermore, it is easy to see that replacing every relation Y_i by Y_{i-1} (respectively Y_{i+1}) reverses the roles of the players, so the following holds:

Lemma 4.27. *A query is* Σ_kPIL*-definable if, and only if, its complement is* Π_kPIL*-definable.*

Using the translation between $\exists$PIL and $\exists$SO as the induction base, we can now show a correspondence between the hierarchy of alternating PIL and the quantifier alternation hierarchy for SO.

Lemma 4.28. $\Sigma_k \leq \Sigma_k\text{PIL}$ *and* $\Pi_k \leq \Pi_k\text{PIL}$ *for every* $k \geq 1$.

Proof. Proof by induction on $k \geq 1$. For $k = 1$, the statement follows from the results for ∃PIL and ∀PIL. Let $k > 1$. Because of Lemma 4.27, it suffices to translate Σ_k to Σ_kPIL. So let $\varphi = \exists X_1 \dots \exists X_m \psi \in \Sigma_k$ with $\psi \in \Pi_{k-1}$. By induction hypothesis, there is a Π_{k-1}PIL-program equivalent to ψ.

To obtain a Σ_kPIL-program equivalent to φ, we add a nullary relation Y_0 to the output signature, and modify the initial interpretation such that it sets Y_0 to true and the other Y_i to false.

Analogously to the translation of ∃PIL (Lemma 4.22), the step-interpretation successively adds tuples to the relations $X_1, \dots, X_m$. Recall that the additional parameters $b_1, \dots, b_m$ determine to which relations the current tuple is added. As soon as the current tuple is not added to any relation, the interpretation sets Y_1 to true. If Y_1 is true, the step-interpretation behaves like the one in the program for ψ.

Then Π accepts $\mathbf{A}$ if, and only if, Eve can choose a sequence of parameters such that ψ is satisfied. □

Furthermore, alternating PIL-programs can be evaluated by alternating Turing machines in a straightforward way. All in all, we obtain:

Theorem 4.29.

1. *For every* $k \geq 1$, $\Sigma_k\text{PIL} \equiv \Sigma_k$ *and* $\Pi_k\text{PIL} \equiv \Pi_k$.
2. APIL ≡ SO.

4.3 Conclusion

We explored different ways in which the characterisation via PIL induces natural fragments and extensions of CPT. Two of these fragments—the restriction to two-dimensional interpretations and the one with a single binary output relation—have the same expressive power as PIL. Unary output relations, however, induce a fragment which is, though more expressive than FO+H, strictly included in PIL. For the one-dimensional fragment, we showed that it is not only strictly included in PIL, but also equivalent to fixed-point logic. More precisely, one-dimensional PIL is equivalent to FP+C, and one-dimensional PIL$^-$ is equivalent to the PTIME-restriction of PFP. Using our alternating extensions of PIL with parameters, we showed that PIL can also be viewed as a natural fragment of second-order logic: The hierarchy of alternating PIL corresponds to the quantifier alternation hierarchy. In particular, existential PIL is equivalent to existential second-order logic.

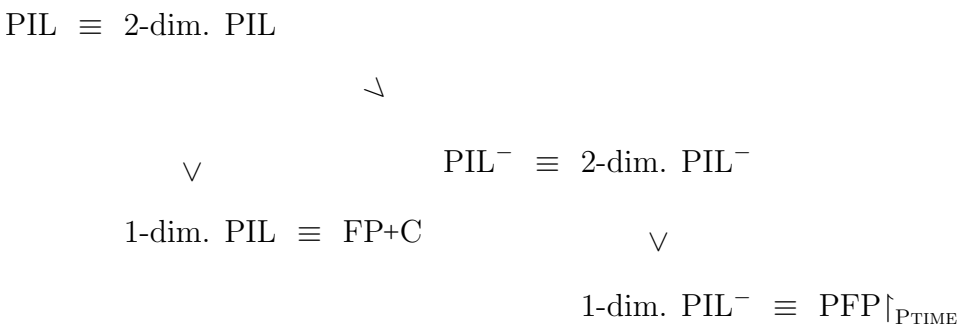

Figure 4.1: Fragments of PIL based on the dimension of interpretations.

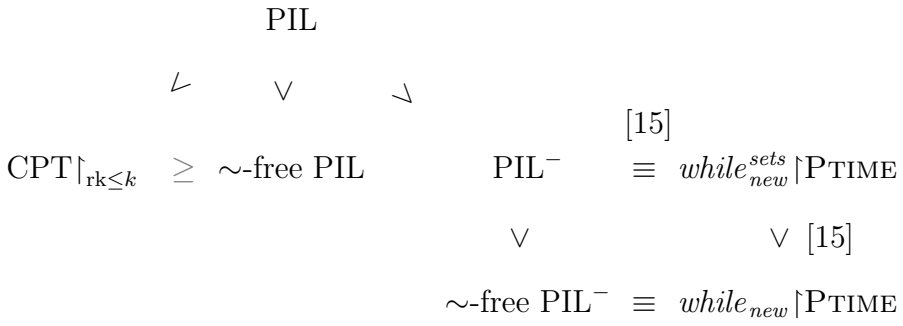

Figure 4.2: Relations between PIL-fragments with and without congruences. The inclusion drawn in grey only holds for structures with bounded colour class size.

Furthermore, we considered the congruence-free fragment of PIL as a formalisation of CPT without the ability to create set-like objects. Both with and without counting, we established that it is strictly weaker than PIL. The expressive power of this fragment is examined further in the next chapter. Our results about fragments of PIL are summed up in Figures 4.1 and 4.2.

So far, the known applications of PIL mainly concern the influence that alterations to the interpretations have on the expressive power. Extended applications may arise when exploiting the fact that the underlying logic is FO. In particular, fragments of PIL based on parameters of the FO-formulae, such as the number of variables or quantifier rank, remain unexplored.

The simplicity of the formalism suggests that it also generates new methods for analysing the expressive power of full CPT. It should, however, be noted that PIL-programs still construct higher-order objects in the form of equivalence classes of tuples—which are necessary to obtain its full expressive power. This indicates that proof techniques for PIL may be as difficult to design as those for CPT in its standard presentation.

5 Expressive power of Choiceless Polynomial Time

The central question about Choiceless Polynomial Time is, in fact, a question about its expressive power: Can it express all queries in PTIME? In this chapter, we review and extend some of the important proof techniques that have been developed for both positive and negative results about CPT's expressive power.

In terms of positive results, we aim to express increasingly difficult PTIME-queries. To push the boundaries of expressibility, the queries that are not FP+C-definable are of particular interest. The prototypical query that comes to mind is the one introduced by Cai, Fürer and Immerman to separate FP+C from PTIME. Recall that some instances of the Cai-Fürer-Immerman query are known to be definable in CPT, whereas the general case is still open.

To make that statement precise, we start with some basic properties of the query, which is explained formally in Section 5.1. The CFI construction associates with every undirected graph a set of *CFI graphs*. When constructing these graphs, every vertex and every edge of the underlying graph is replaced by some gadget, where a vertex gadget can be even or odd. In fact, the CFI graphs can be partitioned into two isomorphism classes, depending on the number of odd gadgets: The even and the odd CFI graphs. Defining the CFI query amounts to deciding whether a given CFI graph is even or odd.

The first definability result, shown by Blass, Gurevich and Shelah [14], concerns the CFI query with padding of exponential size. For the case without padding, Dawar, Richerby and Rossman [23] proved that CPT can define the CFI query if the underlying graph is linearly ordered. A linear order on the underlying graph induces a linear order on the gadgets, and thus a preorder on the CFI graph. We generalise this result to the case where the underlying graphs are already equipped with a preorder, with a certain non-constant bound on the colour class size.

The CPT-algorithm proposed in [23] processes the gadgets as incrementally growing initial sequences of the linear order on the underlying graph. With a preorder on the underlying graph, we can do the same on the level of colour classes

of gadgets. To process the gadgets within a colour class, we make sure that the class is small enough to treat all of its subsets in parallel. This is possible whenever the size of every colour class is logarithmic in the size of the input structure. Our generalised procedure is described in Section 5.3.

A similar idea justifies the side note in [23] that the CFI query over *unordered* complete graphs is CPT-definable as well. For every vertex v of the underlying graph, the CFI graph contains a gadget consisting of half the subsets of v's neighbourhood. Hence if v is connected to n other vertices, the CFI graph is already of size 2^{n-1}. As the CFI graph is the input structure that determines the polynomial bound, a CPT-formula can process *all* subsets of the set of vertex gadgets. Note that this is possible for any class of underlying graphs where at least one vertex of sufficiently large degree is present.

Both the procedure for linearly ordered graphs and our generalisation for pre-ordered graphs make use of highly nested sets. Dawar, Richerby and Rossman [23] show that this nesting of sets is necessary: No CPT-program that activates only sets of constant rank can define the CFI query over ordered graphs.

The sets defined by the CPT-formula for linearly ordered graphs are built up similarly to Kuratowski tuples: Their rank grows linearly with the length of the tuple, or, in our case, the sequence of vertex gadgets. So we can avoid increasing the rank by processing sets instead of sequences. As argued above, this is possible for CFI graphs of exponential size. All in all, we show in Section 5.4 that for every class of graphs possessing a vertex of large degree, CPT can define the CFI query using only sets of small, constant rank.

Note that we *cannot* access all $n!$ many linear orderings of the underlying graph with any resource bound polynomial in 2^n. Intuitively, this means that the power to use sets instead of sequences is vital for our CPT-procedure to respect polynomial bounds. Can this observation be generalised? Do sets, even restricted to bounded rank, lead to greater expressive power than tuples?

In Chapter 4, we established that set-based formalisms are stronger than tuple-based ones. More precisely, PIL is strictly more expressive than its congruence-free fragment. Interestingly, this separation was obtained by translating $\sim$ -free PIL to CPT over sets of bounded rank (on structures of bounded colour class size).

Using our results about CFI graphs, we can now separate these fragments. So the aim is to show that the CFI query over graphs of large degree, which is definable in CPT with bounded rank, cannot be defined in $\sim$ -free PIL. This is where known techniques for inexpressibility results [13, 23] come into play, which we review in Section 5.2.

Inexpressibility proofs for CPT are based on reducing indistinguishability in CPT to games for FP+C played over the active objects. This requires some way to characterise the kind of sets that can be activated by the CPT-fragment to be analysed. Usually, this characterisation is based on the size of supports. To capture the intuition of "tuple-like" sets, we introduce the notion of *strong supports*.

Indeed, CPT over sets with strong supports includes PIL without congruences. In Section 5.5, we extend the method coined in [13, 23] to accommodate sets with our property to show that, even over graphs of large degree, the CFI query cannot be defined in CPT using only tuple-like objects.

5.1 The Cai-Fürer-Immerman construction

The results in this chapter mostly concern expressibility of different variants of the Cai-Fürer-Immerman query. We first present a definition and some properties of that query.

The CFI construction associates with each connected graph G and each subset T of its vertex set a CFI graph $\mathbf{G}^T$. Up to isomorphism, there are only two such graphs, depending on the parity of $|T|$: The even and the odd CFI graph. The CFI query asks, given a CFI graph, whether it is even or odd.

We present here the version of the construction given in [23, 49]. In the following, let $G = (V, E)$ be an undirected graph. All CFI graphs $\mathbf{G}^T$ for $T \subseteq V$ occur as subgraphs of the following graph $\mathbf{G}$: The vertex set of $\mathbf{G}$ is partitioned into vertex gadgets and edge gadgets. For $v \in V$, it contains the *vertex gadget* $v^* = \{v^X : X \subseteq E(v)\}$, and, for $e \in E$, the *edge gadget* $e^* = \{e^0, e^1\}$. We denote by $V^* := \cup_{v \in V} v^*$ the set of *inner vertices*, and by $E^* := \cup_{e \in E} e^*$ the *outer vertices*. For any subset $F \subseteq E$, let $F^* := \cup_{e \in F} e^*$. The edges of $\mathbf{G}$ connect every inner vertex v^X to exactly those e^1 with $e \in X$, and the e^0 for $e \in E(v) \setminus X$. Now let $T \subseteq V$. The subgraph $\mathbf{G}^T$ contains a subset of every vertex gadget v^*, depending on whether $v \in T$. If $v \in T$, then $v^*_T = \{v^X : |X| \text{ is odd}\}$ and we call v^*_T an *odd gadget*. Otherwise, $v^*_T = \{v^X : |X| \text{ is even}\}$ is an *even gadget*. The set of inner vertices in $\mathbf{G}^T$ is $V^*_T := \cup_{v \in V} v^*_T$.

The graph $\mathbf{G}^T$ is the subgraph of $\mathbf{G}$ induced by $E^* \cup \bigcup_{v \in V} v^*_T$. We call $\mathbf{G}^T$ even if $|T|$ is even, and odd otherwise. The construction is illustrated in Figure 5.1.

Consider an automorphism of $\mathbf{G}$ that fixes v^* set-wise for each $v \in V$. Such an automorphism is always completely determined by a set $F \subseteq E$ such that for each $e \in F$, e^0 and e^1 are swapped. Formally, ρ_F is the mapping $e^i \mapsto e^{i-1}$ for $e \in F$ and

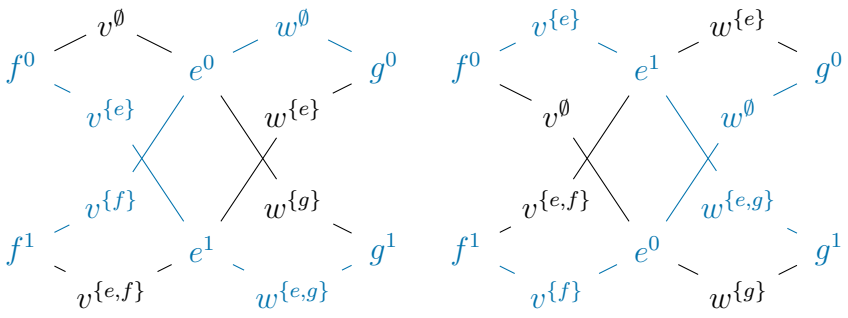

Figure 5.1: A subgraph of the graph $\mathbf{G}$. The subgraph highlighted on the left corresponds to the CFI graph $\mathbf{G}^{\{v\}}$. The automorphism ρ_e maps $\mathbf{G}^{\{v\}}$ to $\mathbf{G}^{\{w\}}$, an isomorphic copy of which is highlighted on the right.

$i \in \{0,1\}$, $e^i \mapsto e^i$ for $e \notin F$, and $v^X \mapsto v^{X \triangle (F \cap E(v))}$. The group of automorphisms of $\mathbf{G}$ that fix each v^* is generated by $\{\rho_e \coloneqq \rho_{\{e\}} : e \in E\}$.

As illustrated in Figure 5.1, each ρ_e maps $\mathbf{G}^T$ to some $\mathbf{G}^S$ such that T and S have the same parity. As shown by Cai, Fürer and Immerman [17], it follows that, for every connected graph G, $\mathbf{G}^S \cong \mathbf{G}^T$ if, and only if, $|S| \equiv |T| \pmod 2$. In other words, the even and odd CFI graphs are uniquely determined up to isomorphism, and the automorphisms of $\mathbf{G}$ fixing every v^* set-wise are exactly the isomorphisms between graphs $\mathbf{G}^T$ and $\mathbf{G}^S$ of the same parity.

By the nature of these isomorphisms, the parity of any given CFI graph can be determined by an algorithm as follows: The input of the algorithm is an encoding of the CFI graph, which induces a linear order on its vertex set. In particular, there is a unique least vertex in every edge gadget. Since flipping any set of edges results in a CFI graph of the same parity, the algorithm can assume that the least vertex in every gadget e^* is e^0, and deduce the parity of every vertex gadget from that information. This is possible in polynomial time, and even in logarithmic space.

The class of CFI graphs over a graph class $\mathcal{C}$ is the class of all $\mathbf{G}^T$ for $G \in \mathcal{C}$. If G is preordered by $\preceq$, then $\preceq$ induces a preorder on $\mathbf{G}$ (and thus all $\mathbf{G}^T$) in the obvious way. Further, if G is linearly ordered, this induces a total preorder on $\mathbf{G}$, which we also denote by $\preceq$.

5.2 Techniques for showing (in-)expressibility results

The results presented in this chapter are shown extending established methods for both positive and negative results about CPT. As we prove positive results about variants of the CFI query, we extend the algorithm proposed for CFI graphs

over ordered graphs by Dawar, Richerby and Rossman [23]. Interestingly, their approach using highly symmetric sets can be generalised to support CPT-definitions for solvability of cyclic linear equation systems [4].

The technique we adopt for our inexpressibility result has been used for negative results about varying fragments of CPT: The variant CPT^- without counting [13], CPT over sets of bounded rank [23] and even CPT itself [54]. These techniques thus constitute a vital part of the standard toolkit for analysing CPT. Before we present our own results, we therefore provide a few details about the approaches they are based on.

5.2.1 Super-symmetry

By the very nature of CPT as a logic, the sets it can define are fixed by all automorphisms of the input structure. The objects constructed in the procedure for the CFI query introduced by Dawar, Richerby and Rossman [23] are additionally fixed by isomorphisms between CFI graphs. In consequence, these objects are called *super-symmetric*. Before we make that notion precise, let us provide some intuition.

A set that is fixed by all isomorphisms between CFI graphs can be seen as canonical in the sense that the sets for all CFI graphs of the same parity (over some fixed, ordered graph) coincide. These "canonical" sets are built in a way that they characterise the parity of the CFI graph. To see how the parity can be extracted, consider the fact that isomorphisms between CFI graphs can flip any subset of the underlying graph's edge set.

For every edge gadget e^* there are two possible labellings assigning names e^0 and e^1 to the two vertices in the gadget. But within a set that is fixed by the isomorphism flipping exactly the edge e, both assignments yield the same result. In case the underlying graph is ordered, the order induced on its edge set can be used to apply these labellings one edge at a time, thus circumventing the need to define a set for each of the exponentially many labellings of the whole graph.

On the one hand, this corresponds to the standard way of solving the CFI query on a Turing machine: Simply assign an arbitrary label to every vertex in an edge gadget and compute the resulting parity. Applying both labellings at once can then be seen as the usual approach to CPT-formulae of replacing arbitrary choice by parallel "computation".

On the other hand, the CFI query can also be reduced to solving a linear equation system over $\mathbb{F}_2$. Then applying the two labellings to an edge gadget means trying both possible assignment of a pair of variables. This view indeed allows for a

generalisation of the sets defined in [23]: In [4], a similar idea is used to solve *cyclic linear equation systems*, a class of equation systems including the CFI query as a special case.

The super-symmetric objects from [23] form the basis of our CPT-formulae for different variants of the CFI query. So we briefly present the original construction here: The starting point are sets τ_v and $\widetilde{\tau}_v$ for every vertex v of the underlying graph. These sets characterise whether the gadget v^* is odd (i. e. $v \in T$ in the graph $\mathbf{G}^T$) in the following way: Every τ_v consists of all sets $\{e^{i_e} : v \in e\}$, where the number of e^{i_e} in every set with $i_e = 1$ is even if, and only if, v^* is an even gadget. The $\widetilde{\tau}_v$ contain the corresponding sets with an odd number of e^i if, and only if, v^* is even. Since the sets in τ_v are exactly the neighbourhoods of the vertices $v^X \in v^*$, these sets are CPT-definable.

Now consider an isomorphism $\rho\colon \mathbf{G}^T \to \mathbf{G}^S$ flipping an edge $\{u, v\}$. Then ρ exchanges τ_v and $\widetilde{\tau}_v$.

Again using the linear order on the underlying graph, the τ_v and $\widetilde{\tau}_v$ are combined into sets $\mu_i, \widetilde{\mu}_i$ for $i \leq |V|$, each of which represents the parity of T restricted to the first i vertices of the underlying graph:

$$\begin{aligned}
&\mu_1 = \tau_{v_1},\ \widetilde{\mu}_1 = \widetilde{\tau}_{v_1},\\
&\mu_{i+1} = \{\langle \mu_i, \tau_{v_{i+1}}\rangle, \langle \widetilde{\mu}_i, \widetilde{\tau}_{v_{i+1}}\rangle\}, \text{and}\\
&\widetilde{\mu}_{i+1} = \{\langle \mu_i, \widetilde{\tau}_{v_{i+1}}\rangle, \langle \widetilde{\mu}_i, \tau_{v_{i+1}}\rangle\}.
\end{aligned}$$

The μ_i can be seen as sequences of $\tau_v, \widetilde{\tau}_v$ containing an even number of $\widetilde{\tau}_v$, and the $\widetilde{\mu}_i$ contain an odd number of $\widetilde{\tau}_v$. In that sense, $\mu_{|V|}$ and $\widetilde{\mu}_{|V|}$ characterise the parity of all of T. Further, an automorphism flipping an edge $\{u, v\}$ replaces τ_u by $\widetilde{\tau}_u$ and τ_v by $\widetilde{\tau}_v$ (and vice versa), so the parity and thus $\mu_{|V|}$ is preserved.

With that argument it can be shown that $\mu_{|V|}$ is super-symmetric, and the assignments labelling vertices in E^* as e^0 or e^1 can be computed as explained above. In the set labelled like this, every τ_v contains sets with an odd number of vertices labelled e^1 if, and only if, $v \in T$. So the parity of T can be extracted from $\mu_{|V|}$.

5.2.2 Bijection games on supports

For many logics, inexpressibility can be shown via pebble games. Choiceless Polynomial Time makes the definition of such pebble games difficult because of both the explicit polynomial bound and the construction of arbitrarily nested hereditarily

finite sets. There is, however, a way of using known pebble games for infinitary logics to show inexpressibility results for certain fragments of Choiceless Polynomial Time.

The technique we present here has been introduced by Blass, Gurevich and Shelah [13] for inexpressibility of the parity query in CPT^- and extended by Dawar, Richerby and Rossman [23] to show that, when restricted to sets of bounded rank, CPT cannot define the Cai-Fürer-Immerman query.

The first step towards using standard pebble games is to translate CPT-formulae to fixed-point logic (with or without counting) in some way. But since CPT can define hereditarily finite sets, the translated formulae must have some means to access these sets. This is achieved by evaluating the FP-formula over the hereditarily finite objects activated by the CPT-formula, i. e. those that occur in a stage of some iteration term. More precisely, for every $\mathrm{CPT}[\sigma]$-formula (φ, p), there is an FP+C-formula $\varphi_{\mathrm{FP+C}}$ such that, for every σ-structure $\mathbf{A}$,

$$\mathbf{A} \models (\varphi, p) \Leftrightarrow \mathbf{act}(\varphi, p, \mathbf{A}) \models \varphi_{\mathrm{FP+C}}.$$

The same holds for CPT^- and FP without counting, which has originally been shown in [13]. As we show in Section 3.4, it is even possible to translate CPT to $\mathrm{FO+H}_\sim$, and CPT^- to FO, in that sense.

It follows that every CPT-sentence can be translated to a sentence in C^m for some m. Hence, if we can show for two structures $\mathbf{A}$ and $\mathbf{B}$ that $\mathbf{act}(\varphi, p, \mathbf{A})$ and $\mathbf{act}(\varphi, p, \mathbf{B})$ cannot be distinguished by any C^m-formula (i. e. $\mathbf{act}(\varphi, p, \mathbf{A}) \equiv_m^C \mathbf{act}(\varphi, p, \mathbf{B})$), then we can conclude that φ cannot distinguish between $\mathbf{A}$ and $\mathbf{B}$ either.

However, there are two serious difficulties in this approach:

- Showing that $\mathbf{act}(\varphi, p, \mathbf{A}) \equiv_m^C \mathbf{act}(\varphi, p, \mathbf{B})$ is combinatorially challenging, because these structures contain highly-nested sets over the two input structures.
- The structures $\mathbf{act}(\varphi, p, \mathbf{A})$ and $\mathbf{act}(\varphi, p, \mathbf{B})$ depend on the sentence (φ, p). Since we want to obtain an undefinability result, for all we know this sentence (φ, p) can behave completely arbitrarily.

The approach from [13, 23] solves both problems at once: Under certain conditions, hereditarily finite sets can be represented as a combination of *form* and *matter*. The idea is to isolate the set structure (represented by the form) from the atoms (the matter) that determine the actual set. A sequence of atoms, called a *molecule* in that setting, characterises a set a if it supports a.

The complexity of the pebble game can then be reduced using the intuition that pebbles are placed on supports instead of sets. This is possible if the supports are of bounded size. So in case all objects activated by CPT-formulae in a certain fragment have supports of bounded size, the game can be played on all hereditarily finite sets with a support of that size. In contrast to the structure over the active objects, this structure is, for every size bound k, independent of the CPT-formula.

For structures over the empty signature, for instance, Blass, Gurevich and Shelah [13] provide a bound for the supports of objects activated by *any* CPT^--formula. Forms specify sets in terms of types, that is, the atoms actually occurring in the set are defined in C^m with respect to the molecule. Whenever the input structure is C^m-homogeneous, there is a correspondence between k-supported objects and those that are represented by forms and molecules. C^m-equivalence of the structures over k-supported objects is shown by giving a winning strategy for Duplicator in the (bijective) pebble game. Using the description of sets in terms of forms and matter, this strategy can be obtained from a winning strategy on the input structures themselves where Spoiler places his pebbles on the molecules. So proving inexpressibility results amounts to transferring winning strategies in that way.

5.3 Graphs with colour classes of logarithmic size

Our first definability result generalises the result from [23] to CFI graphs over certain preordered graphs. Recall that, over ordered graphs, the sets $\mu_i, \widetilde{\mu}_i$ always represent the parity of an initial segment of the (ordered) vertex set of the underlying graph. The main idea is to generalise this to sufficiently small colour classes instead of single vertices. This is possible because a colour class of size logarithmic in the size of the CFI graph has only linearly many subsets. So we can construct sets representing the parity of every subset of a colour class. This leads to the following class of graphs:

Theorem 5.1. *For $c \in \mathbb{N}$, define the class $\mathcal{K}$ of connected, f-bounded graphs with $f\colon |G| \mapsto c \log |\mathbf{G}|$, where $\mathbf{G}$ is the (without loss of generality even) CFI graph over G. Then the CFI query over $\mathcal{K}$ is definable in Choiceless Polynomial Time.*

Implementing this idea requires several technical modifications of the algorithm from [23] explained in the previous section. The first one is due to the fact that we now need to define objects representing the parity of subsets of the unordered colour classes. So, instead of the sets μ_i, $\widetilde{\mu}_i$ described before, we define sets $\mu_W, \widetilde{\mu}_W$ for subsets $W \subseteq V$ defined as follows: First, we construct objects for all subsets of

the first colour class C_1, and obtain sets μ_{C_1} and $\widetilde{\mu}_{C_1}$ for the full colour class. These are combined with all subsets of the second colour class, which yields $\mu_{C_1 \cup C_2}$, and so on for the remaining colour classes.

But we aim to still encode the $\mu_W, \widetilde{\mu}_W$ similarly to the $\mu_i, \widetilde{\mu}_i$ from [23], in the sense that a specific "last" τ_v remains definable. Therefore, for each subset $W \subseteq V$ for which μ_W and $\widetilde{\mu}_W$ are constructed, all ways to decompose W as $W = U \uplus \{v\}$ for some v in the current colour class and $U \subseteq V$ are considered. Here $\uplus$ denotes that the sets U and $\{v\}$ are already disjoint.

This encoding is a preparation for the second modification, which affects the procedure that replaces vertices e^i with their labels from $\{0, 1\}$. The algorithm from [23] uses the linear order on the edges to only process one edge gadget at a time. Since this is not possible in our case, we already assign the labels to edge gadgets during the construction of the $\mu_W, \widetilde{\mu}_W$. Whenever W is decomposed into $U \uplus \{v\}$, it is still possible to define a sufficiently small set of edge gadgets to label: We label the gadgets corresponding to edges in the cut $(U, \{v\})$.

Since the decomposition $W = U \uplus \{v\}$ is only imminent during the construction, our CPT-formula immediately defines partially labelled objects λ_W, $\widetilde{\lambda}_W$ where certain e^i are already processed. By induction, these are exactly the e^i for $e \in E[W]$, i. e. in the subgraph induced by W. Finally, the parity of T is extracted from the object λ_V using a parity function p.

Next, we describe these steps in detail. The sets τ_v, $\widetilde{\tau}_v$ for $v \in V$ are defined as in [23], with one difference: To allow for assigning labels 0 or 1 to several vertices at once without losing information, we equip each neighbourhood with a parity bit, which will encode the parity of the number of edges that are labelled 1.

Definition 5.2. Let $v \in V$. Then

$$\tau_v^T = \{\langle N(v^X), 0\rangle : v^X \in v_T^*\} \text{ and}$$
$$\widetilde{\tau}_v^T = \{\langle N(v^X), 0\rangle : v^X \in v^* \setminus v_T^*\},$$

where $N(v^X)$ is the neighbourhood of v^X in the full graph $\mathbf{G}$.

We omit the superscript T whenever it is clear from the context.

Example 5.3. Consider the vertex v whose gadget v^* is shown in Figure 5.2. Then

$$\tau_v = \{\langle\{e^i, f^\ell, g^m\}, 0\rangle, \langle\{e^j, f^k, g^m\}, 0\rangle, \langle\{e^i, f^k, g^n\}, 0\rangle, \langle\{e^j, f^\ell, g^n\}, 0\rangle\}$$

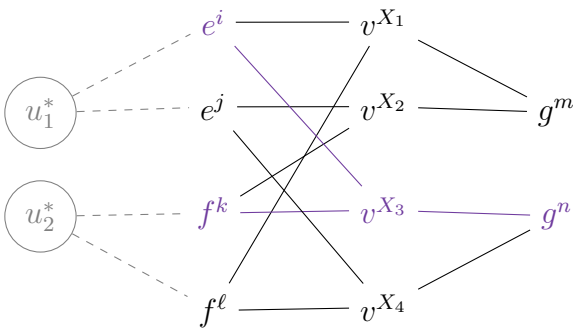

Figure 5.2: A subgraph of an unlabelled CFI graph, where $i, j, k, \ell, m, n \in \{0, 1\}$.

and

$$\widetilde{\tau}_v = \{\langle\{e^i, f^k, g^m\}, 0\rangle, \langle\{e^i, f^\ell, g^n\}, 0\rangle, \langle\{e^j, f^k, g^n\}, 0\rangle, \langle\{e^j, f^\ell, g^m\}, 0\rangle\}.$$

The τ_v, $\widetilde{\tau}_v$ are combined to obtain objects μ_W, $\widetilde{\mu}_W$ for those subsets $W \subseteq V$ that are built up according to the preorder as explained above:

Definition 5.4. Let $\mathcal{W}_i$ be the set of all $W \subseteq V$ such that $W = \bigcup_{j<i} C_j \cup U$ for some $U \subseteq C_i$. Then let $\mathcal{W}$ be the set of all $W \subseteq V$ that are in $\mathcal{W}_i$ for some i.

For subsets $W \in \mathcal{W}$, the objects $\mu_W, \widetilde{\mu}_W$ are constructed inductively as follows:

Definition 5.5. Let $U, W \in \mathcal{W}$, $v \in V \setminus U$ and $T \subseteq V$. Then

$$\begin{aligned}
\mu^T_{\{v\}} &= \tau^T_v, \\
\widetilde{\mu}^T_{\{v\}} &= \widetilde{\tau}^T_v, \\
\mu^T_{U+v} &= \{\langle \mu^T_U, \tau^T_v\rangle, \langle \widetilde{\mu}^T_U, \widetilde{\tau}^T_v\rangle\}, \\
\widetilde{\mu}^T_{U+v} &= \{\langle \mu^T_U, \widetilde{\tau}^T_v\rangle, \langle \widetilde{\mu}^T_U, \tau^T_v\rangle\},
\end{aligned}$$

and, for $|W| > 1$,

$$\begin{aligned}
\mu^T_W &= \{\mu^T_{U+v} : U \in \mathcal{W} \text{ and } U \uplus \{v\} = W\} \text{ and} \\
\widetilde{\mu}^T_W &= \{\widetilde{\mu}^T_{U+v} : U \in \mathcal{W} \text{ and } U \uplus \{v\} = W\}.
\end{aligned}$$

Like the μ^T_i, $\widetilde{\mu}^T_i$ in the procedure by Dawar, Richerby and Rossman, the μ^T_W, $\widetilde{\mu}^T_W$ characterise the parity of $|T \cap W|$, which holds by an easy induction on $|W|$.

Lemma 5.6. $\mu^T_W = \mu^S_W \Leftrightarrow \widetilde{\mu}^T_W = \widetilde{\mu}^S_W \Leftrightarrow |S \cap W| \equiv |T \cap W| \pmod 2$ *and* $\mu^T_W = \widetilde{\mu}^S_W \Leftrightarrow \widetilde{\mu}^T_W = \mu^S_W \Leftrightarrow |S \cap W| \not\equiv |T \cap W| \pmod 2$.

Our aim is to assign labels to certain edges already while building up the objects $\mu_W^T, \widetilde{\mu}_W^T$. These labellings should have the same effect as the "correct" (albeit not CPT-definable) labelling which, on removing a vertex e^i from a neighbourhood, flips its parity bit if, and only if, $i = 1$. We later use this labelling to prove the correctness of the labellings that are CPT-definable.

Definition 5.7. Let $e \in E$. Then $L_e^{\text{edges}}(\langle z, i\rangle) = \langle z \setminus \{e\}, i + j\rangle$, where $j = 1$ if, and only if, $e^1 \in z$ and $+$ is addition modulo 2. For any other set y, $L_e^{\text{edges}}(y) = \{L_e^{\text{edges}}(z) : z \in y\}$, given that $L_e^{\text{edges}}(z)$ is defined for all $z \in y$. Let $F \subseteq E$ and let $e_1, \ldots, e_k$ be an enumeration of F. Then $L_F^{\text{edges}} = L_{e_1}^{\text{edges}} \circ \cdots \circ L_{e_k}^{\text{edges}}$. Note that, for any $e \neq e'$, $L_e^{\text{edges}} \circ L_{e'}^{\text{edges}} = L_{e'}^{\text{edges}} \circ L_e^{\text{edges}}$, so L_F^{edges} is well-defined.

By Lemma 5.6, the objects $\mu_W^T, \widetilde{\mu}_W^T$ characterise the parity of $|T \cap W|$. We proceed to show that, after applying the labelling functions, that parity can be extracted with the following function.

Definition 5.8.

$$p(x) = \begin{cases} i, & \text{if } x = \left\langle N\left(v^X\right), i\right\rangle, \\ p(x_1) + p(x_2) \mod 2, & \text{if } x = \langle x_1, x_2\rangle \text{ and } x_2 \notin \{0, 1\}, \\ \prod_{y \in x} p(y), & \text{if } x \text{ is any other set.} \end{cases}$$

The correctness of the labellings L^{edges} and the parity extraction function is expressed by the following lemma.

