Expressibility and Tractability

by

Richard Gault

A thesis submitted for the degree of PhD
at the University of Leicester

September 2000

Expressibility And Tractability

Richard Gault

Abstract

This thesis is composed of three separate, yet related strands. They have in common the notion that computational problems are regarded not as sets of strings, but as classes of finite structures. Our "computing devices", be they of a logical, traditionally computational, or algebraic nature, all work directly upon such structures. This is in contrast to traditional computability and complexity theory, where machines act instead upon some encoding of structures.

We begin by investigating a restriction of the question of whether or not **NP** = **co-NP**. In particular, we consider the effect of adding a transitive closure operator to monadic **NP**, and show that the resulting logic is a strict extension of it which is not closed under complementation. This extends Fagin's result that monadic **NP** is itself not closed under complementation.

We then investigate the expressive power of a class of program schemes which we call RFDPS. We prove a strong result limiting the expressive power of this class, and use it to obtain a strict, infinite hierarchy of problem classes within RFDPS. To our knowledge, this is the first strict, infinite hierarchy in a polynomial-time logic which properly extends inductive fixed-point logic (with the property that the union of the classes of the hierarchy consists of the class of problems definable in the polynomial-time logic itself).

Finally, we turn our attention to constraint satisfaction problems. This important class of problems is **NP**-hard in general, but many restrictions to it have been identified over the years which ensure its tractability. We introduce a method of combining two tractable classes over disjoint domains, so as to synthesise new tractable classes. We demonstrate that these new classes are genuinely novel, and extend naturally to yet further tractable classes. The algorithms for solving these extended classes can be less than obvious.

Acknowledgements

First and foremost, I would like to thank my supervisor, Iain Stewart, for his continued help, support, and encouragement during the writing of this thesis. Without his guidance, it could never have been written. Many thanks are also due to Peter Jeavons and David Cohen, who have taught me all that I know of constraint satisfaction problems. Chapter 5 was written whilst working in collaboration with these researchers, and I am grateful to them for their permission to include it in this thesis. The constraint satisfaction puzzle which appears in Chapter 1 is due to Paul Roberts.

I have moved around a good deal during the writing of this thesis. I would like to thank the staff and students at each institution I attended for providing me with such an enjoyable and stimulating environment in which to work. In particular, special mention must be made of the members of the Theory Lab at the University of Wales College of Swansea: Yaagoub Ashir, Tom Atkiss, Savita Chauhan, Anuj Dawar, Anthony Fox, Henrik Imhof, Dafydd Rees, and Kristian Stewart; of Matthew Baker and Charles Eaton at the University of Leicester; of Joe Bater at Royal Holloway College, University of London; and of Andrei Bulatov and Andrei Krokhin at the University of Oxford.

Many people have read through various parts of my thesis at one time or another, and have offered useful comments. Apart from the people mentioned above, I would like to thank Rick Thomas and Simon Ambler, who read through preliminary versions of parts of Chapters 3 and 4; along with Florent Madelaine, Victor Dalmau, and Justin Pearson, who read through various versions of Chapter 5.

The unconditional support of my family, and of Kate has been invaluable. Financially, I must thank the EPSRC, whose grants 969035068 and GR/M 12933 have supported me over the past few years.

To those whom I have inevitably forgotten: my apologies. It is impossible to remember everyone who has helped to shape this thesis.

Finally, I would like to thank the participants of BCTCS 14 at St Andrews, for not asking too many awkward questions...

Contents

Chapter 1

Introduction

Finite model theory is the study of the logical properties of finite mathematical structures.

Model theory in general dates back to at least 300BC, when Euclid wrote *The Elements.* In it, he stated five "self-evident" postulates, and used them to systematically deduce a large number of propositions of geometry. However, neither Euclid himself, nor the many mathematicians who came after him, were ever truly satisfied with his fifth postulate: the notorious "parallel lines postulate". Many people tried to deduce the fifth postulate from the other four; all failed. Several investigators, including Saccheri, Lambert, and Legendre experimented with assuming that the postulate was false. However, they did so in the hope of obtaining a contradiction to one of the first four postulates. It does not seem to have occurred to anyone before Gauss that the fifth postulate might actually be independent of the other four. In modern terminology, we would say that all the early researchers assumed that the set of models of the first four postulates is equal to the set of models of all five postulates; that is, that Euclidean geometry is the only possible geometry. (Indeed, Kant described Euclidean geometry as "the inevitable necessity of thought.")

Gauss never published his findings, and it was left to Bolyai and Lobachevsky to publish the first non-Euclidean geometries in the first half of the nineteenth century (though it was not until 1868 that Beltrami proved that the fifth axiom was independent of the others, by constructing a model for the Bolyai-Lobachevsky geometry within three-dimensional Euclidean geometry).

Since those early days, model theory has flourished (see [14]), but until fairly recently, *finite* model theory has remained largely unexplored. Partly, this is because many staple tools of model theory fail to work in the finite case. The completeness theorem of first-order logic, for example, and the compactness theorem both break down when attention is restricted to finite structures. There were a few early results in finite model theory: Trakhtenbrot's Theorem dates from 1950 for example [84], but they tended to be few and far between and, in any case, to be rather negative results.

There are very few results which hold in both the finite and the general case. The theory of Ehrenfeucht-Fraïssé games is one of the few exceptions, and Fagin made heavy use of a variant of these when he showed in the early 1970s that so-called monadic **NP** is not closed under complementation [33]. This result is of significance because it is at least a small step in the direction of proving that the whole of **NP** is not closed under complementation; that is, that **NP** $\neq$ **co-NP**. However, extending his result even to binary **NP** has so far met with little success.

In the first part of this thesis, we extend Fagin's work on monadic **NP** by considering an extension to this logic which is, firstly, a provably strict extension, and secondly, not closed under complementation. More specifically, we add to monadic **NP** the ability to compute the transitive closure of a relation, and show that the problem NON-CONNECTIVITY of undirected graphs is not expressible in the resulting logic.

Although we have been unable to generalise this new result in any significant way, the first part of this thesis concludes with several ideas for possible extensions to the result, as well defining as a tool which may, perhaps, be used to this end.

The next part of the thesis concerns program schemes. Program schemes originated in the seventies and were extensively studied then, although it was not until the mid-to-late eighties that their computational complexity was examined in depth. In [75], Stewart further developed the concept by defining some classes of program schemes which operated on finite, ordered structures; that is, which took such structures as inputs. He was able to demonstrate an intimate tie-up between these classes and finite model theory; in particular he showed that each class he defined had exactly the same expressive power as some natural extension of first-order logic. More recently, Arratia-Quesada, Chauhan, and Stewart have generalised Stewart's constructions [6], and have used them to define proper hierarchies within transitive closure logic and path system logic. Interestingly, the proofs of these results were established without the authors having to play any form of Ehrenfeucht-Fraïssé game: usually the most valuable tool available for proving inexpressibility results.

Such a result should be motivation enough for studying program schemes, but they are intrinsically interesting in their own right because of the curious position they occupy in the world of theoretical computer science. Half-way between logical formulae and high-level programs, their connection to logic means that they can often be easier to reason about than traditional high-level languages [57, 64], yet it is normally simpler to write code for program schemes than to construct a sentence of some logic. In addition, there are tools and paradigms available to the programmer (such as the use of stacks [6] or parallel architectures) which cannot easily be translated into the language of the logician. Because they act directly on structures (rather than on a tape of 0s and 1s, say) we can encode problems much more naturally than if we were working with Turing Machines. At the same time however, we can write code in a sequential, imperative style.

In Chapter 4 of this thesis we introduce a class of program schemes which we call RFDPS. This class is based on arrays, if-instructions, forall-loops, and repeat-loops. The class arose out of our attempts to replace the notion of a while-loop, present in earlier classes of program schemes, with one of a forall-loop which allows parallel execution of blocks of code.

We prove some strong results about the expressive power of RFDPS. Although it is strictly stronger than inflationary fixed-point logic (it can express PARITY, for example) there are still a good many computationally simple problems which it cannot express. We are able to obtain a strict, infinite hierarchy of classes of problems within the class of problems accepted by program schemes of RFDPS. These classes are parameterised by the depth of nesting of forall-loops allowed in the definition of program schemes. To our knowledge, this is the first strict, infinite hierarchy in a polynomial-time logic properly extending inductive fixed-point logic (with the property that the union of the classes of the hierarchy is the same as the class of problems definable in the polynomial-time logic itself). Our results are obtained by a direct analysis of the computations of our program schemes. Note that the existing hierarchy theorems of finite model theory, such as those in [41, 42, 43], are of no use to us here given that all of these hierarchy results are for explicit fragments of bounded-variable infinitary logic (which has a zero-one law), whereas our computational model is, first, not defined in terms of traditional logics, and second, is complicated by its ability to define problems not having a zero-one law.

Finally, we turn in a slightly different direction in Chapter 5, and discuss constraint satisfaction problems (CSPs). A CSP consists of a set of variables, to each of which must be assigned a value (from some domain), subject to one or more constraints.

Constraint satisfaction problems crop up in a huge variety of disciplines: from timetabling ("To each member of a set of lectures, assign a (room, time) pair, subject to constraints such as that no student or lecturer should have to attend two lectures simultaneously") to radio frequency planning ("To each member of a set of mobile phone masts, assign a (geographic location, frequency) pair so that reception is adequate for mobile phone users, whilst the signals from adjacent masts do not significantly interfere with each other"). Many combinatorial problems may be naturally expressed as CSPs [51]. For example, 3-COLOURABILITY can be formulated as "To each vertex of an undirected graph, assign a colour from the set $\{Red, Green, Blue\}$, subject to the constraint that whenever E is an edge in the graph, then the endpoints of E are assigned different colours". Many puzzles can also be framed as CSPs. The interested reader may get a feeling for constraint satisfaction problems in general by attempting to solve the following, slightly frivolous example.

Example 1.1 Arrange the letters A to P in a 4*4 grid, one letter in each of the sixteen squares, subject to the following constraints.

- P is above J, which is to the left of F and N.
- B is not in the third row.
- E is two squares above D.
- O and K are in the same column.
- C is above F, and to the right of A.
- M is above A, and to the right of H.
- E and I are in the same row.
- B is above J.
- K is not in a corner square.
- L is below H, and to the left of P.
- The second row contains exactly two consonants.
- G is adjacent to C.

Here, phrases such as "to the right of" mean that the letters in question appear in the same row. Adjacency includes diagonal adjacency.

In this thesis we will only consider assignments to fixed, finite domains (though infinite domains have also been studied in the literature). In addition, we will assume that our constraints are given by physically enumerating all possible allowed combinations of values which may be taken by sets of variables. Succinct representations, of forms such as "P is to the left of Q" will not be allowed, as they complicate notions such as the size of a problem instance.

The decision problem for CSPs is **NP**-complete in general. Nevertheless, much research has been done on restricting CSPs in various ways to ensure that they may be solved in polynomial time. In particular, restricting the possible constraints which we are permitted to use can ensure tractability in many interesting cases [18, 28, 37, 55, 56, 58, 65, 85]. From the point of view of finite model theory, this is hardly surprising. Consider Example 1.1 once again. Determining whether or not there is a solution to this puzzle is equivalent to determining whether the given set of constraints (which we may view as axioms) has a finite model. It is reasonable to expect that the difficulty in answering this question will depend on the expressive power of the logic we use to define the axioms. If the logic has a good deal of expressive power then the question may well be hard to answer; if it has very little then the question is likely to be easier, but we may no longer be able to express the axioms.

Given that so many tractable classes of constraint relations have been identified, it is pertinent to ask whether they may be combined, to yield new, larger constraint classes which are still tractable. This question has been posed before [15], but whereas the authors of that paper considered the effect of combining tractable classes over some fixed domain, in this thesis we generally consider the effect of combining two tractable classes from *disjoint* domains.

We focus in particular on the "multiple relational union" of two sets of constraint relations. We show that whenever both sets of relations are tractable, then their multiple relation union is a tractable set also. In addition, we show that its tractability cannot in general be deduced from previously known results about tractability. Using the results of [51] we then show that the multiple relational union is itself just one small subset of a much larger set of tractable relations, whose proof of tractability is much less obvious than that of the multiple relational union itself.

Chapter 2

Basic Definitions and Notation

In this chapter we will present the basic definitions and notation which we shall use throughout the remainder of this thesis. Many of our definitions are adapted from those in [30], though we also acknowledge a debt to [5, 51]. Some of our notation has been influenced by Stewart – see [74, 81] amongst others.

2.1 First-Order Logic

A *signature*, also known as a *vocabulary* or a *similarity type*, is a finite tuple $\sigma = \langle R_1, R_2, \ldots, R_r, C_1, C_2, \ldots, C_c \rangle$ of relation symbols (the R_i) and constant symbols (the C_i). Each relation symbol has an associated *arity* $a_i \geq 1$. We do not admit function symbols into our signatures[1].

A *σ-structure* $\mathcal{A}$ is a tuple

$$\mathcal{A} := \langle |\mathcal{A}|, R_1^{\mathcal{A}}, R_2^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, C_1^{\mathcal{A}}, C_2^{\mathcal{A}}, \ldots, C_c^{\mathcal{A}} \rangle$$

[1]Although much of what follows would go through more or less unchanged if we permitted function symbols, allowing them would complicate those of our proofs which depend on the careful counting of quantifiers. For example, a simple expression such as $f(f(f(x))) = y$ silently elides the two quantifiers which would be required to express the same proposition over a purely relational structure: $\exists u \exists v (F(x, u) \wedge F(u, v) \wedge F(v, y))$, where F is the relational analogue of f.

consisting of a non-empty, finite set $|\mathcal{A}|$, which we call the *universe* or *domain* of $\mathcal{A}$, along with relations $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$ and constants $C_i^{\mathcal{A}} \in |A|$. These relations and constants are said to be the *interpretations* of the corresponding relation and constant symbols from σ. Relations, therefore, are sets of tuples of elements of $|\mathcal{A}|$. Given any tuple $\mathbf{t}$ (also known as a *word*), we denote its ith element by $\mathbf{t}[i]$. Where it will aid clarity, symbols representing tuples are often **emboldened** in this way.

The *size* of $\mathcal{A}$ is the cardinality of $|\mathcal{A}|$, and is also denoted by $|\mathcal{A}|$ (this will cause no confusion). We will henceforth assume that every structure under consideration has size at least 2. This is no real restriction, since it is usually easy to treat size-1 structures as a special case if desired.

In accordance with common practice, we will often abuse the strict formalism defined above, and blur the distinction between relation symbols and their interpretations in a structure. We will also allow the use of relation symbols with names different from R_i, and constant symbols with names different from C_i. No confusion will arise as a result of this notational abuse, which is merely designed as a convenient aid to exposition.

We denote the set of all *finite* σ-structures by STRUCT(σ).

For any signature σ, FO(σ) denotes first-order logic over the signature σ, defined in the usual way, and with the usual connectives and quantifiers ($\neg, \rightarrow, \wedge, \vee, \exists, \forall$, etc.) Free and bound variables are defined in the standard way, and we shall use the notation $\varphi(\mathbf{x})$ to denote that the free variables of φ contain (amongst others) the variables of the tuple $\mathbf{x}$, which we assume to be distinct. We denote $\bigcup\{\text{FO}(\sigma) | \sigma \text{ is some signature}\}$ by FO.

We will often have cause to refer to more expressive logics than first-order logic. We write $\mathcal{L}$ to denote an arbitrary logic, and $\mathcal{L}(\sigma)$ to denote those formulae of $\mathcal{L}$ which are over the signature σ.

Let $\varphi(\mathbf{x}) \in \mathcal{L}(\sigma)$ be a formula whose free variables are precisely the variables of $\mathbf{x}$, and let $\mathcal{A}$ be some σ-structure. Let $\mathbf{a} \in |\mathcal{A}|^k$ be a tuple of domain elements of $\mathcal{A}$. We denote by $(\mathcal{A}, \mathbf{a})$ the augmentation of $\mathcal{A}$ with the k extra constants from $\mathbf{a}$, and write $(\mathcal{A}, \mathbf{a}) \models \varphi$ to mean that when the value of each variable $\mathbf{x}[i]$ is assigned to be $\mathbf{a}[i]$ and the rest of φ is interpreted in $\mathcal{A}$, then the result of the interpretation is *True*.

A *problem of arity* $k \geq 0$ over the signature σ is defined to be a subset of

$$\{(\mathcal{A}, \mathbf{a}) \mid \mathcal{A} \in \mathrm{STRUCT}(\sigma), \mathbf{a} \in |\mathcal{A}|^k\}$$

which is closed under isomorphism. That is, if $\mathcal{P}$ is a problem, if $\mathcal{A}$ and $\mathcal{B}$ are σ-structures, and if $(\mathcal{A}, \mathbf{a})$ and $(\mathcal{B}, \mathbf{b})$ are isomorphic, then $(\mathcal{A}, \mathbf{a}) \in \mathcal{P}$ if, and only if, $(\mathcal{B}, \mathbf{b}) \in \mathcal{P}$.

Ultimately, we will only be interested in problems of arity 0. Problems of any arity may be converted to problems of arity 0 by adding constant symbols to the underlying signature. However, it is convenient in what follows to allow problems of positive arity when we build new logics.

A *successor relation* over $\mathcal{A}$ is a binary relation of the form

$$\{\langle u_0, u_1\rangle, \langle u_1, u_2\rangle, \ldots, \langle u_{n-2}, u_{n-1}\rangle\}$$

where each of the u_i is a distinct domain element of $\mathcal{A}$, and n is the size of $\mathcal{A}$. The *minimum* element of the successor relation is u_0; the *maximum* element is u_{n-1}. Given a logic $\mathcal{L}$ over a signature σ (which we may assume without loss of generality does not contain either the special relation symbol *succ* or the special constant symbols 0 or *max*) we may add a successor relation to $\mathcal{L}$ as follows.

The formulae of $\mathcal{L}_s(\sigma)$ are the same as the formulae of $\mathcal{L}(\sigma \cup \langle 0, \mathit{max}, \mathit{succ}\rangle)$, where 0 and *max* are constant symbols, and *succ* is a binary relation symbol. The interpretation of a formula $\varphi \in \mathcal{L}_s(\sigma)$ over a σ-structure $\mathcal{A}$ is exactly as would be expected, save that 0 and *max* are always interpreted as distinct elements of $|\mathcal{A}|$, and *succ* is always interpreted as a binary successor relation whose minimum element is (the interpretation of) 0, and whose maximum element is (the interpretation of) *max*.

Of course, whether or not $\mathcal{A} \models \varphi$ could well depend on the precise successor relation chosen. The usual way around this is to insist that the only well-formed formulae of

$\mathcal{L}_s$ are those whose validity is independent of the chosen successor relation. This is a semantic (as opposed to syntactic) condition, and is somewhat unsatisfactory for that reason. For even in the case where $\mathcal{L}_s = \mathrm{FO}_s$, it is known that FO_s is not recursive. This is an easy consequence of Trakhtenbrot's Theorem: the formula $\psi \rightarrow E(0, max)$ is independent of the particular successor relation chosen if, and only if, ψ is unsatisfiable. Many researchers therefore do not consider "logics" involving a successor relation to be true logics [44]. Indeed, one of the most important open questions in Descriptive Complexity today is to determine whether or not there is a pure logic which captures **PTIME** (several logics are known to capture **PTIME** in the presence of successor relations [40, 48, 78]).

Another, weaker, way to extend a logic is to augment it with two constant symbols. As in the case when we augment a logic with a successor relation, we denote these by 0 and *max*, and ensure that they are always interpreted by distinct domain elements of the structure over which we are interpreting. (Despite the suggestive names, there is no sense in which *max* is any greater or lesser than 0. They are merely constant symbols.) Once again, the usual practice is to insist that formulae which depend in some way on the exact values assigned to 0 and *max* are not well-formed. Once again however, this gives rise to (potentially) non-recursive sets of well-formed formulae.

In both of these cases, we will adhere for the time being to the semantics just given. Nevertheless, we will resume this discussion in Chapter 4, when we will have a little more to say on the subject.

If φ is a *sentence* of some logic $\mathcal{L}(\sigma)$ (that is, it contains no free variables) then the set of structures *defined* by φ is

$$\{\mathcal{A} \in \mathrm{STRUCT}(\sigma) \mid \mathcal{A} \models \varphi\}.$$

For similar reasons to those just outlined, there are logics such that this set of structures is not necessarily closed under isomorphism (that is, does not define a problem). We will not be interested in sentences which do not define problems.

2.2 Extending First-Order Logic

Definition 2.1 Let $\tau = \langle R_1, R_2, \ldots, R_r, C_1, C_2, \ldots, C_c \rangle$ and σ be signatures, where each R_i is a relation symbol of arity $a_i \geq 1$, and each C_i is a constant symbol. Let $k \in \mathbb{N}$, and let

$$T = \langle \varphi_1(\mathbf{x}_1, \mathbf{z}), \varphi_2(\mathbf{x}_2, \mathbf{z}), \ldots, \varphi_r(\mathbf{x}_r, \mathbf{z}), \psi_1(\mathbf{y}_1), \psi_2(\mathbf{y}_2), \ldots, \psi_c(\mathbf{y}_c) \rangle$$

be a tuple of formulae of some logic $\mathcal{L}$, where each φ_i is over the ka_i distinct free variables $\mathbf{x}_i$ (along with the free variables from $\mathbf{z}$) and each ψ_i is over the k distinct free variables $\mathbf{y}_i$ (and no others). We assume that none of the variables of $\mathbf{z}$ occur amongst the variables of any $\mathbf{x}_i$ or $\mathbf{y}_j$; however, we need not assume that any of the other variables are distinct from each other. (In practice, we need not force each φ_i to contain every variable of $\mathbf{z}$ free: merely a subset. However, we can extend the free variables of any φ_i to be the whole of $\mathbf{z}$ by adding trivial clauses such as $z_7 = z_7$, so the present definition is equivalent to the apparently more general one.) Furthermore, let each ψ_i have the property that for any σ-structure $\mathcal{A}$,

$$\begin{aligned} \mathcal{A} \models\; & \exists x_1, \exists x_2, \ldots, \exists x_k (\psi_i(x_1, x_2, \ldots, x_k) \wedge \forall y_1, y_2, \ldots, y_k (\\ & \psi_i(y_1, y_2, \ldots, y_k) \rightarrow (x_1 = y_1 \wedge x_2 = y_2 \wedge \ldots \wedge x_k = y_k))). \end{aligned}$$

A tuple of formulae such as T is called *τ-descriptive of arity k.*

Given any σ-structure $\mathcal{A}$, and any assignment $\mathbf{v} \in |\mathcal{A}|^{|\mathbf{z}|}$ to the variables $\mathbf{z}$, these formulae then give rise to a *translation* of $\mathcal{A}$ to a τ-structure $\mathcal{B}_{\mathbf{v}}$ defined in the following manner.

The domain elements of $\mathcal{B}_{\mathbf{v}}$ are the k-tuples of domain elements of $\mathcal{A}$. For any relation R_i of $\mathcal{B}_{\mathbf{v}}$, we define $R_i(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{a_i})$ to hold if, and only if,

$$(\mathcal{A}, \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{a_i}, \mathbf{v}) \models \varphi_i(\mathbf{x}_i, \mathbf{z})$$

(where the shorthand used should be obvious). We define each constant C_i to be at the (unique) domain element $\mathbf{u} \in |\mathcal{B}_{\mathbf{v}}|$ for which $(\mathcal{A}, \mathbf{u}) \models \psi(\mathbf{y}_i)$. In addition, if there is a successor relation present in $\mathcal{A}$ then we can extend it to a successor relation over $\mathcal{B}_{\mathbf{v}}$ by defining $0^{\mathcal{B}_{\mathbf{v}}} = \langle 0, 0, \ldots, 0 \rangle$, $max^{\mathcal{B}_{\mathbf{v}}} = \langle max, max, \ldots, max \rangle$, and $succ(\mathbf{x}, \mathbf{y})$ holds in $\mathcal{B}_{\mathbf{v}}$ if, and only if, $\mathbf{y}$ is the lexicographic successor (in $\mathcal{A}$) of $\mathbf{x}$.

This notion of translating a structure over one signature into a structure over a possibly different signature is analogous to the complexity-theoretic notion of *reducing* an instance of one problem to be an instance of another problem. In the case of complexity theory, the place of the logic $\mathcal{L}$ is taken by some Turing machine (a *transducer*) whose resources are bounded in some way. Here, the expressiveness of $\mathcal{L}$ is what limits the sorts of translations we can achieve. As in the case of traditional complexity theory, we may exhibit reductions from one *problem* to another.

Example 2.2 Consider the arity-0 problem k-COLOURABILITY, defined by:

INSTANCE: An undirected graph, G.

YES-INSTANCE: G can be k-coloured. That is, each vertex of G can be assigned a natural number from the set $\{0, 1, \ldots, k-1\}$, so that vertices which are connected by an edge are always assigned different numbers.

(We have already met this problem briefly in Chapter 1.)

Let $\sigma = \tau = \langle E \rangle$ be identical signatures containing just the one binary relation E. We may view σ- (and τ-) structures as undirected graphs, via "There is an edge between vertices u and v if, and only if, $E(u,v) \vee E(v,u)$ holds." Consider the formula

$$\varphi(x_1, x_2, y_1, y_2) = (x_1 = 0 \wedge y_1 = 0 \wedge E(x_2, y_2)) \vee (x_1 = max \wedge x_2 = max \wedge y_1 = 0)$$

Given any graph $\mathcal{A}$, the translation of $\mathcal{A}$ by φ is another graph $\mathcal{B}$ containing $|\mathcal{A}|^2$ vertices. As far as edges are concerned, $\mathcal{B}$ contains an isomorphic copy of the graph $\mathcal{A}$ (on the vertices $(0, y)$), along with edges from the vertex (max, max) to every vertex $(0, y)$.

It is easy to see that for any $k \geq 1$, $\mathcal{B}$ is $(k+1)$-colourable if, and only if, $\mathcal{A}$ is k-colourable. Consequently, for each k, φ defines a reduction from the problem k-COLOURABILITY to the problem $(k+1)$-COLOURABILITY.

Another complexity-theoretic notion which we may appropriate to our logical setting is that of an *oracle*. The idea here is that a Turing machine may make repeated use of an oracle, which can solve instances of (potentially computationally difficult) problems in one step. To carry this idea across to the logical world (and, indeed, to generalise it to nested oracles) we introduce the logic $(\pm\Omega)^*[\mathrm{FO}]$.

Definition 2.3 Let σ and τ be as in Definition 2.1, and let Ω be some problem of arity t over τ. The syntax of the logic $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$ is defined to be the smallest set of formulae such that:

- any formula of $\mathrm{FO}(\sigma)$ is a formula of $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$;
- if φ and ψ are formulae of $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$, then so are

  $$(\varphi \wedge \psi),\ (\varphi \vee \psi),\ (\neg\varphi),\ (\exists x\varphi), \text{ and } (\forall x\varphi)\ ; \text{ and}$$

- if T, defined as in Definition 2.1, is a tuple of formulae of $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$ which is τ-descriptive of arity k, then

  $$\begin{aligned}&\Omega[\lambda\mathbf{x}_1, \varphi_1(\mathbf{x}_1, \mathbf{z}), \lambda\mathbf{x}_2, \varphi_2(\mathbf{x}_2, \mathbf{z}), \ldots, \lambda\mathbf{x}_r, \varphi_r(\mathbf{x}_r, \mathbf{z}),\\ &\quad \lambda\mathbf{y}_1, \psi_1(\mathbf{y}_1), \lambda\mathbf{y}_2, \psi_2(\mathbf{y}_2), \ldots, \lambda\mathbf{y}_c, \psi_r(\mathbf{y}_c)](\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_t) \qquad (2.1)\end{aligned}$$

  is a formula of $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$, where each $\mathbf{w}_i$ is a k-tuple of variables and constant symbols, and the variables of each $\mathbf{w}_i$ do not occur in any formula φ_j or ψ_j. The free variables of this formula are the union of the variables of $\mathbf{z}$, and the variables of each $\mathbf{w}_i$. Let us denote the tuple of these variables (according to some canonical ordering) by $\mathbf{x}$.

Semantically, we define the interpretation of a formula $\theta \in (\pm\Omega)^*[\mathrm{FO}(\sigma)]$ in the usual recursive manner. The only non-standard case occurs when θ is of the form given in Formula 2.1. In this case, for any σ-structure $\mathcal{A}$, and for any tuple $\mathbf{v} \in |\mathcal{A}|^{|\mathbf{x}|}$, let $\mathcal{B}_{\mathbf{v}}$ be the translation of $\mathcal{A}$ by T according to the interpretation (from $\mathbf{v}$) of the variables of $\mathbf{z}$. Furthermore, let $\mathcal{B}_{\mathbf{v}}$ be augmented by the constants $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t$, where each $\mathbf{v}_i \in |\mathcal{A}|^k$ is obtained from the tuple $\mathbf{w}_i$ by giving any variable of $\mathbf{w}_i$ the appropriate value from $\mathbf{v}$. Then

$$(\mathcal{A}, \mathbf{v}) \models \theta(\mathbf{x}) \text{ if, and only if, } (\mathcal{B}_{\mathbf{v}}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t) \in \Omega.$$

There are a number of natural sublogics of $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$. For example, the logic $(\Omega)^*[\mathrm{FO}(\sigma)]$ is the same as $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$, save that all occurrences of the "generalised quantifier" Ω must occur *positively*. That is, Ω must not occur within the scope of an odd number of negation signs. The logic $(\pm\Omega)^k[\mathrm{FO}(\sigma)]$ (with $k \in \mathbb{N}$) is the same as $(\pm\Omega)^*[\mathrm{FO}(\sigma)]$, save that occurrences of Ω must not be nested more than k deep. Finally, the logic $(\Omega)^k[\mathrm{FO}(\sigma)]$ contains both of these restrictions.

As might be anticipated, it is possible to add more than one generalised quantifier to a logic, and also to add them to logics other than FO. We will not be concerned with the former in this thesis, though we will return to the latter in the next chapter.

Example 2.4 The traditional quantifiers, $\forall$ and $\exists$, can be viewed as special cases of the above construction. For example, the problem FORALL is defined over the signature $\langle U \rangle$ containing just one unary relation by:

INSTANCE: A collection of points, and a specified subset U of that collection.