Lemma 5.9. *Let $W \in \mathcal{W}$ and $T \subseteq V$. Then, for all $U \in \mathcal{W}$ and $v \in V$ such that $U \uplus \{v\} = W$,*

$$p(L_E^{\text{edges}}(\mu_W^T)) = p(L_E^{\text{edges}}(\mu_{U+v}^T)) = |T \cap W| \bmod 2, \text{ and}$$
$$p(L_E^{\text{edges}}(\widetilde{\mu}_W^T)) = p(L_E^{\text{edges}}(\widetilde{\mu}_{U+v}^T)) = 1 - |T \cap W| \bmod 2.$$

Proof. By Lemma 5.6, the objects μ_W^T (resp. $\widetilde{\mu}_W^T$) coincide for all T where $|T \cap W|$ is even (resp. odd). Therefore it suffices to consider the cases $T \cap W = \emptyset$ and $T \cap W = \{x\}$ for some $x \in W$. We proceed by induction on $|W|$ for both cases simultaneously. For ease of notation, we use the shorthand $P(z)$ for $p(L_E^{\text{edges}}(z))$.

In the induction base, $W = \{v\}$ for some $v \in V$. If $T \cap W = \emptyset$, then v_T^* is an even gadget. Then τ_v^T consists of those neighbourhoods of the vertices v^X that contain an even number of vertices of the form e^1. So all parity bits are 0 in $L_E^{\text{edges}}(\tau_v^T)$, and 1 in $L_E^{\text{edges}}(\widetilde{\tau}_v^T)$. Hence $P(\mu_{\{v\}}^T) = 0$ and $P(\widetilde{\mu}_{\{v\}}^T) = 1$.

If $T \cap W = \{x\}$, then $x = v$, so v^*_T is an odd gadget. With the same reasoning as above, $P(\mu_{\{v\}})^T = 1$ and $P(\widetilde{\mu}_{\{v\}})^T = 0$.

For $|W| > 1$, the induction hypothesis implies that, if $T \cap W = \emptyset$, then

$$P(\mu^T_{U+v}) = (P(\mu^T_U) + P(\tau^T_v)) \cdot (P(\widetilde{\mu}^T_U) + P(\widetilde{\tau}^T_v)) = (0+0) \cdot (1+1) = 0$$

for all decompositions of W as $U \uplus \{v\}$. It follows that $P(y) = 0$ for all $y \in \mu_W$, so $P(\mu^T_W) = 0$.

If $T \cap W = \{x\}$, then, by definition of τ_v and $\widetilde{\tau}_v$,

$$P\left(\mu^T_{\{v\}}\right) = 1 \Leftrightarrow P\left(\widetilde{\mu}^T_{\{v\}}\right) = 0 \Leftrightarrow v = x.$$

Now if $x \in U \uplus \{v\}$, then either $x \in U$ and $P(\mu^T_U) = 1$ by the induction hypothesis, or $v = x$ and $P(\tau^T_v) = 1$. In both cases, $P(\mu^T_{U+v}) = 1$. Finally, since again the value of μ^T_{U+v} only depends on $W = U \uplus \{v\}$,

$$P\left(\mu^T_W\right) = 1 = |T \cap W| \bmod 2, \text{ and } P\left(\widetilde{\mu}^T_W\right) = 0 = 1 - |T \cap W| \bmod 2. \qquad \square$$

Next we define the objects $\lambda_W, \widetilde{\lambda}_W$ with partially labelled edges. To obtain $\lambda_W = L^{\text{edges}}_{E[W]}(\mu^T_W)$, we label the edges in the cut $(U, \{v\})$ whenever the objects μ_{U+v} and $\widetilde{\mu}_{U+v}$ are constructed by combining μ_U and $\widetilde{\mu}_U$ with τ_v and $\widetilde{\tau}_v$. Since v is fixed at that point, a sufficiently small set of labellings with the desired effect can be inferred from v^*: Each v^X defines through its neighbourhood a unique vertex from each adjacent edge gadget, as shown in Figure 5.2 for the vertex v^{X_3}. This allows to define, for each pair v^X, U, a mapping labelling edge vertices with binary values and aggregating these values in the parity check bit.

Definition 5.10. Let $v^X \in V^*$ and $U \subseteq V$. We define a mapping $L^{\text{cut}}_{v^X,U}$ that maps certain sets in $\text{HF}(E^*)$ to sets where the neighbourhoods and parity check bits are modified according to the labelling defined by v^X. For each set z and $i \in \{0,1\}$, let $L^{\text{cut}}_{v^X,U}(\langle z, i\rangle) = \langle z \setminus (U, \{v\})^*, j\rangle$, where $j = i + |(U, \{v\})^* \cap N(v^X)| \bmod 2$. For any other set y such that $L^{\text{cut}}_{v^X,U}(z)$ is defined for all $z \in y$, we define $L^{\text{cut}}_{v^X,U}(y) = \{L^{\text{cut}}_{v^X,U}(z) : z \in y\}$.

Recall that $(U, \{v\})^*$ denotes the set of vertices in edge gadgets corresponding to edges in the cut $(U, \{v\})$.

The objects $\lambda_W, \widetilde{\lambda}_W$ are then constructed by applying the mappings for all decompositions $U \uplus \{v\}$ and all $v^X \in v^*$ in parallel.

Definition 5.11. Let $U, W \in \mathcal{W}$, $v \in V \setminus W$, $T \subseteq V$ and $|W| > 1$. Then

$$\begin{aligned}
\lambda^T_{\{v\}} &= \tau^T_v, \\
\widetilde{\lambda}^T_{\{v\}} &= \widetilde{\tau}^T_v, \\
\lambda^T_{U+v} &= \{\langle \lambda^T_U, \tau^T_v \rangle, \langle \widetilde{\lambda}^T_U, \widetilde{\tau}^T_v \rangle\}, \\
\widetilde{\lambda}^T_{U+v} &= \{\langle \lambda^T_U, \widetilde{\tau}^T_v \rangle, \langle \widetilde{\lambda}^T_U, \tau^T_v \rangle\},
\end{aligned}$$

and, for $|W| > 1$,

$$\begin{aligned}
\lambda_W &= \left\{ L^{\mathrm{cut}}_{v^X,U}(\lambda_{U+v}) : U \in \mathcal{W}, U \uplus \{v\} = W, \text{ and } v^X \in v^*_T \right\} \text{ and} \\
\widetilde{\lambda}_W &= \left\{ L^{\mathrm{cut}}_{v^X,U}(\widetilde{\lambda}_{U+v}) : U \in \mathcal{W}, U \uplus \{v\} = W, \text{ and } v^X \in v^*_T \right\}.
\end{aligned}$$

Applying the mapping $L^{\mathrm{cut}}_{v^X,\{u_1,u_2\}}$ to the set τ_v from Example 5.3 results in the set $\{\langle \{g^m\}, 0\rangle, \langle \{g^n\}, 1\rangle\}$.

Recall that the construction by Dawar, Richerby and Rossman [23] subsequently applies the two possible labellings of each edge gadget. By the super-symmetry property, these two mappings yield the same result. We prove that the sets λ_{U+v} and $\widetilde{\lambda}_{U+v}$ exhibit a restricted form of super-symmetry: they are fixed by all automorphisms of the full CFI graph $\mathbf{G}$ that only flip edges in the cut $(U, \{v\})$. This will imply that all labellings $L^{\mathrm{cut}}_{v^X,U}$ have the same effect, which means that the objects λ_W do not grow too large. In the end, we will use this property to show that these mappings correspond to the "correct" labellings $L^{\mathrm{edges}}_{E[W]}$ of the edges in the subgraph processed so far.

Lemma 5.12. *If $z \in \mathrm{HF}(E^*)$ is fixed by the automorphisms ρ_e for all $e \in (U, \{v\})$, then, for any $v^X \in v^*$, $L^{\mathrm{cut}}_{v^X,U}(z) = L^{\mathrm{edges}}_{(U,\{v\})}(z)$, and $L^{\mathrm{cut}}_{v^X,U}(z)$ is fixed by the same automorphisms ρ_e.*

Proof. The edges in the cut $(U, \{v\})$ that gain different labels from $L^{\mathrm{cut}}_{v^X,U}$ and $L^{\mathrm{edges}}_{(U,\{v\})}$ are exactly those in the set $F = (E(v) \setminus X) \cap (U, \{v\})$. This effect is undone by flipping these edges in z, so $L^{\mathrm{cut}}_{v^X,U}(\rho_F(z)) = L^{\mathrm{edges}}_{(U,\{v\})}(z)$. But, since $F \subseteq (U, \{v\})$, $\rho_F(z) = z$. Hence $L^{\mathrm{cut}}_{v^X,U}(z) = L^{\mathrm{cut}}_{v^X,U}(\rho_F(z)) = L^{\mathrm{edges}}_{(U,\{v\})}(z)$

Further, ρ_e fixes $L^{\mathrm{cut}}_{v^X,U}(z)$ for $e \in (U, \{v\})$, because the corresponding CFI vertices do not occur in $L^{\mathrm{cut}}_{v^X,U}(z)$. □

We use the restricted super-symmetry property to show that the definition of the $\lambda_W, \widetilde{\lambda}_W$ has the desired effect.

Lemma 5.13. *For every $W \in \mathcal{W}$ and $T \subseteq V$, $\lambda^T_W = L^{\mathrm{edges}}_{E[W]}(\mu^T_W)$.*

Proof. We will prove the following statements simultaneously by induction on $|W|$, for all $S, T \subseteq V$, $v \in V$ and $U, W \in \mathcal{W}$ such that $U \uplus \{v\} = W$:

1. a) $\lambda^T_{U+v} = \lambda^S_{U+v} \Leftrightarrow \widetilde{\lambda}^T_{U+v} = \widetilde{\lambda}^S_{U+v} \Leftrightarrow |T \cap W| \equiv |S \cap W| \pmod 2$,
   b) $\widetilde{\lambda}^T_{U+v} = \lambda^S_{U+v} \Leftrightarrow \lambda^T_{U+v} = \widetilde{\lambda}^S_{U+v} \Leftrightarrow |T \cap W| \not\equiv |S \cap W| \pmod 2$.
2. λ_{U+v}, $\widetilde{\lambda}_{U+v}$ are fixed by the automorphism ρ_e for every $e \in (U, \{v\})$.
3. $\lambda_W = L^{\text{edges}}_{E[W]}(\mu_W)$ and $\widetilde{\lambda}_W = L^{\text{edges}}_{E[W]}(\widetilde{\mu}_W)$.
4. a) $\lambda^T_W = \lambda^S_W \Leftrightarrow \widetilde{\lambda}^T_W = \widetilde{\lambda}^S_W \Leftrightarrow |T \cap W| \equiv |S \cap W| \pmod 2$,
   b) $\lambda^T_W = \widetilde{\lambda}^S_W \Leftrightarrow \widetilde{\lambda}^T_W = \lambda^S_W \Leftrightarrow |T \cap W| \not\equiv |S \cap W| \pmod 2$.

For $|W| = 1$, $\lambda_W = \mu_W = \tau_v$ for some v, and there is no decomposition into $U \uplus \{v\}$. So the induction base is trivial. Now let $|W| > 1$.

1. This follows directly from the fact that, by the induction hypothesis, 4 holds for U and $\{v\}$.
2. Let $e \in (U, \{v\})$. So $e = \{u, v\}$ for some $u \in U$. Then ρ_e maps $\mathbf{G}^T$ to $\mathbf{G}^S$ with S defined by $S \triangle T = \{u, v\}$. Since $|S \cap W| \equiv |T \cap W|$, the statement follows from 1.
3. By 2 and Lemma 5.12, $L^{\text{cut}}_{v^X,U}(\lambda_{U+v}) = L^{\text{edges}}_{(U,\{v\})}(\lambda_{U+v})$ for all $v^X \in v^*$. By the induction hypothesis, $\lambda_U = L^{\text{edges}}_{E[U]}(\mu_U)$, so it follows that $\lambda_{U+v} = L^{\text{edges}}_{E[U]}(\mu_{U+v})$, since no edge gadget for any $e \in E[U]$ occurs in τ_v.
   Together, it follows that

$$L^{\text{cut}}_{v^X,U}(\lambda_{U+v}) = L^{\text{edges}}_{(U,\{v\})}(L^{\text{edges}}_{E[U]}(\mu_{U+v})) = L^{\text{edges}}_{E[W]}(\mu_{U+v}).$$

   So, by definition of λ_W, $\lambda_W = L^{\text{edges}}_{E[W]}(\mu_W)$. The proof for $\widetilde{\lambda}_W$ is analogous.
4. The direction from right to left follows directly from 3 and Lemma 5.6. To show the other direction by contraposition, suppose $|T \cap W| \not\equiv |S \cap W| \pmod 2$. Since the direction from right to left implies $\widetilde{\lambda}^T_W = \lambda^S_W$ and $\lambda^T_W = \widetilde{\lambda}^S_W$, it suffices to show that $\lambda^T_W \neq \widetilde{\lambda}^T_W$ (and, analogously, $\lambda^S_W \neq \widetilde{\lambda}^S_W$). Assume $\lambda^T_W = \widetilde{\lambda}^T_W$. Then, by 3, $L^{\text{edges}}_{E[W]}(\mu^T_W) = L^{\text{edges}}_{E[W]}(\widetilde{\mu}^T_W)$. But, as $L^{\text{edges}}_{E \setminus E[W]}(L^{\text{edges}}_{E[W]}(\mu^T_W)) = L^{\text{edges}}_E(\mu^T_W)$, this contradicts the fact that, by Lemma 5.9, $p(L^{\text{edges}}_E(\mu^T_W)) \neq p(L^{\text{edges}}_E(\widetilde{\mu}^T_W))$. So $\lambda^T_W \neq \widetilde{\lambda}^T_W$.
   The proof for the second statement is analogous. □

So the final set λ_V is a hereditarily finite set over $\{0, 1\}$ representing the parity of $|T|$. By Lemma 5.9, that parity can be expressed as $p(\lambda_V)$.

It remains to show that the construction is CPT-definable. All sets used in the computation can be defined by the set-theoretic operations available in CPT. More

precisely, there is a term t that transforms the set of all $\lambda_U, \widetilde{\lambda}_U$ for $U \in \mathcal{W}$ with $|U| = k$ to the set of all $\lambda_W, \widetilde{\lambda}_W$ for sets W of size $k+1$, and leaves the set unchanged (i. e. reaches a fixed point) if λ_V has been constructed. Then it suffices to show that the iteration t^* does not activate too many or too complex objects.

Lemma 5.14. *$|\operatorname{tc}(\mu_W^T)|$ is polynomial in $|\mathbf{G}^T|$ for all $W \in \mathcal{W}$ and $T \subseteq V$.*

Proof. For $k \leq |V| =: n$, let $\mu_k = \{\mu_W, \widetilde{\mu}_W : W \in \mathcal{W} \text{ and } |W| = k\}$. We show by induction on k that the number of elements in $\operatorname{tc}(\mu_k) \setminus \operatorname{tc}(\mu_{k-1})$ is polynomial in n. The lemma then follows because $k \leq n$.

For $k = 1$, consider all $\tau_v, \widetilde{\tau}_v$ for $v \in V$. Each τ_v or $\widetilde{\tau}_v$ is a set of neighbourhoods of the vertices v^X. So the transitive closure contains $\leq |\mathbf{G}|$ many neighbourhoods consisting of vertices in $E(v)^*$.

For $k > 1$, the sets μ_{W+v}, $\widetilde{\mu}_{W+v}$ are constructed for at most $2^{|C_i|}$ many sets and $|C_i|$ many vertices for some colour class C_i of logarithmic size. Furthermore, there are also at most $2^{|C_i|}$ many new sets μ_W and $\widetilde{\mu}_W$, which are built from μ_{U+v} and $\widetilde{\mu}_{U+v}$ for sets U with $|U| = k - 1$. □

By Lemma 5.13, $|\operatorname{tc}(\lambda_W)| \leq |\operatorname{tc}(\mu_W)|$ for all $W \subseteq V$. So our construction is CPT-definable, which completes the proof of Theorem 5.1.

5.4 CFI over graph classes with linear maximal degree

The techniques used for graphs with colour classes of logarithmic size can be further refined for classes of graphs that do not possess any order, but where the maximal degree is bounded from below by a linear function.

Again we use the idea that, for subgraphs of the underlying graph that are sufficiently small compared to the size of the CFI graph, the set of all subsets of these subgraphs is within the polynomial bounds. This is the case whenever the underlying graph possesses a vertex v of large degree (say $\frac{|V|}{k}$ for some constant k), because then, the vertex gadget v^* is of size exponential in $|V|$. The claim that this establishes CPT-definability has already been stated in [23] for CFI graph over complete graphs. Further, Theorem 5.1 implies definability as the special case of a single colour class of logarithmic size.

However, we can modify the construction in a way that all sets defined are of bounded rank. By Theorem 40 in [23], this is not possible for CFI graphs over

ordered graphs, and hence not for colour classes of logarithmic size. Moreover, this result further characterises the bounded-rank-fragment of CPT.

Theorem 5.15. *For every $k \in \mathbb{N}$, the CFI query over the class of graphs $G = (V, E)$ with maximal degree $\geq \frac{|V|}{k}$ is definable in* CPT *activating only sets of constant rank not depending on k.*

We now explain the modifications made to adapt the procedure presented in the previous section to these graph classes.

As previously, we start with an object τ_v for each $v \in V$, that encodes whether $v \in T$. We combine the τ_v and $\widetilde{\tau}_v$ to define the parity of all subsets of V.

Recall that, in the algorithm in [23], the μ_i correspond to sequences of τ_v and $\widetilde{\tau}_v$ with an even number of $\widetilde{\tau}_v$. With access to all subsets of V, these sequences can be replaced by all sets containing at most one of τ_v or $\widetilde{\tau}_v$ for every $v \in V$ where the number of $\widetilde{\tau}_v$ is even. Working with subsets of V allows to circumvent the nesting of sets that is noticeable in Definition 5.5, thus keeping the rank of the sets used here small.

Definition 5.16. Let $v \in V$ and $W \subseteq V$ with $|W| \geq 2$. Then $\mu^T_{\{v\}} = \{\{\tau_v\}\}$ and $\widetilde{\mu}^T_{\{v\}} = \{\{\widetilde{\tau}_v\}\}$. For larger W, let $\dot{\mu}^T_W = \{\{\dot{\tau}^T_v : v \in W\} : \dot{\tau}^T_v \in \{\tau^T_v, \widetilde{\tau}^T_v\}\}$. Then

$$\begin{aligned}\mu^T_W =& \{m \in \dot{\mu}^T_W : \text{ the number of } \widetilde{\tau}^T_v \text{ in } m \text{ is even}\}, \text{ and}\\ \widetilde{\mu}^T_W =& \{m \in \dot{\mu}^T_W : \text{ the number of } \widetilde{\tau}^T_v \text{ in } m \text{ is odd}\}.\end{aligned}$$

The sets μ_W, $\widetilde{\mu}_W$ for $W \subseteq V$ defined in this way again characterise the parity of $|T \cap W|$.

Lemma 5.17. $\mu^T_W = \mu^S_W \Leftrightarrow \widetilde{\mu}^T_W = \widetilde{\mu}^S_W \Leftrightarrow |S \cap W| \equiv |T \cap W| \pmod 2$ *and* $\mu^T_W = \widetilde{\mu}^S_W \Leftrightarrow \widetilde{\mu}^T_W = \mu^S_W \Leftrightarrow |S \cap W| \not\equiv |T \cap W| \pmod 2$.

Proof. The first equivalences both follow directly by induction on $|W|$.

To see that $\mu^T_W = \mu^S_W$ implies $|S \cap W| \equiv |T \cap W|$, consider the set $\{\tau^T_v : v \in W\} \in \mu^T_W$. If that set also occurs as $m \in \mu^S_W$, then the number of $\widetilde{\tau}^S_v$ in m is even. So there is an even number of vertices with $\tau^T_v = \widetilde{\tau}^S_v$ and thus $|S \triangle T|$ is even. With the same argument, if $\widetilde{\mu}^T_W = \mu^S_W$, then $|S \triangle T|$ is odd.

The other direction is another simple induction on $|W|$. □

The μ_W and the objects explained in the following will again have the restricted super-symmetry property from Lemma 5.12. So, by Lemma 5.13, labelling edges

according to all vertices $v^X \in v^*$ in parallel again has the same result as the "correct", but not definable mapping L_W^{edges}.

The definable mappings $L_{v^X,U}^{\text{cut}}$, however, still require a vertex v to be fixed during the construction. So we formalise the construction of the μ_W and $\widetilde{\mu}_W$ as successively enlarging the sets W by a single vertex. For instance, whenever $W = U \uplus \{v\}$, each set in μ_W containing an even number of $\widetilde{\tau}_w$ can be constructed by adding τ_v to a set in μ_U, or $\widetilde{\tau}_v$ to a set in $\widetilde{\mu}_U$. Again, any choice of U and v will lead to the same unique set where all edges in the subgraph induced by W are labelled. So μ_W can be defined as the unique result of all replacements with respect to any decomposition of W as $U \uplus \{v\}$.

Definition 5.18. Let $U, W, T \subseteq V$ and $v \in V$ such that $U \uplus \{v\} = W$.

$$\begin{aligned}
\lambda_{\{v\}}^T &= \{\{\tau_v^T\}\}, \widetilde{\lambda}_{\{v\}}^T = \{\{\widetilde{\tau}_v^T\}\},\\
\lambda_{U+v}^T &= \{X \cup \{\tau_v^T\} : X \in \lambda_U^T\} \cup \{X \cup \{\widetilde{\tau}_v^T\} : X \in \widetilde{\lambda}_U^T\},\\
\widetilde{\lambda}_{U+v}^T &= \{X \cup \{\widetilde{\tau}_v^T\} : X \in \lambda_U^T\} \cup \{X \cup \{\tau_v^T\} : X \in \widetilde{\lambda}_U^T\},
\end{aligned}$$

and $\lambda_W^T = L_{v^X,U}^{\text{cut}}(\lambda_{U+v}^T)$ and $\widetilde{\lambda}_W^T = L_{v^X,U}^{\text{cut}}(\widetilde{\lambda}_{U+v}^T)$ for some (all) $U \subseteq V, v \in V$ such that $U \uplus \{v\} = W$ and $v^X \in v_T^*$.

So the sets λ_W with labelled edges are constructed by splitting W into all possible decompositions $U \uplus \{v\}$, adding τ_v (resp. $\widetilde{\tau}_v$) to the sets in μ_U ($\widetilde{\mu}_U$) and processing the edges along the cut $(U, \{v\})$.

The sets $\lambda_W, \widetilde{\lambda}_W$ are well-defined because, as previously, the λ_{U+v} can also be constructed from μ_{U+v} by labelling all edges with the mappings L_F^{edges} from Definition 5.7.

Lemma 5.19. *Let $U, W \subseteq V$ and $v \in V$ such that $U \uplus \{v\} = W$. Then*

$$\lambda_W = L_{(U,\{v\})}^{\text{edges}}(\lambda_{U+v}) = L_{E[W]}^{\text{edges}}(\mu_W).$$

The proof is analogous to that of Lemma 5.13. To extract the parity of $|T|$ from the final set λ_V, we adapt the aggregation function to the new construction.

Definition 5.20.

$$p(x) = \begin{cases} i & x = \left\langle N\left(v^X\right), i\right\rangle, \\ \sum_{y \in x} p(y) & x \in \lambda_V, \\ \prod_{y \in x} p(y) & x = \lambda_V \text{ or } x = \tau_v \text{ for some } v \in V, \end{cases}$$

where the sums and products are computed in $\mathbb{F}_2$.

Lemma 5.21. $p\left(\lambda_V^T\right) = |T| \mod 2$.

Proof. By Lemma 5.19, $\lambda_V^T = L_E^{\text{edges}}(\mu_V^T)$. First consider a set τ_v^T (the argument for $\widetilde{\tau}_v^T$ is analogous). For each neighbourhood in τ_v^T, the parity bit is 1 if, and only if, $v \in T$. Since all parity bits in τ_v^T coincide, the product is also 1 if, and only if, $v \in T$.

Let $m \in \mu_V^T$. Then the number of $\widetilde{\tau}_v^T$ in m is even. So $p(L_E^{\text{edges}}(m)) = 1$ if, and only if, m contains an odd number of τ_v^T with $p(L_E^{\text{edges}}(\tau_v^T)) = 1$, which holds if, and only if, $|T|$ is odd. All $m \in \mu_V^T$ get the same value, so $p(L_E^{\text{edges}}(\mu_V^T)) = p(\lambda_V^T) = 1$ if, and only if, $|T|$ is odd. □

So it is possible to extract the parity of $|T|$ from λ_V. It remains to show that the λ_W can be constructed in Choiceless Polynomial Time. One of the main obstacles is to compute sums modulo 2 when applying the mappings $L_{v^X,U}^{\text{cut}}$ and p, because counting would in general require ordinals, which do not have bounded rank. Furthermore, we need to show that the size of the transitive closure of each λ_W is polynomial in $|\mathbf{G}|$.

Lemma 5.22. *For any* CPT*-definable set* $x \in \mathrm{HF}(\mathbf{A})$ *of rank* k *with* $|x|$ *logarithmic in* $|\mathbf{A}|$*, the parity of* $|x|$ *can be defined in* CPT *over* $\mathbf{A}$ *with sets of rank* k*.*

Firstly, note that the size of each neighbourhood occurring in the transitive closure of each λ_W is bounded by $|V|$ and thus logarithmic in the size of the CFI graph, so the lemma can be applied to the computation of the mappings $L_{v^X,U}^{\text{cut}}$.

Proof. For all $n \leq |x|$, compute the set of all subsets of x of size n and store their parity. Start with the set of all singletons and parity 1. Given sets of size n, construct the sets of size $n+1$ analogously to the construction of the μ_W and flip the parity. As soon as the set $\{x\}$ itself has been constructed, the parity of $|x|$ can be extracted. □

Lemma 5.23. *For every* $k \in \mathbb{N}$*, there is a* CPT*-term defining* λ_V^T *over every CFI graph* $\mathbf{G}^T$ *where* G *has a vertex of degree* $\geq \frac{|V|}{k}$*.*

Proof. Each set λ_W or $\widetilde{\lambda}_W$ constructed by the algorithm contains $\leq 2^{|W|}$ many sets consisting of (polynomially sized) sets τ_v and $\widetilde{\tau}_v$, and there are $2^{|V|}$ subsets W. The number and size of the λ_W and $\widetilde{\lambda}_W$ is polynomial in $|\mathbf{G}^T|$ because some $v \in V$ has $\geq \frac{|V|}{k}$ neighbours, so the vertex gadget v_T^* is of size $2^{\frac{|V|}{k}-1}$. So the transitive closure of all constructed objects is of polynomial size. □

All in all we have established that, over all graph classes with linear maximal degree, the CFI query is definable in CPT using only sets of bounded rank. We will further use this result in Section 5.5 to separate the fragment of CPT over sets of bounded rank from the fragment that does not permit any set-like objects.

As first mentioned in a presentation by Wied Pakusa, Theorem 5.15 also implies that, on the class of CFI graphs over complete graphs, CPT captures PTIME. More precisely, an ordered copy of the CFI graph can be defined from its parity, since the underlying complete graph can easily be canonised.

Corollary 5.24. CPT *captures polynomial time on the class of CFI graphs over complete graphs.*

Proof. We show that there is a PIL-definable (and thus CPT-definable) canonisation of CFI graphs over complete graphs.

The number of vertices of the underlying graph is definable from the number of vertices of the CFI graph. So there is a PIL-program that, given a CFI graph over the n-clique, outputs the disjoint union of the CFI graph with an ordered copy of the n-clique.

Since, by Theorem 5.15, the parity of a given CFI graph is CPT-definable, it suffices to construct PIL-programs that, given the disjoint union of the ordered n-clique and one of its CFI graphs, output an ordered copy of the even, respectively odd, CFI graph over the n-clique.

In the following, we describe the program for the odd CFI graph. Every interpretation that is used in the program preserves the input structure by representing each element a by the tuple (a, a) (or (a, a, a), respectively). The order on the CFI graph is created by ordering all tuples lexicographically.

The initial interpretation $\mathcal{I}_{\text{init}}$ is the concatenation of interpretations $\mathcal{I}_{\text{edges}}$ and $\mathcal{I}_{\text{vertices}}$. First, $\mathcal{I}_{\text{edges}}$ creates the edge gadgets. For every edge $e = \{u, v\}$ with $u < v$, (u, v) represents e^0 and (v, u) represents e^1. The interpretation also introduces binary relations that map (u, v) to u and v, respectively.

Then, $\mathcal{I}_{\text{vertices}}$ creates vertices v^X with $|X| \leq 1$ for every vertex gadget v^*. The minimal vertex v with respect to the linear order is transformed to the unique odd vertex gadget, so the interpretation creates vertices v^X for all one-element subsets of $E(v)$. To that end, every vertex $v^{\{v,w\}}$ is represented by the tuple (v, w). If v is not the minimal vertex, it suffices to create $v^\emptyset$, which is represented by the equivalence class $\{(v, w) : v \neq w\}$. The new relations defined by $\mathcal{I}_{\text{edges}}$ make it possible to define the edges between gadgets.

The iterated interpretation $\mathcal{I}_{\text{step}}$ creates the remaining vertex gadgets by adding vertices v^X with $|X| \in \{2i, 2i+1\}$ in the ith iteration. Whenever a vertex v^X has already been created in one of the previous steps, the tuple (v^X, w_1, w_2) for $\{v, w_1\}, \{v, w_2\} \notin X$ represents $v^{X \cup \{\{v,w_1\},\{v,w_2\}\}}$. This interpretation is iterated until all vertex gadgets have been created.

Since the input structure already contains a copy of the CFI graph as a substructure, all intermediate structures remain within a polynomial bound. So CFI graphs over complete graphs can be canonised in CPT. □

5.5 CPT over tuple-like objects and PIL without congruences

Using CFI graphs and an extension of the technique for inexpressibility results explained at the beginning of this chapter, we set out to round off the analysis of the PIL-fragment without congruences introduced in Section 4.1.3. As we have seen in Section 4.1.3, PIL without congruences is strictly weaker than full PIL. For both variants, with and without counting, the separation was obtained by translating $\sim$ -free PIL-formulae to logics weaker than CPT. Proving directly that the CFI query over complete graphs is not definable in $\sim$ -free PIL, we get a more fine-grained separation: CPT-variants over tuples are not only weaker than CPT with full set-building abilities, but also weaker than the fragment using only sets of bounded rank.

We show that PIL without congruences, and in fact any CPT-formula that uses only tuple-like objects, cannot define the Cai-Fürer-Immerman query over ordered complete graphs. An important step toward that result is defining what it means for an object to be set-like, as opposed to tuple-like. Together with the expressibility result from the previous section, we obtain a separation from CPT over sets of bounded rank.

Observe that the CPT-procedure that we presented in the previous section strongly makes use of the computational power of sets in order to go through all possible subsets of the vertex set of the underlying complete graph. The simple observation was that if this underlying complete graph has n vertices, then the corresponding CFI graph has about $n \cdot 2^n$ elements, which means that activating all 2^n subsets of the vertex set does not violate the polynomial resource bounds of our CPT-formula. On the other hand, if we had to work with tuples only, then it would be unclear how to represent these 2^n subsets in a succinct way. Indeed, just enumerating them in

any order would lead to about $n!$ different objects, which is clearly super-polynomial in the size of the CFI graph. As we will see, this blow-up cannot be avoided and it is impossible to decide the CFI query with a restriction of CPT to tuples only.

Strong supports To apply the proof technique for inexpressibility results outlined in Section 5.2, we first have to find a characterisation of the sets that can occur as active objects. That is, we define a generalised notion of "tuple-like" objects, as opposed to "set-like" ones. In order to motivate our definition, let us come back to CFI graphs over graphs of large degree. For simplicity, we consider complete graphs now. So let $\mathbf{K}_n = (V, E)$ be the complete graph over $|V| = n$ vertices. The automorphism group of $\mathbf{K}_n$ is the full symmetric group $\Gamma = \mathrm{Sym}(V)$. Now let $x = \{a_1, \dots, a_k\} \subseteq V$ be a set consisting of k distinct vertices, and let $y = (a_1, \dots, a_k) \in V^k$ be some arrangement of these k vertices as a k-tuple over V. Since every CPT-computation on $\mathbf{K}_n$ is invariant under the action of Γ, every CPT-term which defines x also has to define the whole orbit of x. Of course, the same holds for every term which defines y. It is easy to see that $|\mathrm{Orbit}(x)| = \binom{n}{k}$ while $|\mathrm{Orbit}(y)| = \binom{n}{k} \cdot k!$. Hence, for large k the orbit of the *set* x is much smaller than the orbit of the *tuple* y, although they are defined over the same set of atoms.

In light of the orbit-stabiliser theorem, the opposite is true for the stabiliser groups of x and y: $|\mathrm{Stab}_\Gamma(x)| = k! \cdot (n-k)!$ and $|\mathrm{Stab}_\Gamma(y)| = (n-k)!$. So, in terms of stabilisers, this can be explained by the obvious fact that every automorphism fixing y has to fix every one of its entries, whereas the elements of the set $\{a_1, \dots, a_k\}$ can still be permuted while fixing x. Formally, the *point-wise* stabiliser $\mathrm{Stab}^\bullet_\Gamma\{a_1, \dots, a_k\}$ of $a_1, \dots, a_k$ coincides with $\mathrm{Stab}_\Gamma(y)$, whereas it is a subgroup of index $k!$ of $\mathrm{Stab}_\Gamma(x)$.

Definition 5.25. Let $\mathbf{A}$ be a structure and let $\Gamma = \mathrm{Aut}(\mathbf{A})$ and $x \in \mathrm{HF}(A)$. A set $\sigma \subseteq A$ of atoms is a *strong support* for x if $\mathrm{Stab}^\bullet_\Gamma(\sigma) = \mathrm{Stab}_\Gamma(x)$.

Accordingly, we say that an element $x \in \mathrm{HF}(A)$ is *strongly supported* if it has a strong support. If every element in $\mathrm{tc}(x)$ is strongly supported, then x is *transitively strongly supported*. For example, every atom $a \in A$ is strongly supported and every $x \in \mathrm{HF}(A)$ with $\mathrm{Stab}_\Gamma(x) = \Gamma$ has the strong support $\emptyset$. For the example above, the set $\{a_1, \dots, a_k\}$ is a support for the *set* $x = \{a_1, \dots, a_k\}$ and it is a *strong support* for the *tuple* $y = (a_1, \dots, a_k)$. In fact, every CPT-definable set is strongly supported. But note, however, that it is not necessarily transitively strongly supported. A CPT-term can, for instance, define the set of all edges of an undirected graph. But if the elements of an edge can be permuted by an automorphism, they are not strongly supported anymore.

The notion of strongly supported sets is taken as a working definition for objects which are "not set-like", or more intuitively, which are "sequence-like". Those objects enforce an inherent order on the supporting atoms.

Up to now, we have provided some intuition of why this is a reasonable notion of tuple-like objects. To establish a connection to the tuple-based fragment of PIL, we show next that this fragment is contained in CPT over strongly supported sets.

Lemma 5.26. *For every* $\sim$ -free PIL*-program, there is an equivalent* CPT*-sentence that only activates transitively strongly supported objects.*

Proof. As in the proof that PIL $\leq$ CPT ([27]), the CPT-sentence simulates the PIL-computation with an iteration term. The stages of the iteration represent the structures in the run $(\mathbf{A}_i)_{i\leq n}$ on $\mathbf{A}$ as tuples $\langle A_i, (R_i)_{R\in\tau}\rangle$, where τ is the output signature of the PIL-program.

It suffices to show that these stages are transitively strongly supported. The set A_i, as well as each relation R_i (represented as a set of tuples) is CPT-definable and thus has the strong support $\emptyset$. So the tuple $\langle A_i, (R_i)_{R\in\tau}\rangle$ is strongly supported.

The elements of A_i and of the tuples in each R_i are k-tuples over the previous state A_{i-1}, since the step-interpretation does not use congruences. Let $\overline{a} = \langle a_1, \ldots, a_k\rangle$ be a k-tuple over A_{i-1}. By induction, each a_i has a strong support σ_i (and is transitively strongly supported). Then it is easy to verify that $\sigma_1 \cup \ldots \cup \sigma_k$ strongly supports $\overline{a}$. Since the R_i are tuples over those k-tuples, it follows that stage i is transitively strongly supported. □

To separate $\sim$ -free PIL from CPT with sets of bounded rank, we aim to prove Theorem 5.51: no CPT-sentence which can only access strongly supported objects can express the CFI query over ordered complete graphs (the same result for unordered complete graphs follows immediately).

Our proof is based on the techniques developed by Dawar, Richerby and Rossman [23] recapitulated in Section 5.2. To adapt this technique to CFI graphs over ordered complete graphs, these structures should satisfy the following requirements:

- Every strongly supported set activated by a CPT-term has a strong support that is in a normal form and sufficiently small (Lemma 5.28).
- There is a winning strategy that we can transfer to the game on strongly supported sets, i. e. Duplicator wins the bijective pebble game where Spoiler can put pebbles on a constant number of strong supports (Lemma 5.36). This is not evident from known results, because our strong supports do not have constant size.

- The graphs are homogeneous with respect to tuples that can occur as strong supports (Lemma 5.38).

These properties will then imply that the winning strategy on the CFI graphs can be translated to a winning strategy on the strongly supported hereditarily finite sets (Lemma 5.46).

In the following, we denote by $\mathbf{K}_n^<$ the complete graph of size n with a linear order on the vertices. Note that we always assume the CFI graphs over $\mathbf{K}_n^<$ to be equipped with the total preorder induced by the order on $\mathbf{K}_n^<$.

Molecules and sizes of strong supports First we formalise a normal form for strong supports. Since any vertex in a vertex gadget is uniquely determined by a set of adjacent vertices from edge gadgets, it suffices to consider strong supports that only contain edge gadgets. However, a strong support can then contain $n-2$ edge gadgets instead of a single vertex gadget. Note that a vertex gadget is uniquely determined by $n-2$ of its $n-1$ neighbouring edge gadgets since every automorphism flips an even number of incident edges.

Definition 5.27. Fix $k, n \in \mathbb{N}$.

- A (k-sized) *molecule* is either the empty tuple, the singleton a for any atom a, or a tuple $\alpha = (a_1, \dots, a_s)$ of atoms such that $s \leq nk$ and each a_i is an element of an edge gadget.
- For a CFI graph $\mathbf{A}$ over $\mathbf{K}_n^<$, we denote by $\mathrm{HF}(\mathbf{A})_{k-\mathrm{ss}}$ the substructure of $\mathrm{HF}(\mathbf{A})$ consisting of all objects whose transitive closure only contains objects with a strong support that is a k-sized molecule.

When speaking about molecules, we assume the parameter k to be fixed. Note that our definition of molecules differs from that in [23], where the length of molecules is bounded by a constant.

We first show that every strongly supported set occurring in the iterations of a CPT-sentence has a strong support that is a k-sized molecule where k depends only on the sentence. Then every object that is transitively strongly supported is in the domain of $\mathrm{HF}(\mathbf{A})_{k-\mathrm{ss}}$.

Lemma 5.28 (Size of strong supports)**.** *Let $\overline{\varphi} = (\varphi, n^q)$ be a* CPT*-sentence. Every strongly supported set activated by $\overline{\varphi}$ on a CFI graph $\mathbf{G}^T$ over $\mathbf{K}_n^<$ has a strong support that consists of $< 4qn$ vertices, all of which are outer vertices.*

In the following, we fix $k = 4q$. Note that any strong support can be transformed into a strong support consisting only of outer vertices by replacing every vertex v^X by its neighbouring edge gadgets. Furthermore, whenever there is a strong support containing both vertices from the same edge gadget, then the sequence where any one of the two vertices is removed is still a strong support.

In order to show that the strong supports cannot grow too large, we show that the orbit of every set with a large strong support is already too large to be activated in CPT. By the nature of strong supports, it actually suffices to show a lower bound on the orbit of the strong support:

Lemma 5.29. *Let x be strongly supported by α. Then $|Orbit(x)| = |Orbit(\alpha)|$.*

Proof. Immediate from the orbit-stabiliser theorem. □

So Lemma 5.28 follows from

Lemma 5.30. *Let $\mathbf{G}^T$ be a CFI graph over $\mathbf{K}_n^<$, and let α be a tuple of vertices from $> nk$ edge gadgets. Then $|Orbit(\alpha)| > 2^{\frac{nk}{4}}$.*

The lemma is shown by constructing a set of $\frac{nk}{4}$ automorphisms, every one of which flips an edge that occurs in α but is fixed by all other automorphisms in the set. Combining these automorphisms yields $2^{\frac{nk}{4}}$ distinct tuples in Orbit(α). Such automorphisms are induced by sets of edge sets called large automorphism sets.

Definition 5.31. Let $\mathbf{K}_n^< = (V, E, <)$. The set $\mathcal{F} \subseteq \mathcal{P}(E)$ is a *large automorphism set* with respect to $A \subseteq E$ if every set in $\mathcal{F}$ is a cycle and there is a set $D \subseteq A$ of size $\geq \frac{nk - \frac{n}{2}}{2}$ such that, for every edge $d \in D$, there is exactly one $F \in \mathcal{F}$ such that $d \in F$ and $d' \notin F$ for every $d \neq d' \in D$.

D is a sufficiently large set of edges that are flipped by exactly one of the automorphisms induced by the sets in $\mathcal{F}$. The connection between the sets D and $\mathcal{F}$ is illustrated in Figure 5.3.

For any edge $\{u, v\}$ occurring in α that is incident to edges $\{v, w\}$, $\{u, w\}$ that do not occur in α, the automorphism ρ_F for $F = \{\{u, v\}, \{v, w\}, \{u, w\}\}$ flips $\{u, v\}$ and fixes all other edges occurring in α. If such edges do not exist, i. e. if for every vertex w, either $\{v, w\}$ or $\{u, w\}$ occurs in α, we call $\{u, v\}$ *blocked*. Otherwise, $\{u, v\}$ is *non-blocked*. This is illustrated in Figure 5.4.

To show that a large automorphism set exists, we give an algorithm to compute it (see Algorithm 1). It first constructs as many automorphisms as possible from every blocked edge that has not been processed yet, and then constructs the automorphisms described above for the non-blocked edges.

The result of one step of the algorithm is shown in Figure 5.5.

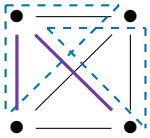

Figure 5.3: The sets D and $\mathcal{F}$ of a large automorphism set. The thick purple edges form the set D, and the triangles mark the cycles in $\mathcal{F}$. The edges in D are included in disjoint cycles in $\mathcal{F}$.

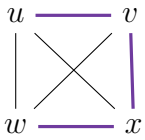

Figure 5.4: A graph where the edges occurring in α are marked. The cycle through w makes $\{u, v\}$ non-blocked, whereas $\{v, x\}$ is blocked since both possible cycles contain another edge in α.

Input: Complete graph $\mathbf{K}_n^< = (V, E, <)$ and a set $A \subseteq E$
Output: A set $\mathcal{F}$ that is a large automorphism set with respect to A
while *there is a blocked edge in* $A \setminus \bigcup \mathcal{F}$ **do**
  choose a blocked edge $e \in A \setminus \bigcup \mathcal{F}$;
  foreach *triangle* $\{e, f, g\}$ *containing* e **do**
    if $f, g \notin \bigcup \mathcal{F}$ **then**
      $\mathcal{F} := \mathcal{F} \cup \{\{e, f, g\}\}$;
    end
  end
end
foreach *edge* $e = \{u, v\} \in A \setminus \bigcup \mathcal{F}$ **do**
  choose a vertex w such that $\{v, w\} \notin A$ and $\{u, w\} \notin A$;
  $\mathcal{F} := \mathcal{F} \cup \{\{e, \{u, w\}, \{v, w\}\}\}$;
end

Algorithm 1: Computing large automorphism sets

Lemma 5.32. *On* $\mathbf{K}_n^<$*, the algorithm traverses the while-loop at most* $\frac{n}{2}$ *times.*

Proof. Let $e = \{u, v\}$ be the edge chosen in the first line of the loop. After the loop, all edges incident to u and all edges incident to v are in $\bigcup \mathcal{F}$. Furthermore, neither u nor v had that property before the loop was entered, because at this point, e was not in $\bigcup \mathcal{F}$. So, after $\frac{n}{2}$ runs of the loop, all n vertices have the property. This means that there is no edge left in $A \setminus \bigcup \mathcal{F}$. □

Lemma 5.33. *If* $|A| > nk$*, then the set* $\mathcal{F}$ *computed by the algorithm is a large automorphism set with respect to* A*.*

Proof. Every edge set added to $\mathcal{F}$ by the algorithm is a cycle. So it suffices to show that there exists a set D as required by the definition.

Whenever an edge other than the blocked edge chosen in line 1 is added to a set in $\mathcal{F}$, it is not in $\bigcup \mathcal{F}$ yet. So the only way an edge can occur in two sets in $\mathcal{F}$ is if it is chosen in line 1. Every set in $\mathcal{F}$ contains at most one such edge. Furthermore,

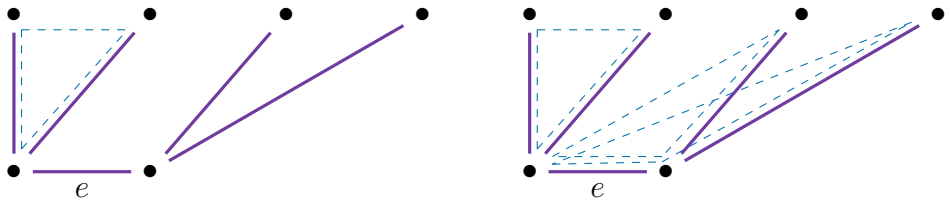

Figure 5.5: One iteration step of the while-loop of the algorithm for large automorphism sets, before (left) and after (right) the blocked edge e is processed. The edges in $E \setminus A$ are omitted. The dashed triangles mark the cycles in $\mathcal{F}$.

if a set in $\mathcal{F}$ is defined starting from a blocked edge, it also contains another edge in A. Otherwise, the edge would not be blocked. So we can choose, from every set in $\mathcal{F}$, an edge that occurs in no other set in $\mathcal{F}$.

By Lemma 5.32, there at most $\frac{n}{2}$ edges that are chosen in line 1 at some point. The algorithm ensures that all $> nk$ edges in A are in $\bigcup \mathcal{F}$. So each of the remaining $> nk - \frac{n}{2}$ edges is contained in a set in $\mathcal{F}$, and at most two of them are in the same set. So there is a set D with $|D| > \frac{nk-\frac{n}{2}}{2}$ as required in the definition. □

The algorithm is correct and produces a set of cycles that induce automorphisms of $\mathbf{G}^T$. This set yields the bound on the size of the orbit of α.

Proof of Lemma 5.30. By Lemma 5.33, there is a set of automorphisms such that each of them flips some edge occurring in α that is not flipped by any of the other automorphisms in the set. Thus, for every subset of these automorphisms, their concatenation yields a unique set of flipped edges in α, i. e. no two of these concatenations map α to the same tuple. So $|\mathrm{Orbit}(\alpha)| > 2^{\frac{nk-\frac{n}{2}}{2}}$, and, since $\frac{nk-\frac{n}{2}}{2} \geq \frac{nk-\frac{nk}{2}}{2} = \frac{nk}{4}$, it follows that $|\mathrm{Orbit}(\alpha)| > 2^{\frac{nk}{4}}$. □

We have shown that the orbit of strong supports of size $> nk$ is too large to be activated by a CPT-sentence with bound $n^{\frac{k}{4}}$. Since, by Lemma 5.29, the orbits of an object and its strong support have the same size, the same holds for every object with a strong support of size $> nk$. So this completes the proof of Lemma 5.28.

Games on molecules We have established that it suffices to consider strong supports that are molecules. The winning strategy for Duplicator in the game on the strongly supported objects will be devised from a winning strategy in the game on the CFI graphs themselves where Spoiler puts pebbles on strong supports. We first describe a winning strategy in the game on the CFI graphs, where Spoiler may put pebbles on a constant number of molecules. But note that pebbling the

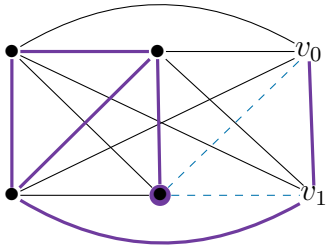

Figure 5.6: A set of pebbled edges and vertices (marked purple) in the 1-molecule bijection game on the 6-clique. Though only a single molecule is pebbled, every vertex is incident to a pebbled edge. The only $(\geq \frac{n}{2})$-free vertices are v_0 and v_1, and the blue dashed edges correspond to an automorphism that moves the error between v_0 and v_1.

molecules themselves does not suffice, because they consist only of edge gadgets. So unless Spoiler can also put pebbles on vertex gadgets, he cannot choose any two adjacent vertices in the CFI graph. However, the $\leq nk$ elements of a molecule also uniquely determine k vertex gadgets. This leads to the following version of the bijection game:

Definition 5.34. The *k-molecule bijection game* is played on a pair of CFI graphs over $\mathbf{K}_n$ with the same rules as the k-pebble bijection game, with the following exception: Instead of constantly many pebbles, Spoiler has nk pebbles that he can place only on edge gadgets, and k additional pebbles that he can place only on vertex gadgets.

In any position of the game on CFI graphs over $\mathbf{K}_n^<$, we call a vertex v (edge e) of $\mathbf{K}_n^<$ *pebbled* if there is a pebble on a vertex in the corresponding gadget v^* (e^*). A vertex or edge that is not pebbled is *free*. Further, a vertex v is $(\geq \frac{n}{2})$-free if it is free and incident to $\geq \frac{n}{2}$ free edges that lead to free vertices. Otherwise, v is $(< \frac{n}{2})$-free. Figure 5.6 shows an example of $(\geq \frac{n}{2})$-free and $(< \frac{n}{2})$-free vertices.

In case the game is played on non-isomorphic CFI graphs, every bijection played by Duplicator has to map some edge in one CFI graph to a non-edge in the other one. This error should always occur in a gadget adjacent to enough free edge gadgets. Note that, when Spoiler puts pebbles on the elements of a single molecule, this can already mean that each vertex is adjacent to a pebbled edge. We can, however, guarantee that the error always occurs in a $(\geq \frac{n}{2})$-free vertex and can be "moved" to another $(\geq \frac{n}{2})$-free vertex via free edges, as illustrated in Figure 5.6.

We first show that there are always enough $(\geq \frac{n}{2})$-free vertices to move the error.

Lemma 5.35. *Fix $\ell \in \mathbb{N}$, let $n \geq 12\ell$ and let $\mathbf{G}^T$ be a CFI graph over $\mathbf{K}_n^<$. In any position of the ℓ-molecule bijection game, there are $> \frac{n}{2}$ many vertices that are $(\geq \frac{n}{2})$-free.*

Proof. Assume otherwise. Then at least $\frac{n}{2}$ vertices are not $(\geq \frac{n}{2})$-free, i. e. they are either pebbled or incident to $\geq \frac{n}{2}$ pebbled edges. The least number of edge pebbles necessary for this situation is when all pebbled edges are again incident to the $(< \frac{n}{2})$-free vertices that are not themselves pebbled. Since there are at most ℓ pebbled vertices, this means that the number of pebbled edges is

$$\begin{aligned}
&\geq \left(\frac{n}{2} - \ell\right)^2 \cdot \frac{1}{2} \\
&= \frac{n^2}{8} - \frac{4n\ell}{8} + \frac{\ell^2}{2} \\
&> \frac{n^2}{8} - \frac{4n\ell}{8} \\
&\geq \frac{12n\ell}{8} - \frac{4n\ell}{8} \\
&= n\ell
\end{aligned}$$

This contradicts the assumption that at most $n\ell$ edges are pebbled. □

We use this observation to show that Duplicator has a winning strategy in the ℓ-molecule bijection game on CFI graphs for suitable ℓ.

Lemma 5.36. *Let $\ell \in \mathbb{N}$ and $n \geq 12\ell$. Further let $\mathbf{K}_n^< = (V, E, <)$, and let $T = \emptyset$, $S = \{v\}$ for some vertex $v \in V$. Then Duplicator wins the ℓ-molecule bijection game on the corresponding CFI graphs $\mathbf{G}^T$, $\mathbf{G}^S$ over $\mathbf{K}_n^<$.*

Duplicator can further guarantee that for every bijection g she chooses, there is a set $F \subseteq E$, a vertex $v_0 \in V$ and an edge $e = \{v_0, v_1\}$ such that

- *e is free,*
- *v_0 and v_1 are $(\geq \frac{n}{2})$-free,*
- $g(x) = \begin{cases} \rho_{F\Delta\{e\}}(x), & \text{if } x \in v_0^*, \\ \rho_F(x), & \text{otherwise.} \end{cases}$

This means that g preserves the parity of each vertex gadget except for v_0^*. We say that the pair (v_0, e) is *inconsistent*. If the parity of a vertex gadget is preserved, the vertex is *consistent*.

Proof. First note that if Duplicator can always play a bijection with the above properties, she wins the bijection game: g preserves all edges except those between vertices in v_0^* and e^*, on which there are no pebbles.

Initially, Duplicator can play a bijection with the required properties by choosing $v_0 = v$ and $e \in E(v_0)$ arbitrarily. Now consider an arbitrary move for Duplicator in the bijection game such that she has previously played a bijection g with the required properties. By the induction hypothesis, we can choose $F \subseteq E$, $v_0, v_1 \in V$ and $e \in E$ as above.

If, in the current move, v_0 and v_1 are still $(\geq \frac{n}{2})$-free and e is still free, then Duplicator can simply play g again. Otherwise, we distinguish several cases.

1. There is a pebble on v_0.
   a) If v_1 is still $(\geq \frac{n}{2})$-free, we choose a $(\geq \frac{n}{2})$-free neighbour v_2 of v_1 and make the pair $(v_1, \{v_1, v_2\})$ inconsistent. Then the edge set associated with the new bijection is $F \triangle \{e\}$.
   b) If v_1 is not $(\geq \frac{n}{2})$-free anymore, it lost this property because of the single pebble on v_0. So it is still incident to $\frac{n}{2} - 1$ free edges leading to free vertices. More than $\frac{n}{2}$ of the vertices are $(\geq \frac{n}{2})$-free. So v_1 has a $(\geq \frac{n}{2})$-free neighbour v_2 such that $\{v_1, v_2\}$ is free. Since v_2 is $(\geq \frac{n}{2})$-free, it is incident to at most $\frac{n}{2} - 1$ pebbled edges. Not counting v_2, there are still $\geq \frac{n}{2}$ many vertices that are $(\geq \frac{n}{2})$-free, so one of them is connected to v_2 with a free edge.
   Call that vertex v_3, and choose $(v_2, \{v_2, v_3\})$ as the inconsistent pair for the new bijection g'. Let g' be the bijection associated with v_2, $\{v_2, v_3\}$ and the edge set $F' = F \triangle \{e, \{v_1, v_2\}, \{v_2, v_3\}\}$. Then, since $e \in F \triangle F'$, v_0 is now consistent. v_1 stays consistent because an even number of edges incident to v_1 is in $F \triangle F'$.
2. There is no pebble on v_0. Then either there is a pebble on e, or v_0 or v_1 is not $(\geq \frac{n}{2})$-free anymore. In all these cases, the new bijection can still be associated with the same edge set F. For the inconsistent pair, simply choose a new $(\geq \frac{n}{2})$-free vertex v_0'. v_0' has a $(\geq \frac{n}{2})$-free neighbour v_1' such that the edge $e' = \{v_0', v_1'\}$ is free. So the inconsistent pair (v_0', e') makes the new bijection satisfy all desired properties. □

Homogeneity Transferring this winning strategy to the structure over strongly supported sets relies on a special representation of every strongly supported set as a combination of a strong support that is a molecule and a form—a kind of template for the structure of the set. This representation is well-defined in case the CFI graphs are homogeneous.

The definition of homogeneity, i. e. that tuples with the same type are mapped to each other by an automorphism, constitutes a small technical difficulty: The

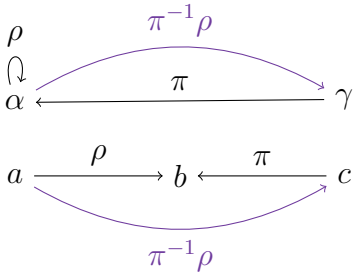

Figure 5.7: It suffices to construct an automorphism ρ from αa to αb.

usual notion of C^m-types is not easily applicable to the tuples that may form strong supports, because their length depends on the size of the input structure. To establish homogeneity in a useful sense, we therefore work with a non-standard notion of types.

Definition 5.37. Fix $k \in \mathbb{N}$. The *type* $\mathrm{tp}(\alpha)$ of the tuple $\alpha = (a_1, \ldots, a_s)$ of length $s \in \mathbb{N}$ from the structure $\mathbf{A}$ is the set of all $\mathrm{C}^{s+4k(4k-1)+3}$-formulae φ such that

- φ has free variables $x_1, \ldots, x_s$,
- $x_1, \ldots, x_s$ **do not occur as bound variables in** φ, and
- $\mathbf{A} \models \varphi(a_1, \ldots, a_s)$.

As the length of the tuple α is not fixed for fixed k, it can be any tuple, in particular a molecule or a concatenation of molecules. The number of variables is chosen in a way that CFI graphs over ordered complete graphs are homogeneous with respect to this definition of types.

Lemma 5.38 (Homogeneity)**.** *Let $k \in \mathbb{N}$, and let $\mathbf{G}^T$ be a CFI graph with underlying graph $\mathbf{K}_n^< = (V, E, <)$. For any two molecules α, γ of the same length,* $\mathrm{tp}(\alpha) = \mathrm{tp}(\gamma)$ *implies that there is an automorphism of $\mathbf{G}^T$ mapping α to γ.*

Proof. Induction on the length i of the molecules α and γ. The statement is clear for $i = 0$, and for $i = 1$, it follows from the fact that the gadgets are definable via the linear order on the underlying graph, because for any two vertices in the same gadget, there is some automorphism exchanging them. In the induction step, let $i \geq 1$ and consider $(i+1)$-tuples αa, γc with the same type.

Since also $\mathrm{tp}(\gamma) = \mathrm{tp}(\alpha)$, there is an automorphism π with $\pi(\gamma) = \alpha$ by induction hypothesis. Choose $b = \pi(c)$. It suffices to show that there is an automorphism ρ with $\rho(\alpha a) = \alpha b$, because then, $\pi^{-1}\rho$ is an automorphism mapping αa to γc (see Figure 5.7).

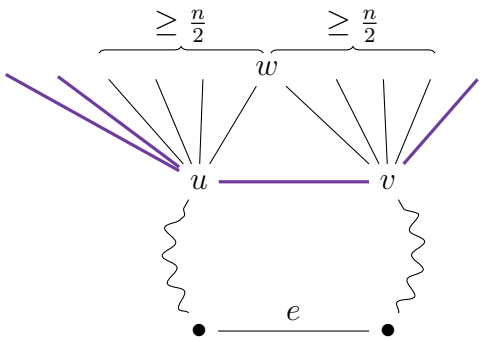

Figure 5.8: Case 1 of the proof of Lemma 5.38. The marked edges are those that occur in α.

Since αa and γc are molecules with at least two elements, they consist only of vertices from edge gadgets. Each edge gadget is definable in C^3 using the order on the vertex gadgets, so, as $\mathrm{tp}(\alpha a) = \mathrm{tp}(\gamma c) = \mathrm{tp}(\alpha b)$, a and b are in the same edge gadget e^*.

We call an edge gadget (or its corresponding edge) *free* if no vertex from the gadget occurs in α (resp. γ), and otherwise the gadget and the edge are *fixed*. A vertex is $(\geq \frac{n}{2})$-*free* if it is incident to $\geq \frac{n}{2}$ free edges, and $(< \frac{n}{2})$-*free* otherwise.

1. First consider the case that there are disjoint paths via free edges from both vertices incident to e to $(\geq \frac{n}{2})$-free vertices u and v. If $u = v$ or the edge $\{u, v\}$ is free, then it follows immediately that e is on a cycle consisting of free edges. If the edge $\{u, v\}$ is fixed, then u and v are each incident to at least $\frac{n}{2}$ free edges to the remaining $n - 2$ vertices. So, as shown in Figure 5.8, there is a vertex w that is connected to both u and v with a free edge, and e is again on a cycle of free edges. Let F be the set of edges on that cycle. Then ρ_F is an automorphism of $\mathbf{G}^T$ that fixes α and exchanges a and b.
2. If there are no such disjoint paths, assume there is no automorphism exchanging a and b (i. e. flipping e) while fixing α. We construct a formula with $4k(4k-1)+3$ additional variables that distinguishes a and b.
   Choose u and v such that $e = \{u, v\}$ and all vertices reachable from u without using e or a fixed edge are $(< \frac{n}{2})$-free. If no such u exists, there are $(\geq \frac{n}{2})$-free vertices u', v' reachable from the endpoints of e via free edges. If the paths to these vertices are disjoint, we are in Case 1. Otherwise, e is on a cycle of free edges.
   Since there is no automorphism flipping e, there is a unique element of $e^* = \{a, b\}$ such that the gadget u^* is even if, and only if, that element is labelled e^1. Our

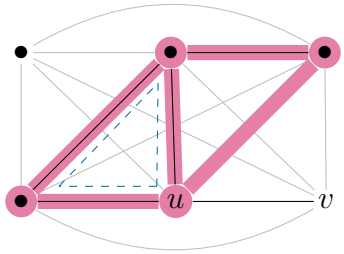

Figure 5.9: Case 2 of the proof of Lemma 5.38. Only the free edges, i. e those that can be used to reach vertices from u, are drawn in black. The subgraph H induced by the vertices reachable from u without using $\{u, v\}$ or a fixed edge is marked. The dashed triangle corresponds to the only automorphism flipping edges incident to u while fixing α.

formula will state that property for all automorphisms flipping only edges incident to $(< \frac{n}{2})$-free vertices. Since there is a constant number of $(< \frac{n}{2})$-free vertices, we can deduce a bound on the number of variables.

Let H be the subgraph of G induced by the free vertices reachable from u without using e or a fixed edge (illustrated in Figure 5.9). Then every automorphism flipping edges incident to u while fixing α is the disjoint union of cycles in H and cycles that do not affect any edges incident to u. So the image of all gadgets corresponding to edges incident to u is already uniquely determined by the cycles in H. Consequently, we can quantify over all relevant automorphisms by quantifying over the free edges in H. Furthermore, all vertices in H are $(< \frac{n}{2})$-free.

For ease of notation, we fix an isomorphic copy of $\mathbf{G}^T$ such that all gadgets corresponding to vertices in H are even. Let $e_1 \dots e_h$ be an enumeration of the edges in H.

We aim to construct a formula $\psi(x)$ defining the vertex $x = e^i$ that, when labelled e^1, makes u^* an even gadget. Every automorphism flipping an edge incident to u corresponds to a labelling of the edges $e_1, \dots, e_h$ that makes all vertex gadgets in H even. We represent such a labelling as an assignment of the variables $x_1, \dots, x_h$ to vertices in $e_1^*, \dots, e_h^*$, respectively. This can be defined with formulae of the form $\varphi_{e^*}(x)$ expressing that $x \in e^*$ for any $e \in E$. Further, let $\varphi_{v^*}(y)$ state that $y \in v^*$ for $v \in V$. These formulae exist because the underlying graph is linearly ordered.

By definition of H, every free edge $\neq \{u, v\}$ incident to a vertex in H is in $E[H] = \{e_1, \dots, e_h\}$. A vertex gadget v^* for $v \in V[H]$ is even if, and only if, it has an even number of neighbours labelled e^1. But that labelling is determined by

the labelling of the edge gadgets occurring in α, the assignment of the variables $x_1, \ldots, x_h$, and the choice of the vertex x to be defined. We first construct a formula $\varphi_{\text{even}}(x_1, \ldots, x_h, x, y)$ stating that y is in an even gadget in that sense. Let $F^1 = \{f \in E : f^1 \text{ occurs in } \alpha\}$ and $F^0 = \{f \in E : f^0 \text{ occurs in } \alpha\}$. For any $f \in F^1 \cup F^0$, let a_f be the corresponding entry of α. Then φ_{even} is the formula

$$\bigvee_{2 \leq i \leq \lfloor \frac{n-1}{2} \rfloor} \exists^{2i} z \left(Eyz \wedge \left(z = x \vee \bigvee_{1 \leq i \leq h} z = x_i \vee \right. \right. \\ \left. \left. \bigvee_{f \in F^1} z = a_f \vee \bigvee_{f \in F^0} (\varphi_{f^*}(z) \wedge z \neq a_f) \right) \right).$$

The final formula ψ should state that if all gadgets corresponding to vertices in $V[H] \setminus \{u\}$ are even, then u^* is even. For any set $W \subseteq V[H]$, the following formula is true if, and only if, all vertex gadgets in W^* are even:

$$\varphi^W_{\text{even}}(x_1 \ldots, x_h, x) \coloneqq \forall y \left(\bigvee_{v \in W} \varphi_{v^*}(y) \to \varphi_{\text{even}}(x_1, \ldots, x_h, x, y) \right).$$

Then the following formula ψ defines the unique vertex x that, for all automorphisms defined by valid assignments to the variables $x_1, \ldots, x_h$, makes the gadget u^* even:

$$\psi(x) \coloneqq \forall x_1 \ldots x_h \left(\bigwedge_{1 \leq i \leq h} \left(\varphi_{e_i^*}(x_i) \wedge \varphi^{V[H] \setminus \{u\}}_{\text{even}} \right) \to \varphi^{\{u\}}_{\text{even}} \right).$$

It remains to provide an upper bound on the number of variables in ψ. The variables used in ψ are $x_1, \ldots, x_h, x, y, z$. Note that, for the formulae of the form φ_{e^*} or φ_{v^*}, it suffices to reuse z.

The number h of edges in H can be computed from the number of $(< \frac{n}{2})$-free vertices in the whole graph. We claim that at most $4k$ vertices are $(< \frac{n}{2})$-free. Every $(< \frac{n}{2})$-free vertex is incident to at least $\frac{n}{2}$ fixed edges. So, if there were $> 4k$ such vertices, the number of fixed vertices would be $> \frac{1}{2} \cdot \frac{n}{2} \cdot 4k = nk$.

This is not possible because α is a molecule. It follows that $h \leq 4k(4k-1)$. So the number of variables in ψ is at most $4k(4k-1) + 3$. □

Transferring winning strategies Next, we give a definition of the representation of sets in terms of their supports introduced in [23]. This definition can directly be transferred to sets supported by molecules and our modified definition of types.

Definition 5.39. Let c be a constant symbol. The set of *forms* is defined inductively as follows: The constant c is a form, and every finite set of pairs (φ, τ) where φ is a form and $\tau = \mathrm{tp}(\alpha\beta)$ for molecules α, β is a form.

Definition 5.40. The *denotation* of a form φ and a molecule α over a structure $\mathbf{A}$ is defined as follows:

- $\varphi \star () = \emptyset$ for every form φ,
- $c \star \alpha = \alpha_1$ for non-empty α (in particular, $c = a$ if $\alpha = (a)$),
- $\varphi \star \alpha = \{\psi \star \beta : (\psi, \mathrm{tp}(\alpha, \beta)) \in \varphi\}$ if φ is a set.

Lemma 5.41 ([23])**.** *α supports $\varphi \star \alpha$.*

The following useful property of denotations is implied by the homogeneity condition:

Lemma 5.42 ([23])**.** *$\pi(\varphi \star \alpha) = \varphi \star \pi(\alpha)$ for any automorphism π.*

The denotations of forms and k-tuples (instead of molecules), as defined in [23], are exactly the objects with a support of size k. The precise characterisation does not transfer to strong supports. Nevertheless, every strongly supported object is the denotation of a form and a strong support.

Lemma 5.43. *Let $x \in \mathrm{HF}(\mathbf{A})_{k-\mathrm{ss}}$ for a CFI graph $\mathbf{A}$ over $\mathbf{K}_n^<$. Then there is a form φ and a molecule α such that $x = \varphi \star \alpha$ and α strongly supports x.*

Proof. If x is an atom, then it is strongly supported by the molecule $\alpha \coloneqq (x)$, and $x = c \star \alpha$. If x is a set, choose a molecule α that strongly supports x. Analogously to the proof of Lemma 38 in [23], one can show that there is a form φ such that $x = \varphi \star \alpha$, using our modified homogeneity condition from Lemma 5.38. □

When transferring winning strategies, we will lift bijections on molecules to bijections on strongly supported sets. The lifted bijections are easier to define if the image of every strongly supported set depends only on the image of a single, canonical support.

Lemma 5.44. *For every linear order on the set of forms and every linear order on the set of types of molecules, every $x \in \mathrm{HF}(\mathbf{A})$ has a unique representation (φ_x, α_x) such that*

- $\varphi_x \star \alpha_x = x$,
- α_x *strongly supports* x,
- $\mathrm{tp}(\alpha_x)$ *is the minimal (w.r.t. the given linear order) type of a molecule strongly supporting* x,
- φ_x *is the minimal (w.r.t. the given linear order) form such that* $\varphi_x \star \alpha = x$ *for any* α *of type* τ.