YES-INSTANCE: The subset U contains every point in the collection.

It is easy to see that using a generalised quantifier corresponding to the problem FORALL is equivalent to using the $\forall$ quantifier. Of course, there is a similar problem EXISTS corresponding to $\exists$.

Example 2.5 The problem TC (short for *Transitive Closure*) is a problem of arity 2 which is defined over the signature $\sigma_2 = \langle E \rangle$ consisting of one binary relation. Structures over σ_2 may be viewed as directed graphs in the obvious way. A yes-instance of TC is a σ_2-structure with two augmented constants, $(\mathcal{A}, u, v)$, with the property that there is a path of edges leading from u to v in the digraph represented by $\mathcal{A}$.

The problem CONNECTIVITY is an arity-0 problem, also defined over the signature σ_2. This time, we will view structures over σ_2 as *undirected* graphs in the same way as in Example 2.2. A yes-instance of CONNECTIVITY is an undirected graph which is completely connected. It may be defined as a sentence of $(\pm\mathrm{TC})^*[\mathrm{FO_s}]$ (actually, of $\mathrm{TC}^1[\mathrm{FO}]$) as follows

$$\forall u \forall v \mathrm{TC}[\lambda x, y, (E(x,y) \vee E(y,x))](u,v).$$

Related to the problem TC is the problem DTC (short for Deterministic Transitive Closure). Like TC, DTC is an arity-2 problem defined over the signature σ_2 whose structures represent digraphs. An augmented structure $(\mathcal{A}, u, v)$ is a yes-instance of DTC if there is a *deterministic* path from the vertex u to the vertex v. That is, if there is a path from u to v in the digraph with the property that all vertices on the path (with the possible exception of v itself) have out-degree exactly 1.

Example 2.6 The problem PARITY of arity 0 is defined on the empty signature by

INSTANCE: A collection of domain elements.

YES-INSTANCE: The collection is of even size.

Computationally, PARITY is a very easy problem to solve. Nevertheless, it cannot be defined using just first-order logic. (This is well known, and easy to prove. The Ehrenfeucht-Fraïssé game, introduced in the next chapter, provides one possible proof method.) We shall show here that it can be defined by a sentence of $(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$.

For consider φ defined in the following manner.

$$\varphi = \mathrm{DTC}[\lambda x_0, x_1, y_0, y_1, (x_0 = 0 \wedge y_0 = max \wedge succ(x_1, y_1)) \vee$$
$$(x_0 = max \wedge y_0 = 0 \wedge succ(x_1, y_1))](0, 0, 0, max)$$

Let φ be interpreted over a structure $\mathcal{A}$, of size n. The digraph constructed by φ contains n^2 vertices. However, we are only interested in $2n$ of these: those of the form $(0, x)$, and those of the form (max, x), with $x \in |\mathcal{A}|$. Given any successor relation, the formula constructs a bipartite digraph on these $2n$ vertices by including an edge from each vertex $(0, x)$ to the vertex $(max, x + 1)$, and from each vertex (max, x) to the vertex $(0, x + 1)$, where $x \neq max$. Consequently, it is easy to see that there is a deterministic path from $(0, 0)$ to $(0, max)$ if, and only if, n is even; *i.e.*, if, and only if, $\mathcal{A}$ is a yes-instance of PARITY.

Definition 2.7 Let $\mathcal{L}$ be a logic. The class of problems *represented* by $\mathcal{L}$ consists of all those problems expressible by a sentence of $\mathcal{L}$. We overload notation, and refer to this class of problems as $\mathcal{L}$ also.

Some logics, as we will shortly discover, have the property that the problems definable in that logic are, in some sense, the same as the problems expressible in a traditional complexity class. We say "in some sense" because problems within traditional complexity classes are considered to be sets of strings from $\{0, 1\}^*$, which are recognised by a resource-bounded Turing machine (or some other equivalent computational model). We must therefore make this connection more explicit.

Definition 2.8 Let $\mathcal{A} \in \mathrm{STRUCT}(\sigma)$ for some signature

$$\sigma = \langle R_1, R_2, \ldots, R_r, C_1, C_2, \ldots, C_c \rangle$$

where each R_i is a relation symbol of arity a_i, and where each C_i is a constant symbol. The *encoding* of $\mathcal{A}$, $enc_\sigma(\mathcal{A})$, is a string from $\{0, 1\}^*$ defined as follows.

- The interpretations of the relations $R_1, R_2, \ldots, R_r$ are encoded in order, with each $R_i^{\mathcal{A}}$ encoded as a sequence of $|\mathcal{A}|^{a_i}$ 0s and 1s, defining the characteristic function of $R_i^{\mathcal{A}}$ in the natural way, according to some ordering on the domain elements of $\mathcal{A}$ (where this ordering depends only on the domain, and not on the particular structure $\mathcal{A}$ itself).
- The interpretations of the constants $C_1, C_2, \ldots, C_c$ are encoded in order, with each $C_i^{\mathcal{A}}$ being encoded as the binary representation of its position in the above ordering, padded with leading zeroes so as to be of length $\lceil \log_2 |\mathcal{A}| \rceil$.
- The two strings thus formed are concatenated.

Note that for a fixed signature, the length of the encoding of any structure depends only on its size; conversely, the domain size of a structure can be deduced knowing only the length of its encoding. Consequently, encodings are reversible.

For any problem Ω over σ, we may define $enc_\sigma(\Omega) = \{enc_\sigma(\mathcal{A}) \mid \mathcal{A} \in \Omega\}$. That is, we can associate with Ω a set of strings over $\{0, 1\}$ which is isomorphic to Ω. Because problems are isomorphism-closed, the precise order chosen in the definition of encodings makes no difference to the resulting set of strings.

Furthermore, *any* set of strings over $S = \{0, 1\}^*$ which is the encoding of at least one problem over some signature, is also an encoding of the following problem Ω' over the signature $\sigma_1 = \langle U \rangle$ consisting of one unary relation symbol. Ω' consists of those structures $\mathcal{A}$ for which $enc_{\sigma_1}(\mathcal{A}) \in S$. Since S is the encoding of some problem, Ω' is also a problem.

A computational complexity class $\mathcal{C}$ is identified with a set of problems $\mathcal{L}$ if, and only if:

1. for each $S \in \mathcal{C}$, there is some $\Omega \in \mathcal{L}$ over some signature σ such that $S = enc_\sigma(\Omega)$; and
2. for each $\Omega \in \mathcal{L}$ over the signature σ, we have that $enc_\sigma(\Omega) \in \mathcal{C}$.

If $\mathcal{L}$ and $\mathcal{C}$ are identified, we shall say that $\mathcal{L}$ *captures* $\mathcal{C}$, and will write $\mathcal{L} = \mathcal{C}$.

The following theorem is due to Immerman [49, 50].

Theorem 2.9 1. Every sentence of $(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$ is equivalent to one of the form

$$\mathrm{DTC}[\lambda\mathbf{x},\mathbf{y},\psi(\mathbf{x},\mathbf{y})](\mathbf{0},\mathbf{max})$$

where $\mathbf{0}$ and $\mathbf{max}$ are tuples $\langle 0,0,\ldots,0\rangle$ and $\langle max, max, \ldots, max\rangle$ respectively, of the appropriate length, and where where ψ is a quantifier-free first-order formula. Furthermore, $(\pm\mathrm{DTC})^*[\mathrm{FO_s}] = \mathbf{L}$.

2. Every sentence of $(\pm\mathrm{TC})^*[\mathrm{FO_s}]$ is equivalent to one of the form

$$\mathrm{TC}[\lambda\mathbf{x},\mathbf{y},\psi(\mathbf{x},\mathbf{y})](\mathbf{0},\mathbf{max})$$

where $\mathbf{0}$ and $\mathbf{max}$ are tuples $\langle 0,0,\ldots,0\rangle$ and $\langle max, max, \ldots, max\rangle$ respectively, of the appropriate length, and where ψ is a quantifier-free first-order formula. Furthermore, $(\pm\mathrm{TC})^*[\mathrm{FO_s}] = \mathbf{NL}$.

L (resp. **NL**) is the class of problems computable by a deterministic (resp. non-deterministic) Turing machine using a logarithmic amount of workspace. Other researchers have captured other complexity classes by adding a generalised quantifier to $\mathrm{FO_s}$ (or sometimes just FO) [7, 24, 72, 73, 74, 76, 77, 78, 79]. Indeed, all of the major traditional complexity classes have been captured in this way.

In fact, the ψ in both of the cases in Theorem 2.9 is not just a quantifier-free first-order formula; it may actually be assumed to be a *quantifier-free projection.*

Definition 2.10 Let σ be a signature. A quantifier-free projection (qfp) over σ is a formula $\psi \in \mathrm{FO_s}(\sigma)$ of the form

$$(\alpha_0 \wedge \beta_0) \vee (\alpha_1 \wedge \beta_1) \vee \ldots (\alpha_m \wedge \beta_m)$$

for some $m \geq 0$, where

1. each α_i is a conjunction of the logical atomic relations (*succ* and $=$), and their negations;

2. each β_i is atomic or negated atomic; and

3. if $i \neq j$ then α_i and α_j are mutually exclusive.

The qfp is an extremely weak form of reduction: the fact (implicit in Theorem 2.9) that **L** and **NL** (along, in fact, with many other complexity classes) have complete problems with respect to qfp's is a very strong result. Since the presence or absence of an edge in the target graph defined by the reduction depends only on the presence or absence of just one bit in the input structure, it is difficult to envisage a much weaker form of reduction under which complexity classes may continue to have complete problems.

Note that in some earlier work, qfp's have been called *projection translations.* The term quantifier-free projection was originally reserved for an apparently more general notion of reduction, in which the first condition of Definition 2.10 is replaced by:

1. Each α_i is quantifier-free, and does not involve any relation symbol of σ.

Immerman was the first to observe that the two notions of reduction are, in fact, the same (see [80]). A corollary of this fact is that the composition of two qfp's is another qfp. This is easy to see given the apparently more general definition, but less clear when qfp's are defined as in Definition 2.10.

As an alternative (or, indeed, as a supplement) to the use of generalised quantifiers such as those we have already discussed, we may add *fixed point* operators to first-order logic. Several of these operators have been defined in the literature, but we will be concerned here with the Inflationary Fixed Point operator, IFP.

Definition 2.11 Let σ be some signature, let R be some k-ary relation symbol not in σ, and let $\mathcal{L}$ be some logic. Let $\varphi(\mathbf{x}) \in \mathcal{L}(\sigma \cup \langle R \rangle)$, with $|\mathbf{x}| = k$ be some formula, and let $\mathcal{A} \in \text{STRUCT}(\sigma)$.

We may define a sequence of k-ary relations as follows. Define $R_0 = \{\}$: the empty relation of arity k over $|\mathcal{A}|$. Define

$$R_{i+1} = \{\mathbf{u} \in |\mathcal{A}|^k : \varphi(\mathbf{u}, R_i) \text{ holds}\} \cup R_i.$$

Then $R_i \subseteq R_{i+1}$ for each i. Since $\mathcal{A}$ is finite, there must be some $i \leq |\mathcal{A}|^k$ for which $R_i = R_{i+1}$. Of course, every further $j > i$ also has the property that $R_i = R_j$. We call this R_i the *inflationary fixed point* of $\varphi^{\mathcal{A}}$, and denote it by $\mathrm{IFP}[\lambda\mathbf{x}, R, \varphi^{\mathcal{A}}(\mathbf{x}, R)]$.

As with the generalised quantifiers defined earlier, we may augment a logic $\mathcal{L}$ with the IFP operator, yielding logics such as $(\pm\mathrm{IFP})^*[\mathrm{FO}]$, $\mathrm{IFP}^7[\mathrm{FO_s}]$, and so on.

Example 2.12 Consider the following formula over $\sigma_{2++} \cup \langle R\rangle$:

$$\varphi(x, y) \equiv (x = y) \vee E(x, y) \vee \exists z(E(x, z) \wedge E(z, y)).$$

Then for any $\mathcal{A} \in \mathrm{STRUCT}(\sigma_{2++})$ it is not hard to see that

$$(\mathcal{A}, u, v) \models \mathrm{IFP}[\lambda x, y, R, \varphi(x, y)](u, v)$$

if, and only if, there is a path from u to v in the digraph represented by $\mathcal{A}$. Consequently, the sentence

$$\mathrm{IFP}[\lambda x, y, R, \varphi(x, y)](c, d)$$

defines the problem TC. Indeed, with a little more work, it can be shown that any sentence of $(\pm\mathrm{TC})^*[\mathrm{FO}]$ can be defined in $(\pm\mathrm{IFP})^*[\mathrm{FO}]$.

Both $(\pm\mathrm{IFP})^*[\mathrm{FO}]$ and $(\pm\mathrm{TC})^*[\mathrm{FO}]$ are fragments of *bounded variable infinitary logic*, $\mathcal{L}^{\omega}_{\infty\omega}$ (see [30]). This is defined as $\bigcup_{d=1}^{\infty}\{\mathcal{L}^{d}_{\infty\omega}\}$, where $\mathcal{L}^{d}_{\infty\omega}$ is defined in the same way as first-order logic save that the only variables we allow in our formulae (free or bound) are $x_1, x_2, \ldots, x_d$, and where we allow conjunctions and disjunctions of arbitrary (rather than just finite) sets of formulae.

A third way of extending first-order logic, somewhat different in flavour from adding generalised quantifiers and inductive operators, is to move to second-order logic. In second-order logic, we allow our formulae to make use of quantification over *relations* of arbitrary (fixed) arity, rather than just over domain elements of a structure.

Example 2.13 Consider the signature $\sigma_2 = \langle E \rangle$ consisting of one binary edge relation, and treat structures over σ_2 as undirected graphs, as in Example 2.2. Define φ by

$$\begin{aligned}\varphi \equiv \exists R\, \exists G\, \exists B\, \forall x ((R(x) \vee G(x) \vee B(x)) \wedge \\ \neg(R(x) \wedge G(x)) \wedge \neg(R(x) \wedge B(x)) \wedge \neg(G(x) \wedge B(x))) \wedge \\ \forall x \forall y ((E(x,y) \vee E(y,x)) \rightarrow \\ (\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y))))\end{aligned}$$

where R, G, and B are new unary relation symbols. Then for any structure $\mathcal{A} \in \mathrm{STRUCT}(\sigma_2)$,

$$\mathcal{A} \models \varphi \text{ iff the graph } \mathcal{A} \text{ is 3-colourable.}$$

Of course, 3-COLOURABILITY is an **NP**-complete problem. More generally, by Fagin's celebrated result [32], the *existential* fragment of second-order logic (by which is meant that fragment consisting of first-order formulae, prefixed by one or more existentially quantified relation symbols) precisely captures the complexity class **NP**. This fragment is sometimes known as Σ_1^1. Indeed, Stockmeyer went further, and showed that each level of the polynomial hierarchy can be captured by an appropriate fragment of second-order logic [82], but we shall not make use of this result here.

A further restriction of Σ_1^1 can be obtained by limiting the permitted arities of the quantified second-order variables. It is known that the natural hierarchy obtained in this way is strict [2]. We shall be concerned here with the *monadic* fragment – that is, the fragment in which all of the second-order variables have arity 1. Monadic **NP**, as it is sometimes known, was originally studied by Fagin [33]. It is orthogonal to traditional complexity classes: we have already seen that it contains the problem 3-COLOURABILITY, yet it does not contain the computationally simple problem

PARITY. This fact, originally observed by Fagin, is easy to prove given the right machinery. We shall come to this in the next chapter.

2.3 The Constraint Satisfaction Problem

The definitions presented in this section are only used in Chapter 5, and may be omitted, if desired, until that chapter is read.

The constraint satisfaction problem [59, 60, 63] is defined as follows.

Definition 2.14 An instance of a *constraint satisfaction problem* consists of:

- a finite set of variables, V;
- a finite domain of values, D; and
- a finite set of constraints $\{C_1, C_2, \ldots, C_q\}$.

  Each constraint C_i is a pair (s_i, R_i), where:

  - s_i is a tuple of variables of length m_i, called the 'constraint scope'; and
  - R_i is an m_i-ary relation over D, called the 'constraint relation'.

For each constraint, (s_i, R_i), the tuples in R_i indicate the allowed combinations of simultaneous values for the variables in s_i. The length of s_i, and of the tuples in R_i, is called the 'arity' of the constraint. In particular, unary constraints specify the allowed values for a single variable, and binary constraints specify the allowed combinations of values for a pair of variables.

A *solution* to a constraint satisfaction problem instance is a function from the variables to the domain such that the image of each constraint scope is an element of the corresponding constraint relation. Deciding whether or not a given problem instance has a solution is **NP**-complete in general [60], even when the constraints are restricted to binary constraints. In Chapter 5 of this thesis we shall consider how restricting the allowed constraint relations to some fixed subset of all the possible relations affects the complexity of this decision problem. We therefore make the following definition.

Definition 2.15 For any set of relations, Γ, $\mathbf{C}_\Gamma$ is defined to be the following decision problem.

INSTANCE: A constraint satisfaction problem instance, $\mathcal{P}$, in which all constraint relations are elements of Γ.

YES-INSTANCE: $\mathcal{P}$ has a solution.

If there is a deterministic algorithm which solves every problem instance in $\mathbf{C}_\Gamma$ in polynomial time, then we shall say that Γ is a *tractable* set of relations. If $\Gamma = \{R\}$ is a singleton set, then we shall call the relation R itself tractable.

Example 2.16 The binary disequality relation over a set D is defined as follows.

$$\neq_D := \{\langle d_1, d_2\rangle \in D^2 \mid d_1 \neq d_2\}$$

For any finite set D, the class of constraint satisfaction problem instances $\mathbf{C}_{\{\neq_D\}}$ corresponds to the graph $|D|$-COLOURABILITY problem. This problem is tractable when $|D| \leq 2$ and **NP**-complete when $|D| \geq 3$. Consequently, when $|D| \leq 2$, $\{\neq_D\}$ is a tractable set of relations.

Note that $\mathbf{C}_\Gamma$ always defines a problem – that is, the set of structures which it defines is always closed under isomorphism. Furthermore, as is the case with monadic **NP**, there are many computationally simple problems which cannot be defined as $\mathbf{C}_\Gamma$ for any Γ. The easiest way to see this is to observe that if $(V, D, \{C_1, C_2, \dots, C_q\})$ is a yes-instance of some $\mathbf{C}_\Gamma$, then $(V \cup V', D, \{C_1, C_2, \dots, C_q, C_1', C_2', \dots, C_q'\}$ is another yes-instance, where V' is a set of $|V|$ variables which are disjoint from those of V, and where each C_i' is identical to the corresponding C_i, save that its constraint scope refers to the variables of V', rather than V. Consequently, each $\mathbf{C}_\Gamma$ which has at least one yes-instance, has an infinite collection of yes-instances; it is impossible to define any non-empty finite problem in this framework.

There are other, less trivial examples of undefinable problems in the literature: see [35, 61] for some examples.

2.4 Operations on Tuples

Theorem 5.2 from Chapter 5 will allow us to define a variety of previously unknown tractable classes of relations. In order to generalise the theorem however, and make it more widely applicable, we need to work within an algebraic framework. This section sets up the basic definitions and results which we shall need.

A k-ary operation φ on a set D is simply a function from D^k to D. We call k the *arity* of the operation. The domain, $dom(\varphi)$, and range, $range(\varphi)$, of φ are defined in the obvious way; in particular, the domain of a partial operation on D will, in general, be a subset of D.

Any operation on the elements of a set D can be extended to an operation on tuples over D by applying the operation to the values in each coordinate position separately.

Hence, any operation defined on the domain of a relation can be used to define an operation on the elements of that relation, in the following manner.

Definition 2.17 Let R be an n-ary relation over a domain D, and let $\varphi : D^k \to D$ be a k-ary operation on D.

For any collection of tuples, $t_1, t_2, \ldots, t_k \in R$ (not necessarily all distinct) define the tuple $\varphi(t_1, t_2, \ldots, t_k)$ by:

$$\varphi(t_1, t_2, \ldots, t_k) =$$
$$\langle \varphi(t_1[1], t_2[1], \ldots, t_k[1]), \varphi(t_1[2], t_2[2], \ldots, t_k[2]), \ldots, \varphi(t_1[n], t_2[n], \ldots, t_k[n]) \rangle$$

Note that in this definition, and indeed throughout, we allow the possibility that $k = 0$; that is, that φ is a nullary operation. Such an operation takes no arguments, but always returns some domain element, d. Nullary operations are a special kind of constant operation, defined below.

We may now define the following closure property of relations.

Definition 2.18 Let R be a relation over a domain D, and let $\varphi : D^k \to D$ be a k-ary (possibly partial) operation on D.

R is said to be *closed under* φ if, for all $t_1, t_2, \ldots, t_k \in R$ (not necessarily all distinct) such that $\varphi(t_1, t_2, \ldots, t_k)$ is defined:

$$\varphi(t_1, t_2, \ldots, t_k) \in R$$

Lemma 2.19 Let R be a relation over a domain D which is closed under some (possibly partial) operation φ. Let R' be a second relation over D which has been generated from R by permuting the order of elements in each tuple of R in the same way. Then R' is also closed under φ.

Proof Follows immediately from the definition of closure. □

Notation 2.20 Let Γ be a set of relations over a domain D, and let $\varphi : D^k \to D$ be a k-ary operation on D. Then Γ is said to be closed under φ if every relation $R \in \Gamma$ is closed under φ.

The set of all total operations under which Γ remains closed is called the set of *polymorphisms* of Γ, and is denoted by $Pol(\Gamma)$. In a similar vein, let Φ be a set of operations over D. The set of all relations which are closed under every operation of Φ is called the set of *invariants* of Φ, and denoted by $Inv(\Phi)$.

There are certain relations which we can guarantee will always occur in $Inv(\Phi)$. For example, the equality relation on D (of any arity) is closed under every possible φ. Similarly, the relation D^a is a member of $Inv(\Phi)$ for any a.

An observation which will be useful later is that $Inv(\Phi)$ normally also contains the empty relation of every arity ≥ 0. The exception to this rule is that if Φ contains an operation of arity 0 (which must always be a constant operation), then $\mathit{Inv}(\Phi)$ can only contain relations of strictly positive size. This is because every relation in $\mathit{Inv}(\Phi)$ of positive arity must then contain the appropriate constant tuple of that arity. Between them, the mappings $\mathit{Pol}()$ and $\mathit{Inv}()$ establish a *Galois connection* between sets of relations and sets of operations [16, 62]. That is, the following four conditions all hold.

1. If Φ and Φ' are sets of operations, and $\Phi \subseteq \Phi'$, then $\mathit{Inv}(\Phi') \subseteq \mathit{Inv}(\Phi)$.

2. If Γ and Γ' are sets of relations, and $\Gamma \subseteq \Gamma'$, then $\mathit{Pol}(\Gamma') \subseteq \mathit{Pol}(\Gamma)$.

3. If Φ is a set of operations, then $\Phi \subseteq \mathit{Pol}(\mathit{Inv}(\Phi))$.

4. If Γ is a set of relations, then $\Gamma \subseteq \mathit{Inv}(\mathit{Pol}(\Gamma))$.

From these conditions, we can also deduce that for all sets of relations Γ, we have $\mathit{Pol}(\Gamma) = \mathit{Pol}(\mathit{Inv}(\mathit{Pol}(\Gamma)))$; and for all sets of operations Φ, we have $\mathit{Inv}(\Phi) = \mathit{Inv}(\mathit{Pol}(\mathit{Inv}(\Phi)))$. That is, $\mathit{Pol}(\mathit{Inv}())$ and $\mathit{Inv}(\mathit{Pol}())$ are "closure functions" on sets of operations and sets of relations respectively (albeit of a different type to the closure operations we have so far considered).

The following theorem was proved in [51].

Theorem 2.21 A set of relations Γ over some domain is tractable if, and only if, $\mathit{Inv}(\mathit{Pol}(\Gamma))$ is tractable.

A set of relations of the form $\mathit{Inv}(\mathit{Pol}(\Gamma))$, for some Γ, is known as a *relational clone.*

We will sometimes want to refer to operations with various special properties, as specified in the following definition.

Definition 2.22 Let $\varphi : D^k \to D$ be a k-ary operation.

- If there is some $d \in D$ such that for all $\langle d_1, d_2, \ldots, d_k \rangle \in D^k$ we have that $\varphi(d_1, d_2, \ldots, d_k) = d$, then we say that φ is a *constant operation.*

- If there exists an index $1 \leq i \leq k$ such that for all $\langle d_1, d_2, \ldots, d_k \rangle \in D^k$ we have that $\varphi(d_1, d_2, \ldots, d_k) = d_i$, then φ is called a *projection*.

- If $k = 2$, and for all $d_1, d_2, d_3 \in D$ we have that

  1. $\varphi(\varphi(d_1, d_2), d_3) = \varphi(d_1, \varphi(d_2, d_3))$ (Associativity);
  2. $\varphi(d_1, d_2) = \varphi(d_2, d_1)$ (Commutativity); and
  3. $\varphi(d_1, d_1) = d_1$ (Idempotence)

  then φ is said to be a *binary ACI operation*[2].

- If $k = 3$ and for every $d_1, d_2 \in D$ it is the case that $\varphi(d_1, d_1, d_2) = \varphi(d_1, d_2, d_1) = \varphi(d_2, d_1, d_1) = d_1$ then φ is called a *majority operation*.

- If $k = 3$ and for all $d_1, d_2, d_3, d_4 \in D$ we have that

  1. $\varphi(\varphi(d_1, d_2, d_3), d_3, d_4) = \varphi(d_1, d_2, d_4)$;
  2. $\varphi(d_1, d_2, d_3) = \varphi(d_3, d_2, d_1)$; and
  3. $\varphi(d_1, d_1, d_2) = d_2$

  then φ is said to be *affine*. An alternative way of defining affine operations is as those ternery operations φ for which there exists some Abelian group $(D, +, ^{-1}, 0)$, such that given any $d_1, d_2, d_3 \in D$

  $$\varphi(d_1, d_2, d_3) = d_1 + (d_2)^{-1} + d_3.$$

  The two definitions are equivalent [83], and we shall use them interchangeably.

- If there is some $1 \leq i \leq k$ such that for all $d_1, d_2, \ldots, d_k, d_i' \in D$ it is the case that

  $$\varphi(d_1, d_2, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k) = \varphi(d_1, d_2, \ldots, d_{i-1}, d_i', d_{i+1}, \ldots, d_k),$$

  then we say that φ *does not depend on its ith argument*.

[2]This is more commonly known as a *2-semilattice operation* in the Universal Algebra literature.

Note that constant operations are precisely those that do not depend on any of their arguments. Projections on the other hand depend on exactly one of their arguments (though not all such functions are projections).

The properties of being a majority operation, a binary ACI operation, or an affine operation are three examples of properties which are *defined by identities.* A property is said to be defined by identities if it is defined by giving one or more equations which must be true for every $d_1, d_2, \ldots d_r \in D$.

Jeavons, Cohen, and Gyssens [53] were able to identify four distinct classes of tractable sets of relations. These were characterised as those sets of relations which were closed under a constant operation, a majority operation, a binary ACI operation, or an affine operation respectively. At the time, these four classes together captured all known tractable sets of relations. More recently however, other researchers have discovered some more tractable sets [8, 11, 12, 25, 26, 56]. These too may be characterised by closure under sets of operations. In this thesis we shall refer explicitly only to the four classes from [53]. Most of what we shall say however, will also apply to the novel sets of relations.

Chapter 3

Adding Transitive Closure to Monadic NP

3.1 Introduction

In this chapter, we will examine the expressibility of logics such as mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$, formed by adding variants of the TC-operator to monadic existential second-order logic. We will examine this power both in the presence and the absence of a built-in successor relation.

The motivation for this strand of research stems ultimately from the question of whether or not **NP** = **co-NP**. Fagin [32] showed that this question is equivalent to the one of whether the fragments Σ^1_1 and Π^1_1 of second-order logic are equally expressive, and was able to make a start in investigating it by showing that the *monadic* fragment of Σ^1_1 is in fact not closed under complementation. Whilst this provides a small degree of evidence that full Σ^1_1 is not closed under complementation either, we are a long way from actually finding a proof of this conjecture.

It therefore seems sensible to concentrate our attention on a fragment of Σ^1_1 which, whilst more expressive than monadic Σ^1_1, is still not powerful enough to capture the whole of **NP**. Showing that such a logic is not closed under complementation may give clues as to how to go about proving the general case. Our hope (which we have been unable to realise) is to determine whether mon-$\Sigma_1^1(\pm\mathrm{TC})^*[\mathrm{FO}]$ is an example of such a logic. Despite this failure, we do exhibit a sublogic of mon-$\Sigma_1^1(\pm\mathrm{TC})^*[\mathrm{FO}]$ which is a strict extension of monadic Σ_1^1 and which is not closed under complementation.

Adding a transitive closure operator is not by any means the only natural extension of mon-Σ_1^1, of course. Binary Σ_1^1 (so-called binary **NP**), for example, has been studied in a certain amount of detail by Durand, Lautemann, and Schwentick [29]. However whether or not binary **NP** = binary **co-NP** is still an open question. Similarly, Ajtai, Fagin, and Stockmeyer have investigated the closure of monadic Σ_1^1 under first-order quantification and existential unary second-order quantification [4]. The resulting logic is more robust than monadic Σ_1^1, but it is unknown whether the class is closed under complementation.

Courcelle has examined the expressive power of monadic **NP** and, indeed, of mon-$\Sigma_1^1(\pm\mathrm{TC})^*[\mathrm{FO}]$ in a long-running series of papers (see [20] for a good overview). The bulk of his research has been concerned with investigating how the representation chosen for a structure, or set of structures, can affect its descriptive complexity. In particular, he shows how representing graphs in a way which allows quantification over edges and sets of edges (as well as the more usual vertices and sets of vertices) allows us to express graph properties in monadic **NP** which would otherwise require a stronger logic to express. DIRECTED-REACHABILITY for example, shown by Ajtai and Fagin to be inexpressible in monadic **NP** under the usual representation of graphs [3], becomes expressible under Courcelle's representation. Courcelle has also investigated the extent to which being able to guess an orientation on the edges of an undirected graph gives a logic added expressive power [21, 22, 23].