We call the pair (φ_x, α_x) *the* canonical representation *of* x.

Proof. Given linear orders on forms and types of molecules, it is easy to define a minimal form φ_x and a molecule α_x of minimal type satisfying the conditions from the lemma. It remains to show that α_x is unique.

Suppose there are molecules $\alpha \neq \beta$ of type τ such that $\varphi_x \star \alpha = \varphi_x \star \beta = x$ and both α and β are strong supports. By the homogeneity condition, there is an automorphism π mapping α to β, because they are of the same type. But then $\pi(x) = \pi(\varphi_x \star \alpha) = \varphi_x \star \pi(\alpha) = \varphi_x \star \beta = x$, even though $\pi(\alpha) \neq \alpha$. This contradicts the fact that α is a strong support. So α_x is unique. □

The representation in terms of forms and molecules further yields a characterisation of the set structure of $\mathrm{HF}(\mathbf{A})_{k-\mathrm{ss}}$ based solely on the form and the type of the molecule. This will ensure that the lifted bijections preserve the set structure, since the underlying bijections obtained from the ℓ-molecule bijection game on the CFI graphs will preserve the types of molecules.

Lemma 5.45 ([23])**.** *There are relations* In *and* Eq *such that, for all forms* φ, ψ *and all molecules* α, β *of the same length,* $\varphi \star \alpha = \psi \star \beta$ *if, and only if,* $\mathrm{Eq}(\varphi, \psi, \mathrm{tp}(\alpha, \beta))$ *and* $\varphi \star \alpha \in \psi \star \beta$ *if, and only if,* $\mathrm{In}(\varphi, \psi, \mathrm{tp}(\alpha, \beta))$.

The proof is completely analogous to that of Lemma 39 in [23], the modified definition of types does not change the definition of the relations In and Eq.

We now use that machinery together with the winning strategy in the game on the CFI graphs to describe a winning strategy in the standard m-pebble bijection game on the strongly supported sets.

Lemma 5.46. *Let* $k, m \in \mathbb{N}$, $\ell = km + 4k(4k-1) + 3$, $n \geq 12\ell$, *and let* $\mathbf{G}^{\emptyset}, \mathbf{G}^{\{v\}}$ *be CFI graphs over* $\mathbf{K}_n^{\leq}$ *for some vertex* v. *Then Duplicator wins the* m*-pebble bijection game on* $\mathrm{HF}(\mathbf{G}^{\emptyset})_{k-\mathrm{ss}}$ *and* $\mathrm{HF}(\mathbf{G}^{\{v\}})_{k-\mathrm{ss}}$.

Consider any position $(x_1, y_1), \ldots, (x_m, y_m)$ in the game. We will later use as an invariant that there are forms $\varphi_1, \ldots, \varphi_m$ and molecules $\alpha_1, \ldots, \alpha_m$, $\beta_1, \ldots, \beta_m$,

such that $x_i = \varphi_i \star \alpha_i$, $y_i = \varphi_i \star \beta_i$ and Duplicator wins the ℓ-molecule bijection game from the position $(\alpha_1, \beta_1), \ldots, (\alpha_m, \beta_m)$ (where the pair (α_i, β_i) means that there are pebbles on all corresponding pairs of entries of the molecules α_i, β_i). We use Duplicator's winning strategy in the molecule bijection game on the structures $\mathbf{G}^{\emptyset}$ and $\mathbf{G}^{\{v\}}$ to obtain a bijection g for Duplicator in the game on the strongly supported sets.

Consider a position in the game on $\mathbf{G}^{\emptyset}, \mathbf{G}^{\{v\}}$ with pebbles on $(\alpha_2, \beta_2), \ldots, (\alpha_m, \beta_m)$. Assume Duplicator has a winning strategy in the ℓ-molecule bijection game with $\ell > km$ (which will follow from Lemma 5.36 by induction). Then there is a bijection f between molecules such that Duplicator wins the game starting from the position $(\alpha, f(\alpha)), (\alpha_2, \beta_2), \ldots, (\alpha_m, \beta_m)$ for any molecule α: If Spoiler successively pebbles the elements $a_1, \ldots, a_{nk}$ of α, Duplicator plays bijections h_1,$h_2[a_1]$,$h_3[a_1, a_2], \ldots, h_{nk}[a_1, \ldots, a_{nk-1}]$ according to her winning strategy. (In case the molecules are tuples of length $s < nk$, fix any additional vertices $a_{s+1}, \ldots, a_{nk}$ from edge gadgets.) Choose

$$f(\alpha) = (h_1(a_1), h_2[a_1](a_2), h_3[a_1, a_2](a_3), \ldots, h_{nk}[a_1, \ldots, a_{nk-1}](a_{nk})).$$

Note that f is a bijection, because its inverse can be constructed using the same bijections $h_1, h_2[a_1], \ldots, h_{nk}[a_1, \ldots, a_{nk-1}]$. Define $g\colon \mathrm{HF}(\mathbf{G}^{\emptyset})_{k-\mathrm{ss}} \to \mathrm{HF}(\mathbf{G}^{\{v\}})_{k-\mathrm{ss}}$ through $g(x) = \varphi_x \star f(\alpha_x)$, where (φ_x, α_x) is the canonical representation of x. We show that g is well-defined and a bijection.

Lemma 5.47. *Let π be an automorphism and α a molecule, where we identify the automorphism $\pi = \rho_F$ of $\mathbf{G}^{\emptyset}$ with the automorphism of $\mathbf{G}^{\{v\}}$ induced by the same edge set F. Then* $\mathrm{tp}(\alpha, \pi(\alpha)) = \mathrm{tp}(f(\alpha), \pi(f(\alpha)))$.

Proof. By definition of f, Duplicator wins the $(4k(4k-1)+3)$-pebble bijection game on $\mathbf{G}^{\emptyset}, \mathbf{G}^{\{v\}}$ while always mapping α to $f(\alpha)$. But this also means that, for every entry α_i of α, $f(\alpha)_i$ already uniquely determines how her bijection acts on the whole gadget containing α_i.

- If α consists only of edge gadgets, then $\mathrm{tp}(\alpha, \pi(\alpha))$ is already completely determined by equalities. If $\pi(\alpha_i) = \alpha_i$, then clearly also $\pi(f(\alpha)_i) = f(\alpha)_i$. If $\pi(\alpha_i) \neq \alpha_i$, then $\alpha_i = e^j$ and $\pi(\alpha_i) = e^{1-j}$ for some $j \in \{0, 1\}$, so $e \in F$. But since $f(\alpha)_i$ can be either e^j or e^{1-j}, π also flips $f(\alpha)_i$.
- Now consider the case that $\alpha = v^X$ for some vertex v. Then $\mathrm{tp}(\alpha, \pi(\alpha))$ is completely determined by the symmetric difference of the outer vertices adjacent to v^X and $\pi(v^X)$.

Let $\pi(\alpha) = v^Y$ and $f(\alpha) = v^{X \triangle Z}$. Assume $v^{Y'} := f(\pi(\alpha)) \neq v^{Y \triangle Z}$. Then there is some edge $e \in (Y \triangle Y') \setminus Z$. Then Spoiler wins by putting a pebble on v^Y and then on e^1: Since $e \in (Y \triangle Y') \setminus Z$, e^1 is incident to v^X if, and only if, it is incident to $v^{X \triangle Z} = f(v^X)$. But e^1 is incident to v^Y if, and only if, it is *not* incident to $v^{Y'}$. □

Lemma 5.48. *g is well-defined.*

Proof. We need to show that for every set $\varphi \star \alpha$ that is strongly supported by α, $\varphi \star f(\alpha)$ is also strongly supported by $f(\alpha)$.

Let π be an automorphism of $\mathbf{G}^\emptyset$ and $\mathbf{G}^{\{v\}}$, where we identify automorphisms as in Lemma 5.47.

By Lemma 5.47, $\mathrm{tp}(\alpha, \pi(\alpha)) = \mathrm{tp}(f(\alpha), \pi(f(\alpha)))$. Since $\pi(\varphi \star f(\alpha)) = \varphi \star \pi(f(\alpha))$, it follows that

$$\begin{aligned}
&\varphi \star \alpha = \pi(\varphi \star \alpha) \\
\Leftrightarrow &\varphi \star \alpha = \varphi \star \pi(\alpha) \\
\Leftrightarrow &\mathrm{Eq}(\varphi, \varphi, \mathrm{tp}(\alpha, \pi(\alpha))) \\
\Leftrightarrow &\mathrm{Eq}(\varphi, \varphi, \mathrm{tp}(f(\alpha), \pi(f(\alpha)))) \\
\Leftrightarrow &\varphi \star f(\alpha) = \varphi \star \pi(f(\alpha)) \\
\Leftrightarrow &\varphi \star f(\alpha) = \pi(\varphi \star f(\alpha))
\end{aligned}$$

So π fixes $\varphi \star \alpha$ if, and only if, π fixes $\varphi \star f(\alpha)$. Since, by definition of molecules, π also fixes α if, and only if, it fixes $f(\alpha)$, it follows that α strongly supports $\varphi \star \alpha$ if, and only if, $f(\alpha)$ strongly supports $\varphi \star f(\alpha)$. □

The following property will imply that g is a bijection:

Lemma 5.49. *For every set x that is strongly supported by a molecule α, (φ, α) is the canonical representation of $x = \varphi \star \alpha$ if, and only if, $(\varphi, f(\alpha))$ is the canonical representation of $\varphi \star f(\alpha)$.*

Proof. Since f is a bijection, the proofs for both directions are symmetric. Let $\varphi \star \alpha$ be canonical, and suppose that $\varphi \star f(\alpha)$ is not canonical. Then either $\mathrm{tp}(f(\alpha))$ is not minimal, or it is minimal, but φ is not.

$\mathrm{tp}(f(\alpha))$ **is not minimal:** Then there is a molecule β with $\mathrm{tp}(f(\beta)) < \mathrm{tp}(f(\alpha))$ such that $f(\beta)$ strongly supports $\varphi \star f(\alpha)$. Since f preserves types of molecules,

also $\mathrm{tp}(\beta) < \mathrm{tp}(\alpha)$. Furthermore, β strongly supports $\varphi \star \alpha$: Let π be an automorphism. Then

$$\begin{aligned}
&\pi(\varphi \star \alpha) = \varphi \star \alpha \\
\Leftrightarrow &\pi(\alpha) = \alpha \\
\Leftrightarrow &\pi(f(\alpha)) = f(\alpha) \\
\Leftrightarrow &\pi(\varphi \star f(\alpha)) = \varphi \star f(\alpha) \\
\Leftrightarrow &\pi(f(\beta)) = f(\beta) \\
\Leftrightarrow &\pi(\beta) = \beta,
\end{aligned}$$

since α strongly supports $\varphi \star \alpha$, $f(\beta)$ strongly supports $\varphi \star f(\alpha)$, and α and $f(\alpha)$ (respectively β and $f(\beta)$) are fixed by the same automorphisms (i. e. those we identify because they flip the same edges).

φ is not minimal: Then there are a form ψ and a tuple β such that $\psi < \varphi$ and $\psi \star f(\beta) = \varphi \star f(\alpha)$. Since $\mathrm{tp}(f(\alpha))$ is minimal, the tuples $f(\beta), f(\alpha), \alpha$ and β all have the same type.

By the homogeneity condition shown in Lemma 5.38, there is an automorphism π that maps $f(\alpha)$ to $f(\beta)$ and α to β. So, by Lemma 5.47, $\mathrm{tp}(f(\alpha), f(\beta)) = \mathrm{tp}(\alpha, \beta)$. But then, because of the relation Eq, $\varphi \star f(\alpha) = \psi \star f(\beta)$ implies $\varphi \star \alpha = \psi \star \beta$, which contradicts the assumption that (φ, α) is canonical. □

Lemma 5.50. *g is a bijection.*

Proof. First we show that g is injective. Let $\varphi \star f(\alpha) = \psi \star f(\beta)$, such that $f(\alpha), f(\beta)$ are strong supports and (φ, α) and (ψ, β) are canonical representations. By Lemma 5.49, $(\varphi, f(\alpha))$ and $(\psi, f(\beta))$ are also canonical representations. So $\varphi = \psi$ and $f(\alpha) = f(\beta)$, and thus $\varphi \star \alpha = \psi \star \beta$.

To show surjectivity, let $x \in \mathrm{HF}(\mathbf{G}^{\{v\}})$ be strongly supported, and let (φ, α) be its canonical representation. By Lemma 5.49, $(\varphi, f^{-1}(\alpha))$ is also canonical. So $\varphi \star \alpha$ is in the image of g (note that, analogously to the well-definedness proof, $\varphi \star f^{-1}(\alpha)$ is strongly supported by $f^{-1}(\alpha)$). □

It remains to show that Duplicator wins if she plays these bijections.

Proof of Lemma 5.46. We show that Duplicator wins the m-pebble bijection game on $\mathrm{HF}(\mathbf{G}^{\emptyset})_{k-\mathrm{ss}}$, $\mathrm{HF}(\mathbf{G}^{\{v\}})_{k-\mathrm{ss}}$ by playing the bijections g defined above. That strategy preserves the following invariant: In every position $(x_1, y_1), \ldots, (x_m, y_m)$,

there are forms $\varphi_1, \ldots, \varphi_m$ and molecules $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m$ such that, for every i, (φ_i, α_i) is the canonical representation of x_i and (φ_i, β_i) is the canonical representation of y_i, and Duplicator wins the ℓ-molecule bijection game from the position $(\alpha_1, \beta_1), \ldots, (\alpha_m, \beta_m)$.

In the initial position, we assume that all pebbles are on $\emptyset$, and choose all forms and molecules to be empty. Duplicator wins the game by Lemma 5.36. For the induction step, let $(x_1, y_1), \ldots, (x_m, y_m)$ be the position after Duplicator has played the bijection g and Spoiler has placed a pebble on the pair $(x_1, y_1 \coloneqq g(x_1))$. Let (φ_1, α_1) be the canonical representation of x_1 and choose $\beta_1 = f(\alpha_1)$. Since f is played as part of a winning strategy, and by Lemmas 5.48 and 5.49, the invariant holds.

It remains to show that $x_1, \ldots, x_m \mapsto y_1, \ldots, y_m$ is a partial isomorphism. The edge relation between atoms is preserved because Duplicator wins the game on the strong supports. Spoiler only needs nkm pebbles on edge gadgets and km pebbles on vertex gadgets to fix the strong supports of $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$. Since $\ell = km + 4k(4k-1) + 3$, there are $\geq 4k(4k-1) + 3$ pebbles for both edge and vertex gadgets left to verify that $\mathrm{tp}(\alpha_1, \ldots, \alpha_m) = \mathrm{tp}(\beta_1, \ldots, \beta_m)$. It follows that the relations $=$ and $\in$ between sets are preserved because of the relations In and Eq:

$$\begin{aligned} & x_i = x_j \\ \Leftrightarrow\, & \mathrm{Eq}(\varphi_i, \varphi_j, \mathrm{tp}(\alpha_i, \alpha_j)) \\ \Leftrightarrow\, & \mathrm{Eq}(\varphi_i, \varphi_j, \mathrm{tp}(\beta_i, \beta_j)) \\ \Leftrightarrow\, & \varphi_i \star \beta_i = \varphi_j \star \beta_j \\ \Leftrightarrow\, & y_i = y_j. \end{aligned}$$

The proof for $\in$ is analogous. □

Inexpressibility of the Cai-Fürer-Immerman query The previous results are combined as explained in Section 5.2 to obtain

Theorem 5.51. *Let $(\varphi, p) \in \mathrm{CPT}$ be a sentence which activates only strongly supported sets. Then (φ, p) cannot define the Cai-Fürer-Immerman query over ordered complete graphs.*

Proof. To obtain a contradiction, assume that there is a CPT-sentence $\overline{\varphi} = (\varphi, p)$ defining the query. First we translate $\overline{\varphi}$ into an equivalent C^m-formula φ_{act} which simulates $\overline{\varphi}$ on $\mathbf{act}(\overline{\varphi}, \mathbf{G}^T)$, i. e. $\mathbf{G}^T \models \overline{\varphi}$ if, and only if, $\mathbf{act}(\overline{\varphi}, \mathbf{G}^T) \models \varphi_{\mathrm{act}}$.

Let, without loss of generality, $p\colon n \mapsto n^q$ for some $q \geq 1$, and let $k = 4q$. By Lemma 5.28, every object activated by $\overline{\varphi}$ on a CFI graph $\mathbf{G}^T$ over $\mathbf{K}_n^<$ has a strong support that consists of $< nk$ vertices that are all in edge gadgets. Hence $\mathrm{act}(\overline{\varphi}, \mathbf{G}^T) \subseteq \mathrm{HF}(\mathbf{G}^T)_{k-\mathrm{ss}}$ for every such CFI graph.

Let $\ell = km + 4k(4k-1) + 3$ and let $\mathbf{G}^T$ be an even and $\mathbf{G}^S$ be an odd CFI graph over the ordered complete graph $\mathbf{K}_n^<$ for some $n \geq 12\ell$. By Lemma 5.46, Duplicator wins the m-pebble bijection game on $\mathrm{HF}(\mathbf{G}^T)_{k-\mathrm{ss}}$ and $\mathrm{HF}(\mathbf{G}^S)_{k-\mathrm{ss}}$. So $\overline{\varphi}$ cannot distinguish between $\mathbf{G}^T$ and $\mathbf{G}^S$. □

5.6 Conclusion

We examined expressibility of the Cai-Fürer-Immerman query to expand the classification of the expressive power of CPT. For CFI graphs over preordered graphs with colour classes of logarithmic size, as well as over graph classes with vertices of large degree, we established CPT-definability. In the latter case, the CFI query is definable using only sets of bounded rank. We used this result as a starting point for a deeper analysis of the expressive power of PIL without congruences, i. e. the fragment of CPT with restricted set-building abilities. Using a characterisation of set-like objects based on automorphism groups, we proved that congruence-free PIL—and, in fact, any fragment of CPT over tuple-like objects—cannot define the CFI query over graphs with large degree. Recall that, as shown in Chapter 4, congruence-free PIL is included in CPT over sets of bounded rank over structures of bounded colour class size. The precise relation between these two fragments of CPT over general structures is open. In particular, it is still possible that they are incomparable.

Similarly to padding constructions, our results depend on the size of the input structure, i. e. the CFI graphs. Nevertheless, extending the known techniques to these graph classes is already technically challenging. Consequently, it is not obvious how—and if—these techniques can be extended to prove expressibility of the CFI query over more general classes of graphs.

The proof of our inexpressibility result extends known proof techniques beyond fragments of CPT that can define only sets with supports of constant size. However, there are several restrictions that make full CPT inaccessible to these techniques. The decomposition of hereditarily finite sets into forms and molecules depends on homogeneity of the input structure. Thus, if there is a Ptime-query that is not CPT-definable, that query has to be reduced to homogeneous structures. More

importantly, the method requires some restriction on the types of sets that can be defined by the given fragment of CPT, such as sets with constant support or strongly supported sets. But as we know from [23] and our own result for CFI over complete graphs, CPT can in general define sets that do not have these properties. Applying this proof technique to full CPT would therefore require a new characterisation of CPT-definable sets which allows a reduction to pebble games over simpler objects.

In contrast, pebble games that are played directly on hereditarily finite sets are presumably too complicated to be of use, at least if they were defined in a straightforward fashion. Such a definition would include moves like "choose a polynomial bound", or "choose a hereditarily finite set". This is another reason why a characterisation of CPT without polynomial bounds is desirable. Altogether, it seems plausible that more elaborate inexpressibility results require novel proof techniques. As a first application of such proof techniques, one could consider NP-hard queries instead of those that are known to be in PTIME.

6 Choiceless Logarithmic Space

So far we have seen why Choiceless Polynomial Time constitutes a promising candidate for a logic capturing Ptime. Thus we set out to transfer the notion of choiceless computation to other complexity classes. Logarithmic space has been a subject of interest since the early days of descriptive complexity: When Chandra and Harel [18] enquired about a classification of Ptime-queries, they already asked about Logspace as well. Moreover, Logspace has gained practical importance in recent years, as it formalises efficient computation over large data sets with small working memory. In other words, it can be seen as a formalisation of efficient computation for big data. Therefore, we aim to develop a model of choiceless computation for Logspace.

Logics for Logspace have evolved in a similar way as logics for Ptime. On ordered structures, Logspace can be captured by an extension of FO by an appropriate recursion mechanism: Deterministic transitive closure logic (DTC). But, even with an added counting operation, it does not capture Logspace on arbitrary structures. Some of the shortcomings of DTC are elegantly countered by the logic LREC (short for L-recursion), especially in its stronger variant, called $\text{LREC}_=$ in the literature. The core of LREC is an elaborate recursion operator, augmented by a mechanism defining symmetric transitive closures to obtain closure under interpretations. Though LREC captures Logspace on a larger class of structures than transitive closure logics, it is contained in FP+C and thus does not capture Logspace on the class of all structures. Known logics for Logspace are reviewed in detail in Section 6.1. As LREC is, to our knowledge, the strongest logic suggested to capture Logspace so far, there is currently no candidate comparable to the role CPT and FP+rk play for Ptime.

We aim to fill that gap with a restriction of CPT to logarithmic space. The central concepts of choiceless computation, i. e. symmetry-preserving computation with sets as data structures, should be maintained. The obvious change necessary when adjusting the approach to logarithmic space is at the same time the most challenging one: The polynomial time and space bounds have to be changed to a logarithmic space bound. CPT permits polynomially many computation steps involving sets

whose transitive closure is of polynomial size. Thus the naïve approach would be to discard the time bound and allow sets with a transitive closure of logarithmic size.

The problem is that the size of the transitive closure is measured in terms of the number of objects. In particular, this would make it possible to define sets containing logarithmically many atoms. But this admits no straightforward evaluation in Logspace: To represent such a set—over an ordered input structure—on a Turing machine, one would store every atom as a number with logarithmically many bits (in the size of the input structure). So we would actually have to represent logarithmically many numbers of logarithmic size. Unless Logspace $= \mathrm{NC}^2$, this should not be possible.

The central question is thus what it means that a set is of logarithmic size. Even though choiceless computation is supposed to be invariant under encodings, the size of a single atom already depends on its encoding. A classical algorithm receives as input an ordered encoding of a structure, so every atom can be referenced by its index—a binary number with logarithmically many bits. Assuming this straightforward encoding, the algorithm can only store constantly many atoms at once. With this reasoning, a set of logarithmic size is actually a set whose transitive closure is bounded by a constant. A previous approach to defining choiceless logarithmic space by Grädel and Spielmann [29, 57] operates on exactly those *bounded sets*. More precisely, it can be characterised by the logic BDTC that provides an operator for deterministic transitive closures over bounded sets. As shown in [29], BDTC lacks the ability to count and therefore cannot define Even, not even on padded structures. But, as we show in the following section, counting is not its only deficiency: BDTC-formulae are only as expressive as pure pointer programs, a mechanism introduced by Hofmann and Schöpp [39] specifically to formalise the (strict) fragment of Logspace that can handle only constantly many elements of the input structure.

The assumption that a set of logarithmic size can only contain constantly many atoms does not take into account that there are alternative ways to represent atoms. It is, for instance, possible to store the unique root of a tree in constant space by just storing its defining property. In a structure with a unary predicate P, every atom in P can be represented as "the kth element of P", so the size of the atom representation is logarithmic in $|P|$ instead of the size of the whole structure. Atoms can even be represented in terms of other atoms, say, "the ith successor of the jth successor of the root".

The size of an atom with respect to Logspace-algorithms can hence vary depending on its representation. This observation leads to the main idea behind Choiceless

LOGSPACE or CLogspace, the logic we introduce: When evaluating formulae with logarithmic bounds, we do not assume a fixed size for every atom. Therefore, the sizes of atoms are part of the semantics. Consequently, the logic is evaluated over *size-annotated hereditarily finite sets*.

In the examples above, atoms are represented using logical formulae. To formalise this idea in our logic, we introduce terms of the form "Atoms $.\varphi$" to make sure that, whenever a set of atoms is defined, the definition is guarded by a formula. The size of every atom in the set will be defined assuming a representation as "the ith entry of the kth tuple satisfying φ". So the size of a hereditarily finite object will depend on the terms of the form Atoms $.\varphi$ generating the atoms in its transitive closure.

The formula φ, however, does not have any additional free variables as parameters, as these would complicate evaluation in LOGSPACE. So this way of defining atoms does not allow recursion of non-constant depth. To make this kind of recursion possible, we simply add (a slightly modified version of) the recursion operator from LREC to our logic. As we will see in Section 6.4.3, LREC over hereditarily finite sets does not suffice to capture LOGSPACE. Just like in CPT, recursive creation of sets is necessary to obtain a formalism that is stronger than common logics.

Some characteristics of LOGSPACE-computations can be incorporated into our logic by modifying technical details in the definition of iteration terms. Firstly, LOGSPACE-algorithms are closed under sequential composition. So creating some output of *polynomial* size and running a LOGSPACE-algorithm on that output is again in LOGSPACE. We implement this by allowing iteration terms to have outputs of polynomial size (while the size of every stage has to be logarithmic). More precisely, the value of an iteration term is not the last stage of its iteration, but the set of *all* intermediate stages.

Secondly, a LOGSPACE-algorithm can iterate over a set of polynomial size and run a subroutine for each element of that set. So we add free variables to iteration terms to allow for evaluating an iteration term for different initial values in parallel.

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

Before we formally define the logic CLogspace in Section 6.2, we review some known logics within LOGSPACE. To justify that our logic is a model of choiceless computation for LOGSPACE, we prove in Section 6.4.2 that it is contained in both CPT and LOGSPACE. Further, as shown in Section 6.4.4, our logic captures LOGSPACE on certain padded structures, which we use in Section 6.4.5 to separate it from known logics in LOGSPACE.

6.1 Logics for Logspace

We aim to show that the logic we introduce in this chapter is strictly more expressive than known logics for Logspace. Therefore, we briefly define these logics here, and state some important results about their expressive power. Further, to allow for a deeper analysis of the expressive power (and weaknesses) of BDTC, we show that it can be simulated by pure pointer programs, a formalism whose expressive power has been studied extensively.

6.1.1 Transitive closure logics

Logics with restricted operators for transitive closures play a similar role for Logspace as fixed-point logic does for Ptime. These logics not only add a weaker (i. e. Logspace-computable) iteration mechanism to FO, they also permit capturing results on the class of ordered structures, and are hence a useful starting point for the analysis of logics for Logspace.

We will define the operators for both deterministic and symmetric transitive closures as restrictions of the general transitive closure operator.

Definition 6.1. A formula is in *transitive closure logic* TC if it is built from the rules for constructing FO-formulae, together with the following additional rule: Let $\varphi(\overline{u},\overline{v})$ be a formula with $\overline{u} = u_1,\dots,u_k, \overline{v} = v_1,\dots,v_k$ among its free variables, and let $\overline{x},\overline{y}$ be k-tuples of variables. Then

$$[\mathsf{tc}_{\overline{u},\overline{v}}\varphi(\overline{u},\overline{v})](\overline{x},\overline{y})$$

is a formula. The formula $[\mathsf{tc}_{\overline{u},\overline{v}}\varphi(\overline{u},\overline{v})](\overline{x},\overline{y})$ states that the pair $(\overline{x},\overline{y})$ is included in the transitive closure of the relation defined by φ. So $\mathbf{A} \models [\mathsf{tc}_{\overline{u},\overline{v}}\varphi(\overline{u},\overline{v})](\overline{a},\overline{b})$ if, and only if, there are $n \in \mathbb{N}$ and k-tuples $\overline{c}_0,\dots,\overline{c}_n$ such that $\overline{c}_0 = \overline{a}$, $\overline{c}_n = \overline{b}$, and $\mathbf{A} \models \varphi(\overline{c}_i,\overline{c}_{i+1})$ for all $i \leq n$.

Theorem 6.2 (Immerman [43])**.** TC *captures nondeterministic* Logspace *on the class of ordered structures.*

Syntactically, the logics DTC and STC are defined like TC, but with the operator dtc (respectively stc) in place of tc. The deterministic transitive closure operator dtc defines the transitive closure of a deterministic relation, so

$$[\mathsf{dtc}_{\overline{u},\overline{v}}\varphi(\overline{u},\overline{v})](\overline{x},\overline{y}) \equiv [\mathsf{tc}_{\overline{u},\overline{v}}\varphi(\overline{u},\overline{v}) \wedge \forall\overline{w}(\varphi(\overline{u},\overline{w}) \to \overline{v} = \overline{w})](\overline{x},\overline{y}).$$

The operator stc, on the other hand, defines the transitive closure of a symmetric relation, i. e. reachability in the undirected graph defined by φ. Thus

$$[\mathsf{stc}_{\overline{u},\overline{v}}\varphi(\overline{u},\overline{v})](\overline{x},\overline{y}) \equiv [\mathsf{tc}_{\overline{u},\overline{v}}\varphi(\overline{u},\overline{v}) \wedge \varphi(\overline{v},\overline{u})](\overline{x},\overline{y}).$$

Both DTC and STC capture Logspace on the class of ordered structures.

Theorem 6.3 (Immerman [43])**.**

1. DTC *captures* Logspace *on the class of ordered structures.*
2. STC *captures symmetric* Logspace *on the class of ordered structures.*

The second result implies that STC also captures Logspace on the class of ordered structures since, by Reingold's famous algorithm for undirected reachability [53], Logspace equals symmetric Logspace. Both DTC and STC, however, fail to define Even on unordered structures. In the variant with counting (which is defined as usual), not even TC can define isomorphism of directed trees [24], so stronger logics are necessary to capture Logspace on classes of unordered structures.

6.1.2 Choiceless computation over bounded sets

Our search for a formalism for Choiceless Logspace is based on an approach by Grädel and Spielmann [29, 57], which we call BDTC since it defines deterministic transitive closures over sets with constant bounds instead of atoms. The aim is to extend that formalism to achieve greater expressive power. Indeed, as we show in Section 6.4, our logic is strictly more expressive than BDTC.

Originally, BDTC was defined as a variant of abstract state machines with rather complex mechanisms for iteration within Logspace. That formalism was also denoted Choiceless Logspace. The equivalent logic BDTC we consider here was introduced by Spielmann in his PhD thesis [57].

BDTC is based on the idea that Logspace-algorithms can represent constantly many objects of logarithmic size. So it allows to define sets with a constant bound on the transitive closure. This implies a constant bound on the number of atoms in the transitive closure, so $c \log |A|$ bits suffice to represent c many atoms from a structure A.

For any σ-structure $\mathbf{A}$ and $c \in \mathbb{N}$, we define the set $\mathrm{HF}_c(A)$ as $\{a \in \mathrm{HF}(A) : |\operatorname{tc}(a)| \leq c\}$. $\mathrm{HF}_c(A)$ is the set of *bounded objects*, and we call $\mathrm{HF}_c(A) \setminus A$ the set of *bounded sets*.

Like CPT, BDTC consists of basic set-theoretic operations with an iteration mechanism. Note that ordinary BGS-terms can already define sets whose transitive closure exceeds a constant bound, so BDTC-terms defining sets are already equipped with an explicit constant bound. Iteration is realised as bounded deterministic transitive closure, i. e. a dtc-operator that ranges over bounded objects.

Definition 6.4 (Syntax of BDTC)**.** The terms and formulae of BDTC are defined inductively:

- Every variable is a term.
- $\emptyset$ is a term.
- If s is a term, then $\mathrm{Unique}(s)$ is a term.
- If $s_1, \ldots, s_k$ are terms and $c \in \mathbb{N}$, then $\{s_1, \ldots, s_k\}_c$ is a term.
- If s is a term, φ is a formula, c is a natural number, and r is either Atoms or a term without free occurrences of x, then $\{s : x \in r : \varphi\}_c$ is a term.
- If $s_1, \ldots, s_k$ are terms and R is a k-ary relation symbol, then $Rs_1 \ldots s_k$ is a formula.
- If s, t are terms, then $s = t$ and $s \in t$ are formulae.
- Boolean combinations of formulae are formulae.
- If ψ is a formula with $\overline{u} = u_1, \ldots, u_k, \overline{v} = v_1, \ldots, v_k \in \mathrm{free}(\psi)$, c is a natural number, and $\overline{r}, \overline{s}$ are two k-tuples of terms, then $[\mathsf{dtc}_{\overline{u},\overline{v}}\psi]_c(\overline{r}, \overline{s})$ is a formula.

The semantics is defined similarly to that of CPT, where the subscript c acts as a bound on the size of transitive closures.

Definition 6.5 (Semantics of BDTC)**.** Except for the following cases, the semantics is defined analogously to BGS. For these cases, let $\mathbf{A}$ be a σ-structure and let $\overline{a} = a_1, \ldots, a_\ell \in A$.

- If $t = \{s_1, \ldots, s_k\}_c$, then $t^{\mathbf{A}}(\overline{a}) = \{s_1^{\mathbf{A}}(\overline{a}), \ldots, s_k^{\mathbf{A}}(\overline{a})\}$ if $|\operatorname{tc}(\{s_1^{\mathbf{A}}(\overline{a}), \ldots, s_k^{\mathbf{A}}(\overline{a})\})| \leq c$, and $\emptyset$ otherwise.
- If $t = \{s : x \in r : \varphi\}_c$, then $t^{\mathbf{A}}(\overline{a}) = \{s^{\mathbf{A}}(\overline{a}, b) : b \in r^{\mathbf{A}}(\overline{a}) : \mathbf{A} \models \varphi(\overline{a}, b)\}$, if that set is in $\mathrm{HF}_c(A)$, and $\emptyset$ otherwise.
- If $\varphi = [\mathsf{dtc}_{\overline{u},\overline{v}}\psi]_c(\overline{r}, \overline{s})$, then $\mathbf{A} \models \varphi(\overline{a})$ if, and only if, there are $n \in \mathbb{N}$ and tuples $\overline{c}_0, \ldots, \overline{c}_n \in \mathrm{HF}_c(A)^k$ such that $\overline{c}_0 = \overline{r}^{\mathbf{A}}(\overline{a})$, $\overline{c}_n = \overline{s}^{\mathbf{A}}(\overline{a})$ and $\mathbf{A} \models \psi(\overline{c}_i, \overline{c}_{i+1})$ for all $i \leq n$.

Even though BDTC is strictly more expressive than DTC [57], it cannot define Even on padded structures and thus does not capture Logspace. For a more fine-grained analysis of the expressive power of BDTC, we show that it is actually included in the pure pointer language.

6.1.3 Bounded sets and pure pointer programs

Intuitively, the weakness of BDTC is that it can access only constantly many elements of the input structure at once. This corresponds to the idea of the pure pointer language (purple), which permits constantly many pointers to atoms, together with a means to iterate over all atoms. In contrast to the logical formalisms we study, purple enforces isomorphism-invariance by requiring that the computation has the same output for every enumeration of the atoms. So a classical algorithm evaluating a purple-program has to run the program for all linear orders on the input structure. Therefore, purple cannot be a logic for LOGSPACE in the sense of descriptive complexity, regardless of its expressive power.