In the present chapter however, we will be concerned only with the usual representation of graphs over the binary signature $\sigma_2 = \langle E \rangle$. The material herein is motivated largely by Grädel's research [39] into $\mathrm{TC}^*[\mathrm{FO}]$.

3.2 The Ehrenfeucht-Fraïssé Game

We begin this chapter with a discussion of the following well-known game, devised by Ehrenfeucht [31] and Fraïssé [36]. It is (usually) used to prove inexpressibility results in first-order logic[1].

Definition 3.1 For every word p over the alphabet $\{\exists, \forall\}$ we define the *first-order quantifier class* $L^0(p)$ in FO inductively as follows:

- $L^0(\epsilon)$ contains the quantifier-free formulae; and
- for a quantifier $Q \in \{\exists, \forall\}$, the class $L^0(Qp)$ is the closure under conjunctions and disjunctions of the class $L^0(p) \cup \{(Qx_i)\varphi : \varphi \in L^0(p)\}$.

Definition 3.2 Let σ be a signature containing $s \geq 0$ constant symbols, and let Ω be a problem over σ. The Ehrenfeucht-Fraïssé game for Ω is a two-player game (the players are normally called *Spoiler* and *Duplicator*), which is played as follows.

- Spoiler begins by choosing some $k \in \mathbb{N}$, and fetches k pairs of pebbles: (u_1, v_1), (u_2, v_2), ..., (u_k, v_k).
- Duplicator then chooses some σ-structure $\mathcal{A} \in \Omega$. (Of course, her choice will depend on k.)
- She next chooses some σ-structure $\mathcal{B} \notin \Omega$.
- The game proper now begins. In the i^{th} round, Spoiler can choose to make one of two types of move.
  - **The $\exists$-move.** Spoiler takes pebble u_i, and places it on one of the domain elements of $\mathcal{A}$. Duplicator responds by placing pebble v_i on one of the domain elements of $\mathcal{B}$.

[1]It can, in principle, be used to show expressibility results also, but it is usually simpler just to give a suitably expressive first-order sentence in such cases.

  - **The $\forall$-move.** Spoiler takes pebble v_i, and places it on one of the domain elements of $\mathcal{B}$. Duplicator responds by placing pebble u_i on one of the domain elements of $\mathcal{A}$.

- The game ends when all $2k$ pebbles have been placed.

At the end of the game, Duplicator wins if, and only if, the pebbles determine a local isomorphism from $\mathcal{A}$ to $\mathcal{B}$. More precisely, let $a_1, \ldots, a_k$ and $b_1 \ldots, b_k$ be the elements of $\mathcal{A}$ and $\mathcal{B}$ respectively which are carrying the pebbles $u_1, \ldots, u_k$ and $v_1, \ldots, v_k$, and let $c_1, \ldots, c_s$ and $d_1, \ldots, d_s$ be the interpretations of the constant symbols of σ in the two structures. If the mapping $f : \mathcal{A} \longrightarrow \mathcal{B}$ with

$$f(a_i) = b_i \text{ for } i = 1, \ldots, k \text{ and } f(c_i) = d_i \text{ for } i = 1, \ldots, s$$

is well defined, and is an isomorphism between the substructures of $\mathcal{A}$ and $\mathcal{B}$ that are generated by the pebbled elements and constants, then Duplicator wins; otherwise Spoiler wins.

This game is defined in a slightly different manner to how it is usually defined in the literature. It is much more common to omit the first three steps, and to define the "k-round Ehrenfeucht-Fraïssé game on structures $\mathcal{A}$ and $\mathcal{B}$". We believe that the present definition (which is equivalent from the point of view of the following proposition) is more elegant, however.

The point of the game is encapsulated in the following proposition (which is stated in its "negative" form since that is the form in which it is most useful).

Proposition 3.3 [31, 36] Duplicator has a winning strategy in the Ehrenfeucht-Fraïssé game for Ω if, and only if, Ω is *not* definable in first-order logic.

In fact, this proposition can be strengthened. If $p \in \{\forall, \exists\}^*$ is a word, then we may define a restricted version of the Ehrenfeucht-Fraïssé game – the $L^0(p)$ game – which is the same as the usual game, save that the choice of whether to make a $\forall$ or $\exists$ move during the i^{th} round is not left up to Spoiler, but is instead determined by $p[i]$. (Thus, the length of p also determines the value chosen by Spoiler for k at the very beginning of the game.)

Theorem 3.4 [31, 36] Duplicator has a winning strategy in the $L^0(p)$ game for Ω if, and only if, Ω is not definable in $L^0(p)$.

3.3 Extending the Ehrenfeucht-Fraïssé Game

Theorem 3.4 is a very useful tool, and many important results can be proved using it. For example, it can be used to provide a simple proof that the problem PARITY is not definable in FO. However, one of its most important properties is that it can be extended in various ways so as to capture logics other than FO. A huge variety of such extensions have been considered in the literature; we shall be concerned here with just two of them. (We shall mention a third variation, the *infinitary game*, in Chapter 4.)

3.3.1 The Ajtai-Fagin Game

The Ajtai-Fagin game is a reworking by Ajtai and Fagin [3] of an earlier game by Fagin [33] which characterises definability in monadic **NP**.

Definition 3.5 Let $\mathcal{A}$ be some structure over a signature σ. A *colouring* of $\mathcal{A}$ by the s sets $S_1, S_2, \ldots, S_s$ is an assignment of a subset of $\{S_1, S_2, \ldots, S_s\}$ to each domain element of $\mathcal{A}$. Equivalently, it is an assignment of a subset of $|\mathcal{A}|$ to each set S_i.

By a *coloured structure*, we just mean a pair consisting of a structure and some colouring of it.

Note that once again, this definition varies throughout the literature. Some researchers refer to colouring a structure by c colours. Here, each colour represents a particular subset of sets. Of course, colouring a structure by s sets is equivalent to colouring it with 2^s colours.

Definition 3.6 The Ajtai-Fagin game for a problem Ω is played by Spoiler and Duplicator as follows.

- Spoiler chooses a number of rounds, k, and a number of sets, s. As in the original game, he fetches k pairs of pebbles.
- Duplicator chooses a structure $\mathcal{A} \in \Omega$.
- Spoiler colours $\mathcal{A}$ with the s sets $S_1, S_2, \dots, S_s$, forming the coloured structure $\mathcal{A}'$.
- Duplicator chooses a structure $\mathcal{B} \notin \Omega$.
- She colours it with the s sets $S_1, S_2, \dots, S_s$, forming $\mathcal{B}'$.
- The game now proceeds in the same way as does the main part of the Ehrenfeucht-Fraïssé game, for k rounds. As before, the game ends when all $2k$ pebbles have been placed.

The winning condition is essentially the same as before: Duplicator wins if, and only if, the mapping defined by the pebbles and the constant symbols of the structure is a partial isomorphism, where this partial isomorphism must *respect colour.* That is, if domain element $a \in \mathcal{A}'$ is a member of sets S_1, S_7, and S_{13} only (say), and if a is in the domain of the partial mapping f, then $f(a)$ is a member of sets S_1, S_7, and S_{13} only in $\mathcal{B}'$.

The motivation for the game is the following proposition.

Proposition 3.7 [3] Duplicator has a winning strategy in the Ajtai-Fagin game for the problem Ω if, and only if, Ω is not definable in monadic **NP**.

Once again, this proposition may be strengthened in a natural way. If $p \in \{\forall, \exists\}^*$ is a word, and $q \geq 0$, then we may define the *second-order quantifier class* $L^q(p)$ by

$$L^q(p) = \{\exists X_1 \exists X_2 \dots \exists X_q \varphi : \varphi \in L^0(p)\}.$$

We take $L^*(p) \equiv \bigcup_{q=1}^{\infty} L^q(p)$. Note that $L^0(p)$ coincides with our earlier definition, so there is no confusion in notation.

Furthermore, we may define the $L^q(p)$ game to be the same as the usual Ajtai-Fagin game, save that Spoiler is forced to choose q as his value of s and, in addition, must make his k first-order moves according to p. The $L^*(p)$ game is identical, except that Spoiler is not constrained in his choice of s.

Theorem 3.8 [3] For any $p \in \{\forall, \exists\}$ and any $q \geq 0$, Duplicator has a winning strategy in the $L^q(p)$ game (resp. the $L^*(p)$ game) for problem Ω if, and only if, Ω is not definable by sentence of $L^q(p)$ (resp. $L^*(p)$).

This theorem is a strict generalisation of Theorem 3.4, and can be used to prove many non-definabilty results for monadic Σ^1_1, amongst them the result mentioned in the previous chapter that PARITY is not definable in this logic.

3.3.2 Winning the Ajtai-Fagin Game

The game theoretic characterisations of logics which we have so far presented are all very well, but they suffer from one major drawback. It is usually very difficult in practice to prove that one or other player has a winning strategy in the Ω game. Fagin, Stockmeyer, and Vardi [34], following Hanf [47], developed the following technique which can often simplify such proofs.

For any structure $\mathcal{A}$, we say that $a, b \in |\mathcal{A}|$ are *adjacent* if either $a = b$, or there is some tuple t in some relation of $\mathcal{A}$, such that a and b both occur in t. The *degree* of $a \in |\mathcal{A}|$ is the number of domain elements which are adjacent to a, but not equal to a.

Definition 3.9 Let σ be a signature, and let $\mathcal{A}$ be a σ-structure. For any $a \in \mathcal{A}$, and $d \in \mathbb{N}$, we define the *neighbourhood of radius d about a* recursively as follows.

$$\begin{aligned} \mathit{Nbd}(1, a) &= \{a\} \\ \mathit{Nbd}(d+1, a) &= \{x \in |\mathcal{A}| : x \text{ is adjacent to some } b \in \mathit{Nbd}(d, a)\} \end{aligned}$$

Intuitively, $Nbd(d, a)$ consists of those points from $|\mathcal{A}|$ whose distance from a is strictly less than d.

Definition 3.10 Let $\mathcal{A}$ be a structure, or a coloured structure. For any $a \in |\mathcal{A}|$, and $d \in \mathbb{N}$, the d-type of a is the isomorphism type of the substructure of $\mathcal{A}$ which is induced by $Nbd(d, a)$, together with a distinguished constant representing the point a.

Definition 3.11 Let σ be a signature, and $d, m \in \mathbb{N}$. We say that the σ-structures $\mathcal{A}$ and $\mathcal{B}$ are (d, m)-equivalent if, for every d-type τ, either $\mathcal{A}$ and $\mathcal{B}$ have the same number of domain elements with d-type τ, or else they each have at least m domain elements which have d-type τ.

Fagin, Stockmeyer, and Vardi proved the following result (slightly reworded to fit in with our notation).

Theorem 3.12 [34] Let σ be a signature containing just relation symbols (and no constants). Let $l, f \in \mathbb{N}$. Then there are $d, m \in \mathbb{N}$, where d depends only on l, and m depends only on l and f such that whenever $\mathcal{A}$ and $\mathcal{B}$ are (d, m)-equivalent σ-structures (or coloured σ-structures) in which every domain element has degree at most f, then Duplicator has a winning strategy in the restriction of the Ehrenfeucht-Fraïssé game in which Spoiler is obliged to choose l as his value of k.

This theorem can often simplify the proof that Duplicator has a winning strategy in either the usual Ehrenfeucht-Fraïssé game, or in the Ajtai-Fagin game. For instead of actually having to exhibit a strategy, it is sufficient to prove that the two structures chosen by Duplicator (and coloured, in the case of the Ajtai-Fagin game) are, in some sense, "locally equivalent". This is often a considerably easier task.

In fact, the theorem also holds over signatures which contain constant symbols (see [69]). To see this, let $\sigma = \langle R_1, R_2, \ldots, R_r, C_1, C_2, \ldots, C_c\rangle$ be a signature containing the c constant symbols $C_1, C_2, \ldots C_c$, and let $\mathcal{A} \in \mathrm{STRUCT}(\sigma)$. Consider now the purely relational signature $\sigma' = \langle R_1, R_2, \dots, R_r, R'_1, R'_2, \dots, R'_c \rangle$ in which each constant symbol of σ has been replaced by a unary relation symbol. We define the structure $\mathcal{A}'$ over σ' in the following way.

The domain of $\mathcal{A}'$ is equal to the domain of $\mathcal{A}$. Each relation R_i of σ' is interpreted in $\mathcal{A}'$ in the same way as the corresponding relation R_i from σ is interpreted in $\mathcal{A}$. Each relation $R'_i(v)$ of σ' is defined to hold for precisely one value of $v \in |\mathcal{A}'|$; specifically for that value for which $(\mathcal{A}, v) \models C_i = x$. Clearly, the structures $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic. Consequently, for any d, m, a pair of structures $\mathcal{A}$ and $\mathcal{B}$ are (d, m)-equivalent if, and only if, the corresponding structures $\mathcal{A}'$ and $\mathcal{B}'$ are (d, m)-equivalent. Furthermore, Duplicator has a winning strategy for the Ajtai-Fagin game on $\mathcal{A}$ and $\mathcal{B}$ if, and only if, she has a winning strategy on $\mathcal{A}'$ and $\mathcal{B}'$.

We shall make use of this extended version of the theorem when we begin to play the mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$-game, defined below.

3.3.3 Grädel's TC Game

Another extension of the Ehrenfeucht-Fraïssé game, due to Grädel [39], allows us to analyse the expressive power of transitive closure logic, $(\pm\mathrm{TC})^*[\mathrm{FO}]$.

We begin by extending the first- and second-order quantifier classes defined above.

Definition 3.13 For every word p over the alphabet $\{\exists, \forall, \mathrm{TC}, \neg\mathrm{TC}\}$ we define the first-order quantifier class $L^0(p)$ in $(\pm\mathrm{TC})^*[\mathrm{FO}]$ inductively as follows:

- $L^0(\epsilon)$ contains the quantifier-free formulae.
- For a quantifier $Q \in \{\exists, \forall\}$, the class $L^0(Qp)$ is the closure under conjunctions and disjunctions of the class $L^0(p) \cup \{(Qx_i)\varphi : \varphi \in L^0(p)\}$.
- $L^0((\mathrm{TC})p)$ is the closure under conjunctions and disjunctions of the class of formulae $L^0(p) \cup \{\mathrm{TC}[\lambda\mathbf{x}, \mathbf{y}, \varphi(\mathbf{x}, \mathbf{y})](\mathbf{u}, \mathbf{v}) : \varphi \in L^0(p)\}$.

- $L^0((\neg\mathrm{TC})p)$ is the closure under conjunctions and disjunctions of the class of formulae $L^0(p) \cup \{\neg\mathrm{TC}[\lambda\mathbf{x},\mathbf{y},\varphi(\mathbf{x},\mathbf{y})](\mathbf{u},\mathbf{v}) : \varphi \in L^0(p)\ \}$.

These are strict extensions of the classes already introduced, so there will be no confusion over notation. We will henceforth assume that the logic $(\pm\mathrm{TC})^*[\mathrm{FO}]$ is augmented with two constant symbols, 0 and *max*, which are always interpreted as distinct domain elements of any structure over which we interpret a formula of the logic.

Proposition 3.14 [39] Every formula in $(\pm\mathrm{TC})^*[\mathrm{FO}]$ is equivalent to a formula in some $L^0(p)$ where $p \in \{\mathrm{TC}, \neg\mathrm{TC}\}^*$.

Proof We can always replace existential quantifiers by TC-operators: a formula $(\exists z)\psi(z)$ is equivalent to

$$\mathrm{TC}[\lambda x, y, (x = 0 \wedge \psi(y)) \vee (\psi(x) \wedge y = max)](0, max).$$

In a similar vein, universal quantifiers can always be replaced by $\neg$TC-operators. $\square$

Definition 3.15 The *quantifier depth* of any formula $\varphi \in (\pm\mathrm{TC})^*[\mathrm{FO}]$ is defined inductively by:

- $\mathit{depth}(\varphi) = 0$ if φ is atomic
- $\mathit{depth}(\neg\varphi) = \mathit{depth}(\varphi)$
- $\mathit{depth}(\varphi \wedge \psi) = \mathit{depth}(\varphi \vee \psi) = \mathit{max}\{\mathit{depth}(\varphi),\ \mathit{depth}(\psi)\}$
- $\mathit{depth}(\exists x\varphi) = \mathit{depth}(\forall x\varphi) = \mathit{depth}(\varphi) + 1$
- $\mathit{depth}(\mathrm{TC}[\lambda\mathbf{x},\mathbf{y},\varphi](\mathbf{u},\mathbf{v})) = \mathit{depth}(\varphi) + 2k$, where $\mathbf{x}$ and $\mathbf{y}$ are of length k.

Definition 3.16 [39] The Grädel game for the problem Ω is played as follows.

- Spoiler begins by choosing some $k \in \mathbb{N}$, and fetches the k pairs of pebbles (u_1, v_1), $(u_2, v_2), \ldots, (u_k, v_k)$.

- Duplicator chooses some structure $\mathcal{A} \in \Omega$.
- Duplicator chooses some structure $\mathcal{B} \notin \Omega$.
- The game proper now begins. During the i^{th} round, Spoiler chooses to make one of the following four varieties of moves.
  - **The $\exists$-move.** Spoiler takes pebble u_i, and places it on one of the domain elements of $\mathcal{A}$. Duplicator responds by placing pebble v_i on one of the domain elements of $\mathcal{B}$.
  - **The $\forall$-move.** Spoiler takes pebble v_i, and places it on one of the domain elements of $\mathcal{B}$. Duplicator responds by placing pebble u_i on one of the domain elements of $\mathcal{A}$.
  - **The TC-move.** Suppose that r pairs of pebbles are already on the board. Spoiler chooses some $l \leq (k-r)/2$ and selects a sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_m$ ($m \geq 1$) of l-tuples in $\mathcal{A}$ such that $\mathbf{x}_0$ and $\mathbf{x}_m$ consist only of constants and of already pebbled elements. Duplicator then chooses a similar sequence (though not necessarily one of the same length) of l-tuples $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_n$ ($n \geq 1$) in $\mathcal{B}$, where $\mathbf{y}_0 = f(\mathbf{x}_0)$ and $\mathbf{y}_n = f(\mathbf{x}_m)$ where f is the local isomorphism defined by the constants and the pebbles (see below). Spoiler now chooses two l-tuples $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$ which are adjacent in Duplicator's sequence. He lays down $2l$ pebbles on structure $\mathcal{B}$: one on each of the components of $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$ (including repetitions). Duplicator responds by choosing adjacent l-tuples $\mathbf{x}_j$ and $\mathbf{x}_{j+1}$ from Spoiler's sequence, and laying down $2l$ pebbles on the appropriate elements of $\mathcal{A}$.
  - **The $\neg$TC-move.** This is exactly the same as the TC-move, save that the rôles of the structures $\mathcal{A}$ and $\mathcal{B}$ are reversed.
- The game ends after all $2k$ pebbles have been placed. This may occur before k rounds have been completed.

When all of the pebbles have been placed, Duplicator wins if, and only if, the pebbles determine a local isomorphism from $\mathcal{A}$ to $\mathcal{B}$ in the sense which we have already described.

Observe that this game reduces to the usual Ehrenfeucht-Fraïssé game when Spoiler is restricted to making just $\forall$- and $\exists$-moves.

The associated proposition is as one might by now expect.

Proposition 3.17 [39] Duplicator has a winning strategy in the Grädel Game for Ω if, and only if, Ω is not definable by a sentence of $(\pm\mathrm{TC})^*[\mathrm{FO}]$.

As with the other games, this is just a special case of the following theorem. For a word $p \in \{\exists, \forall, \mathrm{TC}, \neg\mathrm{TC}\}$ and for $l \in \mathbb{N}$ we define the $(L^0(p), l)$ game to be the same as the Grädel game, save that Spoiler is restricted to choosing l as his value of k, and must make his moves according to the word p. (Note that unlike in previous games, the value of l is not determined by the length of p; all we have is that $l \geq |p|$.)

Theorem 3.18 [39] Duplicator has a winning strategy in the $(L^0(p), l)$ Grädel Game for Ω if, and only if, Ω is not definable by a sentence of $L^0(p)$ with quantifier depth l.

3.4 Extending monadic NP

In this section, we turn our attention to the logic mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$, and give an Ehrenfeucht-Fraïssé-type game which characterises definability within it. We begin with the definition of the logic itself.

Definition 3.19 A formula of mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$ is composed of $s \geq 0$ existential monadic second-order quantifiers, followed by a formula of $(\pm\mathrm{TC})^*[\mathrm{FO}]$.

Clearly, mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$ is at least as expressive as monadic **NP**, and also as $(\pm\mathrm{TC})^*[\mathrm{FO}]$. Furthermore, it is a strict extension of them both. For it contains the problem 3-COLOURABILITY, which is not definable in $(\pm\text{TC})^*[\text{FO}]$ (this is a consequence of the results in [27]), and also contains the problem CONNECTIVITY (see Example 2.5), which is known not to be definable in monadic **NP** [33, 34].

Before we present our game for precisely determining the expressive power of mon-$\Sigma_1^1(\pm\text{TC})^*[\text{FO}]$ we need the following definition, which establishes several sublogics of our logic. It should be compared with the definition immediately following Proposition 3.7.

Definition 3.20 If $p \in \{\forall, \exists, \text{TC}, \neg\text{TC}\}^*$ is a word, and $q \geq 0$, then we may define the second-order quantifier class $L^q(p)$ by

$$L^q(p) = \{\exists X_1 \exists X_2 \ldots \exists X_q \varphi : \varphi \in L^0(p)\}.$$

We take $L^*(p) \equiv \bigcup_{q=1}^{\infty} L^q(p)$. Note that this is just an extension of our previous definitions, so once again there is no confusion in notation.

Our game is a natural amalgamation of the Ajtai-Fagin game and the Grädel game.

Definition 3.21 Let σ be a signature, and let Ω be some problem over σ. The mon-$\Sigma_1^1(\pm\text{TC})^*[\text{FO}]$ game for Ω is played between Spoiler and Duplicator, and proceeds in the following way.

- Spoiler chooses some $k \in \mathbb{N}$, and a number of sets, s. As usual, he fetches k pairs of pebbles.
- Duplicator chooses a structure $\mathcal{A} \in \Omega$.
- Spoiler colours $\mathcal{A}$ with the s sets $S_1, S_2, \ldots, S_s$, forming the coloured structure $\mathcal{A}'$.
- Duplicator chooses a structure $\mathcal{B} \notin \Omega$.
- She colours it with the s sets $S_1, S_2, \ldots, S_s$, forming $\mathcal{B}'$.

- The game now proceeds in the same way as does the main part of the Grädel game. That is, Spoiler chooses to play either an $\exists$-move, a $\forall$-move, a TC-move, or a $\neg$TC-move, and Duplicator responds appropriately.
- The game continues until all $2k$ pebbles have been placed.

The winning condition is the same as in the Ajtai-Fagin game. That is, Duplicator wins if, and only if, the mapping defined by the pebbles and the constant symbols of the structure is well-defined, and is a partial isomorphism, where this partial isomorphism must respect colour.

The game gives rise to the following proposition, whose proof follows from that of Theorem 3.23 below.

Proposition 3.22 Duplicator wins the mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$ game for Ω if, and only if, Ω is not definable in mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$.

As usual, this result is just a special case of a more general result. For any word $p \in \{\forall, \exists, \mathrm{TC}, \neg\mathrm{TC}\}^*$, and for any $q, l \geq 0$, we may define the $(L^q(p), l)$ game to be the same as that defined above, save that Spoiler is obliged to choose l as his value of k, to choose q as his value of s, and to choose his moves according to the word p. Then we have the following result.

Theorem 3.23 Let Ω be a problem over some signature σ. Let $p \in \{\forall, \exists, \mathrm{TC}, \neg\mathrm{TC}\}^*$, and let $q, l \geq 0$. Then Duplicator has a winning strategy in the $(L^q(p), l)$ (resp. $(L^*(p), l)$ mon-$\Sigma^1_1(\pm\mathrm{TC})^*[\mathrm{FO}]$ game for Ω if, and only if, Ω is not definable in $L^q(p)$ (resp. $L^*(p)$) by a sentence of quantifier depth l.

Proof We shall only prove the theorem in the case where q is given. The case where q is unrestricted is similar.

We begin by proving the "if" direction. This is the most useful direction in practice, and is the easier to prove. In fact, we prove the contrapositive: that is, if Ω is definable by a sentence of $L^q(p)$ whose quantifier depth is l, then Duplicator does not have a winning strategy.

Suppose therefore that Ω is defined by the sentence $\psi = \exists X_1 \ldots \exists X_q \varphi$, where $\varphi \in L^0(p)$ has depth l. During the first step of the game, Spoiler is obliged to take $k = l$ and $s = q$. Duplicator responds by constructing some structure $\mathcal{A} \in \Omega$ which Spoiler must colour. Clearly, $\mathcal{A} \models \psi$. Consequently, there must be some assignment to the sets $X_1, X_2, \ldots, X_q$ so that the colouring $\mathcal{A}''$ of $\mathcal{A}$ by this assignment satisfies $\mathcal{A}'' \models \varphi$. Spoiler chooses the isomorphic colouring with the sets $S_1, S_2, \ldots, S_q$ to form $\mathcal{A}'$. Then $\mathcal{A}' \models \varphi'$, where φ' is the same as φ, save that occurrences of X_i have been replaced with the corresponding S_i. (Henceforth, we shall elide this difference, and refer to both formulae just as φ.)

Duplicator now responds by choosing some $\mathcal{B} \notin \Omega$, and colouring it. Since $\mathcal{B} \not\models \psi$, then no matter how she colours the structure (forming $\mathcal{B}'$), it is always the case that $\mathcal{B}' \not\models \varphi$.

The game now proceeds as does the Grädel game on Ω. The proof that Spoiler has a winning strategy (and hence that Duplicator does not) is identical to that of Theorem 3.18. It may be found in [39]. We need merely observe that the proof presented in [39] still goes through when the structures in question are coloured.

For the converse, we will prove that if Duplicator does not have a winning strategy in the $(L^q(p), l)$ mon-$\Sigma_1^1(\pm\mathrm{TC})^*[\mathrm{FO}]$ game for Ω, then Ω is definable in the corresponding logic.

If Duplicator does not have a winning strategy, then Spoiler must have such a strategy. That is, whichever $\mathcal{A} \in \Omega$ Duplicator chooses, it may be coloured by the q sets in such a way that whichever $\mathcal{B} \notin \Omega$ Duplicator chooses, and however she colours it, Spoiler has a winning strategy in the Grädel-part of the game. For any $\mathcal{A}$, denote Spoiler's winning colouring of $\mathcal{A}$ by $col(\mathcal{A})$.

Since the proof of correctness of the Grädel game goes through even when the structures are coloured, it follows that whichever $\mathcal{A} \in \Omega$ Duplicator chooses as her first structure, and whichever coloured $\mathcal{B}'$ she chooses as her second structure, there is at least one formula $\varphi_{\mathcal{A},\mathcal{B}'} \in L^0(p)$ of depth l such that $col(\mathcal{A}) \models \varphi$ and $\mathcal{B}' \models \neg\varphi$. Of course, $\varphi_{\mathcal{A},\mathcal{B}'}$ will make use of up to q free monadic second-order variables. Let $\Phi_{\mathcal{A}}$ be the conjunction of these $\varphi_{\mathcal{A},\mathcal{B}'}$ over all possible choices of $\mathcal{B}'$. This conjunction is permitted since there are only a finite number of such formulae up to logical equivalence. Note that $\Phi_{\mathcal{A}} \in L^0(p)$ and has quantifier depth l.

Now, clearly $col(\mathcal{A}) \models \Phi_{\mathcal{A}}$. Also, for every $\mathcal{B} \notin \Omega$ and for all possible colourings $\mathcal{B}'$ of $\mathcal{B}$, $\mathcal{B}' \models \neg\Phi_{\mathcal{A}}$. So let Ψ be the conjunction of $\Phi_{\mathcal{A}}$ over all $\mathcal{A} \in \Omega$. Once again, this is permitted since there are only a finite number of $\Phi_{\mathcal{A}}$ up to logical equivalence. Then $\Psi \in L^0(p)$ and has depth l and, moreover, for any $\mathcal{A} \in \Omega$, $col(\mathcal{A}) \models \Psi$. Furthermore, for all $\mathcal{B} \notin \Omega$ and for all colourings $\mathcal{B}'$ of $\mathcal{B}$, $\mathcal{B}' \models \neg\Psi$.

So for every $\mathcal{A} \in \Omega$

$$\mathcal{A} \models \exists X_1 \exists X_2 \ldots \exists X_n \Psi$$

where the X_i are chosen to be the free second order variables of Ψ.

Similarly, for every $\mathcal{B} \notin \Omega$ and

$$\mathcal{B} \models \forall X_1 \forall X_2 \ldots \forall X_n \neg\Psi.$$

Hence, $\exists X_1 \exists X_2 \ldots \exists X_n \Psi$ is an $L^q(p)$ sentence of depth l which defines Ω. □

Having defined this game, we may now use it to show that some (strict) extension of mon-Σ^1_1 is not closed under complementation. Immerman showed [49] that every sentence of $\mathrm{TC}^*[\mathrm{FO_s}]$ is equivalent to a sentence of $\mathrm{TC}^1[\mathrm{FO_s}]$ (see Chapter 2). He did this by showing inductively that for any two formulae $\varphi, \psi \in \mathrm{TC}^1[\mathrm{FO_s}]$, the formulae $\varphi \wedge \psi$, $\varphi \vee \psi$, $\forall x\varphi$, $\exists x\varphi$, and $\mathrm{TC}[\lambda \mathbf{w}, \mathbf{x}, \mathrm{TC}[\lambda \mathbf{y}, \mathbf{z}, \varphi](\mathbf{s}, \mathbf{t})](\mathbf{u}, \mathbf{v})$ are all equivalent to a formula from $\mathrm{TC}^1[\mathrm{FO_s}]$. In fact, his proof goes through in most of these cases even in the absence of a successor relation. The exception is the case of $\forall x\varphi$.

With that in mind, we define the following hierarchy.

Definition 3.24

- TC(0) is the set of all formulae of the form $\mathrm{TC}[\lambda \mathbf{x}, \mathbf{y}, \varphi(\mathbf{x}, \mathbf{y})](\mathbf{u}, \mathbf{v})$, where φ is first-order (and may contain free variables other than those of $\mathbf{x}$ and $\mathbf{y}$).

- $\forall$-TC(m) is the universal closure of TC(m) – *i.e.*, the set of formulae $(\forall \mathbf{x})\psi$, where $\psi \in \mathrm{TC}(m)$.

- TC($m+1$) is the set of formulae $\mathrm{TC}[\lambda \mathbf{x}, \mathbf{y}, \varphi(\mathbf{x}, \mathbf{y})](\mathbf{u}, \mathbf{v})$, where $\varphi \in \forall\text{-TC}(m)$.

Logics such as mon-Σ^1_1-TC(m) are also defined in the natural way, with the second-order quantifiers coming at the front.

Note that each of these classes is equivalent to the union of one or more classes $L^0(p)$ (resp. $L^*(p)$). Consequently, they may (in principle at least) be separated by playing an appropriate version of one of the games already defined in this chapter. The proof of the following lemma does exactly that.

Lemma 3.25 Let $\sigma = \langle E \rangle$ be a signature consisting of a binary relation symbol, E. Furthermore, let σ be augmented with the two distinct constants 0 and *max*. Then CONNECTIVITY cannot be expressed as a formula of the logic mon-Σ^1_1-TC(0) over σ.

Proof The appropriate game to play is the mon-$\Sigma^1_1(\pm\mathrm{TC})^*$[FO] game for CONNECTIVITY in which Spoiler is obliged to begin with a single TC-move, and then to restrict his moves to coming from the set $\{\forall, \exists\}$.

Spoiler begins by choosing $s, k \in \mathbb{N}$, and fetching the k pairs of pebbles. Duplicator's structure $\mathcal{A}$ is going to be a large, undirected cycle, so she takes $f = 2$ and $l = k$, and applies (the proof of) Theorem 3.12 to obtain d and m. Duplicator must now choose the length, p, of her cycle. She does this as follows.

Given an undirected path of length $2d$, there are exactly $(2^s)^{2d}$ ways of colouring it. Consequently, for any r, if Duplicator chooses $p \geq r \cdot (2d) \cdot (2^{2sd})$ then no matter how Spoiler colours the cycle, it must contain at least r identically coloured, non-overlapping regions of length $2d$. Call these regions $\Delta_1, \Delta_2, \ldots, \Delta_{r'}$ with $r' \geq r$. We add that in fact p may be made much smaller than this and this property would still hold. We are interested only in the existence of such a p, not a minimal value.

We must now give a suitable value for r.

The number of distinct isomorphism types of radius d on the cycle (ignoring constants for the moment) is less than 2^{2sd}. So the size of the power set of the set of isomorphism types is less than $2^{2^{2sd}}$. Therefore, if we choose $r \geq (m+k+3) \cdot 2^{2^{2sd}}$, then there must be a "gap" between two adjacent Δ_i regions with the property that the d-isomorphism type of every point in the gap occurs in at least $(m+k+2)$ other gaps between adjacent regions. Similarly, if we choose $r \geq 3(m+k+3) \cdot 2^{2^{2sd}}$, then there must be at least three such gaps.

So Duplicator chooses:

$$p \geq 3(m+k+3) \cdot 2^{2^{2sd}} \cdot (2d) \cdot 2^{2sd}$$

Duplicator constructs the cycle of length p, and places the constants 0 and *max* anywhere on the cycle. This is structure $\mathcal{A}$.

Spoiler colours in the cycle, forming $\mathcal{A}'$.

Duplicator finds one of the three gaps referred to above, so that the gap itself, along with the nearer half of the two identically coloured regions next to it, is constant-free. This is always possible since there are three gaps, and only two distinct constants. She takes this whole constant-free part of the cycle, and joins the ends together to form another, smaller, cycle (see Figure 3.1).

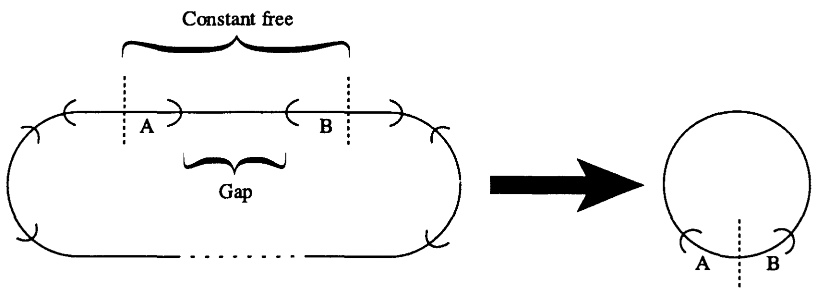

Figure 3.1: Cutting up a cycle

She then takes the disjoint union of this small cycle and (an isomorphic copy of) the original cycle as her second, disconnected graph, $\mathcal{B}$.

Duplicator colours this second graph in the natural way: she colours the first cycle in the same way as Spoiler coloured his cycle, and the second cycle using the colours induced from the first cycle. The resulting graph is $\mathcal{B}'$.

It is now time for the game proper to begin. Spoiler's strategy must consist of first making a TC-move, then carrying on with a normal, first-order game. So he begins by constructing a path of t-tuples in $\mathcal{A}'$.

Duplicator responds with the isomorphic path in the larger cycle of $\mathcal{B}'$.

Spoiler now chooses two adjacent t-tuples from Duplicator's path. Duplicator chooses the isomorphic t-tuples from Spoiler's path.

At this point, there are at most $k + 2$ pebbles and constants on each of the two graphs (and none at all on the smaller cycle of $\mathcal{B}'$). So by construction, $\mathcal{A}'$ and $\mathcal{B}'$ are still (d,m)-equivalent. Hence, by Theorem 3.12 (generalised, as described above, to hold over signatures containing constant symbols) we can conclude that Duplicator has a winning strategy in the game. □

Lemma 3.26 mon-Σ_1^1-TC(0) is a strict extension of mon-Σ_1^1.

Proof That mon-Σ_1^1-TC(0) is an extension is obvious. We need only prove strictness. In fact this is simple given Ajtai and Fagin's result [3] that DIRECTED REACHABILITY $\notin$ mon-Σ_1^1. It is easy to see that DIRECTED REACHABILITY can be expressed in TC(0) however, and hence *a fortiori* in mon-Σ_1^1-TC(0). □

Our theorem now follows almost immediately:

Theorem 3.27 mon-Σ_1^1-TC(0) is a strict extension of mon-Σ_1^1 which is not closed under complementation (provided that our signature contains two distinct constant symbols and a relation symbol of arity at least two).

Proof Fagin showed in [33] that non-CONNECTIVITY is definable in mon-Σ^1_1. Thus it is also definable in mon-Σ^1_1-TC(0). The preceding two lemmas complete the proof.

□

In fact Lemma 3.25 is a generalisation of a result originally due to Martin Otto, and presented (with a different proof) by Erich Grädel in [39]. That result states that CONNECTIVITY cannot be expressed in TC(0). The immediate corollary is that TC(0) $\neq$ $\forall$-TC(0). But Grädel goes further. Provided that we restrict our attention to structures containing at least two distinct constants, then he showed that the following holds.

Theorem 3.28 [39] For all $m \in \mathbb{N}$, TC(m) $\subsetneq$ $\forall$-TC(m) $\subsetneq$ TC($m+1$).

It would be nice to extend Lemma 3.25 along similar lines. That is, to show a result such as

"For all $m \in \mathbb{N}$, mon-Σ^1_1-TC(m) $\subsetneq$ mon-Σ^1_1-$\forall$-TC(m) $\subsetneq$ mon-Σ^1_1-TC($m+1$)".

However, proving such a result appears to be extremely difficult. In particular, Grädel's own methods [39] fail to generalise to the case where we have existential second-order quantifiers available.

Intuitively, the reason for this failure is as follows. We have already observed (see Proposition 3.14) that existential first-order quantifiers may be simulated by a TC quantifier. One consequence of this is that in proving the "even" levels of the TC-hierarchy to be strict; that is, to show that $\forall$-TC(m) $\subsetneq$ TC($m+1$), Grädel merely needed to show that $\forall$-TC(m) $\subsetneq$ $\exists$-$\forall$-TC(m). (We have not defined this class of formulae, but our meaning should be clear.)

However, when we have a second-order $\exists$ quantifier present, then this latter inequality is not strict. For a sentence of mon-Σ^1_1-$\exists$-$\forall$-TC(m) of the form

$$\exists X \exists y \forall \mathbf{z} \varphi$$

with $\varphi \in \mathrm{TC}(m)$, is equivalent to one of the form

$$\exists X \exists Y(\text{“}Y \neq \emptyset\text{”} \wedge \forall y(\neg Y(y) \vee \forall \mathbf{z}\varphi))$$

At first sight, it may appear that we have not gained very much: we still require a first-order existential quantifier to express “$Y \neq \emptyset$”. Note however that “$Y \neq \emptyset$” $\in \mathrm{TC}(0)$, and consequently may be moved inside φ. Thus our original sentence is equivalent to one from mon-Σ_1^1-$\forall$-TC(m).

As a consequence of this, any attempt to prove a hierarchy similar to Grädel's in the presence of monadic existential second-order quantifiers, must make some more essential use of TC than its ability to define first-order $\exists$ quantifiers. It is hard to see how such a use may be made.

3.5 The Effect of a Successor Relation

In this section we examine the logics mon-$\Sigma_1^1(\pm\mathrm{TC})^*[\mathrm{FO_s}]$ and mon-$\Sigma_1^1(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$ – *i.e.*, essentially we examine the effect of adding a successor relation to the logic of the previous section. Although we refer exclusively to the DTC operator in what follows, everything said is applicable to the TC operator also, provided references to the word “deterministic” are ignored.

We begin with an elementary result, but one which is necessary if we are to justify our working with these logics. Specifically, we must show that mon-$\Sigma_1^1(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$ is not the same logic as mon-$\Sigma_1^1(\mathrm{FO_s})$ (and, as indicated above, we must show something analogous for TC). Note that we cannot expect to be able to prove something similar for mon-$\Sigma_1^1(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$ and $(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$, since

$$\mathbf{L} = (\pm\mathrm{DTC})^*[\mathrm{FO_s}] \subseteq \text{mon-}\Sigma_1^1(\pm\mathrm{DTC})^*[\mathrm{FO_s}] \subseteq \mathbf{NP}.$$

So to do so would separate **L** and **NP**.

Lemma 3.29 Let $\sigma = \langle U \rangle$ be a signature containing one unary relation symbol. Then

$$\text{mon-}\Sigma_1^1(\mathrm{FO_s}(\sigma)) \subset \text{mon-}\Sigma_1^1(\mathrm{DTC})^1[\mathrm{FO_s}(\sigma)].$$

Proof That there is a containment is of course obvious. We need only prove strictness.

First note that σ-structures may be considered as strings over the set $\{0,1\}$. By Büchi's Theorem [10], the languages definable in mon-$\Sigma^1_1(\mathrm{FO_s}(\sigma))$ are precisely the regular languages. On the other hand, by the results of Immerman [49] the logic mon-$\Sigma^1_1(\mathrm{DTC})^1[\mathrm{FO_s}(\sigma)]$ is at least as powerful as **L**. To prove the lemma therefore, it is only necessary to exhibit a non-regular language which is recognisable in **L**. In fact there are many such languages; perhaps the simplest is that given by $\{1^n0^n : n \in \mathbb{N}\}$. □

We shall be concerned here with the sequence of logics q-mon-$\Sigma^1_1(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$, defined for any q to be fragment of monadic existential second-order logic with successor and a DTC operator whose formulae have at most q second order quantifiers. This gives rise to a natural hierarchy:

$$\begin{aligned}(\pm\mathrm{DTC})^*[\mathrm{FO_s}] = 0\text{-mon-}\Sigma^1_1(\pm\mathrm{DTC})^*[\mathrm{FO_s}] &\subseteq 1\text{-mon-}\Sigma^1_1(\pm\mathrm{DTC})^*[\mathrm{FO_s}]\\ &\subseteq 2\text{-mon-}\Sigma^1_1(\pm\mathrm{DTC})^*[\mathrm{FO_s}]\\ &\subseteq \cdots\end{aligned}$$

The zeroth level of this hierarchy is clearly equivalent to **L**. On the other hand, we have already observed that several **NP**-complete problems can be defined in the first level (KERNEL for example, and SATISFIABILITY). So proving any level of the hierarchy to be strict would be at least as hard as separating **L** and **NP**.

Whilst it is obvious that the entire hierarchy is contained in **NP**, it is an open question as to whether or not all the levels are equal. Once again, proving that they are not equal would involve separating **L** from **NP**. This time however the converse is not true: if the hierarchy exactly captures **NP** it might be tractable to prove it. Having said that, we believe that the former case is more likely: monadic **NP** seems sufficiently weak that adding just a **L** amount of extra power to it is unlikely to give it the full power of **NP**.

Interesting questions in this area include ones such as:

- What happens if the hierarchy collapses at, say, the nth level? Does this imply that it collapses further down as well? Does it imply some complexity-theoretically unlikely result, such as $\mathbf{L} = \mathbf{NP}$?

- Assuming that $\mathbf{L} \neq \mathbf{NP}$, and that the hierarchy is strict for at least the first few levels, are there any natural problems which are definable in the hierarchy which are not definable in either $(\pm\mathrm{DTC})^*[\mathrm{FO_s}]$ or mon-Σ^1_1?

These questions appear to be very hard, and we have been unable to find answers to them. We do, however, have a couple of tools which may be helpful in making inroads into the questions in the future.

We begin by exhibiting a class of reductions under which the hierarchy remains closed. Even more usefully, we define a class of reductions under which the individual levels of the hierarchy each remain closed. Cosmadakis' monadic first-order reductions provide an example of the former type of reduction; his monadic first-order reductions of factor 1 provide an example of the latter.

Definition 3.30 [19] Let $\tau = \langle R_0, R_1, \dots, R_{r-1}, C_0, C_1, \dots, C_{c-1}\rangle$ and σ be signatures, where for each i, the arity of R_i is denoted by a_i. (Note that we are numbering our relation and constant symbols from 0, rather than from 1 as elsewhere. This is to simplify our exposition slightly.) Let Ω be a class of τ-structures and let Γ be a class of σ-structures. A *monadic first-order reduction* from Γ to Ω is a tuple $\mathfrak{R} = (k, \overline{\xi}, \overline{\varphi^0}, \overline{\varphi^1}, \dots, \overline{\varphi^{r-1}}, \overline{\psi^0}, \overline{\psi^1}, \dots, \overline{\psi^{c-1}})$ with the properties that:

- k is a natural number. We call this the *factor* of the reduction.

- $\overline{\xi}$ is a k-tuple $(\xi_0, \xi_1, \dots, \xi_{k-1})$ of first-order formulae of arity 1.

- For each $0 \leq i < r$, $\overline{\varphi^i}$ is a k^{a_i}-tuple consisting of one first-order formula φ^i_t of arity a_i for every word $t \in \{0, \dots, k-1\}^{a_i}$.

- For each $0 \leq i < c$, $\overline{\psi^i}$ is a k-tuple $(\psi^i_0, \psi^i_1, \ldots, \psi^i_{k-1})$ of first-order formulae of arity 1.

And furthermore such that for every σ-structure $\mathcal{A}$, there is a τ-structure $\mathcal{B}$ such that:

- The universe of $\mathcal{B}$ is the set of all pairs $(x, j), x \in |\mathcal{A}|, j < k$, for which $(\mathcal{A}, x) \models \xi_j$.
- For every $i < r$ and for all tuples $((x_0, j_0), (x_1, j_1), \ldots, (x_{a_i-1}, j_{a_i-1})) \in |\mathcal{B}|^{a_i}$ it holds that:

$$R_i^{\mathcal{B}}((x_0, j_0), \ldots, (x_{a_i-1}, j_{a_i-1})) \iff (\mathcal{A}, x_0, \ldots, x_{a_i-1}) \models \varphi^i_{(j_0, \ldots, j_{a_i-1})}.$$

- For every $i < c$ there is exactly one $(x, j) \in |\mathcal{B}|$ such that $(\mathcal{A}, x) \models \psi^i_j$.
- For every $i < c$ and for every $(x, j) \in |\mathcal{B}|$

$$C_i^{\mathcal{B}} = (x, j) \iff (\mathcal{A}, x) \models \psi^i_j.$$

  Note that in conjunction with the last condition, this ensures that an interpretation of the constant symbols $C_0, \ldots, C_{c-1}$ exists in $\mathcal{B}$, and that it is unique.

- $\mathcal{B} \in \Omega \iff \mathcal{A} \in \Gamma$.

Theorem 3.31 [19] Monadic **NP** is closed under monadic first-order reductions.

Unfortunately, it is not easy to generalise Cosmadakis' proof techniques (see [69]) to the case where a DTC operator is present. To see why, we must first give just the barest bones of his proof.

Proof Let σ, τ, Γ, Ω, and $\mathfrak{R}$ be as in Definition 3.30, and suppose Ω is defined by a formula θ of the form

$$\theta = \exists X_0 \exists X_1 \ldots \exists X_{q-1} \psi$$

where ψ is a first-order formula in prenex normal form. We wish to massage θ into a formula Θ which expresses Γ. In essence, we just have to replace every relation symbol and constant symbol from τ with an appropriate formula from $\mathfrak{R}$. Technical problems arise however, since the universe constructed by the reduction consists of *pairs* (x, j), where x comes from the domain of the source structure, and where $j < k$. We construct Θ via a three-step process.

1. First, replace any subformula of ψ of the form $\forall x \chi(x)$ by
$$\forall x \bigwedge_{j=0}^{k-1} (\xi_j \rightarrow \chi'((x, j)))$$
where $\chi'((x, j))$ is obtained from $\chi(x)$ by replacing every occurrence of x by (x, j).

2. Next, replace any subformula of ψ of the form $\exists x \chi(x)$ by
$$\exists x \bigvee_{j=0}^{k-1} (\xi_j \wedge \chi'((x, j))).$$

3. Finally, replace relation symbols and constant symbols by appropriate formulae from $\mathfrak{R}$. The details are routine, and are omitted here since we shall not need them.

The first two steps in the above proof rely on the fact that universal quantification is the same as repeated conjunction, and existential quantification is the same as repeated disjunction. Because these operations are both associative and commutative, we may "factor out" the syntactically ill-formed $\forall(x, j)(\xi_j(x) \rightarrow \chi'((x, j)))$ and $\exists(x, j))\xi_j(x) \wedge \chi'((x, j)))$ in the manner shown.

Now consider trying to add a similar step to deal with a subformula of the form

$$\mathrm{DTC}[\lambda \mathbf{x}, \mathbf{y}, \chi(\mathbf{x}, \mathbf{y})](\mathbf{0}, \mathbf{max})$$

where $\mathbf{x}, \mathbf{y}, \mathbf{0}, \mathbf{max}$ are all of arity l. In this case, were we to replace $\mathbf{x}$ and $\mathbf{y}$ with $(\mathbf{x}, \mathbf{i})$ and $(\mathbf{y}, \mathbf{j})$ respectively, there would be no apparent way to "factor out" the $\mathbf{i}$'s and $\mathbf{j}$'s. Any path in the digraph built by χ is likely to move freely between vertices of the form $(\mathbf{x}, \mathbf{j})$ and vertices of the form $(\mathbf{x}', \mathbf{j}')$ with $\mathbf{x} \neq \mathbf{x}', \mathbf{j} \neq \mathbf{j}'$. Consequently, Cosmadakis' proof does not easily generalise to the present setting.

In order to show that mon-$\Sigma^1_1(\pm\text{DTC})^*[\text{FO}_\text{s}]$ is closed under monadic first-order reductions, we make use of the fact that the DTC operator gives us the ability to create new vertices from nothing. Consider once again the DTC-formula above, and suppose we interpret it over some structure $\mathcal{A}$. If we double the arity of $\mathbf{x}$ and $\mathbf{y}$, then (provided $|\mathcal{A}| \geq k$), we may treat pairs $(\mathbf{x}, \mathbf{j}), \mathbf{x} \in |\mathcal{A}|^l, \mathbf{j} \in \{0, 1, \ldots, k-1\}^l$ directly as pairs $(\mathbf{x}, \mathbf{x}'), \mathbf{x}, \mathbf{x}' \in |\mathcal{A}|^l$. All that is then required is a certain amount of management to ensure that the k copies of $\mathcal{A}$ fit together in the correct way.

The following proof fills out the details.

Theorem 3.32 mon-$\Sigma^1_1(\pm\text{DTC})^*[\text{FO}_\text{s}]$ is closed under monadic first-order reductions.

Proof Let Ω be a problem which is defined by a formula θ of mon-$\Sigma^1_1(\pm\text{DTC})^*[\text{FO}_\text{s}]$ over the signature $\tau = \langle R_0, R_1, \ldots, R_{r-1}, C_0, C_1, \ldots, C_{c-1}\rangle$ of constant and relation symbols, where the relation symbols have arity $a_0, a_1, \ldots, a_{r-1}$ respectively. By [49] (see Theorem 2.9), we may assume that θ is of the form:

$$\theta = \exists X_0 \exists X_1 \ldots \exists X_{q-1} \text{DTC}[\lambda \mathbf{x}, \mathbf{y}, \delta](\mathbf{0}, \mathbf{max})$$

where $\mathbf{x}$, $\mathbf{y}$, $\mathbf{0}$, and $\mathbf{max}$ are of arity l, and where δ is a quantifier-free projection.

Let $\mathfrak{R} = (k, \overline{\xi}, \overline{\varphi^0}, \ldots, \overline{\varphi^{r-1}}, \overline{\psi^0}, \ldots, \overline{\psi^{c-1}})$ be a monadic first-order reduction from another problem Γ (over a signature σ) to Ω. Let $\Gamma_{|_{\geq k}} \subseteq \Gamma$ consist of all those structures in Γ whose universe has size at least k. We will construct a sentence Θ of mon-$\Sigma^1_1(\pm\text{DTC})^*[\text{FO}_\text{s}]$ which defines $\Gamma_{|_{\geq k}}$. That Γ itself is definable by a sentence of mon-$\Sigma^1_1(\pm\text{DTC})^*[\text{FO}_\text{s}]$ will then follow immediately, since we can define the finite number of structures in $\Gamma \setminus \Gamma_{|_{\geq k}}$ explicitly using just first-order logic.

We have already indicated that we are going to treat domain elements of our target structure as pairs (x, y) of elements of our source structure. However, the formulae from $\mathfrak{R}$ are presented to us in a slightly inconvenient form. We need to be able to refer to, say, the y'th formula from the tuple $\overline{\varphi^2}$ (where y is a logical variable which represents a natural number less than k), or the x'th formula from the tuple $\overline{\xi}$.

An example should make this clearer. Define:

$$\Xi(x,y) =_{def} \bigvee_{j=0}^{k-1} (y = f_j \wedge \xi_j(x)).$$

Assume for the moment that $f_0, \dots, f_{k-1}$ are constants which are interpreted by distinct elements in any structure over which we interpret Ξ. Then for each $j < k$, $\Xi(x, f_j)$ is logically equivalent to $\xi_j(x)$. (In fact when we come to define Θ shortly, we will define $f_0 = 0$, $f_1 = succ(f_0)$, $\dots$, $f_{k-1} = succ(f_{k-2})$. This will prove convenient for us when we come to define 0, max, and a successor relation on the target structure of $\mathfrak{R}$.)

In a similar vein, for each $i < r$ we may define:

$$\Phi^i(x_0, \dots, x_{a_i-1}, y_0, \dots, y_{a_i-1}) =_{def} \bigvee_{t_0,\dots,t_{a_i-1}<k} ((\bigwedge_{j=0}^{a_i-1} y_j = f_{t_j}) \wedge \varphi^i_{(t_0,\dots,t_{a_i-1})}(x_0, \dots, x_{a_i-1}))$$

so that for every word $(t_0, \dots, t_{a_i-1}) \in \{0, \dots, k-1\}^{a_i}$, $\Phi^i(\mathbf{x}, f_{t_0}, \dots, f_{t_{a_i-1}})$ is logically equivalent to $\varphi^i_{(t_0,\dots,t_{a_i-1})}(\mathbf{x})$.

Finally, for each $i < c$ we define:

$$\Psi^i(x,y) =_{def} \bigvee_{j=0}^{k-1} (y = f_j \wedge \psi^i_j(x)).$$

Then as might be expected, for every $j < k$, $\Psi^i(x, f_j)$ is logically equivalent to $\psi^i_j(x)$.

We may use the formulae Ψ^i to define SETCONSTS$(d_0, \dots, d_{c-1}, e_0, \dots, e_{c-1})$. This asserts that the pairs (d_0, e_0), (d_1, e_1), $\dots$, (d_{c-1}, e_{c-1}) of elements from the source structure represent the constants $C_0, C_1, \dots, C_{c-1}$ respectively of the target structure:

$$\text{SETCONSTS}(d_0, \dots, d_{c-1}, e_0, \dots, e_{c-1}) =_{def} \bigwedge_{i=0}^{c-1} \Psi^i(d_i, e_i).$$

We would like to use similar formulae to define the logical constants 0 and max, but there is a small problem to do with the relativisation of the target universe by $\overline{\xi}$. Although we can order its potential members lexicographically from $(0, f_0)$ to (max, f_{k-1}), there is no guarantee that $\Xi(0, f_0)$, say, will hold. So to define the position of the logical constant 0, we must find the lexicographically smallest pair (d, e) such that $\Xi(d, e)$

holds.

$$\begin{aligned}\mathrm{SETZERO}(d,e) =_{def} &\ \Xi(d,e) \wedge ((d = 0 \wedge e = f_0) \vee \mathrm{DTC}[\lambda x, g, y, h, (\\ &((y = succ(x) \wedge g = h) \vee (x = max \wedge y = 0 \wedge h = succ(g))) \wedge \\ &((\neg\Xi(x,g) \wedge \neg\Xi(y,h)) \vee (\neg\Xi(x,g) \wedge y = d \wedge h = e)) \\ &)](0, f_0, d, e))\end{aligned}$$

Note that this formula relies on the fact that $f_{i+1} = succ(f_i)$ for each $i < k-1$. SETMAX(d,e) is defined symmetrically.

We are now ready to define Θ:

$$\begin{aligned}\Theta =_{def} &\ \exists X_{0,0} \ldots \exists X_{0,k-1} \ldots \exists X_{q-1,0} \ldots \exists X_{q-1,k-1} \\ &\exists f_0 \ldots \exists f_{k-1} \exists d_0 \ldots \exists d_{c-1} \exists d_c \exists d_{c+1} \exists e_0 \ldots \exists e_{c-1} \exists e_c \exists e_{c+1} (\\ &f_0 = 0 \wedge f_1 = succ(f_0) \wedge \ldots \wedge f_{k-1} = succ(f_{k-2}) \wedge \\ &\mathrm{SETCONSTS}(d_0, \ldots, d_{c-1}, e_0, \ldots, e_{c-1}) \wedge \\ &\mathrm{SETZERO}(d_c, e_c) \wedge \mathrm{SETMAX}(d_{c+1}, e_{c+1}) \wedge \\ &\mathrm{DTC}[\lambda x_0, \ldots, x_{l-1}, g_0, \ldots, g_{l-1}, y_0, \ldots, y_{l-1}, h_0, \ldots, h_{l-1}, \\ &\bigwedge_{i=0}^{l-1} (\Xi(x_i, g_i) \wedge \Xi(y_i, h_i)) \wedge \Delta]((d_c)^l, (e_c)^l, (d_{c+1})^l, (e_{c+1})^l)).\end{aligned}$$

Given all of the above, this should be fairly self-explanatory. We set up the variables f_i to refer to the various copies of the universe of our source structure; the pairs $(d_0, e_0), \ldots, (d_{c-1}, e_{c-1})$ give the location of the constants $C_0, \ldots, C_{c-1}$ in our target structure; finally, the pairs (d_c, e_c) and (d_{c+1}, e_{c+1}) give the location of the logical constants 0 and *max* respectively.

The DTC sub-formula itself is composed of two conjunctions. The first (itself a large conjunction) ensures that all the pairs (x_i, g_i) and (y_i, h_i) which we deal with are in the relativised universe of the target structure. The second, the sub-formula Δ, has not been discussed yet. It is derived from the qfp δ in the following way:

- Replace any sub-formula of δ of the form $x_i = y_j$ by $x_i = y_j \wedge g_i = h_j$. Other equalities should be dealt with in similar ways. $x_i = C_j$, for example, should be replaced by $x_i = d_j \wedge g_i = e_j$; similarly, $y_i = max$ should be replaced by $y_i = d_{c+1} \wedge h_i = e_{c+1}$.

- Replace any sub-formula of the form $x_i = succ(y_j)$ by

$$\begin{aligned}\mathrm{DTC}[\lambda v, s, w, t, (((&v = x_i \wedge s = h_i \wedge w = y_j \wedge t = g_j) \vee \\ &(v = x_i \wedge s = h_i \wedge \neg\Xi(w,t)) \vee \\ &(\neg\Xi(v,s) \wedge w = y_j \wedge t = g_j) \vee \\ &(\neg\Xi(v,s) \wedge \neg\Xi(w,t))) \wedge \\ &((v = succ(w) \wedge s = t) \vee \\ &(v = 0 \wedge w = max \wedge s = succ(t))))](x_i, g_i, y_j, h_j)\end{aligned}$$

The reason for this complexity is basically the same as the reason for the complexity in the definitions of SETZERO and SETMAX above: the lexicographic successor of (x_i, g_i) is not necessarily a member of the relativised universe of the target structure.

As before, we treat other sub-formulae of the same form similarly.

- Replace any sub-formula of the form $R_i(x_{j_0}, x_{j_1}, \ldots, x_{j_{a_i-1}})$ by

$$\Phi^i(x_{j_0}, x_{j_1}, \ldots, x_{j_{a_i-1}}, g_0, g_{j_1}, \ldots, g_{j_{a_i-1}}).$$

As usual, other similar sub-formulae are handled in the obvious way.

- Define:

$$\chi^i(x,y) =_{def} \bigvee_{s=0}^{k-1} (y = f_s \wedge X_{i,s}(x)).$$

Then for each $s < k$, $\chi^i(x, f_s)$ is logically equivalent to $X_{i,s}(x)$. So we may replace any sub-formula of the form $X_i(x_j)$, $i < q$, by $\chi^i(x_j, g_j)$. As usual, we handle sub-formulae of a similar form in the obvious way.