Pure pointer programs are usually evaluated over *pointer-structures*, which are essentially relational structures with unary functions. But as we are only interested in a translation of BDTC, we only consider relational structures.

Similarly to logics with counting, purple-programs use different types of variables: Boolean variables take values from $\{0, 1\}$, and pointer variables are mapped to elements of the input structure. We denote Boolean variables by upper case letters X, Y, Z and pointer variables by lower case letters x, y, z.

Definition 6.6 (Syntax of purple-formulae)**.** In the following let σ be a relational signature.

- Every Boolean variable X is a formula.
- If $R \in \sigma$ is a relation symbol of arity r and $x_1, \ldots, x_r$ are pointer variables, then $Rx_1 \ldots x_r$ is a formula.
- If φ_1 and φ_2 are formulae, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are formulae.
- If x_1 and x_2 are pointer variables, then $x_1 = x_2$ is a formula.

Definition 6.7 (Syntax of purple-programs)**.** Pointers and formulae over the signature σ are combined into *programs* as follows:

- `skip` is a program.
- If P_1 and P_2 are programs, then $P_1; P_2$ is a program.
- If x_1, x_2 are pointer variables, then $x_1 \coloneqq x_2$ is a program.
- If X is a Boolean variable and φ is a formula, then $X \coloneqq \varphi$ is a program.
- If φ is a formula and P_1 and P_2 are programs, then `if` φ `then` P_1 `else` P_2 is a program.
- If x is a pointer variable and P is a program, then `forall` x `do` P is a program.

Let $\mathbf{A}$ be a σ-structure. An $\mathbf{A}$-configuration is a pair $I = (\rho, q)$, where the *pebbling* ρ maps pointer variables to A, and the *state* q maps Boolean variables to $\{0, 1\}$. The value $[\![\varphi]\!]^I$ of a formula φ is defined using the values of variables from ρ and q in the obvious way.

A program together with a σ-structure determines a successor relation on configurations as follows:

Definition 6.8 (Semantics of purple-programs)**.** Let $\mathbf{A}$ be a σ-structure, and let I, O be $\mathbf{A}$-configurations. For every purple-program P, we say that O is a successor configuration of I, or $P \vdash_{\mathbf{A}} I \longrightarrow O$, if one of the following rules applies:

- $\texttt{skip} \vdash_{\mathbf{A}} I \longrightarrow I$.
- $P_1; P_2 \vdash_{\mathbf{A}} I \longrightarrow O$ if $P_1 \vdash_{\mathbf{A}} I \longrightarrow R$ and $P_2 \vdash R \longrightarrow O$ for some configuration R.
- $x_1 := x_2 \vdash_{\mathbf{A}} (\rho, q) \longrightarrow (\rho[x_1 \mapsto \rho(x_2)], q)$.
- $X := \varphi \vdash_{\mathbf{A}} (\rho, q) \longrightarrow (\rho, q[X \mapsto [\![\varphi]\!]^{(\rho,q)}])$.
- $\texttt{if } \varphi \texttt{ then } P_1 \texttt{ else } P_2 \vdash_{\mathbf{A}} I \longrightarrow O$ if $[\![\varphi]\!]^I = 1$ and $P_1 \vdash_{\mathbf{A}} I \longrightarrow O$.
- $\texttt{if } \varphi \texttt{ then } P_1 \texttt{ else } P_2 \vdash_{\mathbf{A}} I \longrightarrow O$ if $[\![\varphi]\!]^I = 0$ and $P_2 \vdash_{\mathbf{A}} I \longrightarrow O$.
- $\texttt{forall } x \texttt{ do } P \vdash_{\mathbf{A}} I \longrightarrow O$ if there exist an enumeration $a_1, a_2, \ldots, a_n$ of A, as well as configurations $I = (\rho_1, q_1), (\rho_2, q_2), \ldots, (\rho_{n+1}, q_{n+1}) = O$, such that $P \vdash_{\mathbf{A}} (\rho_i[x \mapsto a_i], q_i)) \longrightarrow (\rho_{i+1}, q_{i+1})$ for all $1 \leq i \leq n$.

Let X_{res} be a distinguished Boolean variable. A program P *accepts* (resp. *rejects*) $\mathbf{A}$ if $P \vdash_{\mathbf{A}} (\rho, q) \longrightarrow (\rho', q')$ implies $q'(X_{\text{res}}) = 1$ (resp. $q'(X_{\text{res}}) = 0$) for all configurations $(\rho, q), (\rho', q')$. P *defines* a Boolean query $\mathcal{Q}$ if P accepts all σ-structures in $\mathcal{Q}$ and rejects all others.

Note that the requirement that all sequences of configurations are accepting makes sure that the output is independent of the enumeration of the atoms in `forall`-loops.

We will use a comparison to purple to show that a meaningful extension of BDTC has to change the formalism in more fundamental ways than just by adding a counting operation. To this end, we show that every BDTC-formula can be translated to a purple-program. This indicates that the extension purple$_{\text{C}}$ by counters, which can define the cardinality of all purple-definable relations [38], can be seen as (an extension of) BDTC with counting. Then it will follow that, even with counting, BDTC would not be able to define the tree isomorphism query.

Theorem 6.9 ([38])**.** *Tree isomorphism is not definable in* purple$_{\text{C}}$.

Careful analysis of the proof shows that it survives in the presence of padding, so the query is not purple$_C$-definable on padded structures either. To transfer results from purple to BDTC, we show

Lemma 6.10. *Every* BDTC*-definable query is definable by a* purple*-program.*

The main step of the translation is to represent any set in $\mathrm{HF}_c(A)$ using constantly many pointers. Recall that general hereditarily finite sets can be represented by a "template" for the set structure together with a sequence of atoms: A form and a molecule. Since the transitive closure of bounded sets contains only constantly many atoms, a molecule can now contain *all* atoms occurring anywhere in the transitive closure. This idea has been formalised by Spielmann [57] as presented in the following definitions.

Definition 6.11. Let $c \in \mathbb{N}$, and let $\mathrm{Slots} = \{1, \ldots, c+1\}$. A *molecule* is a c-tuple of atoms, and a *form* is an element of the set

$$\mathrm{Forms}_c = \{x \in \mathrm{Slots} \cup \mathrm{HF}_c(\mathrm{Slots}) : \emptyset \notin \mathrm{tc}(x)\}.$$

We denote by Forms, without the subscript, the set of all forms in $\mathrm{Slots} \cup \mathrm{HF}(\mathrm{Slots})$.

Definition 6.12. Let $F \in \mathrm{Forms}_c$ and let $m = (m_1, \ldots, m_c)$ be a molecule. The *value* $F \star m$ is defined by induction on the rank of F. If $1 \leq F \leq c$, then $F \star m = m_F$, and if $F = c+1$, then $F \star m = \emptyset$. If F is a set, then $F \star m = \{f \star m : f \in F\}$.

For fixed c, there are constantly many forms, so our purple-program can use a Boolean variable for every form in Forms_c, whereas a molecule can be represented as a sequence of c pointer variables. To show that this suffices, we establish the following correspondence between forms and molecules:

Lemma 6.13. *$a \in \mathrm{HF}_c(A)$ if, and only if, there is a form $F \in \mathrm{Forms}_c$ and a molecule $m \in A^c$ such that $F \star m = a$.*

Proof. Since $|\mathrm{tc}(F \star m)| \leq |\mathrm{tc}(F)|$, $F \star m \in \mathrm{HF}_c(A)$ for every $F \in \mathrm{Forms}_c$ and $m \in A^c$. Conversely, given $a \in \mathrm{HF}_c(A)$, a form in Forms_c can be obtained by replacing every occurrence of an atom or the empty set by a number in $\{1, \ldots, c+1\}$. Formally, we define the following mapping $\rho\colon \mathrm{HF}_c(A) \to \mathrm{Forms}_c$: Fix a linear order on A, and let $a_1, \ldots, a_c \in A$ with $a_1 < \ldots < a_c$. Then ρ maps every set $b \in \mathrm{HF}_c(A)$ with $\mathrm{tc}(b) \cap A \subseteq \{a_1, \ldots, a_c\}$ to a set in Forms_c by replacing every occurrence of a_i by i, and every occurrence of $\emptyset$ by $c+1$, throughout the transitive closure of b. By construction, $|\mathrm{tc}(b)| = |\mathrm{tc}(\rho(b))|$, so $\rho(b) \in \mathrm{Forms}_c$. Further $\rho(b) \star (a_1, \ldots, a_c) = b$, which follows from an easy induction on the rank of b. □

By Lemma 6.13, a constant number of forms suffices to represent all of $\mathrm{HF}_c(A)$. We use this fact to define a representation of forms with a bounded number of Boolean variables.

Definition 6.14. Let $\mathcal{X} = \{X_F : F \in \mathrm{Forms}_c\} \cup \{x_1, \ldots, x_c\}$, where $x_1, \ldots, x_c$ are pointer variables, and the X_F are Boolean variables. Then $\mathcal{X}$ *defines the bounded set* $a \in \mathrm{HF}_c(A)$ *in the configuration* (ρ, q), if, and only if, there is a form $F \in \mathrm{Forms}_c$ and a molecule $m = m_1, \ldots, m_c \in A^c$ such that $a = F \star m$ and

- $\rho(x_i) = m_i$ for $1 \leq i \leq c$,
- $q(X_F) = 1$ and $q(X_{F'}) = 0$ for all $F \neq F' \in \mathrm{Forms}_c$.

With this representation of bounded sets, we can formalise what it means for a purple-program to simulate a BDTC-formula with free variables.

Definition 6.15.

- Let t be a BDTC-term with free variables $x_1, \ldots, x_k$, and let $c \in \mathbb{N}$ be the maximal bound occurring as a subscript of any subterm of t. A purple-program P *simulates* t if, for every σ-structure $\mathbf{A}$ and every configuration $I = (\rho, q)$ such that $\mathcal{X}_1, \ldots, \mathcal{X}_k$ represent $a_1, \ldots, a_k \in \mathrm{HF}(A)$ in I, if $P \vdash_{\mathbf{A}} I \longrightarrow O$, then $\mathcal{X} = \{X^t_F : F \in \mathrm{Forms}_c\} \cup \{x^t_1, \ldots, x^t_c\}$ represents $[\![t(a_1, \ldots, a_k)]\!]^{\mathbf{A}}$ in O.
- Let φ be a BDTC-formula with free variables $x_1, \ldots, x_k$, and let $c \in \mathbb{N}$ be the maximal bound in any subterm of φ. A purple-program P *simulates* φ if, for every σ-structure $\mathbf{A}$ and for every configuration $I = (\rho, q)$ such that $\mathcal{X}_1, \ldots, \mathcal{X}_k$ represent $a_1, \ldots, a_k \in \mathrm{HF}(A)$ in I, if $P \vdash_{\mathbf{A}} I \longrightarrow O$, then, in O, the Boolean pointer X^φ is mapped to 1 if, and only if, $\mathbf{A} \models \varphi(a_1, \ldots, a_k)$.

Then Lemma 6.10 follows from

Lemma 6.16. *For every* BDTC*-term* t *(resp. formula* φ*), there is a* purple*-program* P^t *(resp* P^φ*) simulating* t *(resp.* φ*).*

The representation of bounded sets using forms and molecules is not unique. Since some steps of the simulation require comparing whether two combinations of forms and molecules represent the same set, we first construct purple-programs for that purpose. Note that, since the number of forms is bounded, these programs can be defined with a pair of forms as a parameter.

Lemma 6.17. *For every pair of forms* $F_1, F_2 \in \mathrm{Forms}_c$ *and distinguished variables* $X^=, X^\in, x_1, \ldots, x_c, y_1, \ldots, y_c$*, there are* purple*-programs* $P^=_{F_1,F_2}$ *and* $P^\in_{F_1,F_2}$ *such that, for all configurations* $I = (\rho_I, q_I)$ *and* $O = (\rho_O, q_O)$*,*

- ◇ *If* $P^{=}_{F_1,F_2} \vdash_{\mathbf{A}} I \longrightarrow O$, *then* $F_1 \star (\rho_I(x_1), \ldots, \rho_I(x_c)) = F_2 \star (\rho_I(y_1), \ldots, \rho_I(y_c))$ *if, and only if,* $q_O(X^{=}) = 1$,
- ◇ *If* $P^{\in}_{F_1,F_2} \vdash_{\mathbf{A}} I \longrightarrow O$, *then* $F_1 \star (\rho_I(x_1), \ldots, \rho_I(x_c)) \in F_2 \star (\rho_I(y_1), \ldots, \rho_I(y_c))$ *if, and only if,* $q_O(X^{\in}) = 1$.

Proof. We construct $P^{=}$ and $P^{\in}$ simultaneously by induction on the maximal rank of F_1 and F_2.

If both F_1 and F_2 have rank 0, then $P^{\in}_{F_1,F_2}$ is just the assignment $X^{\in} \coloneqq 0$. For $P^{=}$, we distinguish several cases. If $F_1 = F_2 = c + 1$, then $P^{=}_{F_1,F_2}$ is $X^{=} \coloneqq 1$. If $F_1 = i$ and $F_2 = j$ for $i, j \in \{1, \ldots, c\}$, let $P^{=}_{F_1,F_2}$ be

$$\texttt{if } x_i = x_j \texttt{ then } X^{=} \coloneqq 1 \texttt{ else } X^{=} \coloneqq 0.$$

Otherwise, $P^{=}_{F_1,F_2}$ is the assignment $X^{=} \coloneqq 0$.

In the induction step, first suppose $\mathrm{rk}(F_1) < \mathrm{rk}(F_2)$ (otherwise, $P^{\in}_{F_1,F_2}$ always sets $X^{\in}$ to 0). Then $\mathrm{rk}(F_2) > 0$. By the induction hypothesis, there is a program $P^{=}_{F_1,f}$ for every $f \in F_2$. Since F_2 has constantly many elements, $P^{\in}_{F_1,F_2}$ can run the program for every $f \in F_2$ and set $X^{\in}$ to 1 if, and only if, one of them outputs equality.

For the program $P^{=}_{F_1,F_2}$, suppose $\mathrm{rk}(F_1) = \mathrm{rk}(F_2) = r$ (otherwise, the program always outputs inequality). Then F_1 and F_2 have the same value if, and only if, all programs $P^{\in}_{f,F_1}$ for $f \in F_2$ and $P^{\in}_{f,F_2}$ for $f \in F_1$ set their variable $X^{\in}$ to 1. Note that the programs for F_1 and F_2 have already been constructed since $\mathrm{rk}(f) < r$ for $f \in F_1 \cup F_2$. □

Because of the way we represent bounded sets in configurations, a purple-program can iterate over all bounded sets: The set of forms is of constant size and can thus be enumerated explicitly, and every molecule is a c-tuple of domain elements, over which the program can iterate with a `forall`-loop. This observation is the main ingredient for the simulation of dtc-subformulae as well as comprehension terms and terms of the form $\{s_1, \ldots, s_k\}_c$. The dtc-operator in BDTC ranges over $\mathrm{HF}_c(A)$, so the simulating program iterates over all objects in $\mathrm{HF}_c(A)$ to check which one is the unique successor.

When simulating terms, the programs simulating the immediate subterms output arbitrary representations as forms and molecules. Combining these representations may yield a molecule of size $> c$, and a form which is itself not a bounded set. So, instead of combining previously computed forms and molecules, our program will

use $P^{=}$ and $P^{\in}$ from the previous lemma to check for *every* form and molecule whether their value is the desired value of the term.

Proof of Lemma 6.16. We construct the simulating programs by induction on terms and formulae. If $t = \emptyset$, then P^t consists of the assignments $X^t_{c+1} := 1$ and $X^t_F := 0$ for all $F \neq c+1$.

If $t = x_i$, let $\mathcal{X}_i = \{x^i_1, \dots, x^i_c\} \cup \{X^i_F : F \in \mathrm{Forms}_c\}$ be the variables representing a_i. Then P^t is the concatenation of assignments $X^t_F := X^i_F$ for all $F \in \mathrm{Forms}_c$ and $x^t_j := x^i_j$ for $1 \leq j \leq c$.

If $t = \mathrm{Unique}(s)$, first run P^s, which will set some X^s_F to 1. Then, for all $f_1, f_2 \in F$, run $P^{=}_{f_1,f_2}$ to check if the value of s is a singleton. If this is the case, set $X^t_{f_1} := 1$ for the first form $f_1 \in F$, and set $x^t_i := x^s_i$ for all $1 \leq i \leq c$. In case the value of s is not a singleton, set $X^t_{c+1} := 1$.

For the case $t = \{s_1, \dots, s_k\}_c$, note that the programs $P^{s_1}, \dots, P^{s_k}$ might compute forms with distinct molecules for the values of $s_1, \dots, s_k$. But P^t should compute a single, sufficiently small form and molecule for the value of t. Therefore, P^t iterates over all forms and molecules and uses programs of the form $P^{=}_{F_1,F_2}$ and $P^{\in}_{F_1,F_2}$ to check whether the current form and molecule represents the set $s^{\mathbf{A}}_1, \dots, s^{\mathbf{A}}_k$.

Since the number of forms is constant, a purple-program can iterate over all forms with the concatenation of programs P_F for all $F \in \mathrm{Forms}_c$. Further, the size of molecules is bounded by c, so a sequence of `forall`-loops can iterate over all molecules.

For $t = \{s(x) : x \in r : \varphi(x)\}_c$, the idea is the same as for the previous case. Now P^t first runs P^r to obtain a form F_r and molecule m_r for the value of r. Then, for every form F and molecule m, P^t iterates over the elements of $t^{\mathbf{A}}$ and uses $P^{\in}$ to check whether they are contained in $F \star m$. Conversely, for every form $f \in F$, P^t searches for an element of $t^{\mathbf{A}}$ that is equal to $f \star m$. The iteration over elements of $t^{\mathbf{A}}$ is possible by running P^{φ} and P^s for every form $f \in F_r$ together with the molecule m_r.

For a formula $\varphi = Rs_1 \dots s_k$ with $R \in \sigma$, run $P_{s_1}, \dots, P_{s_k}$. If the resulting forms are in $\{1, \dots, c\}$, check whether their values form a tuple in $R^{\mathbf{A}}$.

Formulae $s \in t$ and $s = t$ can be simulated with programs of the form $P^{=}_{F_1,F_2}, P^{\in}_{F_1,F_2}$ for forms F_1, F_2 computed by P^s, P^t.

If $\varphi = \psi \vee \vartheta$ or $\varphi = \neg\psi$, run P_ψ and, in the first case, P_ϑ, and then assign $X^\varphi := X^\psi \vee X^\vartheta$ or $X^\varphi := \neg X^\psi$, respectively.

For $\varphi = [\mathsf{dtc}_{\overline{u},\overline{v}}\psi]_c(\overline{r}, \overline{s})$, first run P^{r_i} for every term r_i in $\overline{r}$ to obtain the initial value of $\overline{u}$. Whenever the value $\overline{s}$ is reached (computed by programs P_{s_i} for entries

s_i of $\overline{s}$) in this or one of the following steps, halt and set $X^{\varphi} := 1$. One step of the dtc-computation consists of iterating over all forms and molecules (as described in previous cases) and checking whether the result satisfies φ. If there is a unique value (determined by a program $P^{=}_{F_1,F_2}$) satisfying φ, continue with the next step. To guarantee termination, the whole dtc-computation is performed within a `forall`-loop over all k-tuples, where k is the length of $\overline{u}$. □

So we have established an upper bound for the expressive power of BDTC.

6.1.4 L-Recursion

None of the logics we have covered so far can define the tree isomorphism query. So their recursion mechanisms (if present) are too weak to capture LOGSPACE. The logic LREC, introduced by Grohe, Grußien, Hernich and Laubner [32, 34], tackles this weakness with a recursion operator that captures the kind of recursion that should intuitively be possible in LOGSPACE.

To see this, consider the recursive traversal of a DAG $G = (V, E)$. Assume the subcomputation at every vertex contributes to the result for all of its parents. So the algorithm maintains, for every vertex that is currently processed, the parent to which the result is passed when the subcomputation is complete. Altogether, the whole path that was taken to reach the current vertex has to be traceable to the root. If a vertex has a unique parent, then nothing has to be stored (this is used in Lindell's [48] tree canonisation algorithm). But for a vertex with k parents, one can store every parent using at most $\log k$ bits. So whenever the sum of the logarithms of the out-degrees—that is, the product of the out-degrees themselves—is small enough, then the whole path can be stored in logarithmically many bits.

The lrec-operator allows to traverse certain DAGs in that way, with the guarantee that not only the paths to the root, but also all intermediate results can be stored in logarithmic space. An lrec-subformula can be seen as an interpretation defining a graph over tuples of the input structure. That graph is unfolded to a DAG by annotating every vertex with a resource $\ell \in \mathbb{N}$. For every edge (a, b) in the graph induced by the interpretation, there is an edge $((a, \ell), (b, \ell'))$ in the DAG, where the new resource ℓ' is obtained from ℓ by dividing by the number of parents of b in the graph. This makes sure that the number of parents to be stored on every path is sufficiently small.

An lrec-formula φ induces a set X of pairs (a, ℓ) satisfying φ. Membership in the set X depends on the *label* of the vertex a in the original graph—a set of natural

numbers defined by another subformula of the interpretation. The label specifies the number of children of (a, ℓ) that have to be in X for (a, ℓ) itself to be added to X. This counting property defining membership in X guarantees that intermediate results can be stored in logarithmic space during the recursive computation.

We directly define the stronger variant of LREC, called $\mathrm{LREC}_=$ in the literature. The difference is that the weak variant, which we denote by LREC^-, is not closed under interpretations. This is the case because the recursion operator in LREC^- cannot be applied to structures over equivalence classes of tuples. In the stronger variant, the interpretation defining the graph for the recursion operator possesses a non-trivial equality formula. For the graph to be well-defined, the operator *implicitly* computes the symmetric transitive closure of the relation defined by the equality formula. As a side effect, LREC can express symmetric transitive closures.

LREC is a logic with counting terms, i. e. it is evaluated over two-sorted structures, and the variables can be domain or number variables. In contrast to the logics with counting terms we have introduced before, LREC possesses an operation for counting tuples as an explicit part of the syntax. Formally, it is an extension of the logic $\mathrm{FO{+}C_{tup}}$, which adds to FO+C exactly that operation.

We say that tuples $u_1, \ldots, u_k$ and $v_1, \ldots, v_k$ of variables are *compatible* if, for $1 \leq i \leq k$, u_i and v_i have the same type.

Definition 6.18 (Syntax of LREC)**.** LREC is obtained from $\mathrm{FO{+}C_{tup}}$ by adding the following rule for defining formulae:

If $\overline{u} = u_1, \ldots, u_k, \overline{v} = v_1, \ldots, v_k, \overline{x} = x_1, \ldots, x_k$ are compatible tuples of variables, $\overline{\lambda} = \lambda_1, \ldots, \lambda_m, \overline{\mu} = \mu_1, \ldots, \mu_m$ are non-empty tuples of number variables, and $\varphi_=, \varphi_E$ and φ_C are LREC-formulae, then

$$\varphi = [\mathsf{lrec}_{\overline{u},\overline{v},\overline{\lambda}} \varphi_=, \varphi_E, \varphi_C](\overline{x}, \overline{\mu})$$

is an LREC-formula. To define the free variables of φ, let $U = \{u_1, \ldots, u_k\}$, $V = \{v_1, \ldots, v_k\}$, $X = \{x_1, \ldots, x_k\}$, $L = \{\lambda_1, \ldots, \lambda_m\}$ and $M = \{\mu_1, \ldots, \mu_k\}$. Then the free variables of φ are $(\mathrm{free}(\varphi_=) \setminus (U \cup V)) \cup (\mathrm{free}(\varphi_E) \setminus (U \cup V)) \cup (\mathrm{free}(\varphi_C) \setminus (U \cup L)) \cup X \cup M$.

In the formula $\varphi = [\mathsf{lrec}_{\overline{u},\overline{v},\overline{\lambda}} \varphi_=, \varphi_E, \varphi_C](\overline{x}, \overline{\mu})$, $\varphi_=$, φ_E and φ_C are considered as an interpretation defining the graph labelled by sets of natural numbers described above. Note that the DAG unfolding does not take place at this point. To define the interpreted graph, let $\mathbf{A}$ be a σ-structure, where σ is the vocabulary of φ.

For ease of notation, we only mention the values of the free variables $\overline{u}, \overline{v}, \overline{x}, \overline{\lambda}$ and $\overline{\mu}$ in the subformulae, even though the formulae may have additional free variables.

If v is a domain variable, then $A^v = A$, and if λ is a number variable, then $A^\lambda = N(A)$. Let $V_0 = A^{x_1} \times \cdots \times A^{x_k}$, and $E_0 = \{(\overline{a}, \overline{b}) \in V_0^2 : \mathbf{A} \models \varphi_E(\overline{a}, \overline{b})\}$. Further, let $\sim$ be the reflexive symmetric transitive closure of $\{(\overline{a}, \overline{b}) \in V_0^2 : \mathbf{A} \models \varphi_=(\overline{a}, \overline{b})\}$. Then our graph is $G = (V, E)$ with $V = V_{0/\sim}$ and $E = \{([\overline{a}], [\overline{b}]) \in V^2 : (\overline{a}, \overline{b}) \in E_0\}$. We abbreviate $[\overline{a}]_\sim$ by $[\overline{a}]$. Every vertex $[\overline{a}] \in V$ is labelled by the set

$$C([\overline{a}]) = \{\mathrm{enc}(\overline{n}) : \text{there is an } \overline{a}' \in [\overline{a}] \text{ with } \mathbf{A} \models \varphi_C(\overline{a}', \overline{n})\}.$$

Recall that enc is the function interpreting tuples as single numbers.

To evaluate the formula φ, the interpreted graph is implicitly unfolded to a DAG over vertices in $V \times \mathbb{N}$. A variable assignment satisfies φ if, and only if, it encodes a DAG vertex in the set X defined recursively as follows: A DAG vertex $([\overline{a}], \ell)$ is in X if, and only if, it is not a leaf and the number of children in X is contained in the label $C([\overline{a}])$, formally:

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

Decreasing the number ℓ keeps the paths in the DAG sufficiently short.

Definition 6.19 (Semantics of lrec-formulae)**.** Let $\mathbf{A}$, φ and X as above, $\overline{a} \in A^{x_1} \times \cdots \times A^{x_k}$, and $\overline{\ell} \in N^m$. Then $\mathbf{A} \models \varphi(\overline{a}, \overline{\ell})$ if, and only if, $(\overline{a}, \mathrm{enc}(\overline{\ell})) \in X$.

Even the weaker variant of LREC, without the formula $\varphi_=$, can define the tree isomorphism query. In fact, it captures Logspace on the class of directed trees [32, 34]. As is also shown in [32, 34], the strong version of LREC considered here captures Logspace on the class of interval graphs. Further, the formula $\varphi_=$ implicitly defines symmetric transitive closures, so LREC also includes STC+C.

On the class of all structures, however, it does not capture Logspace, because LREC $\leq$ FP+C [34]. This is explained in more detail in Section 6.4, when we use that fact to separate our logic from LREC.

6.2 Definition of Choiceless Logspace

As explained in the introduction of this chapter, our logic is based on the idea that an element of a structure can have different sizes, depending on how it is represented. So terms and formulae are evaluated over hereditarily finite sets

together with functions that assign a size to every element of the transitive closure of the respective set. Before we define the syntax and semantics of the different kinds of terms and formulae, we establish how the sizes of sets are defined.

Size annotations A size annotation is a function that defines the size of a hereditarily finite set depending on the sizes of its elements. We define these functions in a way that the size of a set corresponds to the size of its encoding on a Turing machine. To encode the set $\{a_1, \dots, a_k\}$, one would write down all its elements $a_1, \dots, a_k$ and include some signifier, for instance delimiters "{" and "}", to denote the set. This signifier can be represented with a constant number of bits. Ignoring constant factors, this leads to the following definition of the sizes of sets:

Definition 6.20. Let $\mathbf{A}$ be a σ-structure and $a \in \mathrm{HF}(A)$. A function $s\colon \mathrm{tc}(a) \to \mathbb{N}$ is a *size annotation of* a if, for each set $b \in \mathrm{tc}(a)$, $s(b) = 1 + \sum_{c \in b} s(c)$.

The set of *size-annotated hereditarily finite objects* is

$$\mathrm{SHF}(A) \coloneqq \{(a, s_a) : a \in \mathrm{HF}(A) \text{ and } s_a \text{ is a size annotation of } a\}.$$

For size annotations $(a, s_a), (b, s_b)$, we say that $(b, s_b) \in (a, s_a)$ if $b \in a$ and $s_b = s_a{\restriction}\,\mathrm{tc}(b)$.

Note that a size annotation is completely determined by the values it assign to the atoms, so the following holds.

Remark 6.21. *If* $\mathrm{tc}(a) \cap A = \emptyset$ *for a set* $a \in \mathrm{HF}(A)$*, then it has a unique size annotation. We denote that size annotation by* ann_a.

Further, since every set in the transitive closure of a set contributes the summand 1 to any size annotation, the size annotation yields an upper bound for the size of the transitive closure. As atoms may have size 0, we obtain the following bound:

Remark 6.22. *Let* $s\colon \mathrm{tc}(a) \to \mathbb{N}$ *be a size annotation of the set* $a \in \mathrm{HF}(A)$*. Then* $|\,\mathrm{tc}(a) \setminus A| \leq s(a)$.

Size annotations are a crucial part of the semantics of terms and formulae. The size of a value of a term is usually derived from the sizes of its subterms' values. We now define how to combine size annotations in a way that reflects the intention to represent every atom in the smallest known manner.

Definition 6.23. Let $\mathcal{B} \subseteq \mathrm{SHF}(A)$ be a set of size-annotated hereditarily finite sets. We denote by $\mathcal{B}_{\mathrm{SHF}}$ the pair $(b, \mathrm{ann}_b^{\min})$ where $b = \{b' : (b', \mathrm{ann}_{b'}^{\min}) \in \mathcal{B}\}$ and $\mathrm{ann}_b^{\min}$ is the unique size annotation of b with $\mathrm{ann}_b^{\min}(a) = \min\{\mathrm{ann}_{b'}^{\min}(a) : (b', \mathrm{ann}_{b'}^{\min}) \in \mathcal{B}\}$ for every atom $a \in \mathrm{tc}(b)$.

Choiceless Logarithmic Space Recall that CPT is a PTIME-restriction of the underlying logic BGS. We now give a high-level definition of the logic CLogspace as the LOGSPACE-restriction of a certain underlying logic, and then proceed to make precise, and explain, the ingredients of that logic step by step.

Definition 6.24 (Choiceless Logarithmic Space)**.** The logic CLogspace is defined by rules for ordinary terms and formulae, iteration terms and recursion formulae as follows. If t is a term and φ is a formula according to these rules, and f is a function $f : n \mapsto c \log n$ for $c \in \mathbb{N}$, then (t, f) is a CLogspace-term, and (φ, f) is a CLogspace-formula.

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

Syntax of ordinary terms and formulae The basis for the terms and formulae in CLogspace are the set-theoretic operations that are also possible with ordinary BGS-terms. These are later extended by operators for iteration and recursion. Recall that the size of an atom will depend on a logical formula defining it, so, instead of the BGS-term Atoms, we introduce terms of the form Atoms $.\varphi$ guarded by an FO-formula φ.

Definition 6.25 (Syntax)**.** Let σ be a vocabulary. The set of ordinary σ-terms and σ-formulae is defined inductively as follows:

- $\emptyset$ is a term.
- Every variable x is a term.
- If $\varphi \in \mathrm{FO}[\sigma]$ has at least one free variable, then Atoms $.\varphi$ is a term.
- If r, s are terms, then Union(s), Unique(s) and Pair(r, s) are terms.
- If s is a term, then Card(s) is a term.
- If $R \in \sigma$ is of arity k and $s_1, \ldots, s_k$ are terms, then $Rs_1 \ldots s_k$ is a formula.
- If s and t are terms, then $s = t$ and $s \in t$ are formulae.
- Boolean combinations of formulae are formulae.
- If r and s are terms and φ is a formula, then $\{s : x \in r : \varphi\}$ is a (comprehension) term.

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

The value of terms of the form $\mathrm{Card}(s)$ is a number, and the value of $\mathrm{Atoms}.\varphi$ is a tuple which is indexed by numbers. So, before we define the semantics of ordinary terms, we introduce a way of representing natural numbers such that every number that can be represented in logarithmic space corresponds to a set with a logarithmic size annotation.

Counting In CPT, the value of a term $\mathrm{Card}(t)$ is a von Neumann ordinal. So the number n is a set with n elements. In LOGSPACE it is however possible to compute the cardinality of sets of polynomial size. Therefore cardinalities will be denoted by the following set encoding of their binary representation:

Definition 6.26. We define the function $\mathrm{bitset}\colon \{0,1\}^+ \to \mathrm{HF}(\emptyset)$ with the rules

- $\mathrm{bitset}(0) = \emptyset$,
- $\mathrm{bitset}(1) = \{\emptyset\}$,
- $\mathrm{bitset}(bw) = \langle \mathrm{bitset}(b), \mathrm{bitset}(w)\rangle$ for $b \in \{0,1\}$ and $w \in \{0,1\}^+$.

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

Recall that we use the definition $\langle a, b\rangle = \{a, \{a, b\}\}$ of ordered pairs. This makes sure that every word of length two is represented by a set with two elements and can thus be distinguished from $\mathrm{bitset}(0)$ and $\mathrm{bitset}(1)$. It follows that the function bitset is injective.

The purpose of this representation of numbers is to keep their size logarithmic (with respect to any size annotation). The following lemma shows that the size of $\mathrm{bitset}(w)$ is always linear in the word length $|w|$:

Lemma 6.27. *Let* $s\colon \mathrm{tc}(\mathrm{bitset}(w)) \to \mathbb{N}$ *be a size annotation for* $w \in \{0,1\}^+$. *Then* $s(\mathrm{bitset}(w)) \leq 6|w|$.

Proof. Proof by induction on $|w|$. For $w \in \{0,1\}$, $s(\mathrm{bitset}(w)) \leq 2 \leq 6|w|$.