All that remains is to check that Θ does indeed define $\Gamma_{|_{\geq k}}$. Let $\mathcal{A}$ be a σ-structure of size $n \geq k$, and let $\mathcal{B}$ be the corresponding τ-structure given by the reduction $\mathfrak{R}$. Then the construction of Θ implies that Θ holds in $\mathcal{A}$ if and only if θ holds in $\mathcal{B}$. So we have:

$$\begin{aligned} \mathcal{A} \models \Theta &\iff \mathcal{B} \models \theta \\ &\iff \mathcal{B} \in \Omega \\ &\iff \mathcal{A} \in \Gamma \end{aligned}$$

So Θ defines $\Gamma_{|_{\geq k}}$. Thus there is a mon-$\Sigma^1_1(\pm\text{DTC})^*[\text{FO}_\text{s}]$ formula which defines Γ.
$\square$

Note that the above proof goes through essentially unchanged even in the absence of a successor relation. For the successor relation is only used for two purposes: first, to define $f_1, f_2, \ldots, f_{k-1}$, and second, to define SETZERO, SETMAX, and the replacement of subformulae of the form $x_i = succ(y_j)$. For the former purpose, a quantifier-free first-order formula which does not involve the successor realtion will suffice; the latter purpose is, of course, unnecessary in a successor-free logic.

As mentioned above, it is difficult to see how to extend (rather than, as here, to modify) the standard proof that monadic **NP** is closed under monadic first-order reductions to cope with the case when the DTC operator is present.

The following corollary is immediate from the proof of the theorem.

Corollary 3.33 For each $k \in \mathbb{N}$, k-mon-$\Sigma^1_1(\pm\text{DTC})^*[\text{FO}_\text{s}]$ is closed under monadic first-order reductions of factor 1.

Chapter 4

Program Schemes with Forall Instructions

4.1 Program Schemes

One of the central open problems of finite model theory is whether or not there is a logic capturing **PTIME**. That is, does there exist a logic whose sentences define exactly the problems (whose encodings are) recognisable by polynomial-time Turing machines? Of course, in order to make sense of this question, one needs to say exactly what one means by a "logic". This has been done by Gurevich [44] who formulated a very liberal definition which encompasses many different traditional logics as well as a variety of computational models. Over the years, a number of contenders have been suggested as logics which might capture **PTIME**, but so far all such logics, whilst only having the power to define problems within **PTIME**, have eventually been shown to be too weak to to define *every* problem in **PTIME**. Perhaps the best-known contender was *inductive fixed-point logic with counting* (see, for example, [30]). Immerman had suggested that this logic might capture **PTIME**, but this was later shown not to be the case by Cai, Fürer and Immerman [13].

In this chapter we introduce a new logic (in the sense of Gurevich) which defines only problems in **PTIME**. Whilst our logic is strictly more expressive than, for example, inductive fixed-point logic and can define problems such as PARITY, there remain problems in **PTIME** which are not definable in our logic. This comes as no surprise to us as (intuitively) our logic lacks the sophistication we feel any such logic must have were it to capture **PTIME**. In any case, it is not our real aim here to develop a logic which might capture **PTIME** (though we feel that our logic might provide a stepping-off point in the search for such a logic): rather, our primary motivation for introducing our logic is because such a logic arises naturally within the ongoing systematic study of the expressive power of classes of program schemes (see [6, 70, 71, 75, 81]). Broadly speaking, *program schemes* are models of computation which are amenable to logical analysis, yet closer to the general notion of programs than are logical formulae. Program schemes were extensively studied in the seventies, without much regard being paid to an analysis of resources, before a closer complexity analysis was undertaken in, mainly, the eighties. There are connections between program schemes and logics of programs, especially dynamic logic. (The reader is referred to [6] for references relating to the research mentioned in the preceding two sentences.)

We define our program schemes around "high-level" programming constructs such as arrays, while-loops, assignments, non-determinism, and so on, but so that the input comes in the form of a finite structure and, in general, there is no access to a linear ordering of the elements of the input structure. The program schemes defined in [70, 81] all involve arrays, while-loops and non-determinism. Allowing unrestricted access to arrays enables one to accept **PSPACE**-complete problems, whilst by restricting access to arrays (to be, in a sense, "write-once"), one can limit oneself to accept only problems in **NP** (although there is still sufficient power to accept **NP**-complete problems). Other classes of program schemes were defined in [6, 71] in which every problem accepted by such a program scheme is in **PTIME** and, furthermore, there are such program schemes accepting **PTIME**-complete problems. These program schemes involve while-loops, a stack and non-determinism. So as to emphasise that these models of computation are not given an ordering on the elements of an input structure, amongst the results in the aforementioned papers are that the class of problems accepted by any of the above classes of program schemes has a zero-one law (but, interestingly, not necessarily because the problems can be defined in bounded-variable infinitary logic as is often the case in finite model theory). On ordered structures, the above classes of program schemes capture the complexity classes **PTIME**, **NP** and **PSPACE** as appropriate (see also [75]).

In this chapter, we introduce a class of program schemes which we call RFDPS. This class is based on arrays, if-then instructions, and forall-loops, where our forall-loops result in parallel executions of a portion of code, with one execution for each element of the input structure. So as to provide a means for iteration, we allow portions of code to be repeatedly executed n times, where n is the size of the input structure. The class RFDPS arose through our efforts to replace the notion of a while-loop in earlier classes of program schemes with one of a forall-loop. Note that unlike the program schemes mentioned above, our program schemes are deterministic (and every problem accepted by such a program scheme is in **PTIME**).

A related model of computation has recently been examined by Blass, Gurevich and Shelah. In [9], these authors introduced a model of computation called $\tilde{\text{C}}$PTime (*Choiceless Polynomial-Time*), a program $(\rho, p(n), q(n))$ of which is an adapted *Abstract State Machine* ρ (see [45, 46]) augmented with two polynomial bounds, $p(n)$ and $q(n)$, with $p(n)$ bounding the length of any run of the machine on any input and $q(n)$ bounding the number of "parallel executions" in one of their forall-loops, where these polynomial bounds are in terms of the size n of the finite input structure upon which the program works. Although such a program takes a finite structure (over some relational signature) as input, it treats the elements of this finite structure as atoms and has the potential to build certain sets over these atoms and use these sets as new "elements" in its "computational domain". Consequently, without restricting the run-time and the number of parallel executions, the program would have the capacity to build a computational domain of arbitrary size; and, indeed, it is not difficult to show that such an unrestricted program can simulate an arbitrary Turing machine. The instructions, or rules in the terminology of [9], of the programs of $\tilde{C}$PTime have similarities with those of the program schemes of this chapter. For example: there are dynamic function symbols and assignments via update rules, whereas our program schemes have arrays and assignment blocks; there are conditional rules, whereas our program schemes have if-then-fi blocks; and there are do-forall rules, whereas our program schemes have forall-do-od blocks. However, there are a number of important differences between the computational model of Blass, Gurevich and Shelah and ours, including the following. Their computational domain fluctuates, whereas ours is fixed and is always the domain of the input structure. Viewed as a logic, $\tilde{C}$PTime is three-valued (a program may accept, reject, or neither accept nor reject), whereas our program schemes always either accept or reject. A program of $\tilde{C}$PTime has no access to the cardinality of the input structure, and the problem PARITY cannot be accepted by a program of $\tilde{C}$PTime (furthermore, it has been reported in [9] that Shelah has shown that $\tilde{C}$PTime has a zero-one law), whereas our program schemes have access to the size of the input structure and there is such a program scheme accepting PARITY. In order to force the abstract state machine to accept polynomial-time solvable problems, the polynomial bounds $p(n)$ and $q(n)$ must be imposed from without, whereas no such bounds need be imposed upon our program schemes: our program schemes naturally accept only polynomial-time solvable problems.

The motivation for the research in [9] was the search for an answer to the question, stated earlier, of whether there is a logic capturing **PTIME**. In turn, this question has motivated a search for logics capturing an increasing sub-class of the class of polynomial-time solvable problems. The main results of [9] are that the class of problems accepted by the programs of $\tilde{C}$PTime properly contains the class of problems accepted by Abiteboul and Vianu's class of *polynomial-time relational machines* [1] but that there are polynomial-time solvable problems, in particular PARITY and

BIPARTITE MATCHING, that are not accepted by any program of $\tilde{C}$PTime. (BIPARTITE MATCHING consists of those bipartite undirected graphs whose two sets in the partition have equal size, for which there exists a perfect matching.) In fact, it is also shown in [9] that BIPARTITE MATCHING is not accepted by any program of $\tilde{C}$PTime$^+$, an extension of $\tilde{C}$PTime which allows access to a constant symbol whose value is fixed at the size of the input structure (however, PARITY is accepted by a program of $\tilde{C}$PTime$^+$). It is also claimed in [9] that the class of problems accepted by the programs of $\tilde{C}$PTime includes any problem definable in any other "polynomial-time logic" in the literature (the authors presumably mean only "natural polynomial-time logics" and not augmentations of such by, for example, counting quantifiers or Lindström quantifiers).

Our results are of a somewhat different flavour to those of [9] and, in a sense, are more refined. We obtain a strong result which provides limitations on the problems accepted by our program schemes, and we use this result to obtain a strict, infinite hierarchy of classes of problems within the class of problems accepted by the program schemes of RFDPS. These classes are parameterised by the depth of nesting of forall-loops allowed in the defining program schemes, and the union of these classes is the class of problems accepted by the program schemes of RFDPS. Consequently, each class of problems in the hierarchy is definable by a logic in Gurevich's sense. To our knowledge, this is the first strict, infinite hierarchy in a polynomial-time logic properly extending inductive fixed-point logic (with the property that the union of the classes of the hierarchy consists of the class of problems definable in the polynomial-time logic). Our results are obtained by a direct analysis of computations of our program schemes. Note that the existing hierarchy theorems of finite model theory, such as those in [41, 42, 43], are of no use to us here, since all of these hierarchy results are for explicit fragments of bounded-variable infinitary logic (which has a zero-one law). On the other hand, our computational model is, first, not defined in terms of traditional logics, and second, has no zero-one law (and so in particular cannot be a fragment of bounded variable infinitary logic, making the existing hierarchy theorems useless to us).

Like the Choiceless Polynomial-Time model of Blass, Gurevich and Shelah, our program schemes are different from other (polynomial-time) models of computation more prevalent in database theory, such as the relational machines of Abiteboul and Vianu [1], the extension of inductive fixed-point logic with a symmetry-based choice operator proposed by Gire and Hoang [38] and the extension of first-order logic with for-loops proposed by Neven, Otto, Tyszkiewicz, and Van den Bussche [66]. The models of computation proposed by these researchers (and others) allow the construction of whole relations as an atomic operation, whereas our construction of relations (stored in arrays) is, in a sense, "one element at a time". Some of these models are more expressive than our class of program schemes but, unlike our class of program schemes, no hierarchy results have been established. Hence, we do not discuss these models further here (although the reader is referred to our comments in Section 4.5).

4.2 Definitions

The classes of program schemes we shall be concerned with here are yet further generalisations of those presented in [75], and extended in various ways in [6, 70, 71, 81]. We dispense with the WHILE instructions of these previous papers, and replace them with a REPEAT instruction (which iteratively executes a block of instructions a number of times depending on the size of the input structure) and a FORALL instruction (which allows a large degree of parallel computation to take place). We also permit the use of arrays.

Definition 4.1 A *program scheme* $\rho \in$ RFDPS involves a finite set $\{x_1, x_2, \ldots, x_k\}$ of *variables*, for some $k \geq 1$; a finite (possibly empty) set $\{A_1, A_2, \ldots, A_g\}$ of *array symbols*, for some $g \geq 0$; and is over a signature σ. Each array symbol A_i has an associated arity $a_i \geq 1$.

The set of *basic terms* consists of the variables $\{x_1, x_2, \ldots, x_k\}$ together with the constant symbols of σ and the additional constant symbols 0 and *max* (which we assume are not present in σ). An *array term* is a string of the form $A_i[\tau_1, \tau_2, \ldots, \tau_{a_i}]$, where each τ_j is a basic term. Note that we do not allow array terms to be nested. (This could be simulated if required by introducing extra variables. The details are trivial.) The set of *terms* is the union of the set of basic terms and the set of array terms.

Informally, a *program scheme* consists of an *input instruction*, a sequence of *blocks* of instructions, and an *output instruction*. More technically, it is an *input-output block*, defined by

```
INPUT($x_1, x_2, \ldots, x_k$)          input instruction
    $\alpha_1$                                block of instructions
    $\alpha_2$                                block of instructions
    ⋮
    $\alpha_l$                                block of instructions
OUTPUT($x_1, x_2, \ldots, x_k$)         output instruction
```

for some $l \geq 1$. The *scope* of this block is the whole of the program scheme.

A program scheme has only one input-output block. All of its other blocks are defined recursively as follows.

- An *assignment block* α is simply an instruction of the form

  ```
      $\tau := \tau'$                           assignment instruction
  ```

  where τ is a variable or an array term and τ' is a term. The *scope* of α is the actual assignment instruction constituting the block.

- An *if-then-fi block* α is a sequence of instructions of the form

  ```
      IF $\varphi$ THEN                         if instruction
  ```

```
        $\alpha_1$                          *block of instructions*
        $\alpha_2$                          *block of instructions*
        ⋮
        $\alpha_l$                          *block of instructions*
    FI                                      *fi instruction*
```

for some $l \geq 1$, where φ is a quantifier-free first-order formula over $\sigma \cup \{0, max\}$ whose free variables come from $\{x_1, x_2, \ldots, x_k\}$ (note that we do not allow array terms within φ, though once again this could be simulated if required). The *scope* of α is the union of the if instruction, the fi instruction and the scopes of the blocks $\alpha_1, \alpha_2, \ldots, \alpha_l$.

- A *repeat-do-od block* α is a sequence of instructions of the form

```
    REPEAT DO                               *repeat-do instruction*
        $\alpha_1$                          *block of instructions*
        $\alpha_2$                          *block of instructions*
        ⋮
        $\alpha_l$                          *block of instructions*
    OD                                      *repeat-od instruction*
```

for some $l \geq 1$. The *scope* of α is the union of the repeat-do instruction, the repeat-od instruction and the scopes of the blocks $\alpha_1, \alpha_2, \ldots, \alpha_l$.

- A *forall-do-od block* α is a sequence of instructions of the form

```
    FORALL $x_p$ WITH $A_i^j$ DO            *forall-do instruction*
        $\alpha_1$                          *block of instructions*
        $\alpha_2$                          *block of instructions*
        ⋮
        $\alpha_l$                          *block of instructions*
```

OD	*forall-od instruction*

for some $l \geq 1$, where $1 \leq p \leq k$, $1 \leq i \leq g$ and $1 \leq j \leq a_i$. The variable x_p is called the *control variable* and the array symbol A_i is called the *control array symbol* of the forall-do-od block. The *scope* of α is the union of the forall-do instruction, the forall-od instruction and the scopes of the blocks $\alpha_1, \alpha_2, \ldots, \alpha_l$. If there exists at least one assignment instruction in the scope of α where the term on the left-hand side of the assignment is an array term involving A_i, then we say that the control variable x_p is *active* in A_i in α at index j. If there is no such assignment instruction, then we say that x_p is *inactive* in α. Furthermore, there are some additional constraints on α.

1. Any array term $A_i[\tau_1, \tau_2, \ldots, \tau_{a_i}]$ appearing in any assignment instruction (on the left or on the right) in the scope of α must be such that the term τ_j is x_p (the control variable).
2. No array symbol apart from the control array symbol may appear in a term on the left-hand side of any assignment instruction in the scope of α.
3. The control variable x_p must not appear (as a solitary variable) on the left-hand side of any assignment instruction in the scope of α.
4. Any other forall-do-od block (with either an active or an inactive control variable) whose instructions are in the scope of α must not have x_p as its control variable.

We will come to the reason for these constraints shortly. For now, it is enough to observe that they force both x_p, and all entries of arrays other than A_i to remain unchanged in value within the scope of the forall-do-od block.

We say that a block α *appears* in the scope of another block α' if the instructions in the scope of α are in the scope of α'. The *depth of nesting* of an instruction in the scope of ρ or of a block appearing in ρ is the number of forall-do-od blocks in whose scope the instruction or the block appears.

In a similar vein, the *scope* of the program scheme ρ is defined to be the scope of its input-output block; we say that a block α *appears* in ρ if it appears within the scope of the input-output block of ρ; and the *depth of nesting* of ρ is the maximum of the depth of nesting of all instructions of ρ. □

Our name for our class of program schemes, RFDPS, is an acronym for 'Repeat Forall Deterministic Program Schemes'.

With the above definitions in mind, we can explain how our program schemes compute. The following definition is relatively informal; a more rigorous definition of the semantics will be presented in the next section. Nevertheless, these informal semantics will be sufficient for us to give examples of our program schemes, and also to give some lower bound results on their computational power.

Definition 4.2 A program scheme $\rho \in$ RFDPS over σ takes as input a σ-structure $\mathcal{A}$. The variables and array elements all take values from $|\mathcal{A}|$, with array elements being indexed by tuples of elements of the input structure (the length of the tuple is the arity of the array symbol). The program scheme proceeds sequentially through its component instructions in the obvious way, but according to the following rubrics.

- At the beginning of the computation, the constant sybols 0 and *max* are given arbitrary distinct values from $|\mathcal{A}|$. All variables, and all array elements are set to the value 0.
- A repeat-do-od block α of the form

```
REPEAT DO                     repeat-do instruction
    $\alpha_1$                          block of instructions
    $\alpha_2$                          block of instructions
    ⋮
    $\alpha_l$                              *block of instructions*
OD                                          *repeat-od instruction*
```

repeatedly executes the blocks α_1, α_2, $\ldots, \alpha_l$ in sequence, a total of $|\mathcal{A}|$ times. Intuitively, the effect is the same as that of writing

$$\alpha_1; \alpha_2; \ldots; \alpha_l; \alpha_1; \alpha_2; \ldots; \alpha_l; \ldots \alpha_1; \alpha_2; \ldots; \alpha_l$$

in place of the repeat-do-od block, where there are $|\mathcal{A}|$ repetitions of $\alpha_1; \alpha_2; \ldots; \alpha_l$. Of course this is is impossible within our syntax, since program schemes are fixed, whereas the value of $|\mathcal{A}|$ depends on the particular input structure under consideration.

- A forall-do-od block α of the form

```
FORALL $x_p$ WITH $A_i^j$ DO                *forall-do instruction*
    $\alpha_1$                              *block of instructions*
    $\alpha_2$                              *block of instructions*
    ⋮
    $\alpha_l$                              *block of instructions*
OD                                          *forall-od instruction*
```

causes a "multi-way split" in the computation path of ρ. On encountering the instruction, a total of $|\mathcal{A}|$ "child processes" are set off in parallel, each executing the blocks $\alpha_1, \alpha_2, \ldots, \alpha_l$ of instructions, and each with its own local copy of the variables $x_1, x_2, \ldots, x_k$ of ρ. This latter stipulation ensures that the child processes cannot somehow use these variables to communicate with each other. The only difference between the processes is that x_p takes a different value in each. That is, for each $u \in |\mathcal{A}|$, there is a unique child process with the property that x_p takes the value u within it. The value of x_p cannot change within a process (this is a consequence of the third and fourth syntactic constraints in our definition of the forall-do-od block) but other variables are free to change, and hence to take a different value in two different processes.

Arrays, on the other hand, are *not* local to the individual processes: each process makes use of exactly the same set of arrays. This could lead to conflicts and "race conditions" were it not for our four syntactic constraints. The first of these ensures that the processes each have exclusive access to their own unique part of A_i. Specifically, in the child process in which x_p takes the value $u \in |\mathcal{A}|$, only array elements from the set

$$\{A_i[u_1, \ldots, u_{j-1}, u, u_{j+1}, \ldots, u_{a_i}] : u_1, \ldots, u_{j-1}, u_{j+1}, \ldots, u_{a_i} \in |\mathcal{A}|\}$$

may be referred to. By the second constraint however, A_i is the only array which may be changed by an instruction in the scope of α. Of course, two processes might both have read-only access to the same elements of arrays other than A_i, but no process has write access to these arrays. Consequently, conflicts cannot occur and non-determinism is not introduced.

When all of the child processes have reached the forall-od instruction, they terminate, and the computation of ρ proceeds as follows. Call those variables which occur alone on the left hand side of an assignment instruction within the scope of α, the *local variables* of α. (Note that it is quite possible that α may have no local variables.) The main computation resumes at the instruction in ρ following the forall-do-od instruction, but according to the following provisos. The value of x_p is set to be either 0 or max. It is set to max if, in every child process, each of the local variables was set at max on termination. Otherwise, x_p is set to 0. If α has no local variables, then x_p is always set to max. The values of all variables apart from x_p now take the original values they had immediately prior to the execution of α. The values of the arrays remain unchanged at the end of the block. As we have already observed, they are in an entirely consistent state.

To re-cap: execution of a forall-do-od block leaves the values of all variables (save, perhaps, its control variable) unchanged; it has an effect which is signalled by its control variable; and it may alter the value of some of the elements of its control array. All other arrays remain unaffected by it.

The structure $\mathcal{A}$ is *accepted* by ρ if, and only if, there exist distinct values of 0 and max such that the computation described above reaches the output instruction with all variables set at max.

What we are doing is "building in" two distinct constant symbols, 0 and max, but we are building them in with a slightly different semantics than is usual in finite model theory. Traditionally, as explained in Chapter 2, we would not consider a program scheme to be well-formed unless its acceptance, or otherwise, of a structure was independent of the placements of 0 and max. This raises the question of whether or not it is decidable to determine whether or not a program scheme of RFDPS is well-formed.

Under the semantics presented here however, every program scheme is automatically well-formed. Consequently, RFDPS is truly a "logic" in, say, the sense of Gurevich [44]. Moreover, we shall presently come to define some fragments of RFDPS. Using our semantics, it is immediately obvious that these fragments are all logics in the sense of [44]; using the traditional semantics, this fact would be far from clear.

We phrase the following trivial observations in the form of a lemma.

Lemma 4.3 The computation of a program scheme $\rho \in$ RFDPS on some input structure always terminates; and every problem in RFDPS can be solved in polynomial time.

A couple of examples will help to clarify these definitions.

Example 4.4 Given any signature σ, the problem PARITY of deciding whether the size of the universe over which a program scheme is executed is even, can be expressed by the following program scheme of RFDPS.

```
INPUT $(x_1, x_2)$
$x_1 := max$
REPEAT DO
    $x_2 := 0$
    IF $(x_1 = 0)$ THEN DO
        $x_1 := max$
        $x_2 := max$
    OD
    IF $(x_2 = 0)$ THEN DO
        $x_1 := 0$
        $x_2 := max$
    OD
OD
OUTPUT $(x_1, x_2)$
```

We present program schemes in an indented style to aid readability.

During each iteration of the repeat loop, the value of the variable x_1 is swapped between 0 and *max*. It therefore ends with the value *max* if, and only if, the universe has even size. The variable x_2 ends every iteration of the repeat loop with the value *max*. Thus, this program scheme accepts precisely those structures whose universes are of even size.

Remark 4.5 Before proceeding to our next example, it is worth pointing out one or two things concerning the nesting of forall-do-od blocks. Suppose we have one such block, α, nested inside another as follows:

```
FORALL $x_{m_1}$ WITH $A_{i_1}^{j_1}$ DO
    $\ddots$
        FORALL $x_{m_2}$ WITH $A_{i_2}^{j_2}$ DO
```

```
            $\alpha_1$
            $\vdots$
            $\alpha_l$
        OD
      $\iddots$
   OD.
```

The syntactic constraints concerning the nesting of forall-do-od blocks force $m_1 \neq m_2$, but they deliberately say nothing either about the relationship between i_1 and i_2 or about that between j_1 and j_2. There are three possibilities for this relationship:

1. $i_1 \neq i_2$. The second constraint, applied to both forall-do-od blocks, implies that no term of *any* array can appear on the left hand side of any assigment instruction in the scope of α. In other words, arrays within the scope of the two blocks are read-only, and x_{m_2} must consequently be inactive within α. (Of course, whether or not x_{m_1} is inactive within the outer forall-do-od block depends on the instructions which we have elided with ellipses.)

2. $i_1 = i_2$ and $j_1 = j_2$. Applying the first constraint to both forall-do-od blocks implies that any reference to A_{i_1} within the scope of α would be forced to have both x_{m_1} and x_{m_2} simultaneously in the j_1'th position in the array. Consequently, no instruction within the scope of α can make any reference whatsoever (be it read or write) to A_{i_1}. So x_{m_2} is once again inactive within α.

3. $i_1 = i_2$ but $j_1 \neq j_2$. In this case the two sets of constraints dovetail neatly: array terms involving A_{i_1} can appear on either side of any assignment instruction in the scope of α, whilst those involving other arrays may only appear on the right hand side. Similarly, array terms in the scope of α which involve A_{i_1} must all have x_{m_1} as their j_1'th entry, and x_{m_2} as their j_2'th entry; the terms of other arrays are unconstrained in this regard. We cannot say anything *a priori* about

whether or not the control variables are inactive within their blocks.

We reiterate that these remarks are not additional constraints on forall-do-od blocks; rather, they are natural consequences of the four constraints which we enumerated when we originally introduced the blocks.

Having set up our formalism, we now proceed to systematically abuse notation. Firstly, and most trivially, we will allow ourselves a good deal of freedom in our choice of variable names: x, y_3, and *work_var* are examples of the sorts of variables we will be using. Array names will undergo a similar treatment. To prevent confusion however, array names will always be in upper case, whilst variable names will always be in lower case.

Secondly, we will allow ourselves a richer syntax than is strictly permitted under the formalism. The following extended syntaxes will all be used.

- An if-then-else-fi block of the form

```
IF $\varphi$ THEN          if instruction
    $\alpha_1$             block of instructions
    $\alpha_2$             block of instructions
    $\vdots$
    $\alpha_l$             block of instructions
ELSE                       else instruction
    $\beta_1$              block of instructions
    $\beta_2$              block of instructions
    $\vdots$
    $\beta_m$              block of instructions
FI                         fi instruction
```

for some $l, m \geq 1$, where φ is a quantifier-free first-order formula is just an abbreviation for

```
test := 0                           test is a new variable
IF ($\varphi$) THEN DO
    test := max
    $\alpha_1$
    $\vdots$
    $\alpha_l$
OD
IF (test $\neq$ max) THEN DO
    test := max
    $\beta_1$
    $\vdots$
    $\beta_m$
OD
```

and has the obvious semantics. Note that if this code fragment occurs within the scope of a forall-do-od block, then care must be taken to set the value of *test* to *max* before the termination of the program scheme, else the program scheme will reject all input structures.

- An array-free-forall-do-od block, α, of the form

```
FORALL $x_p$ DO                     array-free-forall-do instruction
    $\alpha_1$                      block of instructions
    $\alpha_2$                      block of instructions
    $\vdots$
    $\alpha_l$                      block of instructions
OD                                  array-free-forall-od instruction
```

for some $l \geq 1$, is just an abbreviation for

```
FORALL $x_p$ WITH $A^1_\nu$ DO
```

```
        $\alpha_1$
        $\alpha_2$
        $\vdots$
        $\alpha_l$
    OD
```

where A_ν is a new array symbol of arity 1.

Although just an abbreviation, we may define, informally at least, the *scope* of α to be the union of the array-free-forall-do instruction, the array-free-forall-od instruction, and the scopes of the blocks $\alpha_1, \alpha_2, \ldots, \alpha_l$. Similarly, we may refer to x_p as the *control variable* of α.

The constraints on those instructions which may appear within the scope of α follow from the constraints which must be present on the translation of α. In particular:

- The only array symbol which can occur on the left hand side of an assignment instruction within the scope of α is A_ν. Since A_ν is a new array symbol however, it follows that *no* array term may appear on the left hand side of such an assignment instruction.
- The control variable x_p must not occur alone on the left hand side of any assignment instruction in the scope of α.
- Any other forall-do-od block (whether an abbreviated, array-free block or otherwise) whose instructions are in the scope of α must not have x_p as its control variable.

Note that since A_ν is entirely new, it follows from Remark 4.5 that the *same* array symbol A_ν may be used in the translation of *all* array-free-forall-do-od blocks. Note too, that by the above comments x_p is inactive in the translation of α. In fact, the converse is true as well: it is not hard to see that any forall-do-od block with an inactive control variable may be recast as an array-free-forall-do-od block. Consequently, array-free-forall-do-od blocks are sometimes known as *inactive forall-do-od blocks.* We shall use the terms interchangably.

Our second example is a little more complicated than our first, but shows that even **PTIME**-complete problems may be expressed in RFDPS.

Example 4.6 Let $\sigma = \langle R, c, d\rangle$ be a signature containing a relation symbol R of arity 3, and two constant symbols c and d. A structure $\mathcal{A} \in \text{STRUCT}(\sigma)$ of size n can be viewed as a *path system*; that is, as a set of vertices $\{0, 1, \ldots, n-1\}$ and a set of *rules* of the form *"From x and y deduce z"* (one such rule for each $\langle x, y, z\rangle$ such that $R(x, y, z)$ holds in $\mathcal{A}$), together with two distinguished vertices c and d. The set of *accessible* vertices of $\mathcal{A}$ is constructed inductively in the following way:

- the vertex c is accessible;
- if x and y are accessible vertices (not necessarily distinct), then every z which may be deduced from x and y by an application of some rule is also accessible.

The problem PATH-SYSTEM is defined as follows.

Instance: A structure $\mathcal{A} \in \text{STRUCT}(\sigma)$.

Yes-instance: A structure $\mathcal{A} \in \text{STRUCT}(\sigma)$ such that d is in the set of accessible vertices.[1]

The problem PATH-SYSTEM has long been known to be **PTIME**-complete via logspace reductions [17], and in fact the logic $(\pm\text{PS})^*[\text{FO}_s]$ has more recently been shown to capture **PTIME** [78] (where PS is the generalised quantifier corresponding

[1]Occasionally, PATH-SYSTEM is defined over a signature where c and d are unary relations rather than constant symbols. In this set-up, all the vertices in c are defined to be initially accessible, and the yes-instances of the problem are those structures for which there exists some $x \in d$ such that x is accessible. It is easy to see that the two definitions are interchangeable.

to PATH-SYSTEM). We now show that PATH-SYSTEM may be defined by a program scheme of RFDPS.