Now let $w \in \{0,1\}^+$ and $b \in \{0,1\}$. Then

$$\begin{aligned} s(\mathrm{bitset}(bw)) &= s(\{\mathrm{bitset}(b), \{\mathrm{bitset}(b), \mathrm{bitset}(w)\}\}) \\ &= s(\mathrm{bitset}(w)) + 2s(\mathrm{bitset}(b)) + 2. \end{aligned}$$

Since $s(\mathrm{bitset}(b)) \leq 2$, it follows that $s(\mathrm{bitset}(bw)) \leq s(\mathrm{bitset}(w)) + 6 \leq 6|bw|$. □

Semantics of ordinary terms and formulae The main idea that differentiates our formalism from other logics is that it is evaluated over size-annotated hereditarily finite sets. This means that every term defines not just a hereditarily finite set, but also a size annotation of that set. For this size annotation to be well-defined, the values of the free variables must in turn be equipped with sizes.

But where does the size of an atom originate if it is not defined externally through a variable assignment? This is where the operators $\mathrm{Atoms}\,.\varphi$ come into play.

Recall that we assume every atom to be represented as "the ith entry of the kth tuple satisfying φ". Both φ and i are of constant size, since φ is part of the term. So we can assume that, up to addition with a constant, such an atom can be written as a number with $\log k$ bits. The number k is bounded by the number m of tuples satisfying φ, so the order-independent size of every atom defined like that is $\log m$.

For other terms, the size annotations obtained from the subterms are combined according to Definition 6.23 to reflect the intention to represent every set in the smallest possible way.

For the evaluation of terms and formulae in Logspace in Section 6.4.2, it will be shown that these size annotations indeed correspond to the sizes of the representations occurring in the evaluation algorithm.

Formally, a term or formula is evaluated over a structure $\mathbf{A}$ together with an assignment mapping the free variables to pairs in $\mathrm{SHF}(A)$. Analogously to simpler assignments, if $\beta\colon X \to \mathrm{SHF}(A)$ is a variable assignment and $\alpha \in \mathrm{SHF}(A)$, we write $\beta[x \mapsto \alpha]\colon X \cup \{x\} \to \mathrm{SHF}(A)$ to denote the function that behaves like β on $X \setminus \{x\}$ and maps x to α.

Definition 6.28. Let $\mathbf{A}$ be a σ-structure and let $\beta\colon X \to \mathrm{SHF}(A)$ be a variable assignment. For an ordinary σ-term t (with $\mathrm{free}(t) \subseteq X$), we define the value $t^{\mathbf{A},\beta} = (\llbracket t \rrbracket^{\mathbf{A},\beta}, \mathrm{ann}_t^{\mathbf{A},\beta}) \in \mathrm{SHF}(A)$ as follows:

- If $t = x$ for a variable x, then $t^{\mathbf{A},\beta} = \beta(x)$.
- $\emptyset^{\mathbf{A},\beta} = (\emptyset, \emptyset \mapsto 1)$.
- For $t = \mathrm{Atoms}\,.\varphi$ with $\varphi \in \mathrm{FO}[\sigma]$, let $\llbracket t \rrbracket^{\mathbf{A},\beta} = \{a : \mathbf{A} \models \varphi(a)\}$ if $|\,\mathrm{free}(\varphi)| = 1$, and

$$\llbracket t \rrbracket^{\mathbf{A},\beta} = \{\{\langle \mathrm{bitset}(1), a_1\rangle, \dots, \langle \mathrm{bitset}(k), a_k\rangle\} : \mathbf{A} \models \varphi(a_1, \dots, a_k)\}$$

  if $|\,\mathrm{free}(\varphi)| > 1$. In both cases $\mathrm{ann}_t^{\mathbf{A},\beta}$ is the unique size annotation mapping each $a \in A \cap \mathrm{tc}(\llbracket t \rrbracket^{\mathbf{A},\beta})$ to $\log |\varphi^{\mathbf{A}}|$.
- If $t = \mathrm{Pair}(r, s)$, then $t^{\mathbf{A},\beta} = \left\{r^{\mathbf{A},\beta}, s^{\mathbf{A},\beta}\right\}_{\mathrm{SHF}}$.

- If $t = \mathrm{Unique}(s)$, then

$$t^{\mathbf{A},\beta} = \begin{cases} (a, \mathrm{ann}_s^{\mathbf{A},\beta} \restriction \mathrm{tc}(a)), & \text{if } [\![s]\!]^{\mathbf{A},\beta} = \{a\}, \\ (\emptyset, \emptyset \mapsto 1), & \text{otherwise.} \end{cases}$$

- If $t = \mathrm{Union}(s)$, then $[\![t]\!]^{\mathbf{A},\beta} = \bigcup_{b \in [\![s]\!]^{\mathbf{A},\beta}} b$ and $\mathrm{ann}_t^{\mathbf{A},\beta} = \mathrm{ann}_s^{\mathbf{A},\beta} \restriction \mathrm{tc}([\![t]\!]^{\mathbf{A},\beta})$.
- If $t = \mathrm{Card}(s)$, then $[\![t]\!]^{\mathbf{A},\beta} = \mathrm{bitset}\left(\left|[\![s]\!]^{\mathbf{A},\beta}\right|\right)$, and $\mathrm{ann}_t^{\mathbf{A},\beta} = \mathrm{ann}_{[\![t]\!]^{\mathbf{A},\beta}}$.
- If $t = \{s : x \in r : \varphi\}$, then

$$t^{\mathbf{A},\beta} = \{s^{\mathbf{A},\beta[x \mapsto a]} : a \in r^{\mathbf{A},\beta} : \mathbf{A}, \beta[x \mapsto a] \models \varphi\}_{\mathrm{SHF}}.$$

For a σ-formula φ (again with $\mathrm{free}(\varphi) \subseteq X$), we say that $\mathbf{A}, \beta \models \varphi$ if, and only if, one of the following rules applies:

- $\mathbf{A}, \beta \models Rs_1 \dots s_k$ if, and only if, $([\![s_1]\!]^{\mathbf{A},\beta}, \dots, [\![s_k]\!]^{\mathbf{A},\beta}) \in R^{\mathbf{A}}$.
- $\mathbf{A}, \beta \models s \in t$ if, and only if, $[\![s]\!]^{\mathbf{A},\beta} \in [\![t]\!]^{\mathbf{A},\beta}$.
- $\mathbf{A}, \beta \models s = t$ if, and only if, $[\![s]\!]^{\mathbf{A},\beta} = [\![t]\!]^{\mathbf{A},\beta}$.
- The semantics of $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\neg\varphi$ is defined as usual.

We denote the size $\mathrm{ann}_t^{\mathbf{A},\beta}([\![t]\!]^{\mathbf{A},\beta})$ assigned to the value of a term t by $\|t\|^{\mathbf{A},\beta}$.

The term Atoms without an FO-formula, as defined in BGS, can be expressed as $\mathrm{Atoms}\,.x = x$. We also write true instead of $x = x$, and true^k to denote a tautology with k free variables. Then $\mathrm{Atoms}\,.\,\mathrm{true}^k$ defines the set of all k-tuples over the input structure.

Iteration terms As in Choiceless Polynomial Time, hereditarily finite sets only increase expressive power if they can be defined iteratively (we show this formally in Section 6.4.3). Also in analogy to CPT, the stages of the iteration will be subject to a logarithmic bound.

Nevertheless, because of the intuition given at the beginning of this chapter, the value of an iteration term may be a set of polynomial size. This is achieved by accumulating every stage into the value. Further, the initial stage of every iteration is given by a free variable, which permits iterations for polynomially many values in parallel.

Definition 6.29 (Syntax of iteration terms)**.** If u and x are variables and s is a term, then $[s_u]^*(x)$ is an *iteration term* with free variables $\{x\} \cup \mathrm{free}(s) \setminus \{u\}$.

To define the semantics of iteration terms, we define the i-fold iteration s^i of the subterm s of $[s_u]^*(x)$ by induction as $s^0 = s[u/x]$ and $s^{i+1} = s[u/s^i]$. For any term r, $s[u/r]$ is the term that results from replacing every occurrence of u in s by r.

Definition 6.30 (Semantics of iteration terms)**.** Let (t, f) be a CLogspace-term such that $t = [s_u]^*(x)$ is an iteration term with vocabulary σ, let $\mathbf{A}$ be a σ-structure and $\beta\colon \mathrm{free}(t) \to \mathrm{SHF}(A)$ a variable assignment. Then

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

Note that whenever the logarithmic bound is satisfied by every stage up to the fixed point, all stages contribute to the value of the iteration term.

Recursion formulae With recursion formulae, we basically add to our logic the lrec-operator from the logic LREC. But, since our logic can handle hereditarily finite sets instead of just elements of the input structure, we adjust the operator accordingly.

Recall that the subformulae of an lrec-formula interpret a graph in the input structure. We extend this interpretation to allow arbitrary definable hereditarily finite sets as elements of the graph.

But then, to algorithmically construct that graph, one would have to check, for all hereditarily finite sets, whether they satisfy the edge-formula. So we add to the interpretation a domain term t_δ restricting the domain of the interpreted structure, and thus the objects for which the remaining formulae have to be evaluated. The term t_δ also determines the size annotation of every vertex of the graph.

Further, LREC is defined over two-sorted structures, and the vertex labels are sets of tuples over the number sort. In our setting, the labels are encoded with the bitset-representation to match our encoding of numbers. Note that, if the labels were defined by a formula, an algorithm would again have to check for arbitrarily large bitsets whether they satisfy the formula. Instead, we define the label using a term, with the semantics that the label consists of exactly those elements of the term's value that are bitset representations.

Definition 6.31 (Syntax of recursion formulae)**.** If t_δ and t_C are terms and φ_E and $\varphi_=$ are formulae, then $[\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x, y)$ is a *recursion formula*.

Since tuples of fixed length can be defined as sets, it suffices to consider single variables x, y instead of tuples, which are used in the original setting.

For the definition of the semantics, fix a σ-structure $\mathbf{A}$, a variable assignment β, and a recursion formula $\varphi = [\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x, y)$.

We first describe how t_δ, $\varphi_=$, φ_E and t_C define the labelled graph. The formulae $\varphi_=$ and φ_E are evaluated for every pair of elements from $[\![t_\delta]\!]^{\mathbf{A},\beta}$. For that purpose, we define the extension $\beta_{a,b} \coloneqq \beta[u \mapsto (a, \operatorname{ann}_{t_\delta}^{\mathbf{A},\beta} \restriction \operatorname{tc}(a)), v \mapsto (b, \operatorname{ann}_{t_\delta}^{\mathbf{A},\beta} \restriction \operatorname{tc}(b))]$ of β for every $a, b \in [\![t_\delta]\!]^{\mathbf{A},\beta}$.

The vertex set V consists of the equivalence classes of the relation $\sim \subseteq [\![t_\delta]\!]^{\mathbf{A},\beta} \times [\![t_\delta]\!]^{\mathbf{A},\beta}$, which is defined as the reflexive symmetric transitive closure of the relation $\{(a, b) \in [\![t_\delta]\!]^{\mathbf{A},\beta} \times [\![t_\delta]\!]^{\mathbf{A},\beta} : \mathbf{A}, \beta_{a,b} \models \varphi_=\}$. We write $[a]$ instead of $[a]_\sim$ to denote the congruence class of a. The formula φ_E defines the edge set of the graph as $E \coloneqq \{([a], [b]) : \mathbf{A}, \beta_{a,b} \models \varphi_E\}$. Additionally, each vertex of V is labelled by the set $C([a]) = \{c : \text{ there exists } a' \in [a] \text{ with } \operatorname{bitset}(c) \in [\![t_C]\!]^{\mathbf{A},\beta_{a'}}\}$, where $\beta_{a'} = \beta[u \mapsto (a', \operatorname{ann}_{t_\delta}^{\mathbf{A},\beta} \restriction \operatorname{tc}(a'))]$.

The truth value of the recursion formula φ is defined via the set $X \subseteq V \times \mathbb{N}$, analogously to the lrec-operator. A pair $([a], \ell)$ is in X if, and only if,

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

Definition 6.32 (Semantics of recursion formulae)**.** Let $\varphi, \mathbf{A}, \beta$ and X as above. Then $\mathbf{A}, \beta \models \varphi$, where $\beta(x) = (a, s_a)$ and $\beta(y) = (\operatorname{bitset}(\ell), \operatorname{ann}_{\operatorname{bitset}(\ell)})$ for $\ell \in \mathbb{N}$, if, and only if, $(a, \ell) \in X$.

6.3 Examples

We start with some examples of terms and their induced size annotations. As we have seen, $\operatorname{Atoms}.x = x$ defines the set of all atoms. The associated size annotations maps every atom from the input structure $\mathbf{A}$ to $\log |A|$. Similarly, $\operatorname{Atoms}.Px$ defines the set $P^{\mathbf{A}}$, where every atom is annotated by $\log |P^{\mathbf{A}}|$ instead. If P is sufficiently small, this allows to iterate over subsets of $P^{\mathbf{A}}$ of non-constant size. The same set can be defined with the term $\{y : y \in \operatorname{Atoms}.x = x : Py\}^{\mathbf{A}}$, but then every atom would be annotated with $\log |A|$ again.

The term $\operatorname{Atoms}.Exy$ defines the set $E^{\mathbf{A}}$ as a set of tuples, i. e. $\{\{\langle 1, a\rangle, \langle 2, b\rangle\} : (a, b) \in E^{\mathbf{A}}\}$. The associated size annotation maps every atom a occurring in some edge to $\log |E^{\mathbf{A}}|$. In general, using an equivalent term defining the atoms

with Atoms . true would lead to a smaller size annotation of each atom. The term Atoms $.Exy$ only leads to an advantage on classes of structures where E is small.

To see that the size annotation of a single term can map the atoms to different values, consider the term Atoms $.x = x \cup$ Atoms $.Px$. (Note that we use the same abbreviated terms that are defined for BGS in Section 3.2.1.) The value of that term in a structure $\mathbf{A}$ is again its whole domain. But the size annotation maps every element of $P^{\mathbf{A}}$ to $\log |P^{\mathbf{A}}|$, and the other atoms to $\log |A|$.

Our last example of an ordinary term is Unique(Atoms $.\forall y \neg Eyx$). The evaluation of this term heavily depends on the input structure. If the input structure is a directed tree, then it has a unique root. The size annotation will map that root to 0. But in a directed forest with multiple roots, the term's value is $\emptyset$, and the corresponding size annotation is the function $\emptyset \mapsto 1$.

To demonstrate the use of iteration terms, we construct a formula $\varphi_{\mathrm{dtc}}(x, y)$ defining that there is a deterministic path from x to y. The core of the formula is an iteration term $[t_u]^*(x)$ progressing along the deterministic path starting from x:

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

Note that, in contrast to iteration terms in BGS, the free variable x determines the value of the first stage. So, by definition of Unique, the ith stage of $[t_u]^*(x)$ is the ith element on the deterministic path starting from x. Every stage is added to the value of the term, so the desired formula is $\varphi_{\mathrm{dtc}}(x, y) = y \in [t_u]^*(x)$. The size annotation maps every atom from the input structure $\mathbf{A}$ to $\log |A|$, since all atoms are defined by the term Atoms . true. Thus $n \mapsto \log n$ is a suitable bound for the formula.

6.4 Expressive power

To demonstrate that CLogspace is a reasonable candidate for a logic for Logspace, we now analyse its expressive power. As a first step towards comparison with other logics, we show that it can express every DTC-definable query, including Logspace-computable queries on the natural numbers.

We proceed to show that Choiceless Logspace is indeed the choiceless fragment of Logspace. That means that it is included in both Logspace and Choiceless Polynomial Time. Further, it is stronger than known logics for Logspace. To obtain a separation from LREC, the strongest logic we consider, we use a CPT-style

padding argument, which demonstrates that CLogspace has a similar advantage over other logics as CPT.

6.4.1 Arithmetic operations and DTC

As a first step towards classifying the expressive power of CLogspace, we show that it can express Logspace-computable arithmetic operations on its "number sort". To show the analogous statement for CPT, one would prove that all Ptime-computable arithmetic operations are definable by using that FP captures Ptime on ordered structures. CLogspace, though, uses bitsets instead of ordinals as numbers, so there is no inherent linear order on the number sort. Therefore we explicitly express in CLogspace certain arithmetic operations that we then use to define a linear order on bitsets.

DTC, which is included in CLogspace, captures Logspace on the class of ordered structures. To use this, however, we need to show that CLogspace can still evaluate any DTC-formula over definable, i. e. interpreted, structures. This is non-trivial because CLogspace-formulae are evaluated over size-annotated objects, and the size of an element of the interpreted structure may be larger than the size of an atom. We thus define a suitable notion of CLogspace-interpretations and prove that DTC is weakly closed under these interpretations.

These steps may seem obsolete because CLogspace incorporates an LREC-style recursion operator, which should make it possible to embed LREC in CLogspace. This, however, already requires certain arithmatic operations: One of the differences between our recursion operator and the one in LREC is the *implicit* encoding of numbers as tuples. So the following results are in fact a prerequisite for embedding LREC into CLogspace.

We start defining the construction of the linear order on bitsets with two simple bit operations. Note that whenever we use iteration terms, we have to assume a logarithmic bound on the bitsets processed.

Lemma 6.33. *For every $c \in \mathbb{N}$, there are* CLogspace*-terms* readlast*, with free variables x, y, and* precbit*, with the free variable x, with values as follows. Let* $\mathbf{A}$ *be a structure $\beta\colon x \mapsto (a, \mathrm{ann}_a), y \mapsto (b, \mathrm{ann}_b)$ a variable assignment such that $a = \mathrm{bitset}(v)$ and $b = \mathrm{bitset}(w)$ for words $v, w \in \{0,1\}^+$ of length $\leq c \log |A|$ with the respective unique size annotations. Then*

1. $[\![\mathrm{readlast}(x)]\!]^{\mathbf{A},\beta}$ *is the rightmost bit of the string v, and*
2. $[\![\mathrm{precbit}(x,y)]\!]^{\mathbf{A},\beta}$ *is the bit preceding the suffix w in v if w is a strict suffix of v.*

Proof.

1. $\mathrm{readlast}(x) = \mathrm{Unique}(\{y : y \in [s_u]^*(x) : y \in \{\emptyset, \{\emptyset\}\}\})$, where the iterated term s outputs u if it is already a single bit, and continues the iteration with u's second entry otherwise, i. e. $s(u) = \mathrm{if}_{u \in \{\emptyset, \{\emptyset\}\}}(u, \pi_2(u))$.
2. To obtain $\mathrm{precbit}(x, y)$, we construct a term t that iterates over the suffixes of v, starting with v itself, until a string bw is encountered for some $b \in \{0, 1\}$. So let $t = [s_u]^*(x)$ with $s = \mathrm{if}_{y=\pi_2(u)}(\pi_1(u), \pi_2(u))$, and

$$\mathrm{precbit}(x, y) = \mathrm{Unique}(\{z : z \in t : z \in \{\emptyset, \{\emptyset\}\}\}).$$

Since v and w are of length $\leq c \log |A|$, we can derive logarithmic bounds for the iteration terms. □

The linear order will be constructed successively starting from $\emptyset$. Therefore we define a term incrementing bitsets.

Lemma 6.34. *For every polynomial $p \colon \mathbb{N} \to \mathbb{N}$, there is a term $t_{+1}(x)$ such that, for every structure $\mathbf{A}$ and assignment β with $\beta(x) = (a, s)$, if $a = \mathrm{bitset}(n)$ for $n \leq p(|A|)$ then $[\![t_{+1}]\!]^{\mathbf{A},\beta} = \mathrm{bitset}(n + 1)$.*

Proof. The goal is to construct a term that flips bits starting from the end of the binary representation of n until some bit is flipped from 0 to 1. We construct an iteration term $[t_u]^*(x)$ defining stages of the form $\langle u_1, u_2, u_3 \rangle$, where u_1 represents the original string (i. e. the value of x), u_2 is the suffix of u_1 that has already been processed, and u_3 is the modified version of the suffix where all necessary bits have been flipped. To distinguish the tuple $\langle u_1, u_2, u_3 \rangle$ from a single bitstring, we represent it as $\{\langle \mathrm{bitset}(1), u_1 \rangle, \langle \mathrm{bitset}(2), u_2 \rangle, \langle \mathrm{bitset}(3), u_3 \rangle\}$.

In the first iteration, t initialises its value to $\langle u, \mathrm{readlast}(u), \mathrm{if}_{\mathrm{readlast}(u)=\emptyset}(\{\emptyset\}, \emptyset) \rangle$. Starting from that value, every iteration flips the next bit, determined using precbit, if, and only if, the first bit of u_2 is 1 and the first bit of u_3 is 0. If $u_1 = u_2$, the whole string has been processed and a fixed point is enforced. Otherwise, we distinguish the following cases, where, in every case, u_2 is updated to $\langle \mathrm{precbit}(u_1, u_2), u_2 \rangle$, and u_1 does not change:

- ⋄ If $\pi_1(u_2) = \{\emptyset\}$ and $\pi_1(u_3) = \emptyset$, the next bit is flipped, i. e. u_3 is updated to $\langle \mathrm{if}_{\mathrm{precbit}(u_1,u_2)=\emptyset}(\{\emptyset\}, \emptyset), u_3 \rangle$.
- ⋄ If the current step flips the leftmost bit from 1 to 0, add a 1 in front of u_3. This is the case if, in addition to the conditions for the previous case, the next value

for u_2 will coincide with u_1 and $\mathrm{precbit}(u_1, u_2) = \{\emptyset\}$. Then u_3 is updated to $\langle\{\emptyset\}, \langle\emptyset, u_3\rangle\rangle$ instead.

- If $\pi_1(u_2) = \pi_1(u_3)$ or $\pi_1(u_3) = \{\emptyset\}$, then no more bits are flipped. This is achieved by updating u_3 to $\langle\mathrm{precbit}(u_1, u_2), u_3\rangle$.

The case distinction can be implemented with terms of the form if_φ.

The value of $t_{+1}(x)$ can be extracted from the last stage of $[t_u]^*(x)$, which is definable as the unique tuple $\langle u_1, u_2, u_3\rangle$ with $u_1 = u_2$. All entries of the stages of the iteration are bitsets encoding numbers $\leq p(|A|)$, so their size is bounded logarithmically. □

With the term t_{+1} we can now define the set of bitsets up to a definable cardinality by starting from bitset(0) and incrementing until the bound is reached.

Lemma 6.35. *For every term t without free variables, there is a term* order_t *such that, for every structure* $\mathbf{A}$, $[\![\mathrm{order}_t]\!]^{\mathbf{A}} = \{\langle\mathrm{bitset}(n), \mathrm{bitset}(m)\rangle : n \leq m \leq |[\![t]\!]^{\mathbf{A}}|\}$.

Proof. Let $\mathrm{order}_t = [s_u]^*(\langle\emptyset, \emptyset\rangle)$, where we construct s such that it iterates over all pairs $\langle\mathrm{bitset}(n), \mathrm{bitset}(m)\rangle$ with $n \leq m \leq N$. In particular, the value of s is always a pair $\langle u_1, u_2\rangle$.

- If $u_2 \neq \mathrm{Card}(t)$, increment u_2 using the term t_{+1}.
- If $u_2 = \mathrm{Card}(t)$ and $u_1 \neq \mathrm{Card}(t)$, increment u_1 and replace u_2 by the new value of u_1.
- If $u_1 = u_2 = \mathrm{Card}(t)$, leave u unchanged to enforce a fixed point.

To see that there is a logarithmic bound for the term order_t, note that all bitsets occurring in the stages encode numbers up to $|[\![t]\!]^{\mathbf{A}}|$. As we will show in the next section, the value of t, represented as a list of sets in its transitive closure, can be computed in logarithmic space. So its transitive closure is of polynomial size. □

Given order_t, the set of bitset representations can easily be defined by the term $\{\pi_1(x) : x \in \mathrm{order}_t\}$, so we obtain the following corollary:

Corollary 6.36. *For every term t without free variables, there is a term* $\mathrm{numbers}_t$ *such that, for every structure* $\mathbf{A}$, $[\![\mathrm{numbers}_t]\!]^{\mathbf{A}} = \{\mathrm{bitset}(n) : n \leq |[\![t]\!]^{\mathbf{A}}|\}$.

To formalise that CLogspace can simulate DTC-formulae on the order defined that way, we convert the term order_t to an interpretation. But the standard definition of interpretations does not suffice to translate formulae to CLogspace: The values of the free variables of every formula in the interpretation have to be translated

to size-annotated hereditarily finite objects. But if the domain of the interpreted structure is defined by a CLogspace-term instead of a formula, every element of the interpreted structure can inherit the size annotation from that term. This also makes it possible to use the full set-building abilities of CLogspace to define the domain. A similar notion of interpretations was first defined by Theophil Trippe in his Bachelor's thesis [58].

Definition 6.37. Let σ, τ be signatures. A CLogspace$[\sigma, \tau]$-*interpretation* with parameter z is a tuple $\mathcal{I} = (t_\delta, \varphi_=, (\varphi_R)_{R\in\tau})$ where t_δ is a CLogspace$[\sigma]$-term and $\varphi_=$ and the φ_R are CLogspace$[\sigma]$-formulae. $\varphi_=$ has free variables x_1, x_2, and every φ_R has free variables $x_1, \dots, x_r$, where r is the arity of R. Additionally, z can occur as a free variable in $t_\delta, \varphi_=$ and every φ_R.

$\mathcal{I}$ defines a transformation of σ-structures and parameter assignments to τ-structures as follows: Let $\mathbf{A}$ be a σ-structure, and let $\gamma\colon \{z\} \to \mathrm{SHF}(A)$. We first define a $(\tau \cup \{\sim\})$-structure $\mathbf{B}$ over the domain $B = [\![t_\delta]\!]^{\mathbf{A},\gamma}$. The formulae for φ_R are evaluated using the size annotation induced by t_δ. So for a formula φ_R with $R \in \tau$ of arity r and a tuple $\overline{b} = (b_1, \dots, b_r) \in B^r$, we define the assignment $\beta_{\overline{b}} = \gamma[x_i \mapsto (b_i, \mathrm{ann}^{\mathbf{A},\gamma}_{t_\delta} \restriction \mathrm{tc}(b_i))]$, $1 \le i \le r$. Then $R^{\mathbf{B}} = \{\overline{b} \in B^r : \mathbf{A}, \beta_{\overline{b}} \models \varphi_R\}$. Similarly, the formula $\varphi_=$ defines the relation $\sim$ as the reflexive symmetric transitive closure of $\{(b_1, b_2) \in B^2 : \mathbf{A}, \beta_{(b_1,b_2)} \models \varphi_=\}$.

The *interpreted structure* $\mathcal{I}(\mathbf{A}, \gamma)$ is $\mathbf{B}_{/\sim}$. If the parameter z does not occur in any of the terms and formulae, we write $\mathcal{I}(\mathbf{A})$ as usual.

We will see later that CLogspace can define symmetric transitive closures (because it includes LREC), so the definition of $\sim$ does not add inadequate expressive power to interpretations. In fact, we only allow the formula $\varphi_=$ for uniformity reasons.

Similarly to interpretations for other logics, CLogspace-interpretations also define transformations of formulae. We show the existence of such a transformation for DTC-formulae. To make that transformation precise, we first translate variable assignment suitable for DTC to assignments for CLogspace.

Definition 6.38. Let $\mathcal{I} = (t_\delta, \varphi_=, (\varphi_R)_{R\in\tau})$ be a CLogspace$[\sigma, \tau]$-interpretation with parameter z, let $\mathbf{A}$ be a σ-structure, and let $\gamma\colon \{z\} \to \mathrm{SHF}(A)$ be an assignment of the parameter. Further, let $\mathcal{I}(A, \gamma)$ be the domain of $\mathcal{I}(\mathbf{A}, \gamma)$. We associate with every variable assignment $\beta\colon X \to \mathcal{I}(A, \gamma)$ an assignment $\beta_{\mathcal{I}}\colon X \cup \{z\} \to \mathrm{SHF}(A)$ defined as $\beta_{\mathcal{I}}(x) = (\beta(x), \mathrm{ann}^{\mathbf{A}}_{t_\delta} \restriction \mathrm{tc}(\beta(x)))$ for $x \in X$ and $\beta_{\mathcal{I}}(z) = \gamma(z)$.

This allows to formalise a notion of closure under CLogspace-interpretations:

Lemma 6.39. *For every* CLogspace$[\sigma, \tau]$*-interpretation* $\mathcal{I}$ *with parameter* z *and every* DTC$[\tau]$*-formula* φ*, there is a* CLogspace$[\sigma]$*-formula* $\varphi^{\mathcal{I}}$ *such that, for every* σ*-structure* $\mathbf{A}$ *and all assignments* $\beta\colon \mathrm{free}(\varphi) \to \mathcal{I}(A)$ *and* $\gamma\colon \{z\} \to \mathrm{SHF}(A)$*,*

$$\mathbf{A}, \beta_{\mathcal{I}} \models \varphi^{\mathcal{I}} \text{ if, and only if, } \mathcal{I}(\mathbf{A}, \gamma), \beta \models \varphi.$$

It might seem straightforward to translate dtc-subformulae to iteration terms. Before we present the proof, let us thus remark why we choose the lrec-operator instead. One of the results of Theophil Trippe's Bachelor's thesis [58] is closure of DTC under a certain class of CLogspace-interpretations without the use of recursion formulae. Note, however, that the stages of iteration terms have to be sets of logarithmic size. Accordingly, Trippe's construction only covers the case where every element of the interpreted structure satisfies a logarithmic size bound.

Proof. We proceed by induction on the DTC-formula φ. For atomic formulae, we use the formulae $\varphi_=$ and φ_R, respectively. Boolean connectives are translated directly, and quantifiers are handled as in the translation from FO to BGS. So it remains to translate formulae of the form $\varphi = [\mathsf{dtc}_{u,v}\psi(u,v)](x,y)$. In particular, it suffices to consider formulae with single variables u, v instead of tuples, because tuples can be eliminated with an additional CLogspace-interpretation.

We adopt the translation from dtc-formulae to LREC from [32] with very small modifications. First of all, the domain term and the equality formula are taken from the interpretation. The basic idea behind the formula from [32] is to modify the graph defined by ψ as follows: If a vertex has more than one outgoing edge, remove all of its outgoing edges, and reverse the remaining edges. Since lrec-formulae propagate information along *incoming* edges, this makes it possible to propagate the reachability information from s to t. Thus the edge formula of the recursion formula is

$$\varphi_E(u,v) = \psi^{\mathcal{I}}(v,u) \wedge u = \mathrm{Unique}(\{w : w \in \mathrm{Atoms}\,.\,\mathrm{true} : \psi^{\mathcal{I}}(v,w)\}).$$

By definition of φ_E, every vertex has a unique E-predecessor. So membership in the set X induced by the recursion formula is based on the number of successors b with $(b, \ell - 1) \in X$. In other words, the number ℓ is decreased by one in every step. Since the recursion formula propagates information along a deterministic path through the structure over $[\![t_\delta]\!]^{\mathbf{A}}$, it thus suffices to consider labels up to $|[\![t_\delta]\!]^{\mathbf{A}}|$. Further, the pair (s, ℓ) should be included in X for every relevant ℓ, whereas every

other vertex should only occur in X if it has an E-successor in X. So we define the labels by

$$t_C(u) = \{v : v \in \text{numbers}_{t_\delta} : u = s\} \cup \{v : v \in \text{numbers}_{t_\delta} : u \neq s \wedge v \neq \emptyset\},$$

where $\text{numbers}_{t_\delta}$ is the term from Corollary 6.36.

The full recursion formula checks whether there is some $\ell \leq |[\![t_\delta]\!]^{\mathbf{A}}|$ such that $(t, \ell) \in X$. So the formula φ is transformed to

$$\varphi^{\mathcal{I}} = \{\lambda : \lambda \in \text{numbers}_{t_\delta} : [\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](t, \lambda)\} \neq \emptyset,$$

where $\psi(t, \lambda) \coloneqq [\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](t, \lambda)$ is an abbreviation of the formula

$$\{x : x \in \{t\} : \psi(x, \lambda)\} \neq \emptyset. \qquad \square$$

Our goal is to simulate DTC-formulae on an ordered set of bitsets. By Lemma 6.35 the linear order is definable as a set of tuples. We now translate the term from Lemma 6.35 to an interpretation.

Lemma 6.40. *Let t be a* CLogspace*-term with vocabulary σ. Then there is a* CLogspace$[\sigma, \{<\}]$*-interpretation $\mathcal{I}_t^{\text{bit}}$ such that, for every σ-structure $\mathbf{A}$ where $[\![t]\!]^{\mathbf{A}} = \text{bitset}(N)$ for some $N > 0$, $\mathcal{I}_t^{\text{bit}}(\mathbf{A})$ is the structure over the domain $\{\text{bitset}(n) : n \leq N\}$ with $\text{bitset}(n) <^{\mathcal{I}_t^{\text{bit}}(\mathbf{A})} \text{bitset}(m)$ if, and only if, $n < m$.*

Proof. We use the term order_t from Lemma 6.35 to define both the domain and the relation of the interpreted structure. So $\mathcal{I}_t^{\text{bit}} = (t_\delta, \varphi_=, \varphi_<)$ with $t_\delta = \text{numbers}_t$, $\varphi_=(x_1, x_2) = x_1 = x_2$, and

$$\varphi_<(x_1, x_2) = x_1 \neq x_2 \wedge \{y : y \in \text{order}_t : \pi_1(y) = x_1 \wedge \pi_2(y) = x_2\} \neq \emptyset. \qquad \square$$

Together with Lemma 6.39 and the fact that DTC captures Logspace on the class of ordered structures, we get

Lemma 6.41. *Let $R \subseteq \{0, \ldots, |A|^k\}^\ell$ be any* Logspace*-computable relation. Then there is a* CLogspace*-term defining $\{\langle \text{bitset}(n_1), \ldots, \text{bitset}(n_\ell)\rangle : (n_1, \ldots, n_\ell) \in R\}$.*

6.4.2 Choiceless Logspace is choiceless and in Logspace

Next we show that our logic is not too strong to be considered Choiceless Logspace. Most importantly, we verify that it qualifies as a candidate for a logic for Logspace, i. e. that it is included in Logspace.

Theorem 6.42. *Every query definable in* CLogspace *is* LOGSPACE*-computable.*

The explicit logarithmic bound in CLogspace-sentences is intended to keep those sets sufficiently small that are computed during the algorithmic evaluation of terms and formulae. This bound is defined with respect to size annotations, which are part of the value of every CLogspace-term. In the semantics of terms and formulae, size annotations are defined with a certain representation of hereditarily finite objects in mind. To show that values with a logarithmic size annotation only need logarithmic space on a Turing machine, we formalise that representation.

Note that we now assume ordered structures as input, because the input of a Turing machine is always a string encoding of a structure. Recall that elements of the input structure are always defined by means of terms Atoms $.\varphi$. Given such a formula, one can represent an atom as "the ith entry of the kth tuple satisfying φ" using the linear order on the input structure. We denote the set of k-tuples from a structure $\mathbf{A}$ satisfying an FO-formula φ with k free variables by $\varphi^{\mathbf{A}}$.