Consider the following program scheme over σ which uses the two unary arrays A and T.

```
INPUT $(w, x, y, z)$
$A[c] := max$
REPEAT DO
    FORALL $z$ DO
        $w := 0$                                          *Is $z$ now accessible?*
        FORALL $x$ DO
            FORALL $y$ DO
                IF $(A[x] = 0 \vee A[y] = 0 \vee \neg R(x, y, z))$ THEN
                    $w := max$
                FI
            OD
        OD
        IF $(A[z] = max \vee x = 0)$ THEN                 *If so, set $T[z] = max$*
            $T[z] := max$
        FI
    OD
    FORALL $x$ WITH $A^1$ DO                              *We now copy $T$ over $A$*
        $A[x] := T[x]$
    OD
OD
$w := max$                                                *$A[x] = max$ iff $x$ is accessible.*
$x := max$
$y := max$
```

```
$z := 0$
IF $(A[d] = max)$ THEN
    $z := max$
FI
OUTPUT $(w, x, y, z)$
```

The first thing to observe about ρ is that it is indeed a program scheme of RFDPS. This is routine to check: all that it requires is a verification that the syntactic conditions on instructions within the scope of forall-do-od blocks are not violated. We now show that ρ does in fact accept precisely the yes-instances of PATH-SYSTEM. We prove this in some detail, since the ideas behind the construction of the program scheme will crop up several times in the sequel.

The key loop is the REPEAT loop which surrounds much of the rest of the program scheme. We maintain the invariant that after the i^{th} iteration of this loop, the value of $A[x]$ will be max if, and only if, x is accessible from c after i or fewer applications of a rule. This obviously holds after the zeroth iteration: $A[c] = max$, and the rest of A is zero. Within the loop, for each vertex z we must determine whether z is accessible after this iteration; we store this information temporarily in the array T. By definition, z is accessible after the i^{th} iteration if, and only if, either it was already accessible (*i.e.,* $A[z] = max$) or else it has just become accessible (*i.e.,* it is not the case that for every x and for every y, either $A[x] = 0$, or $A[y] = 0$, or $R(x, y, z)$ fails to hold). Finally, we copy the array T over the array A, ready for the next iteration of the repeat instruction. (We do not need to zero the array T at the end of every iteration, since the set of accessible vertices increases monotonically.)

This process of gradually increasing the set of accessible vertices must reach a stable state after at most n iterations, since if even only one vertex is added to the set after each iteration, n iterations would be sufficient for every vertex to have become accessible. So when the repeat instruction terminates, the array A contains, as claimed, a complete characterisation of the set of accessible vertices. The rest of the program scheme is trivial. We set virtually all of the input-output variables to *max* immediately, but set the final such variable to *max* if, and only if, $A[d]$ holds. Thus ρ accepts a structure $\mathcal{A}$ if, and only if, $\mathcal{A} \in$ PATH-SYSTEM.

We now exhibit some lower bounds on the class of problems accepted by the program schemes of RFDPS.

Theorem 4.7 Given any first-order definable problem Ω, there is a program scheme of RFDPS which accepts Ω.

Proof We shall prove the result by induction on the quantifier-rank d of any first-order formula where our induction hypothesis is:

> *Let σ be some signature and σ' be the expansion of σ with m additional constant symbols. For any first-order formula ψ of quantifier-rank r less than $d \geq 1$ over σ and with free variables $x_1, x_2, \ldots, x_m$, say, there exists a program scheme $\rho' \in$ RFDPS over σ' such that if ψ is considered as a sentence over σ' then for every σ'-structure $\mathcal{A}'$:*
>
> - *if $\mathcal{A}' \models \psi$ then $\mathcal{A}' \models \rho'$; and*
> - *if $\mathcal{A}' \not\models \psi$ then the computation of ρ' on input $\mathcal{A}'$ (no matter what the distinct values given to 0 and max are) is such that the output instruction is reached with all variables involved in ρ' having the value 0.*
>
> *Moreover, ρ' does not involve any array symbols and has depth of nesting r.*

Let φ be a first-order formula of quantifier-rank $d \geq 1$ of the form

$$\exists x_m \psi(x_1, x_2, \ldots, x_m),$$

where $x_1, x_2, \ldots, x_m$ are the free variables of ψ. By the induction hypothesis, there exists a program scheme ρ' over σ' (the expansion of σ with m additional constant symbols) such that for every σ'-structure $\mathcal{A}'$:

- if $\mathcal{A}' \models \psi$ then $\mathcal{A}' \models \rho'$; and
- if $\mathcal{A}' \not\models \psi$ then the computation of ρ' on input $\mathcal{A}'$ is such that the output instruction is reached with all variables involved in ρ' having the value 0.

Moreover, ρ' does not involve any array symbols and has depth of nesting $d-1$. Let ρ denote the program scheme ρ' with the input- and output instructions stripped away; and suppose that the variables involved in ρ' are those of the tuple $\mathbf{y}$. Also, regard x_m now as a variable as opposed to a constant symbol (we assume for the sake of exposition that the name of the constant symbol of σ' which corresponds to the variable x_m is also x_m). Define the program scheme ρ'' over σ'' (the expansion of σ with a constant symbol for each of the variables $x_1, x_2, \ldots, x_{m-1}$) as follows.

```
INPUT($\mathbf{y}$, $x_m$)
FORALL $x_m$ DO
    $\rho'$
    IF $\mathbf{y} = \mathbf{0}$ THEN
        $\mathbf{y} := \mathbf{max}$
    ELSE
        $\mathbf{y} := \mathbf{0}$
    FI
OD
IF $x_m = max$ THEN
    $(\mathbf{y}, x_m) := (\mathbf{0}, 0)$
ELSE
    $(\mathbf{y}, x_m) := (\mathbf{max}, max)$
    FI
    OUTPUT(y, $x_m$)
```

The shorthand used above should be obvious (except that **0** and **max** denote tuples of the constant symbols 0 and *max*, respectively, of the appropriate lengths). The program scheme ρ'' is clearly as required. The case where φ is of the form $\forall x_m \psi(x_1, x_2, \ldots, x_m)$ is similar (indeed, somewhat simpler). As the induction hypothesis holds for all quantifier-free formulae, the result follows. □

So RFDPS is strictly more expressive than first-order logic (the strictness follows from Example 4.4). In fact, this result can be generalised.

Theorem 4.8 Given any problem Ω which is definable in inductive fixed-point logic, there is a program scheme of RFDPS which accepts Ω.

Proof Let $\varphi(\mathbf{y}, \mathbf{z})$ be a formula of inductive fixed-point logic of the form

$$\mathrm{IFP}[\lambda\mathbf{x}, R, \psi(\mathbf{x}, \mathbf{y}, R)](\mathbf{z}),$$

where: $|\mathbf{x}| = |\mathbf{z}| = k$; R is a relation symbol of arity k, not in the underlying signature σ; and ψ is a first-order formula whose free variables are those of the k-tuple $\mathbf{x}$ and the m-tuple $\mathbf{y}$. Suppose that there is a program scheme ρ' over σ' (the extension of σ with $k + m$ additional constant symbols called $x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_m$) with the following properties. Involved in ρ' is an array symbol B of arity k which does not appear on the left-hand side of an assignment instruction. We shall regard the array symbol B as being 'free' in the sense that we shall set the values of its elements from without (and so B will not be zeroed whenever execution of ρ' commences); and we shall only be interested in valuations of B for which every element is either 0 or *max*. In this way, B models a k-ary relation over the elements of the input structure. Furthermore, the program scheme ρ' is such that for every σ-structure $\mathcal{A}$, for every

k-tuple $\mathbf{u}$ and m-tuple $\mathbf{v}$ over $|\mathcal{A}|$ and for every array valuation val(A) modelling the k-ary relation R, as above,

$$((\mathcal{A}, \mathbf{u}, \mathbf{v}), \text{val}(B)) \models \rho' \text{ if, and only if, } \psi^{\mathcal{A}}(\mathbf{u}, \mathbf{v}, R)$$

(of course, $(\mathcal{A}, \mathbf{u}, \mathbf{v})$ is the σ'-structure obtained from $\mathcal{A}$ by augmenting $\mathcal{A}$ with the $k + m$ constants $(\mathbf{u}, \mathbf{v})$).

Suppose that the variables involved in the program scheme ρ' are those of the tuple $\mathbf{w}$, and regard the additional constant symbols $x_1, x_2, \ldots, x_k$ now as variables. Consider the program scheme ρ over σ'' (the extension of σ with m additional constant symbols $y_1, y_2, \ldots, y_m$) built as follows, where A is another array symbol of arity k.

```
INPUT(w, x)
REPEAT DO                                              k nested repeat-do-od
    REPEAT DO                                          blocks
        ⋱
            REPEAT DO
                FORALL $x_1$ WITH $B^1$ DO                      copy A to B
                    FORALL $x_2$ WITH $B^2$ DO
                        ⋱
                            FORALL $x_k$ WITH $B^k$ DO
                                $B[x_1, x_2, \ldots, x_k] := A[x_1, x_2, \ldots, x_k]$
                            OD
                        ⋰
                    OD
                OD
                FORALL $x_1$ WITH $A^1$ DO                      Iterate B, placing
                    FORALL $x_2$ WITH $A^2$ DO                  result in A
                        ⋱
                            FORALL $x_k$ WITH $A^k$ DO
                    $\mathbf{w} := \mathbf{0}$
                    $\rho'$
                    IF $\mathbf{w} = \mathbf{max}$ THEN
                        $A[x_1, x_2, \ldots, x_k] := max$
                    FI
                OD
              ⋰
            OD
          OD
        OD
      ⋰
  OD
OD
$(\mathbf{w}, \mathbf{x}) := (\mathbf{max}, \mathbf{max})$
OUTPUT($\mathbf{w}$, $\mathbf{x}$)
```

where ρ' has its input- and output- instructions stripped away. We claim that the program scheme ρ is such that for every σ-structure $\mathcal{A}$ and for every m-tuple $\mathbf{u}$ over $|\mathcal{A}|$, $(\mathcal{A}, \mathbf{u}) \models \rho$ and on termination, the array element $A[\mathbf{z}]$ encodes $\varphi^{\mathcal{A}}(\mathbf{u}, \mathbf{z})$, the inductive fixed point of $\psi^{\mathcal{A}}(\mathbf{x}, \mathbf{u}, R)$. To see that this claim is true, note that ρ computes in a similar way to the program scheme for PATH-SYSTEM in Example 4.6. Each iteration of the innermost repeat-do-od block performs one iteration of the formula ψ, temporarily storing its result in the array A. Array A is then copied back over B before execution resumes at the start of the loop. Because the arity of R is k, this process must have reached a stable state after at most $|\mathcal{A}|^k$ iterations of the loop.

Immerman [48] proved that every problem definable in inductive fixed-point logic can be defined by a sentence of the form

$$\exists z_1 \exists z_2 \ldots \exists z_k \mathrm{IFP}[\lambda \mathbf{x}, R, \psi(\mathbf{x}, R)](z_1, z_2, \ldots, z_k),$$

where ψ is first-order. A combination of the above construction and that of Theorem 4.7 yields the required result. □

4.3 A More Formal Semantics

Having introduced the program schemes of RFDPS, let us now give a more formal description of a computation of a program scheme on some finite structure. This more formal description will be necessary when we come to prove some (very refined) limitations of our program schemes.

Let the program scheme $\rho \in$ RFDPS be over the signature σ, and involve the variables of $\{x_1, x_2, \ldots, x_k\}$ and have h instructions. Let $\mathcal{A}$ be a σ-structure of size n. An *instantaneous description* (*ID*) of ρ on $\mathcal{A}$ is a tuple $(\mathbf{V}, \mathbf{A}, I, \mathbf{R})$ consisting of:

- a tuple $\mathbf{V}$ which contains a value of $|\mathcal{A}|$ for every variable of $\{x_1, x_2, \ldots, x_k\}$ (with the components ordered in some canonical fashion);
- a tuple $\mathbf{A}$ which contains a value of $|\mathcal{A}|$ for every array element of

  $$\{A[u_1, u_2, \ldots, u_a] \;:\; A \text{ is an array symbol in } \rho, \text{ of arity } a, \text{ and } u_1, u_2, \ldots, u_a \in |\mathcal{A}|\}$$

  (with the components ordered in some canonical fashion);
- a number $I \in \{1, 2, \ldots, h\}$; and
- if instruction I of ρ is within the scope of r repeat-do-od blocks then an r-tuple $\mathbf{R}$ of numbers from the set $\{1, 2, \ldots, n\}$.

We can represent a computation of ρ on $\mathcal{A}$ as a labelled acyclic digraph $\mathcal{G}(\rho^{\mathcal{A}})$, whose vertices are labelled with IDs of ρ on input $\mathcal{A}$, built as follows. Start with two vertices q_0 and q_1, and an edge (q_0, q_1). Label the vertex q_0 with the ID $(\mathbf{V}, \mathbf{A}, I, \mathbf{R}) = (\mathbf{0}, \mathbf{0}, 1, \epsilon)$ (this represents the values of the variables and the array elements and the instruction about to be executed in the computation of ρ on input $\mathcal{A}$ initially, where ϵ denotes the empty tuple). If the second instruction is not a repeat-do instruction then label the vertex q_1 with the ID $(\mathbf{0}, \mathbf{0}, 2, \epsilon)$, otherwise label q_1 with the ID $(\mathbf{0}, \mathbf{0}, 2, (1))$ (this represents the values of the variables and the array elements and the instruction about to be executed after execution of the first instruction, the input instruction, of ρ on input $\mathcal{A}$). Now apply the following rules as many times as possible.

- If the instruction associated with (the ID labelling a) vertex q is an assignment instruction of the form $\tau := \tau'$, where τ and τ' are terms, then create a new vertex q' and include the edge (q, q'). Label the vertex q' with the same ID as that labelling vertex q except:
  - with the value of the term τ altered so that it is made equal to the value of the term τ' (where value means according to the ID labelling vertex q);
  - with the value of I increased by 1; and
  - if the instruction whose number corresponds to the new value of I is a repeat-do instruction then tag an extra component onto the tuple $\mathbf{R}$ and give this component the value 1.
- If the instruction associated with vertex q is an if instruction, involving some test φ, then create a new vertex q' and include the edge (q, q'). Label the vertex q' with the same ID as that labelling vertex q except:
  - with the value of I increased by 1 if the test φ holds in $\mathcal{A}$ when the values of any terms in φ are taken according to the ID labelling vertex q;
  - with the value of I made equal to 1 plus the number of the fi instruction corresponding to the if instruction if the test φ does not hold; and
  - if the instruction whose number corresponds to the new value of I is a repeat-do instruction then tag an extra component onto the tuple $\mathbf{R}$ and give this component the value 1.

- If the instruction associated with vertex q is a fi instruction then create a new vertex q' and include the edge (q, q'). Label the vertex q' with the same ID as that labelling vertex q except:
  - with the value of I increased by 1; and
  - if the instruction whose number corresponds to the new value of I is a repeat-do instruction then tag an extra component onto the tuple $\mathbf{R}$ and give this component the value 1.
- If the instruction associated with vertex q is a repeat-do instruction then create a new vertex q' and include the edge (q, q'). Label the vertex q' with the same ID as that labelling vertex q except:
  - with the value of I increased by 1; and
  - if the instruction whose number corresponds to the new value of I is a repeat-do instruction then tag an extra component onto the tuple $\mathbf{R}$ and give this component the value 1.
- If the instruction associated with vertex q is a repeat-od instruction then create a new vertex q' and include the edge (q, q'). Label the vertex q' with the same ID as that labelling vertex q except:
  - if the value of the final component of $\mathbf{R}$ is not equal to n then increase this value by 1 and set the value of I to be the value of the corresponding repeat-do instruction; or
  - if the value of the final component of $\mathbf{R}$ is equal to n then remove the final component from $\mathbf{R}$ and increase the value of I by 1, unless the instruction whose number corresponds to the new value of I is a repeat-do instruction when we do not remove this final component but simply reset it to 1.

- If the instruction associated with vertex q is a forall-do instruction, for which the control variable is x_p, then create n new vertices $q_0, q_1, \ldots, q_{n-1}$ and include edges $(q, q_0), (q, q_1), \ldots, (q, q_{n-1})$. Label the vertices of $\{q_0, q_1, \ldots, q_{n-1}\}$ with the same ID as that labelling vertex q except:
  - with the values of x_p (in $\mathbf{V}$) in each of the IDs set at a unique value of $|\mathcal{A}|$;
  - with the value of I increased by 1; and
  - if the instruction whose number corresponds to the new value of I is a repeat-do instruction then tag an extra component onto the tuple $\mathbf{R}$ and give this component the value 1.
- If the instruction associated with vertex q is a forall-od instruction of a forall-do-od block α, where the control variable corresponding to this instruction is x_p, then find the (unique) first ancestor q'' of q (working backwards up the already constructed acyclic digraph) for which the instruction associated with q'' is the forall-do instruction corresponding to our forall-od instruction. Let Q be the set of leaves, *i.e.*, vertices of out-degree 0, of the already constructed acyclic digraph that are descendants of q''. If the instruction associated with every vertex of Q is our forall-od instruction then create a new vertex q' and include edges $\{(q, q') : q \in Q\}$. Label the vertex q' with the following ID.
  - The value of $\mathbf{V}$ is the same as the value of $\mathbf{V}$ in the ID labelling vertex q'' except if the values of the local variables of α in the IDs labelling the vertices of Q are all *max* then the value of x_p is made equal to *max*; otherwise it is made equal to 0. If α has no local variables then the value of x_p is made equal to *max*.
  - The values of the array elements of $\mathbf{A}$ are as they are in the ID labelling vertex q'' except that if any of these array elements has a different value in any of the IDs labelling vertices of Q then the value of the array element

in the ID labelling the vertex q' is the new value at this vertex of Q (note that because of our syntactic restrictions on forall-do-od blocks, all array elements in the ID labelling q' are well defined).

- The value of I is increased by 1.
- The values of $\mathbf{R}$ are the same as in the ID labelling vertex q'', unless the instruction whose number corresponds to the new value of I is a repeat-do instruction, when we tag an extra component onto the tuple $\mathbf{R}$ and give this component the value 1.

For any block α appearing in ρ, there might be a number of connected subgraphs of $\mathcal{G}(\rho^{\mathcal{A}})$ corresponding to α (where by 'connected' we mean with respect to the underlying undirected graph obtained from $\mathcal{G}(\rho^{\mathcal{A}})$ by replacing all directed edges with undirected ones): this is because the block α might appear in the scope of a repeat-do-od block or a forall-do-od block. We call the subgraphs of $\mathcal{G}(\rho^{\mathcal{A}})$ corresponding to these connected subgraphs *images* of α in $\mathcal{G}(\rho^{\mathcal{A}})$, and we denote an image by $\mathrm{Im}^{\mathcal{A}}(\alpha)$ (it is always clear as to which image of α we are referring). Note that every image has a *source*, the unique vertex of in-degree 0, and a *sink*, the unique vertex of out-degree 0. Note also that the sink of one image will generally be the source of another image, and that the digraph $\mathcal{G}(\rho^{\mathcal{A}})$ is formed by gluing together images of blocks by identifying sources and sinks. The *source* of $\mathcal{G}(\rho^{\mathcal{A}})$ is the unique vertex of in-degree 0, and the *sink* of $\mathcal{G}(\rho^{\mathcal{A}})$ is the unique vertex of out-degree 0. We can clearly talk of a child and a parent of a vertex of $\mathcal{G}(\rho^{\mathcal{A}})$ (indeed, we have already spoken of ancestors and descendants).

Let q be a vertex of $\mathcal{G}(\rho^{\mathcal{A}})$ and let τ be some term. We denote by $q(\tau)$ the value of the term τ in the ID labelling the vertex q (note that if τ is an array term then we must instantiate the appropriate values for the index terms). The input structure $\mathcal{A}$ is accepted by ρ if, and only if, the ID labelling the sink, s, of $\mathcal{G}(\rho^{\mathcal{A}})$ is such that $s(x_1) = max$, $s(x_2) = max$, ..., $s(x_k) = max$.

A *cut* in $\mathcal{G}(\rho^{\mathcal{A}})$ is a set U of vertices such that the source of $\mathcal{G}(\rho^{\mathcal{A}})$ is in U and the vertices of U form a connected subgraph (in the above sense). A vertex q of $\mathcal{G}(\rho^{\mathcal{A}}) \setminus U$ is a *successor vertex* of the cut U if there exists an edge from a vertex of U to q. A *leaf* of U is a vertex of U from which there is no edge to another vertex of U.

4.4 Some Limitations of our Program Schemes

We begin by proving some limitations on the actual values held by variables and array elements throughout a computation of a program scheme of RFDPS on some input structure.

Lemma 4.9 Let $\rho \in$ RFDPS involve the variables $x_1, x_2, \ldots, x_k$ (and no others) and be over the signature σ whose constant symbols are $C_1, C_2, \ldots, C_c$, where $c \geq 0$. Let $\mathcal{A}$ be some σ-structure and let q be some vertex of $\mathcal{G}(\rho^{\mathcal{A}})$ for which the associated instruction I is in the scope of forall-do-od blocks in ρ whose control variables are (without loss of generality) $x_1, x_2, \ldots, x_m$, for some $m \geq 0$.

(*i*) If I is not a forall-do instruction then for every $j \in \{m+1, m+2, \ldots, k\}$,

$$q(x_j) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q(x_1), q(x_2), \ldots, q(x_m)\};$$

and if I is a forall-do instruction, with control variable x_m, say, then for every $j \in \{m, m+1, \ldots, k\}$,

$$q(x_j) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q(x_1), q(x_2), \ldots, q(x_{m-1})\}.$$

(*ii*) Let A be any array symbol, of arity a, say, and let $(u_1, u_2, \ldots, u_a) \in |\mathcal{A}|^a$. Then

$$q(A[u_1, u_2, \ldots, u_a]) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{u_1, u_2, \ldots, u_a\}.$$

Proof We shall show that if the two conditions hold for all vertices in a cut of $\mathcal{G}(\rho^{\mathcal{A}})$ then they hold for any successor vertex of this cut. As the statement trivially holds for the source of $\mathcal{G}(\rho^{\mathcal{A}})$, the result will follow by induction. There are a number of cases, depending upon the type of the instruction associated with a leaf or leaves of our cut.

Suppose that the instruction associated with a leaf q of our cut is a repeat-do instruction, a repeat-od instruction, an if instruction, a fi instruction or a forall-do instruction. Then (i) and (ii) trivially hold for any successor vertex of q in $\mathcal{G}(\rho^A)$. (Let us remark that if the instruction associated with a successor vertex of q is a forall-do instruction then we have another control variable to contend with. However, note that this control variable was not a control variable at q and so (i) still holds at a successor vertex of q . This remark applies throughout.)

Suppose that the instruction I associated with a leaf q of our cut is an assignment instruction and let the successor vertex of q in $\mathcal{G}(\rho^A)$ be q'.

- If I is of the form $x_i := \tau$, for some variable or constant symbol τ, then (i) and (ii) can easily be seen to hold for q' (note that $i \not\in \{1, 2, \ldots, m\}$).

- If I is of the form $x_i := B[\tau_1', \tau_2', \ldots, \tau_b']$, for some array symbol B, of arity b, say, then $q(B[\tau_1', \tau_2', \ldots, \tau_b']) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q(\tau_1'), q(\tau_2'), \ldots, q(\tau_b')\}$ and $q(\tau_j') \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q(x_1), q(x_2), \ldots, q(x_m)\}$, for each $j \in \{1, 2, \ldots, b\}$. So, $q'(x_i) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q(x_1), q(x_2), \ldots, q(x_m)\}$ and $q(x_j) = q'(x_j)$, for each $j \in \{1, 2, \ldots, m\}$ (again, note that $i \not\in \{1, 2, \ldots, m\}$). Hence, (i) and (ii) hold for q'.

- If I is of the form $A[\tau_1, \tau_2, \ldots, \tau_a] := \tau$, where A is an array symbol, of arity a, say, and where τ is a variable or a constant symbol, then $q'(A[\tau_1, \tau_2, \ldots, \tau_a]) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q(x_1), q(x_2), \ldots, q(x_m)\} = \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q'(x_1), q'(x_2), \ldots, q'(x_m)\}$. However, because I is in the scope of forall-do-od blocks with control variables $x_1, x_2, \ldots, x_m$, we have that $x_j \in \{\tau_1, \tau_2, \ldots, \tau_a\}$, for each $j \in \{1, 2, \ldots, m\}$. Hence, $q'(A[\tau_1, \tau_2, \ldots, \tau_a]) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q'(\tau_1), q'(\tau_2), \ldots, q'(\tau_a)\}$, and (i) and (ii) hold for q'.

- If I is of the form $A[\tau_1, \tau_2, \ldots, \tau_a] := B[\tau_1', \tau_2', \ldots, \tau_b']$, where A and B are array symbols of arities a and b, then $q'(A[\tau_1, \tau_2, \ldots, \tau_a]) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup$

$\{q(\tau'_1), q(\tau'_2), \ldots, q(\tau'_b)\}$. Also, we know that $q(\tau'_j) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q(x_1), q(x_2), \ldots, q(x_m)\}$, for each $j \in \{1, 2, \ldots, b\}$. However, because I is in the scope of forall-do-od blocks with control variables $x_1, x_2, \ldots, x_m$, we have that $x_j \in \{\tau_1, \tau_2, \ldots, \tau_a\}$, for each $j \in \{1, 2, \ldots, m\}$. Hence, as $q(x_j) = q'(x_j)$, for each $j \in \{1, 2, \ldots, m\}$, we have that (*i*) and (*ii*) hold for q'.

Let α be a forall-do-od block, with control variable x_m, say, and let $\mathrm{Im}^{\mathcal{A}}(\alpha)$ be an image of α in $\mathcal{G}(\rho^{\mathcal{A}})$ so that every vertex of $\mathrm{Im}^{\mathcal{A}}(\alpha)$ apart from the sink is in our cut. Denote the sink of $\mathrm{Im}^{\mathcal{A}}(\alpha)$ by q' and the source by p. Consider $q'(A[u_1, u_2, \ldots, u_a])$, where A is some array symbol of arity a, say, and $u_1, u_2, \ldots, u_a \in |\mathcal{A}|$. As (*i*) and (*ii*) hold for p and every parent of q' in $\mathrm{Im}^{\mathcal{A}}(\alpha)$, we have that (*ii*) also holds for q'. It is also the case that $q'(x_m) \in \{0, max\}$ and $q'(x_j) = p(x_j) \in \{0, max, C_1, C_2, \ldots, C_c\} \cup \{p(x_1), p(x_2), \ldots, p(x_{m-1})\} = \{0, max, C_1, C_2, \ldots, C_c\} \cup \{q'(x_1), q'(x_2), \ldots, q'(x_{m-1})\}$, for each $j \in \{m+1, m+2, \ldots, k\}$. Hence, (*i*) holds for vertex q'.

The result follows by induction. □

Before proceeding to our main result, we make the following small diversion.

Recall from Chapter 2 that we denote d-variable infinitary logic by $\mathcal{L}^d_{\infty\omega}$. Let $\mathcal{A}$ and $\mathcal{B}$ be σ-structures, for some σ, and let $\mathbf{u} \in |\mathcal{A}|^e$ and $\mathbf{v} \in |\mathcal{B}|^e$, where $0 \leq e \leq d$. If

$$\begin{aligned} &\{\varphi : \varphi \in \mathcal{L}^d_{\infty\omega} \text{ has free variables } x_1, x_2, \ldots, x_e, \mathcal{A} \models \varphi(u_1, u_2, \ldots, u_e)\} \\ = \quad &\{\varphi : \varphi \in \mathcal{L}^d_{\infty\omega} \text{ has free variables } x_1, x_2, \ldots, x_e, \mathcal{B} \models \varphi(v_1, v_2, \ldots, v_e)\} \end{aligned}$$

then we write

$$(\mathcal{A}, u_1, u_2, \ldots, u_e) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, v_1, v_2, \ldots, v_e).$$

There is a well-known game-theoretic characterisation of definability in $\mathcal{L}^d_{\infty\omega}$, based on the Ehrenfeucht-Fraïssé game described in Chapter 3.

The Barwise-Immerman-Poizat game (see [30]) for a problem Ω proceeds in the same manner as the traditional game, described in Definition 3.2, but with one change.

Instead of using a fresh pebble on every move, Spoiler is instead free to move an already-placed pebble from one domain element of a structure to another domain element of the same structure. As before, Duplicator must respond by moving the other pebble of the pair. Potentially therefore, the game may have an infinite duration: Spoiler and Duplicator may shuffle pebbles around the structures without end.

In fact, if the game ends after a finite number of moves then Duplicator must lose. More precisely, if ever Duplicator's turn ends with the mapping defined by the pebbled elements and constant symbols of the two structures failing to induce a partial isomorphism, then Spoiler wins. If each of Duplicator's turns ends with the mapping inducing a partial isomorphism, then Duplicator wins. In this case however, the game must necessarily be of infinite length.

We say that Duplicator has a winning strategy for the Barwise-Immerman-Poizat game for Ω if she has a strategy by which she can win every play of the game.

Theorem 4.10 Duplicator has a winning strategy in the Barwise-Immerman-Poizat game for Ω if, and only if, Ω is not definable in $\mathcal{L}^{\omega}_{\infty\omega}$.

This game can be generalised in the same way as the other variants described in Chapter 3. In particular, definability in $\mathcal{L}^{d}_{\infty\omega}$ can be characterised by a version of the game in which Spoiler is forced to begin the game by selecting d pairs of pebbles.

Furthermore, if a game is already underway, with d pairs of pebbles having been selected by Spoiler; with structures $\mathcal{A}$ and $\mathcal{B}$ having been selected by Duplicator; and with $0 \leq e \leq d$ pairs of pebbles having already been placed (on domain elements $u_1, u_2, \ldots, u_e$ of $\mathcal{A}$ and $v_1, v_2, \ldots, v_e$ of $\mathcal{B}$ respectively), and if Duplicator has a winning strategy for the remainder of the game, then we say that she *wins the d-pebble game on* $(\mathcal{A}, \mathbf{u})$ *and* $(\mathcal{B}, \mathbf{v})$. Theorem 4.10 can then be reformulated as follows.