Definition 6.43 (Representations)**.** Let $\Phi \subseteq \mathrm{FO}[\sigma]$ for some vocabulary σ.

The alphabet Σ_Φ consists of a symbol φ for every $\varphi \in \Phi$, numbers $0, 1$, parentheses "(" and ")", braces "{" and "}", and a delimiter ",".

A Φ*-representation* is a word in Σ_Φ that is either a set representation or an atom representation:

- An *atom representation* is a word (φ, i, m), where $\varphi \in \Phi$ with $k \geq 1$ free variables, i is the binary encoding of a natural number $\leq k$ and m is the binary encoding of a natural number $\leq |\varphi^{\mathbf{A}}|$. If $|\varphi^{\mathbf{A}}| = 1$, then m is the empty string.
- If $r_1, \ldots, r_k$ are representations, then the word $\{r_1 \cdots r_k\}$ (i. e. the concatenation of $r_1, \ldots, r_k$ with a new pair of braces) is a set representation.

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

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

The word length of any representation occurring in the simulation of CLogspace-formulae is supposed to be expressible in terms of a size annotation. But note that the definition of representations permits a set to be listed multiple times, and an atom to occur encoded by different atom representations within the same representation. Thus we define a class of representations without these artifacts—which can then be associated with meaningful size annotations.

Definition 6.44. Let $\mathbf{A}$ be a $\sigma \cup \{<\}$-structure, and let r be a Φ-representation with $\Phi \subseteq \mathrm{FO}[\sigma]$. A representation r is $\mathbf{A}$-*minimal* if,

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

We omit the structure $\mathbf{A}$ and speak about minimal representations if $\mathbf{A}$ can be inferred from the context.

Minimal representations are the core of our evaluation algorithm: The value of each term will be encoded as a minimal representation. Since we aim to use the bound on size annotations as a bound on the size of representations, we associate with every minimal representation a size annotation.

Definition 6.45. Let $\mathbf{A}$ be a $\sigma \cup \{<\}$-structure, and let r be an $\mathbf{A}$-minimal Φ-representation for some $\Phi \subseteq \mathrm{FO}[\sigma]$. We define the size annotation $\mathrm{ann}_r^{\mathbf{A}}$ by induction on the representation r:

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

The *size* of r under $\mathbf{A}$ is $\|r\|^{\mathbf{A}} = \mathrm{ann}_r^{\mathbf{A}}(r^{\mathbf{A}})$. We say that r is a representation of $(r^{\mathbf{A}}, \mathrm{ann}_r^{\mathbf{A}}) \in \mathrm{SHF}(A)$.

Note that the size annotation does not depend on the numbers i and m in the atom representations, and is hence independent of the linear order $<$. So, when we derive a bound on the length of a representation from its size annotation, that bound is also independent of the linear order, i. e. the encoding of the structure.

Lemma 6.46. *Let r be an $\mathbf{A}$-minimal Φ-representation, and let k be the maximal number of free variables of any formula in Φ. Then the length $|r|$ of the word r is $\leq (\log k + 6)\|r\|^{\mathbf{A}}$.*

Proof. Let $c = \log k + 5$. We show that $|r| \leq (c+1)\|r\|^{\mathbf{A}}$. If r is an atom representation, then r is a word (φ, i, m), where φ and the commas and parentheses are alphabet symbols, i is a binary string of length $\leq \log k$, and m is a binary string of length $\leq |\varphi^{\mathbf{A}}| = \|r\|^{\mathbf{A}}$. So $|r| \leq \log k + 5 + \|r\|^{\mathbf{A}} = c + \|r\|^{\mathbf{A}} \leq (c+1)\|r\|^{\mathbf{A}}$.

Now let $r = \{r_1 \dots r_\ell\}$ be a set representation. Then r is a word of length

$$\begin{aligned}
&\sum_{i=1}^{\ell} |r_i| + 2 \\
\leq &\sum_{i=1}^{\ell} |r_i| + (c+1) \\
\overset{\text{ind. hyp.}}{\leq} &\sum_{i=1}^{\ell} (c+1)\|r_i\|^{\mathbf{A}} + (c+1) \\
= &(c+1)\|r\|^{\mathbf{A}}.
\end{aligned}$$

□

We aim to construct for every CLogspace-term a LOGSPACE-algorithm that computes a representation of its value. In most cases, this means that representations for the subterms are computed and then combined to form a new set. Consider, for example, the term $\mathrm{Pair}(t_1, t_2)$. When we simply compute representations r_1 and r_2 for the values of t_1 and t_2 and output the representation $\{r_1 r_2\}$, minimality is not guaranteed anymore. First, we have to check whether r_1 and r_2 actually represent the same set.

Lemma 6.47. *There is an algorithm that, given a $\sigma \cup \{<\}$-structure $\mathbf{A}$ (where $<^{\mathbf{A}}$ is obtained from the string representation of $\mathbf{A}$) and $\mathbf{A}$-minimal representations r_1, r_2, decides in logarithmic space whether $r_1^{\mathbf{A}} = r_2^{\mathbf{A}}$.*

Proof. We reduce equality of representations to isomorphism of coloured trees, where the colours are numbers $1, \dots, |A|$. So we define, by induction on r_1, r_2, trees T_{r_1}, T_{r_2} that are isomorphic if, and only if, $r_1^{\mathbf{A}} = r_2^{\mathbf{A}}$. The leaves of the trees are coloured by (indices of) atoms. Thus, if r is an atom representation, then T_r is a single node coloured by $r^{\mathbf{A}}$.

If $r = \{r_1 \dots r_k\}$ is a set representation, then T_r is the disjoint union of $T_{r_1}, \dots, T_{r_k}$ with a new root node with the roots of $T_{r_1}, \dots, T_{r_k}$ as children.

For atom representations, it follows directly from the definition that the trees are isomorphic if, and only if, the representations have the same value. For set representations, this property follows from the induction hypothesis together with the fact that, since the representations are minimal, every isomorphism type occurs at most once as a subtree of both roots.

It remains to show that the trees can be computed in logarithmic space. Note that there is a one-to-one correspondence between vertices in the trees and opening "{" and "(" in the strings r_1 and r_2, respectively. So the algorithm can compute

adjacency matrices of the trees by maintaining three counters: Iterate over the rows of the adjacency matrix by storing the index of the current "{" or "(". For each row, iterate over the "{" and "(" again to find successors in the tree. At the same time, count the number of closing "}" to maintain the rank of the sets whose representations are encountered, and to determine the end of the representation corresponding to the current row in the adjacency matrix. The rank information then determines whether the current entry should be 1. □

With the equality test, we will be able to construct representations where every set is listed only once. In a second step, we ensure that no two representations of the same atom occur.

Lemma 6.48. *There is an algorithm that, given a $\sigma \cup \{<\}$-structure $\mathbf{A}$ (where $<^{\mathbf{A}}$ is is obtained from the encoding of $\mathbf{A}$) and $\mathbf{A}$-minimal representations $r_1, \dots, r_k$, computes in logarithmic space a minimal representation r of the size-annotated set $\{(r_1^{\mathbf{A}}, \mathrm{ann}_{r_1}^{\mathbf{A}}), \dots, (r_k^{\mathbf{A}}, \mathrm{ann}_{r_k}^{\mathbf{A}})\}_{\mathrm{SHF}}$.*

Proof. We sequentially execute two LOGSPACE-algorithms. The first one iterates over the representations $r_1, \dots, r_k$ and copies r_i to the output tape if, and only if, the value $r_i^{\mathbf{A}}$ has not been encountered before (using the equality algorithm from Lemma 6.47). The second one computes unique atom representations as follows: It iterates over the input representations and copies everything except for atom representations to the output tape. Whenever an atom representation is encountered, the algorithm iterates over all atom representations that occur as substrings of the input and compares their values. The shortest representation of the current atom is written to the output tape. If there are two equally sized representations of the same atom, chose the one that occurs first in the input. □

Another operation that is crucial to the evaluation of terms and formulae from CLogspace is checking whether the logarithmic bound is exceeded. Note that it does not suffice to consider the length of the representation. Firstly, the bound applies to the size generated by the annotation, instead of the actual word length, which may differ by constant factors. Secondly, the size annotation is order-independent; the size annotation of an atom representation (φ, i, m) only depends on the number of tuples satisfying φ.

Lemma 6.49. *There is a* LOGSPACE*-algorithm that, given a σ-structure $\mathbf{A}$ and a representation r, computes $\|r\|^{\mathbf{A}}$.*

Proof. Read the input representation r while maintaining a counter for the size. Whenever the symbol "{" is encountered, add 1 to the counter, and for every atom representation (φ, i, m), compute $\log |\varphi^{\mathbf{A}}|$ and add that value to the counter. □

Using the subroutines from the previous lemmas, we can inductively construct algorithms evaluating all terms and formulae.

Lemma 6.50.

1. *For every formula φ with free variables $x_1, \dots, x_k$ and every function $f\colon n \mapsto c \log n$, there is a* Logspace *algorithm that, given a σ-structure $\mathbf{A}$ and representations $r_1, \dots, r_k$ of $\alpha_1, \dots, \alpha_k \in \mathrm{SHF}(A)$, decides whether $\mathbf{A}, \beta\colon x_i \mapsto \alpha_i \models (\varphi, f)$.*
2. *For every term t with free variables $x_1, \dots, x_k$ and every function $f\colon n \mapsto c \log n$, there is a* Logspace *algorithm that, given a σ-structure $\mathbf{A}$ and representations $r_1, \dots, r_k$ of $\alpha_1, \dots, \alpha_k \in \mathrm{SHF}(A)$, computes a representation of $(t, f)^{\mathbf{A}, \beta\colon x_i \mapsto \alpha_i}$.*

Proof. We proceed by induction on the construction of terms and formulae. In the induction step, we use that Logspace is closed under concatenation, and operate on the outputs of the algorithms for the subterms even if those outputs may be too large to be represented on the work tape. Further, note that the bound f only affects the evaluation of iteration terms, so it can be ignored in the other cases.

For $t = \emptyset$, output $\{\}$, and for $t = x_i$, copy r_i to the output tape.

For $\varphi = Rs_1 \dots s_k$, run the algorithms for $s_1, \dots, s_k$. If all outputs are atom representations, compute their values and check if they form a tuple from $R^{\mathbf{A}}$.

If φ is a Boolean combination of formulae, the truth value of φ can be deduced directly from the values of the subformulae.

For $\varphi = s = t$, run the equality algorithm from Lemma 6.47 on the output of the algorithms for s and t.

For $\varphi = s \in t$, compute representations r_s and r_t for s and t and run the equality test on the pair r_s, r for every representation r occurring within the outer pair of braces in r_t.

For $t = \mathrm{Atoms}\,.\varphi$, compute the number of tuples satisfying the FO-formula φ. Then enumerate the tuples with entries of the form (φ, i, m) for $0 \leq m \leq |\varphi^{\mathbf{A}}|$.

For $t = \mathrm{Pair}(q, s)$, run the algorithms for q, s and minimise the resulting representation with the algorithm from Lemma 6.48.

For $t = \mathrm{Unique}(s)$, check if the output of the algorithm for s (which is already a minimal representation) is a singleton. If it is, remove one pair of braces, and output "{}" otherwise.

For $t = \mathrm{Union}(s)$, run the algorithm for s, remove a pair of braces around each element of the output, and minimise the resulting representation with the algorithm from Lemma 6.48.

For $t = \mathrm{Card}(s)$, compute the value of s and count the number of elements.

For $t = \{s : x \in q : \varphi\}$, compute the value of q first. Then, for every element of that value, run the algorithm $\mathcal{A}_\varphi$ for φ and copy the value to the output tape if, and only if, $\mathcal{A}_\varphi$ returns true. Run the algorithm for s on every element of the output. Finally, minimise with the algorithm from Lemma 6.48.

Let $t = [s_u]^*$ be an iteration term with free variable x_j. In its ith iteration, the algorithm writes a representation of $(s^i)^{\mathbf{A},\beta}$ to the work tape if, and only if, $\|s^i\|^{\mathbf{A},\beta} \leq f(|A|)$, and compares the result to $(s^{i-1})^{\mathbf{A},\beta}$. Given a representation of $(s^{i-1})^{\mathbf{A},\beta}$, the next value $(s^i)^{\mathbf{A},\beta}$ can be computed with the algorithm for s from the induction hypothesis. The size of that value can be computed by Lemma 6.49, and the representations are compared using the equality algorithm from Lemma 6.47.

By Lemma 6.46, the bound on the size of each $(s^i)^{\mathbf{A},\beta}$ induces a bound on the word length of the representations. This implies that there is a logarithmic bound on the values written to the work tape in every iteration. In particular, there is a polynomial number of different values that are allowed to occur as stages. This ensures termination, since a counter can track the number of values that have been produced.

The output should consist of the set of all intermediate stages, so the algorithm additionally writes the value of every stage to the output tape. If the bound is exceeded, a special marker is placed on the output tape. In a postprocessing step, the result is converted to a minimal representation using the algorithm from Lemma 6.48, or replaced by $\{\}$ if the bound was exceeded.

Let $\varphi = [\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x, y)$ be a recursion formula. It is known that the lrec-operator in LREC can be evaluated in logarithmic space [34]. So we reduce evaluation of lrec-formulae over hereditarily finite sets to common lrec-formulae.

The reduction algorithm computes a structure whose domain is the disjoint union of $D \coloneqq [\![t_\delta]\!]^{\mathbf{A},\beta}$ with a linear order N of size $|D|$ with relations $\sim, E, C$ that allow to evaluate a trivial LREC-formula. More precisely, the relations are defined in a way that it suffices to evaluate $\psi = [\mathsf{lrec}_{u,v} \psi_=, \psi_E, \psi_C]$ with $\psi_=(u, v) = u \sim v$ and $\psi_E(u, v) = Euv$. For the formula ψ_C, choose $k \in \mathbb{N}$ such that the largest value in any label $[\![t_C]\!]^{\mathbf{A},\beta[u \mapsto (a,s)]}$ for $a \in D$ (with size annotation s) is at most $|D|^k$. Then define a relation $C \subseteq D \times N^k$ such that $(a, \overline{b}) \in C$ if, and only if, $\overline{b}$ is the nth tuple in N^k for some n in the label of a. The formula $\psi_C(u, \overline{\lambda})$ expresses that $\overline{\lambda}$ encodes the index of a tuple assigned to u by the relation C.

So the algorithm computes a linearly ordered structure over $[\![t_\delta]\!]^{\mathbf{A},\beta}$, uses the algorithms for $\varphi_=, \varphi_E, t_C$ to compute relations $\sim$, E and C, and evaluates ψ over that structure. □

So we have established that CLogspace describes a fragment of Logspace. To verify that this is the *choiceless* fragment, we embed CLogspace in CPT.

Theorem 6.51. CLogspace $\leq$ CPT.

On the level of sentences, it is clear what it means to translate CLogspace-formulae to CPT. But as soon as terms and formulae have free variables, we encounter two technical difficulties: CLogspace is evaluated over size-annotated hereditarily finite sets, and iteration terms in CLogspace can have free variables whose values determine the first stage of the iteration.

We handle the first difficulty by defining a representation of size-annotations as hereditarily finite sets. The CPT-formulae will then be constructed assuming that the values of the free variables carry a size annotation of that form.

Definition 6.52. Let $\mathbf{A}$ be a σ-structure, and let $(a, s) \in \mathrm{SHF}(A)$. The *set representation* $\mathrm{set}((a, s)) \in \mathrm{HF}(A)$ is the set $\langle a, \{\langle b, [s(b)]\rangle : b \in \mathrm{tc}(a) \cap A\}\rangle$ (recall that $[s(b)]$ is the ordinal corresponding to $s(b)$). We say that the second entry $\{\langle b, [s(b)]\rangle : b \in \mathrm{tc}(a) \cap A\}$ *represents the size annotation* s. Let X be a set of variables and let $\beta\colon X \to \mathrm{SHF}(A)$ be a variable assignment. Then we define the assignment $\beta_{\mathrm{HF}}\colon X \to \mathrm{HF}(A)$ through $\beta_{\mathrm{HF}}(x) = \mathrm{set}(\beta(x))$.

We can address the second problem by evaluating every iteration term for all possible values of its free variables at once. More precisely, the CPT-term without free variables first replaces the initial value $\emptyset$ by the set of all possible values of the free variables and then simulates the original CLogspace-term on all elements of that set in parallel. But this is only possible in CPT if the set of possible values of the free variables is already CPT-definable.

Therefore, all lemmas in the remainder of this section are phrased to require a term t_{var} defining the possible values of all free variables. In the inductive translation from CLogspace to CPT, we will make sure that such a term always exists.

For ease of notation, we omit the explicit bound in all terms and formulae we define in the remainder of the proof. So we write t (resp. φ) to denote (t, p) (resp. (φ, p)) for some polynomial p.

Following the definition of the semantics of CLogspace-terms in Section 6.2, we first describe a CPT-term defining the combination of size annotations from Definition 6.23.

Lemma 6.53. *There is an ordinary* BGS*-term* t_{SHF} *such that for every* σ*-structure* $\mathbf{A}$*, if* $a \in \mathrm{HF}(A)$ *is a set of objects representing the elements of* $\mathcal{B} \subseteq \mathrm{SHF}(A)$*, then* $[\![t_{\mathrm{SHF}}]\!]^{\mathbf{A}}(a)$ *represents* $\mathcal{B}_{\mathrm{SHF}}$*.*

Proof. It suffices to determine the minimal size of the atoms occurring in the transitive closure of some $\alpha \in \mathcal{B}$. So we define all pairs $\langle a, n\rangle$ where n is the minimal value of a occurring in any of the size annotations as

$$t_{\min}(x) = \{y : y \in t_{\mathrm{ann}}(x) : \\ \{z : z \in t_{\mathrm{ann}}(x) : \pi_1(z) = \pi_1(y) \wedge \pi_2(y) \in \pi_2(z)\} = \emptyset\},$$

where t_{ann} defines the set of size annotations from $\mathcal{B}$ as $t_{\mathrm{ann}}(x) = \mathrm{Union}(\{\pi_2(y) : y \in x\})$. Then

$$t_{\mathrm{SHF}}(x) = \langle \{\pi_1(y) : y \in x\}, t_{\min}(x)\rangle. \qquad \square$$

Further, the semantics of CLogspace uses restrictions of size annotations. Restricting a set representation of a size annotation to a set a means restricting its domain to $\mathrm{tc}(a)$. So, to show that restrictions of size annotations are CPT-definable, we first show that the transitive closure of definable sets is again definable.

Lemma 6.54. *For every* CPT*-term* t_{var}*, there is a* CPT*-term* t_{tc} *such that, for every* σ*-structure* $\mathbf{A}$ *and every* $a \in [\![t_{\mathrm{var}}]\!]^{\mathbf{A}}$*,* $[\![t_{\mathrm{tc}}]\!]^{\mathbf{A}}(a) = \mathrm{tc}(a)$*.*

Proof. The transitive closure of a set of arbitrary rank can be defined by means of an iteration term. Since iteration terms have no free variables, the value a cannot be passed to the iteration term. But, since t_{var} restricts the possible values of a, we construct a term that defines the transitive closure for all values in the set $[\![t_{\mathrm{var}}]\!]^{\mathbf{A}}$ at once.

More precisely, we define the set $\{\langle a, \mathrm{tc}(a)\rangle : a \in [\![t_{\mathrm{var}}]\!]^{\mathbf{A}}\}$ with a term t'_{tc}. Then the desired term $t_{\mathrm{tc}}(x)$ is $\pi_2(\mathrm{Unique}(\{y : y \in t'_{\mathrm{tc}} : \pi_1(y) = x\}))$.

The term t'_{tc} is the iteration s^*_u of a term s that, whenever the value of its free variable u is $\emptyset$, takes the value $\{\langle v, v\rangle : v \in t_{\mathrm{var}}\}$. In stage $i > 0$, it applies the term $\{\langle \pi_1(v), \pi_2(v) \cup \mathrm{Union}(\pi_2(v))\rangle : v \in u\}$ to the previous stage. $\square$

We can now define the restrictions of size annotations with the encoding from Definition 6.52.

Lemma 6.55. *For every* CPT*-term* t_{var}*, there is a* CPT*-term* $t_{\restriction}$ *such that, for every* σ*-structure* $\mathbf{A}$*, every* $a \in t^{\mathbf{A}}_{\mathrm{var}}$ *and every* $r \in \mathrm{HF}(A)$ *that represents a size annotation* s *of some* $b \supseteq a$*,* $[\![t_{\restriction}]\!]^{\mathbf{A}}(r, a)$ *represents* $s \restriction \mathrm{tc}(a)$*.*

Proof. $t_\restriction(x) = \{y : y \in x : \pi_1(y) \in t_{\text{tc}}(a)\}$, where t_{tc} is the term obtained from Lemma 6.54 using t_{var} as a guard for the free variable. □

With these preparation steps, the translation from CLogspace to CPT becomes a rather straightforward induction.

Lemma 6.56.

1. *For every* CLogspace*-term* t *and every* CPT*-term* t_{var} *without free variables, there is a* CPT*-term* t_{CPT} *such that, for every* σ*-structure* $\mathbf{A}$ *and assignment* $\beta : \text{free}(t) \to \{(a, s) \in \text{SHF}(A) : a \in t_{\text{var}}^{\mathbf{A}}\}$, $[\![t_{\text{CPT}}]\!]^{\mathbf{A},\beta_{\text{HF}}} = \text{set}(t^{\mathbf{A},\beta})$.
2. *For every* CLogspace*-formula* φ *and every* CPT*-term* t_{var} *without free variables, there is a* CPT*-formula* φ_{CPT} *such that, for every* σ*-structure* $\mathbf{A}$ *and assignment* β: $\text{free}(\varphi) \to \{(a, s) \in \text{SHF}(A) : a \in t_{\text{var}}^{\mathbf{A}}\}$, $\mathbf{A}, \beta_{\text{HF}} \models \varphi_{\text{CPT}}$ *if, and only if,* $\mathbf{A}, \beta \models \varphi$.

Proof. We inductively translate the terms and formulae from CLogspace. For every term t, we define terms t_{val} and t_{ann} such that $t_{\text{CPT}} = \langle t_{\text{val}}, t_{\text{ann}} \rangle$. If $t = x_i$, then $t_{\text{val}} = \pi_1(t)$ and $t_{\text{ann}} = \pi_2(t)$. If $t = \emptyset$, then $t_{\text{val}} = t_{\text{ann}} = \emptyset$. The translation of formulae $Rs_1 \dots s_r$, $s = t$, $s \in t$ and all Boolean combinations of formulae is a direct application of the induction hypothesis.

For $t = \text{Atoms}\,.\varphi$ with $\varphi(x_1, \dots, x_k) \in \text{FO}$, let

$$t_{\text{val}} = \{\{\langle \text{bitset}(1), x_1 \rangle, \dots, \langle \text{bitset}(k), x_k \rangle\} : x_1, \dots, x_k \in \text{Atoms} : \varphi_{\text{CPT}}\},$$

where φ_{CPT} is a CPT-formula equivalent to the FO-formula φ with free variables $x_1, \dots, x_k$. The corresponding size annotation should map each atom that occurs in any tuple in $\varphi^{\mathbf{A}}$ to $\log |[\![t_{\text{val}}]\!]^{\mathbf{A}}|$. So we define the formula

$$\varphi_{\text{occurs}}(x) = \{x_1, \dots, x_k \in \text{Atoms} : \varphi_{\text{CPT}}(x_1, \dots, x_k) \wedge x = x_i\} \neq \emptyset,$$

and finally

$$t_{\text{ann}} = \{\langle x, t_{\log}(\text{Card}(t_{\text{val}})) \rangle : x \in \text{Atoms} : \varphi_{\text{occurs}}(x)\}.$$

Note that a CPT-term $t_{\log}$ defining the logarithm of every ordinal exists, because ordinals are linearly ordered, and $\text{FP} \leq \text{CPT}$ already captures PTIME on ordered structures.

For the case $t = \text{Card}(s)$, it suffices to evaluate s and then translate its cardinality from the ordinal defined by the operator in CPT to the bitset-representation used in

CLogspace. Note that, since the bitset-representation of every number is computable in PTIME, it is also definable in fixed-point logic over a set containing the two kinds of number sorts, i. e. the disjoint union of an ordinal $[n]$ and the sets bitset(m) for $m \leq n$. Such a set is definable by an iteration term that successively extends bitsets by both 0 and 1 until those for strings of length $\log n$ have been constructed. The number n is determined as the maximal size of the value of s for any values of free variables in t_{var}. So there is a CPT-term translating the necessary ordinals to bitsets.

If $t = \text{Pair}(r, s)$, define $t_{\text{CPT}} = t_{\text{SHF}}(\text{Pair}(r_{\text{CPT}}, s_{\text{CPT}}))$. If $t = \text{Union}(s)$, then $t_{\text{val}} = \text{Union}(\pi_1(s_{\text{CPT}}))$, and $t_{\text{ann}} = t_{\restriction}(\pi_2(s_{\text{CPT}}), t_{\text{val}})$. The translation for Unique(s) is analogous.

Let $t = \{s(x, \overline{y}) : x \in r(\overline{y}) : \varphi(x, \overline{y})\}$ with $\overline{y} = y_1, \ldots, y_k$. By induction hypothesis, r can be translated to r_{CPT} using t_{var}. The new guard t'_{var} for s and φ should include all possible values of x, so we define $t'_{\text{var}} = t_{\text{var}} \cup \text{Union}(\{r(\overline{y}) : y_1, \ldots, y_k \in t_{\text{var}}\})$. Then the induction hypothesis implies the existence of a term s_{CPT} and a formula φ_{CPT} for s and φ. Then $t_{\text{CPT}} = t_{\text{SHF}}(\{s_{\text{CPT}}(x, \overline{y}) : x \in r_{\text{CPT}}(\overline{y}) : \varphi_{\text{CPT}}(x, \overline{y})\})$.

Let $\varphi = [\mathsf{lrec}_{u,v} t_\delta, \varphi_=, \varphi_E, t_C](x, y)$ be a recursion formula. To evaluate φ, we construct a structure over $[\![t_\delta]\!]^{\mathbf{A},\beta}$ with relations $\sim$, E and C as in the proof of Lemma 6.50. The relations are CPT-definable by the induction hypothesis. Then we evaluate the same simple LREC-formula over that structure. Since LREC $\leq$ FP+C $\leq$ CPT, the resulting simplified formula can be evaluated in CPT.

Let $t = [s_u]^*(x)$ be an iteration term. As explained above, t is evaluated for all possible tuples of free variables (given by t_{var}) at once. More precisely, t_{CPT} defines, in the ith iteration, the set of all triples $\langle a, \text{set}((s^i)^{\mathbf{A},\beta_a}), \text{set}(\bigcup_{j \leq i} (s^i)^{\mathbf{A},\beta_a})\rangle$ for $a \in [\![t_{\text{var}}]\!]^{\mathbf{A}}$, where $\beta_a = \beta[x \mapsto (\pi_1(a), \pi_2(a))]$. By induction hypothesis, there is a translation s_{CPT} of s, so such a term exists. Since there is a term t_{tc} defining the transitive closure of a given set, the term defining the stages of t can be modified such that it replaces the current value by $\emptyset$ whenever the transitive closure exceeds the logarithmic bound.

Further, t_{var} is a CPT-term, so there is a polynomial bound on the number of elements of $t_{\text{var}}^{\mathbf{A}}$. So, because of the logarithmic bound on the values of s^i, which also implies a polynomial bound on the maximal number i of iterations, there is also a polynomial bound on the size of the transitive closure of each stage of t_{CPT} and the number of stages. □

For CLogspace-sentences, the restrictions on the free variables can be omitted. So this shows that every CLogspace-sentence can be translated to a CPT-sentence,

which completes the proof of Theorem 6.51. Altogether, our logic can be seen as the choiceless fragment of LOGSPACE.

6.4.3 Are iteration and recursion necessary?

An obvious question about our formalism is whether it needs both kinds of recursion operators: iteration terms and recursion formulae. We can give a definitive answer for iteration terms, whereas we can only speculate about recursion formulae. The translation to CPT makes it possible to show that iteration is indeed necessary to obtain the full expressive power of CLogspace. Similarly to the proof that CPT needs iteration terms, we show that CLogspace without iteration terms is equivalent to LREC.

With the exception of the Card-operator, every ordinary term can be translated to an ordinary CPT-term t_{val} as in the proof of Lemma 6.56. By Lemma 3.12, these terms can be translated to (FO+H$_\sim$)*-interpretations. LREC has the built-in ability to count equivalence classes of tuples, so these interpretations can also be translated to (LREC+H)*. Here, counting is replaced by the Härtig quantifier to allow for modifications of the number sort by the interpretation. Since counting of tuples can be simulated by the Härtig quantifier analogously to FP+C (see [49]), (LREC+H)* has the same expressive power as LREC.

For translating counting terms, the proof of Lemma 6.56 uses iteration terms to translate ordinals to bitsets. This, however, is already possible with a DTC-interpretation on a structure with a number sort. Since the number sort is linearly ordered, the element relation on bitsets is DTC-definable. So terms with counting can also be simulated by (LREC+H)*-interpretations.

Recursion formulae are essentially LREC-formulae over hereditarily finite sets defined by the domain term t_δ. By induction, there is an (LREC+H)*-interpretation simulating t_δ, so it suffices to evaluate a standard LREC-formula on the interpreted structure. All in all, we have shown the following remark:

Remark 6.57. *The fragment of* CLogspace *without iteration terms is included in* LREC.

Together with the results of the following section, it follows that this fragment is in fact equivalent to LREC.

For the question whether recursion is necessary, we can only provide an intuition as to why there is no obvious way to express recursion formulae with iteration terms. First evidence can be obtained from two examples presented by Theophil

Trippe in his Bachelor's thesis [58]. The first example is a definability result for tree isomorphism in the fragment of CLogspace without recursion formulae. It turns out that the straightforward way to use iteration terms to check whether two trees are isomorphic is to encode pairs of isomorphic subtrees (as pairs of vertices) in the stages of the iteration. Then, algorithmically speaking, the stages encompass multiple branches of the recursion tree. Since this approach exceeds the space bounds on general directed trees, the result is only shown for a certain class of padded trees.

The second example demonstrates that the last step of Reingold's [53] algorithm for symmetric reachability can be expressed in CLogspace. Reingold uses an elaborate reduction to symmetric reachability over graphs with bounded degree and logarithmic diameter. These properties guarantee that every path through the resulting graph can be represented as a sequence of logarithmic length containing statements of the form "the ith successor of the previous vertex", where i is restricted by the bound on the degree. As the motivation behind CLogspace is to allow non-trivial ways to represent atoms, CLogspace-definability of reachability over these graphs is of particular interest. Trippe shows that the last step of Reingold's algorithm is definable in CLogspace, but the natural way to express it uses recursion formulae. It is an open question whether this is possible using only iteration terms.

Note further that CPT does not need a recursion operator because all the branches of the recursive computation—i. e. all paths through the DAG induced by recursion formulae—can be evaluated in parallel. But this means that the stages of the iteration can contain polynomially many paths. This is not possible in CLogspace because of the logarithmic bound. Processing only one path at a time, however, would require an order on the paths.

For recursive computations over (tuples of) atoms, the core of recursion formulae is to express atoms in terms of each other, e. g. as "the ith child of the jth child of the root" or "the kth parent of v". Note that our current formalisation of the terms $\mathrm{Atoms}\,.\varphi$ does not allow that kind of recursive definition of atoms. This is the case because the only free variables of φ are those defining the entries of the tuples in the value of the term. Allowing additional parameters in φ could enable recursion. It is not obvious, though, how to evaluate such a formalism in Logspace. In particular, sets would be represented in a less straightforward way during the evaluation.

Another way to make the two kinds of recursion operators obsolete could be to enhance the capabilities of iteration terms. But some attempts at capturing the kind of recursion that is possible in Logspace again lead to variations of the lrec-operator.

6.4.4 Capturing Logspace on padded structures

A significant strength of CPT is that it benefits from padding of the input structure. More precisely, CPT captures PTIME on a certain class of padded structures. To emphasise the similarities between CPT and CLogspace, we show a capturing result for structures with (even larger) padding for CLogspace. In particular, this result separates CLogspace from other logics in LOGSPACE.

Theorem 6.58. CLogspace *captures* LOGSPACE *on the class of padded structures with* $2^{u!(u^2(3\log u+2)+1)} \leq n$.

The size of the structure is chosen in a way that, analogously to the proof by Blass, Gurevich and Shelah [13] for CPT, we can define the set of all linear orders on the input structure.

Lemma 6.59. *There is a* CLogspace*-term* (orders, f) *that, on every padded structure* $\mathbf{A}$ *with* $2^{u!(u^2(3\log u+2)+1)} \leq n$*, defines the set*

$$\{\{\langle a,b\rangle : a,b \in U^{\mathbf{A}} : a < b\} : < \textit{ is a linear order on } A\}.$$

Proof. Our goal is to use an iteration term that, given a set of linear orders on subsets of $U^{\mathbf{A}}$, extends every linear order by all possible next elements. This term is then iterated starting from the set of all linear orders on two-element subsets until there are no new elements of $U^{\mathbf{A}}$ to add. The following term $\text{ext}(\ell, v)$ extends the linear order ℓ by the element v, which becomes the new largest element:

$$\text{ext}(\ell, v) = \ell \cup \{\langle \pi_1(x), v\rangle : x \in \ell\} \cup \{\langle \pi_2(x), v\rangle : x \in \ell\}.$$

Using ext, we can define the iteration term $\text{extset} = [s_u]^*(x)$, where

$$s(u) = \{\text{ext}(\ell, v) : \ell \in u, v \in \text{Atoms}\,.Uy : \{x : x \in \ell : \pi_1(x) = v \vee \pi_2(x) = v\} = \emptyset\}$$

defines the set of extensions of the linear orders in u by all elements v that do not occur in the respective linear order. When the iteration of extset starts from all two-element linear orders in parallel, the resulting term defines all linear orders on subsets of $U^{\mathbf{A}}$:

$$\text{allorders} = \text{Union}(\{\text{extset}(x) : x \in \{\{y\} : y \in \text{Atoms}\,.(Uz_1 \wedge Uz_2 \wedge z_1 \neq z_2)\}\}).$$

The value of allorders still contains linear orders on strict subsets of $U^{\mathbf{A}}$, because the value of extset incorporates all intermediate stages. So the final step to constructing

the term orders is to select those linear orders that are defined on all of $U^{\mathbf{A}}$. These are exactly the orders ℓ satisfying

$$\varphi_{\mathrm{all}U}(\ell) = \{x : x \in \mathrm{Atoms}\,.Ux : \{y : y \in \ell : \pi_1(y) = x \vee \pi_2(y) = x\} = \emptyset\} = \emptyset,$$

so

$$\mathrm{orders} = \{\ell : \ell \in \mathrm{allorders} : \varphi_{\mathrm{all}U}(\ell)\}.$$

It remains to show that there is a logarithmic bound for orders. The only iteration subterm of orders is extset. Its value in the ith stage is the set of all orders of the form $\{\langle a, b\rangle : a, b \in V : a < b\}$ for subsets $V \subseteq U$ of size $i + 2$.