Theorem 4.11 Let $\mathcal{A}$ and $\mathcal{B}$ be two structures over the same signature and let $\mathbf{u} \in |\mathcal{A}|^e$ and $\mathbf{v} \in |\mathcal{B}|^e$, where $0 \leq e \leq d$. The following are equivalent.

(*i*) $(\mathcal{A}, u_1, u_2, \ldots, u_e) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, v_1, v_2, \ldots, v_e)$.

(*ii*) Duplicator wins the d-pebble game on $(\mathcal{A}, \mathbf{u})$ and $(\mathcal{B}, \mathbf{v})$.

We do not actually use the above characterisation result, only the notion of Duplicator winning the d-pebble game on $(\mathcal{A}, \mathbf{u})$ and $(\mathcal{B}, \mathbf{v})$. However, it is useful for the reader to know the logical significance of the Duplicator winning this game.

We now turn to our main result in this chapter (note that, by Theorem 4.11, we think of $\mathcal{A} \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}$ in game-theoretic terms).

Theorem 4.12 Let $\rho \in$ RFDPS be over the signature σ and have depth of nesting $d \geq 0$. Let $\mathcal{A}$ and $\mathcal{B}$ be $\sigma \cup \{0, max\}$-structures of equal size such that $0^{\mathcal{A}} \neq max^{\mathcal{A}}$, $0^{\mathcal{B}} \neq max^{\mathcal{B}}$ and $\mathcal{A} \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}$. Then

$$\mathcal{A} \models \rho \text{ if, and only if, } \mathcal{B} \models \rho.$$

Proof Suppose that the variables involved in ρ are $x_1, x_2, \ldots, x_k$ and that the constant symbols of σ are $C_1, C_2, \ldots, C_c$, where $c \geq 0$. Let α be any block of instructions appearing in ρ. Suppose that α is in the scope of forall-do-od blocks $\beta_1, \beta_2, \ldots, \beta_m$ with control variables $x_1, x_2, \ldots, x_m$, respectively, for some $m \geq 0$; and suppose further that block β_{i+1} is in the scope of block β_i, for each $i \in \{1, 2, \ldots, m-1\}$.

Let $\text{Im}^{\mathcal{A}}(\alpha)$ and $\text{Im}^{\mathcal{B}}(\alpha)$ be images of α in $\mathcal{G}(\rho^{\mathcal{A}})$ and $\mathcal{G}(\rho^{\mathcal{B}})$, respectively, and let $s^{\mathcal{A}}$ and $t^{\mathcal{A}}$ be the source and the sink of $\text{Im}^{\mathcal{A}}(\alpha)$, and $s^{\mathcal{B}}$ and $t^{\mathcal{B}}$ the source and sink of $\text{Im}^{\mathcal{B}}(\alpha)$. Write cons$(\mathcal{A})$ for $\{0^{\mathcal{A}}, max^{\mathcal{A}}, C_1^{\mathcal{A}}, C_2^{\mathcal{A}}, \ldots, C_c^{\mathcal{A}}\}$, with cons$(\mathcal{B})$ defined similarly.

We write $\mathcal{A}_s \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_s$ to denote that the following two conditions hold:

(*i*) $(\mathcal{A}, s^{\mathcal{A}}(x_1), s^{\mathcal{A}}(x_2), \ldots, s^{\mathcal{A}}(x_m)) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}(x_1), s^{\mathcal{B}}(x_2), \ldots, s^{\mathcal{B}}(x_m))$; and

(*ii*) for each $i \in \{m+1, m+2, \ldots, k\}$, one of the following is true:

- $s^{\mathcal{A}}(x_i) = s^{\mathcal{A}}(x_j)$ and $s^{\mathcal{B}}(x_i) = s^{\mathcal{B}}(x_j)$, for some $j \in \{1, 2, \ldots, m\}$, or

– $s^{\mathcal{A}}(x_i) = C^{\mathcal{A}}$ and $s^{\mathcal{B}}(x_i) = C^{\mathcal{B}}$, for some $C \in \{0, max, C_1, C_2, \ldots, C_c\}$.

Consider the forall-do-od blocks $\beta_1, \beta_2, \ldots, \beta_m$. Note that if x_i is active in the array symbol A in β_i then x_j is active in (the same array symbol) A in β_j, for each $j \in \{1, 2, \ldots, i-1\}$; and if x_i is inactive in β_i then x_j is inactive in β_j, for each $j \in \{i+1, i+2, \ldots, m\}$. In particular: either there exists a unique array symbol A so that x_i is active in A in β_i, for at least one $i \in \{1, 2, \ldots, m\}$, when we say that A is the array symbol associated with α; or x_i is not active in β_i, for each $i \in \{1, 2, \ldots, m\}$, when we say that α has no associated array symbol. (The intuition behind the notation is that the associated array symbol, if present, represents the unique array whose entries change during the execution of the β_i. If no entries change, then we say that there is no associated array symbol, even though the β_i all have their own control array symbols.)

Whenever α has an associated array symbol, which we always take to be the array symbol A, of arity a, say, and $f \in \{1, 2, \ldots, m\}$ is the maximal such element for which x_f is active in A in β_f then w.l.o.g. we assume that x_i is active in A in β_i at index i, for every $i \in \{1, 2, \ldots, f\}$ (throughout, f always refers to this particular index if α has an associated array symbol).

Suppose that $\mathcal{A}_s \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_s$. Let $u_{m+1}, u_{m+2}, \ldots, u_d \in |\mathcal{A}|$ and $v_{m+1}, v_{m+2}, \ldots, v_d \in |\mathcal{B}|$ be such that

$$(\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d).$$

Define π_s to be the natural map from $\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\}$ to $\text{cons}(\mathcal{B}) \cup \{s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d\}$ (note that this map is well-defined and depends upon $u_{m+1}, u_{m+2}, \ldots, u_d$, but we have suppressed this fact in the notation). Suppose that

$$\pi_s(s^{\mathcal{A}}(B[\mathbf{w}])) = s^{\mathcal{B}}(B[\pi_s(\mathbf{w})])$$

(with π_s applied point-wise) whenever

- B is an array symbol, of arity b, say, and different from the associated array symbol of α, if there is one, and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^b$

or

- there is an associated array symbol A of α, $B = A$ and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$ with $w_i = s^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, f\}$

(note that, by Lemma 4.9, $\pi_s(s^{\mathcal{A}}(B[\mathbf{w}]))$ is always well-defined). If the above holds for every $u_{m+1}, u_{m+2}, \ldots, u_d \in |\mathcal{A}|$ and $v_{m+1}, v_{m+2}, \ldots, v_d \in |\mathcal{B}|$ for which

$$(\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d)$$

then we say that $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are *array-consistent* (note that the notion of array-consistency is symmetric).

We shall proceed by induction on the building process for the constituent blocks of ρ. The following will be our induction hypothesis: '*For all images $Im^{\mathcal{A}}(\alpha)$ and $Im^{\mathcal{B}}(\alpha)$, if $\mathcal{A}_s \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_s$ and $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are array-consistent then $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$ and $t^{\mathcal{A}}$ and $t^{\mathcal{B}}$ are array-consistent*'.

Base Case The block α is an assignment block and let $\text{Im}^{\mathcal{A}}(\alpha)$ and $\text{Im}^{\mathcal{B}}(\alpha)$ be such that $\mathcal{A}_s \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_s$ and $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are array-consistent.

Suppose that α consists of an instruction of the form $x_i := \tau$, for some term τ. If τ is a variable or a constant symbol then trivially $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$. Hence, suppose that τ is an array term of the form $B[\tau_1, \tau_2, \ldots, \tau_b]$. Let π_s be the natural map from $\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), s^{\mathcal{A}}(x_2), \ldots, s^{\mathcal{A}}(x_m)\}$ to $\text{cons}(\mathcal{B}) \cup \{s^{\mathcal{B}}(x_1), s^{\mathcal{B}}(x_2), \ldots, s^{\mathcal{B}}(x_m)\}$. As $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are array consistent, we have that

$$\begin{aligned}
&\pi_s(s^{\mathcal{A}}(B[s^{\mathcal{A}}(\tau_1), s^{\mathcal{A}}(\tau_2), \ldots, s^{\mathcal{A}}(\tau_b)])) \\
&= \quad s^{\mathcal{B}}(B[\pi_s(s^{\mathcal{A}}(\tau_1)), \pi_s(s^{\mathcal{A}}(\tau_2)), \ldots, \pi_s(s^{\mathcal{A}}(\tau_b))]).
\end{aligned}$$

That is, $\pi_s(t^{\mathcal{A}}(x_i)) = \pi_s(s^{\mathcal{A}}(\tau)) = s^{\mathcal{B}}(\tau) = t^{\mathcal{B}}(x_i)$, and so $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$.

Suppose that $u_{m+1}, u_{m+2}, \ldots, u_d \in |\mathcal{A}|$ and $v_{m+1}, v_{m+2}, \ldots, v_d \in |\mathcal{B}|$ are such that:

$$(\mathcal{A}, t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, t^{\mathcal{B}}(x_1), \ldots, t^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d),$$

and let π_t be the natural map from $\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\}$ to $\text{cons}(\mathcal{B}) \cup \{t^{\mathcal{B}}(x_1), \ldots, t^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d\}$. Rewriting, we obtain that

$$(\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d),$$

and let π_s be the natural map from $\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\}$ to $\text{cons}(\mathcal{B}) \cup \{s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d\}$. Note that π_s and π_t are identical. Also, no value of any array element changes in the transitions from $s^{\mathcal{A}}$ to $t^{\mathcal{A}}$ and from $s^{\mathcal{B}}$ to $t^{\mathcal{B}}$. By assumption,

$$\pi_s(s^{\mathcal{A}}(B[\mathbf{w}])) = s^{\mathcal{B}}(B[\pi_s(\mathbf{w})])$$

whenever

- B is an array symbol, of arity b, say, and different from the associated array symbol of α, if there is one, and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^b$

or

- there is an associated array symbol A of α, $B = A$ and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$ with $w_i = s^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, f\}$.

Thus

$$\pi_t(t^{\mathcal{A}}(B[\mathbf{w}])) = t^{\mathcal{B}}(B[\pi_t(\mathbf{w})])$$

whenever

- B is an array symbol, of arity b, say, and different from the associated array symbol of α, if there is one, and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$

or

- there is an associated array symbol A of α, $B = A$ and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$ with $w_i = t^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, f\}$.

Hence, $t^{\mathcal{A}}$ and $t^{\mathcal{B}}$ are array-consistent.

Suppose that α consists of an instruction of the form $A[\tau_1, \tau_2, \ldots, \tau_a] := \tau$, for some terms $\tau_1, \tau_2, \ldots, \tau_a, \tau$, and where A is the associated array symbol of α, if α is in the scope of a forall-do-od block with an associated array symbol. As no variable value has changed in the transitions from $s^{\mathcal{A}}$ to $t^{\mathcal{A}}$ and from $s^{\mathcal{B}}$ to $t^{\mathcal{B}}$, we have that $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$. Let $u_{m+1}, u_{m+2}, \ldots, u_d \in |\mathcal{A}|$ and $v_{m+1}, v_{m+2}, \ldots, v_d \in |\mathcal{B}|$ be such that:

$$(\mathcal{A}, t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, t^{\mathcal{B}}(x_1), \ldots, t^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d),$$

and let π_t be the natural map from $\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\}$ to $\text{cons}(\mathcal{B}) \cup \{t^{\mathcal{B}}(x_1), \ldots, t^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d\}$. Rewriting, we obtain that

$$(\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d),$$

and let π_s be the natural map from $\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\}$ to $\text{cons}(\mathcal{B}) \cup \{s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d\}$. By assumption,

$$\pi_s(s^{\mathcal{A}}(B[\mathbf{w}])) = s^{\mathcal{B}}(B[\pi_s(\mathbf{w})])$$

whenever

- B is an array symbol, of arity b, say, and different from the associated array symbol of α, if there is one, and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^b$

or

- there is an associated array symbol A of α, $B = A$ and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$ with $w_i = s^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, f\}$.

Hence,

$$\pi_t(t^{\mathcal{A}}(B[\mathbf{w}])) = t^{\mathcal{B}}(B[\pi_t(\mathbf{w})])$$

whenever

- B is an array symbol of arity b, say, and different from the associated array symbol of α, if there is one, and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^b$ (as the value of neither $B[\mathbf{w}]$ nor $B[\pi_t(\mathbf{w})]$ changes between $s^{\mathcal{A}}$ and $t^{\mathcal{A}}$ and between $s^{\mathcal{B}}$ and $t^{\mathcal{B}}$, respectively)

or

- there is an associated array symbol A of α, $B = A$ and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$ with $w_i = t^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, f\}$ (which holds as the value of $t^{\mathcal{A}}(A[t^{\mathcal{A}}(\tau_1), t^{\mathcal{A}}(\tau_2), \ldots, t^{\mathcal{A}}(\tau_a)])$ is $s^{\mathcal{A}}(\tau)$ and the value of $t^{\mathcal{B}}(A[t^{\mathcal{B}}(\tau_1), t^{\mathcal{B}}(\tau_2), \ldots, t^{\mathcal{B}}(\tau_a)])$ is $s^{\mathcal{B}}(\tau)$).

Hence, $t^{\mathcal{A}}$ and $t^{\mathcal{B}}$ are array-consistent.

Thus, whatever the form of the assignment block α, we have that $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$ and $t^{\mathcal{A}}$ and $t^{\mathcal{B}}$ are array-consistent.

Case (i) Let α be an active forall-do-od block of the form

```
FORALL $x_{m+1}$ WITH $A^p$ DO          *forall-do instruction*
    $\alpha_1$                          *block of instructions*
    $\alpha_2$                          *block of instructions*
    $\vdots$
    $\alpha_l$                          *block of instructions*
OD                                      *forall-od instruction*
```

where our induction hypothesis holds for the blocks $\alpha_1, \alpha_2, \ldots, \alpha_l$. Let $\text{Im}^{\mathcal{A}}(\alpha)$ and $\text{Im}^{\mathcal{B}}(\alpha)$ be images of α such that $\mathcal{A}_s \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_s$ and $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are array-consistent. Note that A is the associated array symbol of α and that we may assume that x_i is active in A for β_i at index i, for each $i \in \{1, 2, \ldots, m+1\}$.

Pick some $u_{m+1} \in |\mathcal{A}|$ and let $s^{\mathcal{A}}_{m+1}$ be the child of $s^{\mathcal{A}}$ in $\mathcal{G}(\rho^{\mathcal{A}})$ for which $s^{\mathcal{A}}_{m+1} = u_{m+1}$. Let $v_{m+1} \in |\mathcal{B}|$ be such that

$$(A, s^{\mathcal{A}}(x_1), s^{\mathcal{A}}(x_2), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}) \equiv^{\mathcal{L}^d_{\infty\omega}} (B, s^{\mathcal{B}}(x_1), s^{\mathcal{B}}(x_2), \ldots, s^{\mathcal{B}}(x_m), v_{m+1})$$

(at least one such v_{m+1} exists since $m < d$) and let $s^{\mathcal{B}}_{m+1}$ be the son of $s^{\mathcal{B}}$ in $\mathcal{G}(\rho^{\mathcal{B}})$ for which $s^{\mathcal{B}}_{m+1} = v_{m+1}$. Rewriting, we obtain that

$$\begin{aligned}&(A, s^{\mathcal{A}}_{m+1}(x_1), s^{\mathcal{A}}_{m+1}(x_2), \ldots, s^{\mathcal{A}}_{m+1}(x_{m+1}))\\ &\quad \equiv^{\mathcal{L}^d_{\infty\omega}} (B, s^{\mathcal{B}}_{m+1}(x_1), s^{\mathcal{B}}_{m+1}(x_2), \ldots, s^{\mathcal{B}}_{m+1}(x_{m+1}))\end{aligned}$$

and so $\mathcal{A}_{s_{m+1}} \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_{s_{m+1}}$.

Let $u_{m+2}, u_{m+3}, \ldots, u_d \in |\mathcal{A}|$ and $v_{m+2}, v_{m+3}, \ldots, v_d \in |\mathcal{B}|$ be such that

$$\begin{aligned}&(\mathcal{A}, s^{\mathcal{A}}_{m+1}(x_1), \ldots, s^{\mathcal{A}}_{m+1}(x_{m+1}), u_{m+2}, \ldots, u_d)\\ &\quad \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}_{m+1}(x_1), \ldots, s^{\mathcal{B}}_{m+1}(x_{m+1}), v_{m+2}, \ldots, v_d).\end{aligned}$$

Rewriting, we obtain that

$$(\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d),$$

and the corresponding maps $\pi_{s_{m+1}}$ and π_s are identical. By assumption,

$$\pi_s(s^{\mathcal{A}}(B[\mathbf{w}])) = s^{\mathcal{B}}(B[\pi_s(\mathbf{w})])$$

whenever

- B is an array symbol, of arity b, say, and different from A, and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^b$

or

- $B = A$ and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$ with $w_i = s^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, m\}$.

Note that the transitions from $s^{\mathcal{A}}$ to $s^{\mathcal{A}}_{m+1}$ and from $s^{\mathcal{B}}$ to $s^{\mathcal{B}}_{m+1}$ cause no array element to change value. Hence,

$$\pi_{s_{m+1}}(s^{\mathcal{A}}_{m+1}(B[\mathbf{w}])) = s^{\mathcal{B}}_{m+1}(B[\pi_{s_{m+1}}(\mathbf{w})])$$

whenever

- B is an array symbol, of arity b, say, and different from A, and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}_{m+1}(x_1), \ldots, s^{\mathcal{A}}_{m+1}(x_{m+1}), u_{m+2}, \ldots, u_d\})^b$

or

- $B = A$ and $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{s^{\mathcal{A}}_{m+1}(x_1), \ldots, s^{\mathcal{A}}_{m+1}(x_{m+1}), u_{m+2}, \ldots, u_d\})^a$ with $w_i = s^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, m+1\}$.

Hence, $s^{\mathcal{A}}_{m+1}$ and $s^{\mathcal{B}}_{m+1}$ are array-consistent.

By the induction hypothesis applied to $\alpha_1, \alpha_2, \ldots, \alpha_l$, we have that $\mathcal{A}_{t_{m+1}} \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_{t_{m+1}}$ and $t^{\mathcal{A}}_{m+1}$ and $t^{\mathcal{B}}_{m+1}$ are array-consistent, where $t^{\mathcal{A}}_{m+1}$ (resp. $t^{\mathcal{B}}_{m+1}$) is the parent of $t^{\mathcal{A}}$ (resp. $t^{\mathcal{B}}$) for which $t^{\mathcal{A}}_{m+1}(x_{m+1}) = u_{m+1}$ (resp. $t^{\mathcal{B}}_{m+1}(x_{m+1}) = v_{m+1}$).

We have just proved that for every 'child process' in the image $\text{Im}^{\mathcal{A}}(\alpha)$, there is a 'similar' process in $\text{Im}^{\mathcal{B}}(\alpha)$. Now we must show that the converse is true. Pick some $v_{m+1} \in |\mathcal{B}|$ and let $s^{\mathcal{B}}_{m+1}$ be the child of $s^{\mathcal{B}}$ in $\mathcal{G}(\rho^{\mathcal{B}})$ for which $s^{\mathcal{B}}_{m+1}(x_{m+1}) = v_{m+1}$. Let $u_{m+1} \in |\mathcal{A}|$ be such that

$$(\mathcal{B}, s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}).$$

An identical argument to the above yields that $B_{t_{m+1}} \equiv^{\mathcal{L}^d_{\infty\omega}} A_{t_{m+1}}$ and $t^{\mathcal{B}}_{m+1}$ and $t^{\mathcal{A}}_{m+1}$ are array-consistent (where the notation is as above). Consequently, either

- $t^{\mathcal{A}}(x_{m+1}) = 0$ and $t^{\mathcal{B}}(x_{m+1}) = 0$

or

- $t^{\mathcal{A}}(x_{m+1}) = max$ and $t^{\mathcal{B}}(x_{m+1}) = max$;

and so $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$.

Let $u_{m+1}, u_{m+2}, \ldots, u_d \in |\mathcal{A}|$ and $v_{m+1}, v_{m+2}, \ldots, v_d \in |\mathcal{B}|$ be such that

$$(\mathcal{A}, t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, t^{\mathcal{B}}(x_1), \ldots, t^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d),$$

with π_t defined as usual.

Case $(i)(a)$ Let B be an array symbol, of arity b, say, and different from A, and let $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^b$. Rewriting, we obtain that

$$(\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, s^{\mathcal{B}}(x_1), \ldots, s^{\mathcal{B}}(x_m), v_{m+1}, \ldots, v_d)$$

and, with π_s defined as usual, π_s is identical to π_t. Consequently, by assumption,

$$\pi_s(s^{\mathcal{A}}(B[\mathbf{w}])) = s^{\mathcal{B}}(B[\pi_s(\mathbf{w})]).$$

Note that no transition from $s^{\mathcal{A}}$ to $t^{\mathcal{A}}$ and from $s^{\mathcal{B}}$ to $t^{\mathcal{B}}$ causes an array element of B to change value and so

$$\pi_t(t^{\mathcal{A}}(B[\mathbf{w}])) = t^{\mathcal{B}}(B[\pi_t(\mathbf{w})]).$$

Case $(i)(b)$ Let $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d\})^a$ with $w_i = t^{\mathcal{A}}(x_i)$, for each $i \in \{1, 2, \ldots, m\}$. Note that the value of $A[\mathbf{w}]$ at $t^{\mathcal{A}}$ (resp. $A[\pi_t(\mathbf{w})]$ at $t^{\mathcal{B}}$) is identical to the value of $A[\mathbf{w}]$ at $t^{\mathcal{A}}_{m+1}$ (resp. $A[\pi_t(\mathbf{w})]$ at $t^{\mathcal{B}}_{m+1}$) where $t^{\mathcal{A}}_{m+1}$ (resp. $t^{\mathcal{B}}_{m+1}$) is the parent of $t^{\mathcal{A}}$ (resp. $t^{\mathcal{B}}$) for which $t^{\mathcal{A}}(x_{m+1}) = w_{m+1}$ (resp. $t^{\mathcal{B}}(x_{m+1}) = \pi_t(w_{m+1})$).

Suppose that $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), w_{m+1}, u_{m+1}, \ldots, \hat{u}_i, \ldots, u_d\})^a$, for some $i \in \{m+1, m+2, \ldots, d\}$, where $\hat{u}_i$ means with the value of u_i removed. Then $\mathbf{w} \in (\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}_{m+1}(x_1), \ldots, t^{\mathcal{A}}_{m+1}(x_{m+1}), u_{m+1}, \ldots, \hat{u}_i, \ldots, u_d\})^a$ and also

$$\begin{aligned}(\mathcal{A}, t^{\mathcal{A}}_{m+1}(x_1), \ldots, t^{\mathcal{A}}_{m+1}(x_{m+1}), u_{m+1}, \ldots, \hat{u}_i, \ldots, u_d)&\\ \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, t^{\mathcal{B}}_{m+1}(x_1), \ldots, t^{\mathcal{B}}_{m+1}(x_{m+1}), v_{m+1}, \ldots, \hat{v}_i, \ldots, v_d),&\end{aligned}$$

with $\pi_{t_{m+1}}$ defined as usual. Note that either $\pi_{t_{m+1}}$ is a restriction of π_t or identical to π_t. By assumption,

$$\pi_{t_{m+1}}(t^{\mathcal{A}}_{m+1}(A[\mathbf{w}])) = t^{\mathcal{B}}_{m+1}(A[\pi_{t_{m+1}}(\mathbf{w})])$$

and so in any case

$$\pi_t(t^{\mathcal{A}}(A[\mathbf{w}])) = t^{\mathcal{B}}(A[\pi_t(\mathbf{w})]).$$

Consequently, we are left with the situation where

- $u_i \neq u_j$, for each $i, j \in \{m+1, m+2, \ldots, d\}, i \neq j$;
- $u_i \notin \text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), t^{\mathcal{A}}(x_2), \ldots, t^{\mathcal{A}}(x_m), w_{m+1}\}$, for each $i \in \{m+1, m+2, \ldots, d\}$; and
- u_i is a component of $\mathbf{w}$, for each $i \in \{m+1, m+2, \ldots, d\}$.

Suppose that the value of the array element $A[\mathbf{w}]$ is changed between $s^{\mathcal{A}}$ and $t^{\mathcal{A}}$. As $\mathbf{w}$ contains $d - m$ components that are pairwise distinct and different from every value of $\text{cons}(\mathcal{A}) \cup \{t^{\mathcal{A}}(x_1), t^{\mathcal{A}}(x_2), \ldots, t^{\mathcal{A}}(x_m), w_{m+1}\}$, Lemma 4.9 implies that any change to the array element $A[\mathbf{w}]$ must be done via an instruction in the scope of $m + 1 + (d - m) = d + 1$ forall-do-od blocks. This yields a contradiction as ρ has depth of nesting d. Similarly, the value of the array element $A[\pi_t(\mathbf{w})]$ does not change between $s^{\mathcal{B}}$ and $t^{\mathcal{B}}$. Thus we have that

$$(\mathcal{A}, s^{\mathcal{A}}(x_1), \ldots, s^{\mathcal{A}}(x_m), u_{m+1}, \ldots, u_d) \equiv^{\mathcal{L}^d_{\infty\omega}} (\mathcal{B}, t^{\mathcal{A}}(x_1), \ldots, t^{\mathcal{A}}(x_m), v_{m+1}, \ldots, v_d),$$

and with π_s defined as usual, π_s is identical to π_t. By assumption,

$$\pi_s(s^{\mathcal{A}}(A[\mathbf{w}])) = s^{\mathcal{B}}(A[\pi_s(\mathbf{w})])$$

and so

$$\pi_t(t^{\mathcal{A}}(A[\mathbf{w}])) = t^{\mathcal{B}}(A[\pi_t(\mathbf{w})]).$$

Thus, $t^{\mathcal{A}}$ and $t^{\mathcal{B}}$ are array-consistent.

In conclusion, we have that $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$ and $t^{\mathcal{A}}$ and $t^{\mathcal{B}}$ are array-consistent.

Case *(ii)* Let α be an inactive forall-do-od block of the form

```
FORALL $x_{m+1}$ WITH $A^p$ DO          forall-do instruction
    $\alpha_1$                           block of instructions
    $\alpha_2$                           block of instructions
    ⋮
    $\alpha_l$                           block of instructions
```

```
OD                                        forall-od instruction
```

where our induction hypothesis holds for the blocks $\alpha_1, \alpha_2, \ldots, \alpha_l$. Let $\mathrm{Im}^{\mathcal{A}}(\alpha)$ and $\mathrm{Im}^{\mathcal{B}}(\alpha)$ be images of α such that $\mathcal{A}_s \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_s$ and $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are array-consistent. The proof of Case (i) can be applied when some β_i, for $i \in \{1, 2, \ldots, m\}$ has an active control variable and also when no such β_i has an active control variable.

Case (iii) Let α be a repeat-do-od block or an if-then-fi block. In both cases, immediate applications of the induction hypothesis yield the required result.

Consequently, we have that the induction hypothesis holds for every constituent block of ρ. Let $s^{\mathcal{A}}$ and $t^{\mathcal{A}}$ be the source and the sink of $\mathcal{G}(\rho^{\mathcal{A}})$, with $s^{\mathcal{A}}$ and $t^{\mathcal{A}}$ the source and the sink of $\mathcal{G}(\rho^{\mathcal{B}})$. Clearly, $\mathcal{A}_s \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_s$ and $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are array-consistent. Hence, $\mathcal{A}_t \equiv^{\mathcal{L}^d_{\infty\omega}} \mathcal{B}_t$ and $t^{\mathcal{A}}$ and $t^{\mathcal{B}}$ are array-consistent, and our result follows. □

Let RFDPS$_d$ be those program schemes of RFDPS with depth of nesting at most d (and also the class of problems definable by such program schemes). Note that RFDPS$_d$ is a logic (in Gurevich's sense).

Corollary 4.13

$$\mathrm{RFDPS}_0 \subset \mathrm{RFDPS}_1 \subset \ldots \subset \mathrm{RFDPS}_d \subset \mathrm{RFDPS}_{d+1} \subset \ldots$$

Proof Let $\sigma = \langle E, C, D \rangle$, where E is a binary relation symbol and C and D are constant symbols. Hence, a σ-structure can be thought of as a directed graph with two distinguished vertices. Fix $d \geq 1$. Define the σ-structure $\mathcal{A}_d$ as follows. The vertices $C^{\mathcal{A}_d}$ and $D^{\mathcal{A}_d}$ are distinct vertices of in-degree 0 and out-degree $d+3$ so that they have no neighbour in common (this constitutes all vertices and edges of $\mathcal{A}_d$). Define the σ-structure $\mathcal{B}_d$ as follows. The vertices $C^{\mathcal{B}_d}$ and $D^{\mathcal{B}_d}$ are distinct vertices of in-degree 0 and out-degree $d+2$ and $d+4$, respectively, so that they have no neighbour in common (this constitutes all vertices and edges of $\mathcal{B}_d$).

Consider the following program scheme ρ of RFDPS$_{d+1}$.

```
INPUT($x_1$, $x_2$, ..., $x_{d+1}$, $y$)
    FORALL $x_1$ DO
        FORALL $x_2$ DO
            ⋱
                FORALL $x_{d+1}$ DO
                    $y := max$
                    IF $\bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_i (x_i \neq 0 \wedge x_i \neq max) \wedge \bigwedge_i E(C, x_i)$
                    $\wedge E(C, 0) \wedge E(C, max)$ THEN
                        $y := 0$
                    FI
                OD
            ⋰
        OD
    OD
    IF $x_1 = 0$ THEN
        (x,y) = (max, max)
    ELSE
        (x,y) = (0, 0)
    FI
OUTPUT($x_1$, $x_2$, ..., $x_{d+1}$, $y$)
```

Clearly, $\mathcal{A}$ is accepted by ρ but $\mathcal{B}$ is not. By Theorem 4.12, the problem defined by ρ is not in RFDPS$_d$. The result follows (as clearly RFDPS$_0 \subset$ RFDPS$_1$). □

Note that the proof of Corollary 4.13 can be used to show that the problem consisting of all those digraphs for which every vertex has even out-degree is not in RFDPS.