Recall that the size of an atom depends on the term defining it. The only subterms of orders defining atoms are $\mathrm{Atoms}\,.Ux$ and $\mathrm{Atoms}\,.(Uz_1 \wedge Uz_2 \wedge z_1 \neq z_2)$, so the size annotation $\mathrm{ann}_{s^i}^{\mathbf{A},\beta}$ maps every atom to a number $\leq 2\log|U|$.

Then the pair $\langle a, b\rangle = \{a, \{a, b\}\}$ is mapped to $6\log|U| + 2$. Every linear order contains $< |V|^2 \leq |U|^2$ many pairs of that form, and there are $|V|! \leq |U|!$ many linear orders in the ith iteration.

So $\|s^i\|^{\mathbf{A},\beta} \leq (|U|^2 \cdot (6\log|U| + 2) + 1) \cdot |U|! + 1 =: g(u)$. This is logarithmic in $|A|$ since, by assumption, $|A| \leq 2^{g(u)}$. □

By Lemma 6.39, DTC is weakly closed under CLogspace-interpretations with parameters. So to prove that CLogspace captures Logspace on the class of padded structures with u, n as specified in Theorem 6.58, we provide a CLogspace-interpretation $\mathcal{I}$ that takes a linear order as a parameter and defines the expansion of the underlying structure by that linear order. Formally, $\mathcal{I} = (t_\delta, \varphi_<, (\varphi_R)_{R\in\tau})$ is the interpretation with parameter z where $t_\delta = \mathrm{Atoms}\,.Uy$, $\varphi_=(x_1, x_2) = x_1 = x_2$, $\varphi_<(x_1, x_2) = \langle x_1, x_2\rangle \in z$ and $\varphi_R(x_1, \ldots, x_r) = Rx_1 \ldots x_r$ for $R \in \tau$. Then every DTC-formula can be evaluated on the linear orders obtained from Lemma 6.59, which completes the proof of Theorem 6.58.

6.4.5 Choiceless Logspace is stronger than known logics

We now compare Choiceless Logspace to two logics within Logspace, which have been reviewed in Section 6.1: BDTC, which was previously considered the choiceless fragment of Logspace, and LREC, the strongest logic for Logspace that has been known so far. In the following, we show that, in terms of expressive power, both logics are included in CLogspace. Since BDTC can express neither simple counting

queries [57] nor tree isomorphism, the first inclusion is strict. A separation from LREC is obtained using the result for padded structures.

Let us first consider BDTC.

Theorem 6.60. CLogspace $\gneq$ BDTC.

BDTC can be characterised as DTC over hereditarily finite sets of constant size, where the size of a set is the number of elements of its transitive closure. The different ways to measure sizes pose the main difficulty in translating formulae to CLogspace.

Consider the set $\{a, \{a, b\}\}$, where a and b are atoms. The transitive closure of that set contains four elements. But, if a and b are defined by an FO-formula, their sizes depend on the number of elements satisfying the formula. So a size annotation arising from a CLogspace-term will never assign a constant size to the whole set. This is, however, not problematic, since the sizes of atoms are always at most logarithmic in the size of the structure, so a constant number of atoms can always be defined.

The more severe difficulty is that every element counts only once toward the size of the transitive closure, whereas a size annotation sums over the elements of every set. So, in our example, every size annotation would count the size of a twice. As a first step towards the connection between the two size measures, we formalise the multiplicity of an object in a set.

Definition 6.61. Let $a, b \in \mathrm{HF}(A)$. The *multiplicity* $M(b, a)$ of b in a is defined by induction on a:

- If a is an atom, then $M(a, a) = 1$, and $M(b, a) = 0$ for all $b \neq a$.
- If $a = \{a_1, \ldots, a_k\}$ and $a_1, \ldots, a_k$ are distinct, then $M(a, a) = 1$ and $M(b, a) = \sum_{i=1}^{k} M(b, a_i)$ for $b \neq a$.

Together with the values assigned to the atoms in $\mathrm{tc}(a)$, the multiplicities of the elements of $\mathrm{tc}(a)$ uniquely determine every size annotation of $a \in \mathrm{HF}(A)$.

Claim 6.62. *If s is a size annotation of $a \in \mathrm{HF}(A)$, then*

$$s(a) = \sum_{b \in \mathrm{tc}(a) \cap A} s(b) M(b, a) + \sum_{b \in \mathrm{tc}(a) \setminus A} M(b, a).$$

Proof.

- If $a \in A$, then $s(a)M(a, a) = s(a)$ by definition of $M(a, a)$.

◇ If $a = \{a_1, \dots, a_k\}$, then

$$\begin{aligned} s(a) &= \sum_{i=1}^{k} s(a_i) + 1 \\ &\overset{\text{ind. hyp.}}{=} \sum_{i=1}^{k} \sum_{b \in \operatorname{tc}(a_i) \cap A} s(b) M(b, a_i) + \sum_{i=1}^{k} \sum_{b \in \operatorname{tc}(a_i) \setminus A} M(b, a_i) + 1 \\ &= \sum_{b \in \operatorname{tc}(a) \cap A} s(b) M(b, a) + \sum_{i=1}^{k} \sum_{b \in \operatorname{tc}(a_i) \setminus A} M(b, a_i) + M(a, a) \\ &= \sum_{b \in \operatorname{tc}(a) \cap A} s(b) M(b, a) + \sum_{b \in \operatorname{tc}(a) \setminus A} M(b, a). \end{aligned}$$ □

The claim implies that a bound on the multiplicities of all objects in the transitive closure of a set also yields a bound on size annotations of a, whenever there is a reasonable bound on the sizes of atoms. So we derive a bound on the multiplicity of hereditarily finite objects that will suffice for bounded sets as they are used in BDTC.

Claim 6.63. *For $a, b \in \operatorname{HF}(A)$, $M(b, a) \leq |\operatorname{tc}(a)|^{\operatorname{rk}(a)}$.*

Proof.

◇ If a is an atom, then $M(b, a) \leq 1$.
◇ If $a = \{a_1, \dots, a_k\}$, then

$$\begin{aligned} M(b, a) = \sum_{i=1}^{k} M(b, a_i) &\leq \sum_{i=1}^{k} |\operatorname{tc}(a_i)|^{\operatorname{rk}(a_i)} \\ &\leq \sum_{i=1}^{k} |\operatorname{tc}(a)|^{\operatorname{rk}(a)-1} \leq |\operatorname{tc}(a)| \cdot |\operatorname{tc}(a)|^{\operatorname{rk}(a)-1}. \end{aligned}$$ □

Some BDTC-terms have explicit constant bounds, where the semantics changes if the transitive closure of the value of such a term would exceed the bound. To evaluate such terms, we define the transitive closure of their preliminary value in CLogspace.

Lemma 6.64. *For every $c \in \mathbb{N}$, there is a* CLogspace*-term $(t_{\operatorname{tc}}, f)$ with free variable x such that, for every $\beta \colon x \mapsto (a, s)$,*

◇ $[\![(t_{\operatorname{tc}}, f)]\!]^{\mathbf{A}, \beta} \in \{\operatorname{tc}(a), \emptyset\}$, *and*
◇ *if $|\operatorname{tc}(a)| \leq c$ and $s(b) \leq \log |A|$ for all $b \in \operatorname{tc}(a) \cap A$, then $[\![(t_{\operatorname{tc}}, f)]\!]^{\mathbf{A}, \beta} = \operatorname{tc}(a)$.*

Proof. Use the iteration term $[t_u]^*(x)$ with $t(u) = u \cup \mathrm{Union}(u)$, and the function $f = n \mapsto c^c \log n$. So $t_{\mathrm{tc}}(x) = \mathrm{Union}([t_u]^*(\{x\}))$. The value at every stage is a subset of $\mathrm{tc}(a)$. So, assuming $|\mathrm{tc}(a)| \leq c$ and $s(b) \leq \log |A|$ for every $b \in A$, Claims 6.62 and 6.63 imply that f bounds the size of every stage i with respect to $\mathrm{ann}_{t^i}^{\mathbf{A},\beta}$. □

We use the term for the transitive closure to translate BDTC to CLogspace.

Lemma 6.65.

1. *For every* BDTC*-term* t *with free variables* $x_1, \ldots, x_k$*, there is a* CLogspace*-term* t_{CL} *such that, for every* σ*-structure* $\mathbf{A}$ *and every assignment* $\beta \colon x_i \mapsto (a_i, s_i)$ *with* $s_i(b) \leq \log |A|$ *for all* $b \in \mathrm{tc}(a_i) \cap A$*,* $[\![t_{\mathrm{CL}}]\!]^{\mathbf{A},\beta} = t^{\mathbf{A}}(a_1, \ldots, a_k)$*.*
2. *For every* BDTC*-formula* φ *with free variables* $x_1, \ldots, x_k$*, there is a* CLogspace*-formula* φ_{CL} *such that, for every* σ*-structure* $\mathbf{A}$ *and every assignment* $\beta \colon x_i \mapsto (a_i, s_i)$ *with* $s_i(b) \leq \log |A|$ *for all* $b \in \mathrm{tc}(a_i) \cap A$*,* $\mathbf{A}, \beta \models \varphi_{\mathrm{CL}}$ *if, and only if,* $\mathbf{A} \models \varphi(a_1, \ldots, a_k)$*.*

Proof. We proceed by induction on the terms and formulae of BDTC. For $t = \mathrm{Atoms}$, set $t_{\mathrm{CL}} = \mathrm{Atoms}\,.\,\mathrm{true}$. Variables, constants (including $\emptyset$), and applications of functions (including Unique), are translated directly. The same is true for Boolean combinations of formulae.

For $t = \{s_1, \ldots, s_k\}_c$, note that there is a CLogspace-term t_{pre} defining $\{s_1, \ldots, s_k\}$. To ensure the constant bound, let

$$\varphi = (t_{\mathrm{pre}} \neq \emptyset \to t_{\mathrm{tc}}(t_{\mathrm{pre}}) \neq \emptyset) \wedge \mathrm{Card}(t_{\mathrm{tc}}(t_{\mathrm{pre}})) \leq \mathrm{bitset}(c).$$

Then φ verifies that the value of the term t_{tc} from Lemma 6.64 has not been set to $\emptyset$ for size reasons, and the transitive closure computed is not too large. Note that the order $\leq$ on bitsets is definable. The desired term is $t_{\mathrm{CL}} = \mathrm{if}_{\varphi}(t_{\mathrm{pre}}, \emptyset)$.

With the technique from the previous case for the constant bound, terms of the form $t = \{s : x \in r : \varphi\}_c$ can be translated directly.

For the case $\varphi = [\mathsf{dtc}_{\overline{uv}}\psi]_c(\overline{x}, \overline{y})$, note that the deterministic transitive closure can be evaluated by an iteration term $[t_u]^*$, where

$$t(u) = \mathrm{Unique}(\{v : v \in t^k_{\mathrm{guard}} : \psi_{\mathrm{CL}}(\pi_1(u), \ldots, \pi_k(u), \pi_1(v), \ldots, \pi_k(v))\}).$$

The term t^k_{guard} should define the set of all possible successors, i. e. the set $(A \cup \mathrm{HF}_c(A))^k$, where k is the length of $\overline{u}$. By Lemma 5.4.3 in [57], a term defining $A \cup \mathrm{HF}_c(A)$ exists in the fragment of BDTC without dtc-operators, so it can already be translated to CLogspace using the previous cases. □

To see that the inclusion is strict, note that, by Theorem 6.1.4 in [57], the parity of a definable subset of padded structures is not BDTC-definable. This query can, however, easily be defined in CLogspace using the cardinality operator.

Though this already illustrates that CLogspace benefits from padding whereas BDTC does not, this particular separation could already be mitigated by adding a counting operation to BDTC.

To see that the enhanced expressive power is not just due to counting, consider the fact that BDTC can be embedded into purple (this is Lemma 6.16). Thus BDTC cannot express the tree isomorphism query, even on those padded structures where CLogspace captures LOGSPACE.

To complete our comparison of CLogspace to other logics, we show that it is strictly more expressive than LREC. Because of the lrec-operator in CLogspace, proving the inclusion simply amounts to showing that our technical adjustments do not reduce the expressive power. The inclusion is strict because of the capturing result on padded structures.

Theorem 6.66. CLogspace $\gneq$ LREC.

As an extension of FO+C, LREC is evaluated over structures with a number sort. Further, the number sort is used extensively in the definition of the lrec-operator itself. CLogspace, on the other hand, uses the bitset representation instead of a distinguished number sort.

To define a notion of translation for formulae with free variables, we therefore define a translation from variable assignments with number variables to those using size-annotated hereditarily finite objects.

Let $X = \{x_1, \dots, x_k, \lambda_1, \dots, \lambda_\ell\}$, where $x_1, \dots, x_k$ are domain variables, and $\lambda_1, \dots, \lambda_\ell$ are number variables. For every assignment $\beta \colon X \to A \cup [|A|]$ for some σ-structure $\mathbf{A}$, we define the set B_{SHF} of assignments $\beta_{\mathrm{SHF}} \colon X \to \mathrm{SHF}(A)$ with the following property:

For $1 \leq i \leq k$, $\beta_{\mathrm{SHF}}(x_i) = (\beta(x_i), s_i)$ for some size annotation s_i, and for $1 \leq i \leq \ell$, $\beta_{\mathrm{SHF}}(\lambda_i) = (\mathrm{bitset}(\beta(\lambda_i)), \mathrm{ann}_{\mathrm{bitset}(\beta(\lambda_i))})$. Since the size annotations for the variables x_i are not unique, we show the translation for all assignments in the set B_{SHF}.

Lemma 6.67. *For every* LREC*-formula* φ *with* $\mathrm{free}(\varphi) \subseteq X$ *(with* X *defined as above), there is a* CLogspace*-formula* $(\varphi_{\mathrm{CL}}, f)$ *such that, for every* σ*-structure* $\mathbf{A}$*, every assignment* $\beta \colon X \to A \cup [|A|]$*, and every* $\beta_{\mathrm{SHF}} \in B_{\mathrm{SHF}}$*, it holds that* $\mathbf{A}, \beta \models \varphi$ *if, and only if,* $\mathbf{A}, \beta_{\mathrm{SHF}} \models \varphi_{\mathrm{CL}}$.

Proof. To handle the encoding of tuples of numbers as single numbers, we use, for every $\ell \geq 1$, a term t^ℓ_{enc} with free variables $x_1, \dots, x_\ell$ such that, if $\beta(x_i) = (\text{bitset}(\lambda_i), s_i)$, then $[\![t^\ell_{\text{enc}}]\!]^{\mathbf{A},\beta} = \text{bitset}(\text{enc}(\lambda_1, \dots, \lambda_\ell))$. By Lemma 6.41, such terms exist.

We translate LREC-formulae inductively, considering only the non-trivial cases.

- If $\varphi = \lambda \leq \mu$ for number variables λ, μ, let $\varphi_{\text{CL}} = \langle \lambda, \mu \rangle \in \text{order}_{\text{Atoms}\,.\,\text{true}}$, where $\text{order}_{\text{Atoms}\,.\,true}$ is the term from Lemma 6.35 defining the linear order on bitsets up to $|A|$.
- For a counting formula $\varphi = \#\overline{u}\psi = \overline{\lambda}$, let t_{var} be a term defining all tuples $\overline{u}$ with $u_i \in \text{Atoms}\,.\,\text{true}$ if u_i is a domain variable, and $u_i = \pi_1(x)$ for some $x \in \text{numbers}_{\text{Atoms}\,.\,\text{true}}$ if u_i is a number variable (where $\text{numbers}_{\text{Atoms}\,.\,\text{true}}$ is the term from Corollary 6.36 defining the bitsets up to $|A|$). Then

$$\varphi_{\text{CL}} = \text{Card}(\{u : u \in t_{\text{var}} : \psi_{\text{CL}}(\pi_1(u), \dots, \pi_k(u))\}) = t^\ell_{\text{enc}}(\lambda_1, \dots, \lambda_\ell),$$

  where ℓ is the length of the tuple $\overline{\lambda}$, and k is the length of $\overline{u}$.
- If $\varphi = \exists x \psi$, then let $\varphi_{\text{CL}} = \{x : x \in \text{Atoms}\,.\,\text{true} : \psi_{\text{CL}}(x)\} \neq \emptyset$.
- Let $\varphi = [\mathsf{lrec}_{\overline{u},\overline{v},\overline{\lambda}}\varphi_=, \varphi_E, \varphi_C](\overline{x}, \overline{\mu})$, where $\overline{u}, \overline{v}$ and $\overline{x}$ are tuples of length k. Then φ is translated to

$$\varphi_{\text{CL}} = [\mathsf{lrec}_{u,v} t_\delta, (\varphi_=)_{\text{CL}}, (\varphi_E)_{\text{CL}}, t_C](x, y),$$

  where t_δ defines that u is of the form $\langle u_1, \dots, u_k \rangle$ with each u_i of the correct type (number or domain variable), and t_C defines the set of encodings of tuples satisfying φ_C, i. e.

$$t_C = \{t^\ell_{\text{enc}}(\lambda_1, \dots, \lambda_\ell) : \lambda_1, \dots, \lambda_\ell \in \text{numbers}_{\text{Atoms}\,.\,\text{true}} : (\varphi_C)_{\text{CL}}(\pi_1(u), \dots, \pi_k(u), \lambda_1, \dots, \lambda_\ell)\}.$$

Note that the only iteration terms occurring as subterms of the formulae φ_{CL} are those that define the linear order and tuple encoding on bitsets. In particular, the size annotations of the free variables do not influence the size of the stages. So there is an appropriate bound f for every φ such that φ_{CL} is equivalent to (φ_{CL}, f). □

So we have shown that LREC is included in CLogspace. To separate CLogspace from LREC, we use that LREC is included in FP+C [34]. FP+C does not benefit from padding, i. e. the proof that the Cai-Fürer-Immerman query is not FP+C-definable survives in the presence of padding. But the CFI query is computable

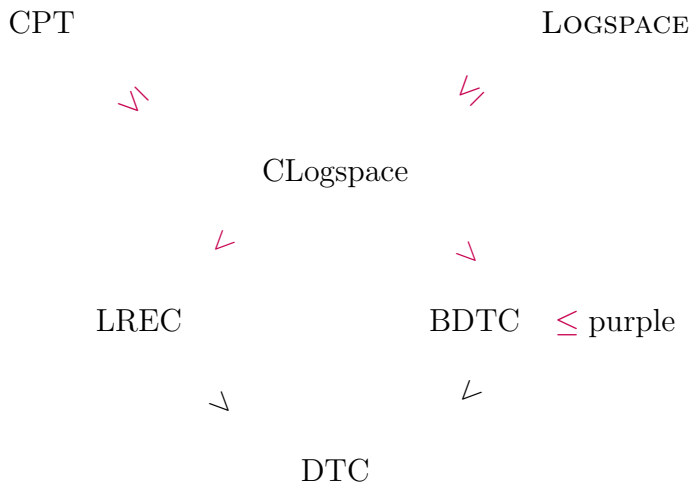

Figure 6.1: The expressive power of CLogspace compared to other logics within Logspace. The highlighted inclusions follow from our results.

in Logspace (see Section 5.1). So by Lemma 6.59 it is CLogspace-definable on a class of padded structures where it is inexpressible by FP+C and thus LREC.

The results of this section are summed up in Figure 6.1.

6.5 Conclusion

We presented a formalism that serves as our working definition of Choiceless Logarithmic Space, i. e. choiceless computation for Logspace. In comparison to Ptime, Logspace is sensitive to the encoding of elements of the input structure, whereas choiceless computation should be oblivious to encoding. Our logic addresses that problem with a semantics based on size-annotated hereditarily finite sets. Using size annotations in the semantics takes into account that atoms can be represented in multiple ways, in particular using logical formulae.

Several results about the expressive power of our logic justify calling it Choiceless Logarithmic Space: First of all, there is an effective translation from formulae to Logspace-algorithms, which verifies one of the conditions for a logic capturing Logspace. Further, our logic is contained in Choiceless Polynomial Time. Analogously to CPT, it captures Logspace on certain classes of padded structures. This also separates it from logics in the classical sense, in particular LREC.

It is, however, still open whether CLogspace is closed under interpretations, a criterion that has been successfully applied to both LREC and BDTC. For the class Logspace, closure under interpretations is of particular importance because

Logspace algorithms are closed under sequential execution. That is, running a Logspace algorithm on the (polynomially-sized) output of another Logspace algorithm is again in Logspace. But the atomic entities a CLogspace-formula operates on are the elements of definable sets instead of single bits. In particular, the domain of an interpreted structure is a set whose elements are not necessarily of logarithmic size, so they can in general not occur as the stages of an iteration term. On the one hand, it seems vital to permit sets of polynomial size to mimic sequential execution of Logspace-algorithms. On the other hand, these sets prevent the trivial approach to closure under interpretations—at least for our definition of interpretations.

Hence our first open question is whether CLogspace is closed under interpretations (with respect to our definition of CLogspace-interpretations). In case the answer is negative, this could point to a weakness either in the definition of interpretations, or in the definition of CLogspace itself. While a non-standard definition of interpretations is necessary to correctly handle size annotations, it seems possible to narrow it down to the case where the domain of the interpreted structure consists of equivalence classes of tuples over the input structure. Further, a simple but somewhat artificial solution would be to permit only iteration terms as domain terms.

Concerning the definition of CLogspace, the original, more algorithmic version of BDTC [29] hints at a modification of the formalism that could yield closure under interpretations. A program in that model of choiceless computation is a sequence of subprograms, where each subprogram operates on a fresh copy of the structure computed by the preceding program. As the size bounds in BDTC do not depend on the input structure, this can be simulated by nested dtc-operators. But for CLogspace, starting the computation from a fresh structure would mean that the size annotations would depend solely on that structure. In particular, this could strengthen terms of the form "Atoms $.\varphi$", as the atoms in later computation steps would be obtained from higher-order objects over the original input structure. A similar effect might be achieved by the hierarchy of CLogspace-extensions suggested by Trippe [58], where, starting from CLogspace itself, every stage is obtained by permitting formulae from the previous stage in any term "Atoms $.\varphi$". It is open whether these extensions lead to greater expressive power, or, equivalently, whether the hierarchy collapses.

Closure under interpretations in a more classical sense, i. e. with equivalence classes of tuples as new domain elements, is closely linked to symmetric transitive closures. For a straightforward proof of closure under that kind of interpretations,

equivalence classes of tuples should be sets of logarithmic size. A starting point may be to allow STC-formulae as guards in the "Atoms $.\varphi$"-terms. But this would only lead to a concise representation of every atom in an equivalence class instead of the equivalence class itself. This is the case because our logic is based on the assumption that the only formulae occurring in the encoding of a set define single atoms. Assuming a different encoding of sets seems like a promising direction for future research. However, it is not clear how such a formalism can be evaluated in Logspace.

Definability of symmetric transitive closures is significant beyond closure under interpretations: Though the problem is computable in Logspace, Reingold's algorithm is highly non-trivial. Moreover, it demonstrates that Logspace algorithms depend to a great extent on an appropriate encoding of atoms. The only known way to express symmetric transitive closures in CLogspace is using the stc-operator built into recursion formulae. If Reingold's algorithm can be implemented in CLogspace, this would indicate that the logic successfully simulates the encoding of computations used in the algorithm. However, the straightforward way to express the last step of the Reingold algorithm (see [58]) is based on the recursion operator. It is open whether this step, or even the whole algorithm, can be simulated using iteration only.

This would also answer the question whether the two kinds of recursion used in CLogspace are necessary. The intuition is that the lrec-operator permits recursive definitions of objects, without the constant bound imposed on the depth of the recursion by the number of free variables in "Atoms $.\varphi$"-terms. But it remains open whether this intuition can be turned into a formal argument. If the recursion operator cannot be omitted, it would be desirable to rewrite it in a way that it extends the core mechanisms of CLogspace in a more natural way. This could be achieved either by extending iteration terms to a more powerful recursion mechanism, or by changing the assumptions about how atoms are represented. Recursive definitions of atoms can be made possible by allowing additional free variables as parameters in the formula φ in Atoms $.\varphi$. Similarly to alternative representations of sets, though, it is probably not trivial to encode such recursive definitions in logarithmic space.

We can conclude that the definition of CLogspace alone leaves room for further research. The components whose adaption seems to be most influential are the terms defining atoms and the underlying assumption about representations of hereditarily finite sets.

The reason why the definition of CLogspace may still be subject to modifications is the central open question about this logic: Can it express all queries computable

in logarithmic space? In terms of expressibility, natural problems in Logspace, like symmetric transitive closure or tree isomorphism, are already covered by the lrec-operator. Analogously to the first results about CPT, we know that the padded version of the Cai-Fürer-Immerman query is expressible in CLogspace. So a natural question is whether the CFI query over linearly ordered graphs, and finally over general graphs, is CLogspace-definable as well. The known CPT-algorithm for the CFI query over ordered graphs, however, does not suffice for CLogspace, as it uses sets containing linearly many atoms.

Inexpressibility proofs for CLogspace constitute an additional challenge. Inexpressibility results for fragments of CPT are shown by characterising the kind of sets that can be defined by the fragment considered. But in CLogspace, it is possible that a set is definable with respect to one size annotation but not another, which may further complicate such a classification. Nevertheless, the definition of size annotations implies that the stages of iteration terms have to be of logarithmic rank. Since the fragment of CLogspace without iteration terms can be simulated in FP+C, it might be sufficient to consider pebble games over sets of logarithmic rank—assuming these sets can be characterised in a meaningful way, for instance via their supports.

A more unified definition of CLogspace, in particular with only one type of recursion, may already remove some of the obstacles that currently complicate the analysis of CLogspace. In particular, a characterisation of CLogspace in terms of interpretation logic may lead to new insights about its structure and expressive power. Contrastingly, strengthening the definition, even without a formal justification through inexpressible queries, also seems desirable if this leads to new positive results.

We conclude with a question that is independent of the expressive power of CLogspace itself: Is there a nondeterministic extension of CLogspace that can serve as a candidate for a logic capturing nondeterministic Logspace? Iteration terms can be viewed as a way to define deterministic transitive closures over hereditarily finite sets. This can be generalised to arbitrary transitive closures by choosing the next stage nondeterministically. It is, however, not clear whether a true nondeterministic variant of CLogspace would additionally require a nondeterministic lrec-operator. Similarly to the case of CLogspace itself, the question is how choiceless nondeterministic Logspace can be formalised in a natural way.

Bibliography

[1] Serge Abiteboul and Victor Vianu. Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 1989)*, pages 71–79. IEEE Computer Society Press, 1989.

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

[3] Faried Abu Zaid, Anuj Dawar, Erich Grädel, and Wied Pakusa. Definability of summation problems for Abelian groups and semigroups. In *Proceedings of the Thirty second Annual IEEE Symposium on Logic in Computer Science (LICS 2017)*, pages 1–11. IEEE Computer Society Press, 2017.

[4] Faried Abu Zaid, Erich Grädel, Martin Grohe, and Wied Pakusa. Choiceless Polynomial Time on structures with small Abelian colour classes. In *Mathematical Foundations of Computer Science 2014*, volume 8634 of *LNCS*. Springer, 2014.

[5] Matthew Anderson and Anuj Dawar. On symmetric circuits and fixed-point logics. *Theory of Computing Systems*, 60(3), 2017.

[6] Matthew Anderson, Anuj Dawar, and Bjarki Holm. Maximum matching and linear programming in fixed-point logic with counting. In *Proceedings of the Twenty-Eighth Annual IEEE Symposium on Logic in Computer Science (LICS 2013)*, pages 173–182. IEEE Computer Society Press, 2013.

[7] Matthew Anderson, Anuj Dawar, and Bjarki Holm. Solving linear programs without breaking abstractions. *Journal of the ACM (JACM)*, 62(6), 2015.

[8] Albert Atserias, Andrei Bulatov, and Anuj Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18), 2009.

[9] Andreas Blass and Yuri Gurevich. Choiceless polynomial time computation and the zero-one law. In *International Workshop on Computer Science Logic*. Springer, 2000.

[10] Andreas Blass and Yuri Gurevich. Strong extension axioms and Shelah's zero-one law for choiceless polynomial time. *The Journal of Symbolic Logic*, 68(1), 2003.

[11] Andreas Blass and Yuri Gurevich. A new zero-one law and strong extension axioms. In *Current Trends in Theoretical Computer Science*. World Scientific, 2004.

[12] Andreas Blass and Yuri Gurevich. A quick update on the open problems in Blass-Gurevich-Shelah's article "On polynomial time computations over unordered structures", 2005.

[13] Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1-3), 1999.

[14] Andreas Blass, Yuri Gurevich, and Saharon Shelah. On polynomial time computation over unordered structures. *The Journal of Symbolic Logic*, 67(3), 2002.

[15] Andreas Blass, Yuri Gurevich, and Jan van den Bussche. Abstract state machines and computationally complete query languages. In *International Workshop on Abstract State Machines*. Springer, 2000.

[16] Mikołaj Bojańczyk and Szymon Toruńczyk. On computability and tractability for infinite sets. In *Proceedings of the Thirty third Annual IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 145–154. IEEE Computer Society Press, 2018.

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

[18] Ashok Chandra and David Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1), 1982.

[19] Derek Corneil and Mark Goldberg. A non-factorial algorithm for canonical numbering of a graph. *Journal of Algorithms*, 5(3), 1984.

[20] Laszlo Csanky. Fast parallel matrix inversion algorithms. In *16th Annual Symposium on Foundations of Computer Science*. IEEE, 1975.

[21] Anuj Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1), 2015.

[22] Anuj Dawar, Martin Grohe, Bjarki Holm, and Bastian Laubner. Logics with rank operators. In *Proceedings of the Twenty-Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 2009)*, pages 113–122. IEEE Computer Society Press, 2009.

[23] 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), 2008.

[24] Kousha Etessami and Neil Immerman. Tree canonization and transitive closure. In *Proceedings of the Tenth Annual IEEE Symposium on Logic in Computer Science (LICS 1995)*, pages 331–341. IEEE Computer Society Press, 1995.

[25] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 1974.

[26] Erich Grädel and Martin Grohe. Is Polynomial Time Choiceless? In *Fields of Logic and Computation II.*, volume 9300 of *LNCS*. Springer, 2015.

[27] Erich Grädel, Łukasz Kaiser, Wied Pakusa, and Svenja Schalthöfer. Characterising Choiceless Polynomial Time with first-order interpretations. In *Proceedings of the Thirtieth Annual IEEE Symposium on Logic in Computer Science (LICS 2015)*, pages 677–688. IEEE Computer Society Press, 2015.

[28] Erich Grädel and Wied Pakusa. Rank logic is dead, long live rank logic! *CoRR (a conference version appeared in the proceedings of CSL'15)*, abs/1503.05423, 2015.

[29] Erich Grädel and Marc Spielmann. Logspace Reducibility via Abstract State Machines. In J. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods (FM '99)*, volume 1709 of *LNCS*. Springer, 1999.

[30] Martin Grohe. Fixed-point logics on planar graphs. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS 1998)*, pages 6–15. IEEE Computer Society Press, 1998.

[31] Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *Journal of the ACM (JACM)*, 59(5), 2012.

[32] Martin Grohe, Berit Grußien, André Hernich, and Bastian Laubner. L-recursion and a new logic for logarithmic space. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 12. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2011.

[33] Martin Grohe and Julian Mariño. Definability and descriptive complexity on databases of bounded tree-width. In *International Conference on Database Theory*. Springer, 1999.

[34] Berit Grußien. *Capturing Polynomial Time and Logarithmic Space using Modular Decompositions and Limited Recursion*. PhD thesis, Humboldt-Universität zu Berlin, 2017.

[35] Yuri Gurevich. Logic and the challenge of computer science. In *Current Trends in Theoretical Computer Science*. Computer Science Press, 1988.

[36] Yuri Gurevich and Saharon Shelah. On finite rigid structures. *The Journal of Symbolic Logic*, 61(2), 1996.

[37] Wilfrid Hodges. *Model theory*, volume 42. Cambridge University Press, 1993.

[38] Martin Hofmann, Ramyaa Ramyaa, and Ulrich Schöpp. Pure pointer programs and tree isomorphism. In *International Conference on Foundations of Software Science and Computational Structures*, pages 321–336. Springer, 2013.

[39] Martin Hofmann and Ulrich Schöpp. Pure pointer programs with iteration. *ACM Transactions on Computational Logic (TOCL)*, 11(4):26, 2010.

[40] Bjarki Holm. *Descriptive complexity of linear algebra*. PhD thesis, University of Cambridge, 2010.

[41] Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[42] Neil Immerman. *Expressibility as a complexity measure: results and directions*. Yale University, Department of Computer Science, 1987.

[43] Neil Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.

[44] Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*. Springer, 1990.

[45] Stephan Kreutzer. Expressive equivalence of least and inflationary fixed-point logic. *Annals of Pure and Applied Logic*, 130(1-3), 2004.

[46] Michal Krynicki. On some applications of games for Härtig quantifier. *Mathematical Logic Quarterly*, 33(4), 1987.

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

[48] Steven Lindell. A logspace algorithm for tree canonization. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 400–404. ACM, 1992.

[49] Martin Otto. *Bounded variable logics and counting*, volume 9. Cambridge University Press, 2017.

[50] Wied Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen University, 2016.

[51] Wied Pakusa, Svenja Schalthöfer, and Erkal Selman. Definability of Cai-Fürer-Immerman problems in choiceless polynomial time. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

[52] Wied Pakusa, Svenja Schalthöfer, and Erkal Selman. Definability of Cai-Fürer-Immerman problems in choiceless polynomial time. *ACM Transactions on Computational Logic (TOCL)*, 19(2), 2018.

[53] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.

[54] Benjamin Rossman. Choiceless computation and symmetry. In *Fields of Logic and Computation*, LNCS. Springer, 2010.

[55] Svenja Schalthöfer. Computing on abstract structures with logical interpretations. Master's thesis, RWTH Aachen University, 2013.

[56] Saharon Shelah. Choiceless polynomial time logic: inability to express. In *International Workshop on Computer Science Logic*. Springer, 2000.

[57] Marc Spielmann. *Abstract state machines: Verification problems and complexity*. PhD thesis, RWTH Aachen University, 2000.

[58] Theophil Trippe. Structure and expressive power of Choiceless Logspace. Bachelor's thesis, RWTH Aachen University, 2018.

[59] Moshe Vardi. The complexity of relational query languages. In *STOC '82*, pages 137–146. ACM Press, 1982.