Corollary 4.14 There are problems in **PTIME** which are not in RFDPS. □

Let φ be a formula of inductive fixed-point logic. The *quantifier-rank* q.r.(φ) of φ is defined inductively as follows.

- If φ is first-order quantifier-free then $\text{q.r.}(\varphi) = 0$.
- If φ is of the form $\neg\psi$ then $\text{q.r.}(\varphi) = \text{q.r.}(\psi)$.
- If φ is of the form $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$ then $\text{q.r.}(\varphi) = \max\{\text{q.r.}(\psi_1), \text{q.r.}(\psi_2)\}$.
- If φ is of the form $\exists\psi$ or $\forall\psi$ then $\text{q.r.}(\varphi) = 1 + \text{q.r.}(\psi)$.
- If φ is of the form $\text{IFP}[\lambda\mathbf{x}, R, \psi(\mathbf{x}, \mathbf{y}, R)](\mathbf{z})$ then $\text{q.r.}(\varphi) = |\mathbf{x}| + \text{q.r.}(\psi)$.

Let IFP_d be those formulae of inductive fixed-point logic with quantifier rank at most d (and also the class of problems definable by such sentences). The proof of Corollary 4.13 suffices to prove the following.

Corollary 4.15

$$\text{IFP}_0 \subset \text{IFP}_1 \subset \ldots \subset \text{IFP}_d \subset \text{IFP}_{d+1} \subset \ldots$$

The above corollary has not been studied before but, as pointed out by Martin Grohe in a personal communication, it follows quite easily from known results. The fragment IFP_d is contained in $L^d_{\infty\omega}$. This implies that the problem consisting of all those structures over the empty signature having at least $d + 1$ elements is not expressible in IFP_d; but it clearly is in IFP_{d+1} (actually in FO_{d+1}). We remark that our proof of Corollary 4.15 relies on no existing results from finite model theory (and not even on an understanding and appreciation of bounded-variable infinitary logic).

4.5 Conclusion

Whilst our concerns in this chapter have been the development of the class of program schemes RFDPS and an investigation of its refined structure, we feel that RFDPS will make a good stepping-off point in the quest for a logic for **PTIME**, as we now explain. Throughout any computation by a program scheme of RFDPS, we construct arrays of values. It will be relatively straightforward to incorporate Lindström quantifiers (see

[30]) into the program schemes of RFDPS by extending if instructions so that the test can be an application of some Lindström quantifier to some arrays, the values of whose elements are either 0 or *max* (so that the arrays model relations as in the proof of Theorem 4.8). It will also be entirely natural to include variables of a different type. For instance, one might allow an additional universe $\{0, 1, \ldots, n-1\}$, when the input to some program scheme is a structure of size n, with some appropriate numeric relations and a mechanism for 'tying' the two universes together; for example, an instruction $x := \sharp\varphi(y)$, where x has numeric type and φ is first-order, whose semantics are such that the number of values of y for which $\varphi(y)$ holds is assigned to the variable x. We shall pursue such extensions in future work.

A natural question to consider is how the class of problems accepted by the program schemes of RFDPS (and any extensions that we might develop, as explained in the preceding paragraph) compares with those accepted by the programs of $\tilde{\text{C}}$PTime and by other models more prevalent in database theory. We have not so far considered this question: however, let us remark that the problem consisting of those digraphs for which every vertex has even out-degree is not accepted by any program scheme of RFDPS yet can be accepted by a program of [66].

Whereas we feel that it will be fruitful to extend the program schemes of RFDPS, as hinted above, and investigate the expressive power of any resulting class of program schemes, there are still questions to be asked of RFDPS. For example, as was the case for the program schemes NPS, NPSS and NPSA of [6, 70, 81], can the class of problems accepted by the program schemes of RFDPS be realized as a *vectorised Lindström logic*? Does this class of problems have a complete member (via some suitable logical translation)? Is this class of problems nothing other than an extension of inductive fixed-point logic?

Chapter 5

Generating New Tractable Problems From Old

5.1 Introduction

In previous chapters, we have defined problems as sets of structures (over some signature) which satisfy some logical formula or program scheme. We now turn our attention in a slightly different direction, and consider problems which can be defined as a *constraint satisfaction problem* (CSP), in which the aim is to find an assignment to a given set of variables which satisfies one or more constraints. We refer the reader to Section 2.3 for a more detailed definition of the constraint satisfaction problem, as well as for other definitions which we shall use throughout this chapter.

The constraint satisfaction problem is known to be **NP**-hard in general [60]. Furthermore, many combinatorial problems within **NP** can be naturally expressed as instances of a CSP [51]. However, by imposing conditions on the forms of constraints allowed (see, amongst others, [18, 28, 37, 55, 56, 58, 65, 85]) it is possible to obtain restricted versions of the problem which are tractable (that is, provably solvable in polynomial time).

Given that so many tractable classes have been identified, it is pertinent to ask whether they may be combined, to yield new, larger constraint classes which are still tractable. This question has been posed before [15], but whereas the authors of that paper considered the effect of combining tractable classes over some fixed domain, in the present chapter we generally consider the effect of combining two tractable classes from *disjoint* domains.

We focus in particular on the "multiple relational union" of two sets of constraint relations, defined in Section 5.2. We show that whenever both sets of relations are tractable, then their multiple relation union is a tractable set also. In addition, we show that its tractability cannot in general be deduced from previously known results about tractability. Using the results of [51] we then show that the multiple relational union is itself just one small subset of a much larger set of tractable relations, whose proof of tractability is much less obvious than that of the multiple relational union itself.

The results presented in this chapter are joint work with Peter Jeavons, at the University of Oxford, and David Cohen, at Royal Holloway, University of London.

5.2 Combining Sets of Relations

5.2.1 Multiple Relational Unions

Definition 5.1 Let Γ_1 and Γ_2 be sets of relations over the non-empty sets D_1 and D_2 respectively. Then the *multiple relational union* of Γ_1 and Γ_2, denoted $\Gamma_1 \dot{\bowtie} \Gamma_2$, is defined to be the following set of relations over $D_1 \cup D_2$:

$$\Gamma_1 \dot{\bowtie} \Gamma_2 := \{R_1 \cup R_2 : R_1 \in \Gamma_1, R_2 \in \Gamma_2 \wedge \mathit{arity}(R_1) = \mathit{arity}(R_2)\}$$

Note that if $|\Gamma_1| = |\Gamma_2| = 1$, and the two relations have the same arity, then $\Gamma_1 \dot{\bowtie} \Gamma_2$ simply reduces to being the union of the two relations. In this case we shall sometimes write $R_1 \dot{\bowtie} R_2$ rather than $R_1 \cup R_2$. This is for the sake of clarity. Also, note that it is quite possible to have $\Gamma_1 \dot{\bowtie} \Gamma_2 = \emptyset$ even though $\Gamma_1 \neq \emptyset$ and $\Gamma_2 \neq \emptyset$: it may be the case that no relation in Γ_1 has the same arity as any relation in Γ_2.

Operations similar to the multiple relational union have been considered before in the literature. The definition of the disjunction operation $\overset{\times}{\vee}$ of [15] for example, is such that whenever each Γ_i contains the empty relation of every arity, then

$$\Gamma_1 \overset{\times}{\vee} \Gamma_2 = \Gamma_1 \dot{\bowtie} \Gamma_2$$

One significant difference between the $\overset{\times}{\vee}$ and the $\dot{\bowtie}$ operators however, is that the former is only defined when $D_1 = D_2$: that is, when the relations of Γ_1 and Γ_2 are defined over the same set. By contrast, in the present chapter we shall generally consider the case when D_1 and D_2 are *disjoint* sets. In particular, we shall show how this allows us to combine well-studied tractable sets of relations in order to generate previously unknown tractable sets. The following theorem is our main tool in this regard.

Theorem 5.2 Let Γ_1 and Γ_2 be two tractable sets of relations over the disjoint domains D_1 and D_2 respectively. Then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is also a tractable set of relations.

Proof Let $\mathcal{P}$ be a problem instance of $\mathbf{C}_{\Gamma_1 \dot{\bowtie} \Gamma_2}$. The following algorithm decides whether $\mathcal{P}$ is a yes-instance of the problem. The algorithm assumes that the hypergraph of constraint scopes is connected; if it is not then we may simply apply the algorithm over each connected component in turn.

First, determine whether or not there is an assignment to the variables of $\mathcal{P}$ in which every variable takes a value from D_1. This test is tractable since every relation $R \in \Gamma_1 \dot{\bowtie} \Gamma_2$ can be decomposed into the union of two relations R_1 and R_2 over the domains D_1 and D_2 respectively, and the R_2 part simply "thrown away". The tractability of the test then follows from the tractability of Γ_1. If there is such an assignment, then return True.

If there is no such assignment, then determine whether or not there is an assignment to the variables of V in which every variable takes a value from D_2. If there is, then return True. Otherwise, return False.

It is easy to see that this algorithm runs in polynomial time, and that if it returns True then $\mathcal{P}$ is a yes-instance of $\mathbf{C}_{\Gamma_1 \dot{\bowtie} \Gamma_2}$. To show the converse, we must observe that in *any* satisfying assignment to the variables of V, if some variable v is assigned a value from D_1 (say), then all the variables from V must be assigned values from D_1. This follows immediately from the definition of the $\dot{\bowtie}$ operator, and the connectivity of the constraint hypergraph. □

Technically, the above proof relies on the implicit assumption that it is tractable to decompose a relation $R \in \Gamma_1 \dot{\bowtie} \Gamma_2$ into its two component parts. The assumption is valid provided that we represent relations in such a way that every relation is the same size (to within a polynomial factor) as it would have been if it had been encoded in the natural way as a set of tuples.

Theorem 5.2 provides us with a method of combining two tractable sets of relations to produce another tractable set of relations but it is not, in and of itself, a particularly far-reaching result. The relations of $\Gamma_1 \dot{\bowtie} \Gamma_2$ are all easily decomposable into their constituent parts: there is no tuple in any relation which contains elements from both D_1 and D_2.

Example 5.3 The relation R defined by

$$\begin{aligned} R \ := \ & \{\langle 2,3,2,2,2\rangle, \\ & \langle 0,1,2,0,0\rangle, \\ & \langle 0,0,2,0,1\rangle, \\ & \langle 1,0,2,0,0\rangle, \\ & \langle 2,2,2,3,3\rangle, \\ & \langle 1,1,2,0,1\rangle\} \end{aligned}$$

cannot be shown to be tractable using previously known methods. That is, it is not closed under any operation which is known to guarantee tractability. Furthermore, there is no apparent hope of showing it to be tractable using Theorem 5.2, since the tuples of R cannot be split apart into tuples over disjoint domains. We shall return to this example later, when we have learned how to apply the above tractability result to more general sets of relations.

There is a partial converse to Theorem 5.2, which states that if either Γ_1 or Γ_2 is an **NP**-complete set of relations, then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is normally **NP**-complete also.

Proposition 5.4 Let Γ_1 be a relational clone over the domains D_1, and suppose that $Pol(\Gamma_1)$ contains no constant operations. Let Γ_2 be any set of relations over some disjoint domain D_2. Then whenever $\mathbf{C}_{\Gamma_2}$ is **NP**-complete, $\mathbf{C}_{\Gamma_1 \dot{\bowtie} \Gamma_2}$ is **NP**-complete also.

Proof Since $Pol(\Gamma_1)$ does not contain any constant operations, it follows that it contains no operations of arity 0, and hence that Γ_1 contains the empty relation of every arity. Thus $\Gamma_2 \subseteq \Gamma_1 \dot{\bowtie} \Gamma_2$. □

(Of course, the proof does not make full use of the assumption that Γ_1 is a relational clone; merely that it contains the empty relation of every arity. However, the most commonly encountered sets of relations which satisfy this property are the relational clones. Consequently, this is the form in which we state the proposition.)

As indicated above, Theorem 5.2 holds out the promise that we may be able to construct novel tractable sets of relations by taking the multiple relational union of two known tractable sets, Γ_1 and Γ_2, over disjoint domains. For this promise to be fulfilled however, it had better not be the case that $\Gamma_1 \dot{\bowtie} \Gamma_2$ is tractable for previously known reasons. The remainder of this section is devoted to developing a theoretical framework which will allow us to prove that in many cases (though not all) one does indeed end up with a genuinely new tractable class.

The tractability of a set of relations is intimately bound up with its polymorphisms (see the end of Section 2.4, as well as [51]). Consequently, to compare the tractable classes generated by the $\dot{\bowtie}$ operation with the four classes given in [53], we need to be able to compute $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$. The next example shows that the relationship between the sets $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$, $Pol(\Gamma_1)$, and $Pol(\Gamma_2)$ is not straightforward.

Example 5.5 Let $D = \{0,1\}$ and $D' = \{2,3\}$ be sets, and consider the following three binary relations over D, D, and D' respectively.

$$\begin{aligned} R_1 &= \{\langle 0,0\rangle, \langle 0,1\rangle\} \\ R_2 &= \{\langle 0,0\rangle, \langle 1,0\rangle\} \\ R' &= \{\langle 2,3\rangle, \langle 3,3\rangle\} \end{aligned}$$

Of course, R_1 and R_2 are permutations of each other, and so it follows immediately from Lemma 2.19 that $Pol(\{R_1\}) = Pol(\{R_2\})$. But consider $\varphi : (D \cup D')^2 \to (D \cup D')$ defined by the following table:

φ	0	1	2	3
0	0	1	3	3
1	0	1	0	3
2	0	1	2	3
3	0	1	2	3

(so, for example, $\varphi(1,2) = 0$, and so on).

It is easy to verify that $\varphi \in Pol(R_1 \dot{\bowtie} R') \setminus Pol(R_2 \dot{\bowtie} R')$. In particular therefore, knowledge of $Pol(\{R_1\})$ and $Pol(\{R'\})$ is not in itself sufficient to determine $Pol(R_1 \dot{\bowtie} R')$: more information is required.

Definition 5.6 Let D be a set, and fix $k \in \mathbb{N}$ (note that k may be 0). A *k-ary pattern on D* is just a k-tuple of subsets of D.

Definition 5.7 Let D be a set, and let p be a k-ary pattern on D. For any total k-ary operation φ on D, the *restriction of* φ *to* p, written φ_p, is the partial operation defined so that

$$\varphi_p(x_1, x_2, \ldots, x_k) = \varphi(x_1, x_2, \ldots, x_k)$$

whenever each $x_i \in p[i]$, and which is undefined everywhere else. We say that the arguments of φ_p must all be *of pattern* p.

Example 5.8 Consider the operation φ defined in Example 5.5. The partial operations $\varphi_{\langle D,D\rangle}$, $\varphi_{\langle D',D\rangle}$, and $\varphi_{\langle D',D'\rangle}$ are all simply projections onto their second argument; the partial operation $\varphi_{\langle D,D'\rangle}$ is defined by

$$\varphi_{\langle D,D'\rangle}(x,y) := \begin{cases} 0 & \text{if } (x,y) = (1,2) \\ 3 & \text{otherwise.} \end{cases}$$

Clearly, a k-ary operation is completely defined by giving its restriction to all possible k-ary patterns. The next lemma goes further, and says that if we consider just patterns from $\{D_1, D_2\}^k$ (where D_1 and D_2 are disjoint) then we can build all polymorphisms of $\Gamma_1 \dot{\bowtie} \Gamma_2$ by putting together arbitrary partial closure operations: one for each possible pattern.

Lemma 5.9 Let Γ_1 and Γ_2 be sets of relations over disjoint domains D_1 and D_2 respectively. Let $\varphi : (D_1 \cup D_2)^k \to (D_1 \cup D_2)$ be some operation. Then $\varphi \in \mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2)$ if, and only if, $\Gamma_1 \dot{\bowtie} \Gamma_2$ is closed under φ_p for every $p \in \{D_1, D_2\}^k$.

Proof We begin by proving the "if" direction. If $\Gamma_1 \dot{\bowtie} \Gamma_2 = \emptyset$ then the result is trivially true. Otherwise, pick an arbitrary relation $R \in \Gamma_1 \dot{\bowtie} \Gamma_2$; we shall show that R is closed under φ. Consider an arbitrary k-tuple $\langle t_1, t_2, \ldots, t_k \rangle$ of tuples from R. By definition, $R = R_1 \cup R_2$ for some relations $R_1 \in \Gamma_1, R_2 \in \Gamma_2$. Thus, each t_i comes either from R_1 or from R_2. Define the pattern $p \in \{D_1, D_2\}^k$ by:

$$p[i] := \begin{cases} D_1 & \text{if } t_i \in R_1; \\ D_2 & \text{if } t_i \in R_2. \end{cases}$$

Then by assumption, $\varphi_p(t_1, t_2, \ldots, t_k)$ is defined and equal to some tuple $t \in R$. Thus $\varphi(t_1, t_2, \ldots, t_k) = t \in R$, and we are done. The proof of the converse is trivial: *any* set Γ of relations is closed under all partial versions of each of its polymorphisms. $\square$

Definition 5.10 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively. Fix $k \in \mathbb{N}$, and let p be a pattern of arity k. Then we define the *restriction of* $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ *to* p to be the set of partial operations given by:

$$Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p := \{\varphi_p : \varphi \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2) \wedge \varphi \text{ has arity } k\}$$

An immediate consequence of Lemma 5.9 is that if φ_p is a partial closure operation on $\Gamma_1 \dot{\bowtie} \Gamma_2$ which is defined for all tuples of pattern p (where $p \in \{D_1, D_2\}^k$), then φ_p can be extended to a total closure operation. This follows because $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ always contains at least one function of every arity: in particular, it contains every projection.

Lemma 5.9 thus tells us that $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ is precisely characterised by the set $\{Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p : p \in \{D_1, D_2\}^*\}$ of restrictions of $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ to each possible "natural" pattern of each possible arity. It is therefore reasonable to ask how each set $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p$ depends upon $Pol(\Gamma_1)$ and $Pol(\Gamma_2)$, since this will tell us how $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ itself depends on $Pol(\Gamma_1)$ and $Pol(\Gamma_2)$. Proposition 5.13 will show that for each $p \in \{D_1, D_2\}^*$, there are certain partial operations which we can always expect to be present in $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p$. Before we can state and prove this result precisely however, we first need the following definition.

Definition 5.11 Let D be a set, and let Ψ be a set of operations over D, where each operation $\psi \in \Psi$ is a total operation $\psi : D_\psi^i \to D_\psi$ for some i and $D_\psi \subseteq D$.

We say that a total operation $\varphi : D^k \to D$ is *synthesisable* from Ψ if the following conditions both hold.

- There is some $\Psi' \subseteq \Psi$ such that $\bigcup\{D_\psi : \psi \in \Psi'\} = D$; and
- For every $p = \langle D_{\psi_1}, D_{\psi_2}, \ldots, D_{\psi_k}\rangle$, with each $\psi_i \in \Psi'$ (not necessarily all distinct), there is some set D_ψ, which occurs at positions $u_1, u_2, \ldots, u_r$ of p, and

some r-ary operation $\psi \in \Psi'$, such that $\psi : D_\psi^r \to D_\psi$ and the following identity holds:

$$\varphi_p(x_1, x_2, \ldots, x_k) = \psi(x_{u_1}, x_{u_2}, \ldots, x_{u_r})$$

That is, φ_p ignores all of its arguments, except for some subset which all lie in D_ψ, and on these it behaves exactly like ψ.

The set of all operations which are synthesisable from Ψ will be denoted $Syn(\Psi)$.

Note that any total operation φ is synthesisable from $\{\varphi\}$. In addition, if $\varphi \in Syn(\Phi)$ for some Φ, then $\varphi \in Syn(\Phi')$ for each $\Phi' \supseteq \Phi$.

Example 5.12 Suppose that $D = \{0, 1, 2, 3\}$, that Ψ_1 consists precisely of the projections of every arity > 0 on the domain $\{0, 1\}$, and that Ψ_2 consists precisely of the negations of the projections of every arity > 0 on the domain $\{2, 3\}$. That is, Ψ_1 contains operations such as $\psi(x, y, z) = y$, where $x, y, z \in \{0, 1\}$; and Ψ_2 contains operations such as $\psi(x, y, z) = 5 - z$, where $x, y \in \{2, 3\}$.

If $\Psi = \Psi_1 \cup \Psi_2$ then the restriction of any k-ary operation $\varphi \in Syn(\Psi)$ to a pattern p from $\{\{0, 1\}, \{2, 3\}\}^k$ will be such that φ_p will depend on exactly one of its arguments: x_i, say, and furthermore:

$$\varphi_p(x_1, x_2, \ldots, x_k) = \begin{cases} x_i & \text{if } p[i] = \{0, 1\} \\ 5 - x_i & \text{if } p[i] = \{2, 3\}. \end{cases}$$

We are now in a position to state the following proposition.

Proposition 5.13 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively. Then

$$Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2)) \subseteq Pol(\Gamma_1 \bowtie \Gamma_2).$$

Proof Fix any $\varphi \in Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2))$, of arity k.

By Lemma 5.9, it is enough to show that for every $p \in \{D_1, D_2\}^k$, it is the case that $\Gamma_1 \dot{\bowtie} \Gamma_2$ is closed under φ_p. So fix p. Without loss of generality, there exists some $\psi \in Pol(\Gamma_1)$ of arity r with the property that

$$\varphi_p(x_1, x_2, \ldots, x_k) = \psi(x_{u_1}, x_{u_2}, \ldots, x_{u_r})$$

where the u_i are those values, in ascending order, for which $p[u_i] = D_\psi = D_1$. (The case where $\psi \in Pol(\Gamma_2)$ is entirely similar.)

Now, take any $R \in \Gamma_1 \dot{\bowtie} \Gamma_2$, and suppose that R is the union of $R_1 \in \Gamma_1$ and $R_2 \in \Gamma_2$. We shall show that R is closed under φ_p. Consider a k-tuple $\langle t_1, t_2, \ldots, t_k \rangle$ of tuples from R with the property that for each i, t_i comes from relation R_1 if $p[i] = D_1$, and from R_2 otherwise. (By the definition of closure, these are the only tuples which we need to consider.) Then

$$\begin{aligned} \varphi_p(t_1, t_2, \ldots, t_k) &= \psi(t_{u_1}, t_{u_2}, \ldots, t_{u_r}) \\ &= t \end{aligned}$$

for some t where, since R_1 is closed under ψ, $t \in R_1$. But $R_1 \subseteq R$, so $t \in R$, and hence R is closed under φ_p.

Example 5.14 To continue Example 5.12, observe that the domains $\{0, 1\}$ and $\{2, 3\}$ of Ψ_1 and Ψ_2 respectively are disjoint. Consequently, all of the operations described in that example are contained within $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$.

Example 5.15 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively, and suppose that every relation in Γ_1 contains the constant tuple $\langle d, d, \ldots, d \rangle$ for some fixed $d \in D_1$. Then for every $0 \leq a$ it is the case that $Pol(\Gamma_1)$ contains the constant-d function on D_1 of arity a.

Now, given any arity $k \geq 0$, define $\varphi : (D_1 \cup D_2)^k \to (D_1 \cup D_2)$ to be the constant-d function of arity k. Then by the above comment, it is easy to see that $\varphi \in Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2))$, for we may always take the ψ required by the definition of synthesis to be a member of $Pol(\Gamma_1)$. Hence, by Proposition 5.13, $\varphi \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$.

Of course, this could have been immediately deduced from the definition of the $\dot{\bowtie}$ operator – all relations in $\Gamma_1 \dot{\bowtie} \Gamma_2$ must by definition contain a constant-d tuple. The point of the example is that because we allow operations of arity 0, the result falls cleanly out of Proposition 5.13. This observation will simplify several proofs below which would otherwise have had to deal with constant functions as a special case.

5.2.2 Restricting the Possible Polymorphisms

Proposition 5.13 is a positive result: it tells us that no matter which sets Γ_1 and Γ_2 of relations we begin with, we can always guarantee the existence of certain operations in $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$. In this section we turn our attention in the other direction, and examine how placing restrictions on the allowable Γ_i restricts the polymorphisms of their multiple relational union. The motivation for doing this is provided by Example 5.5. That example shows that the containment demonstrated in Proposition 5.13 is not, in general, an equality. By restricting the polymorphisms of the multiple relation union of two sets of operations however, we will be able to give a sufficient condition to ensure that equality does hold in many important cases.

Definition 5.16 Let Γ be a set of relations over the domain D, and let p be some k-ary pattern on D. For any a-ary relation $R \in \Gamma$, we define the R-induced graph of pattern p to be the undirected graph $G_p(R)$ constructed in the following way:

- The vertices of $G_p(R)$ are all those elements of D^k which are of pattern p; *i.e.*, the vertices are elements of $p[1] \times p[2] \times \ldots \times p[k]$.
- There is an edge between vertices u and v if, and only if, there is a sequence $r_1, r_2, \ldots, r_k$ of tuples from R (not necessarily distinct), and some c, d with $1 \leq c, d \leq a$ such that for each $i \leq k$, $r_i[c] = u[i]$ and $r_i[d] = v[i]$.

Example 5.17 Consider again the relation R from Example 5.3. If we take $p = \langle\{0,1\},\{0,1,2\},\{1,3\}\rangle$ then $G_p(R)$ contains twelve vertices, *viz:* $\langle 0,0,1\rangle$, $\langle 0,0,3\rangle$,

$\langle 0,1,1\rangle$, ..., $\langle 1,2,3\rangle$. There is an edge between vertices $\langle 1,2,1\rangle$ and $\langle 0,2,1\rangle$ (say) because we may take r_1, r_2, r_3 to be the fourth, fifth, and sixth rows respectively of R, along with $c = 1$ and $d = 2$. Similarly, if we take r_1, r_2, r_3 to be the third, third, and sixth rows of R, and take $c = 1$ and $d = 5$, then we obtain that there is an edge between $\langle 0,0,1\rangle$ and $\langle 1,1,1\rangle$. On the other hand, there is no edge between $\langle 0,0,1\rangle$ and $\langle 0,0,3\rangle$ for there is no row of R which contains both a 1 and a 3.

We shall primarily be interested in the question of whether or not $G_p(R)$ is connected. The reason for this is given by the following proposition.

Proposition 5.18 Let Γ_1 and Γ_2 be sets of relations over disjoint domains D_1 and D_2 respectively. Fix k, and let $p \in \{D_1, D_2\}^k$. If there is some $R \in \Gamma_1 \dot{\bowtie} \Gamma_2$ such that $G_p(R)$ is a connected graph, then for every $\varphi_p \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p$ it is the case that either $range(\varphi_p) \subseteq D_1$ or $range(\varphi_p) \subseteq D_2$.

Proof Choose any $\varphi_p \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p$. Since $R \in \Gamma_1 \dot{\bowtie} \Gamma_2$, it must be that R is closed under φ_p. Now, choose any k-tuple t over $(D_1 \cup D_2)$ which has pattern p. Of course, t is one of the vertices of $G_p(R)$. Let t' be any k-tuple to which t is directly connected. Then by the definition of the graph, there are tuples $\langle r_1, r_2, \ldots, r_k\rangle$ from R, and some c and d such that $t[i] = r_i[c]$ and $t'[i] = r_i[d]$ for every i. Now, $\varphi_p(r_1, r_2, \ldots, r_k)$ is well defined (since for each i, $\langle r_1[i], r_2[i], \ldots, r_k[i]\rangle$ is of pattern p), and hence equal to some $r \in R$. Of course, r consists either entirely of elements from D_1 or else entirely of elements from D_2. But $\varphi_p(t) = r[c]$ and $\varphi_p(t') = r[d]$, so in particular, $\varphi_p(t)$ and $\varphi_p(t')$ are either both elements of D_1 or else are both elements of D_2. A simple transitivity argument therefore allows us to conclude that for any k-tuple s (of pattern p) which can be reached from t in $G_p(R)$, it is the case that $\varphi_p(s)$ lies in the same domain as $\varphi_p(t)$. Since the graph is connected, all such vertices may be reached, and the result follows. □

Proposition 5.18 places some severe restrictions on the variety of functions which may be found within $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p$, but it is rather unwieldy to use: the condition of connectivity of $G_p(R)$ is not always an easy one to check. We now define the "star property" of relations, a very simple condition which will allow us to state a much more usable sufficient condition to ensure the connectivity of $G_p(R)$.

Definition 5.19 Let R be a relation of arity a over some domain D. Assume without loss of generality that $D = \{1, 2, \dots, |D|\}$. R is said to have the (c, d)-*star property*, for $c, d \leq a$, if there exist tuples $r_1, r_2, \dots, r_{|D|} \in R$, along with $u \in D$, such that for each $i \leq |D|$, $r_i[c] = u$ and $r_i[d] = i$.

Example 5.20 If $D = \{1, 2, 3\}$ then the relation R defined by

$$R := \{\langle 1,1,1\rangle, \langle 2,3,1\rangle, \langle 3,1,2\rangle, \langle 1,2,1\rangle\}$$

has the $(3, 2)$-star property, since on taking $u = 1$ we find that tuples $r_1 = \langle 1,1,1\rangle$, $r_2 = \langle 1,2,1\rangle$, and $r_3 = \langle 2,3,1\rangle$ have the desired property.

Lemma 5.21 Let Γ_1 and Γ_2 be sets of relations over disjoint domains D_1 and D_2 respectively. Suppose that there is some $R_1 \in \Gamma_1$ and $R_2 \in \Gamma_2$, along with $c, d \in \mathbb{N}$ so that R_1 and R_2 are both of the same arity, a, and both have the (c, d)-star property. Then there is an $R \in \Gamma_1 \bowtie \Gamma_2$ such that for any pattern $p \in \{D_1, D_2\}^k$, $G_p(R)$ is connected.

Proof Assume without loss of generality that $D_1 = \{1, 2, \dots, |D_1|\}$ and that $D_2 = \{|D_1| + 1, |D_1| + 2, \dots, |D_1| + |D_2|\}$. Fix k, and let $p \in \{D_1, D_2\}^k$. Since R_1 (resp. R_2) has the (c, d)-star property, there are r_1, r_2, …, $r_{|D_1|} \in R_1$ (resp. $r_{|D_1|+1}$, $r_{|D_1|+2}$, …, $r_{|D_1|+|D_2|} \in R_2$), along with $u_1 \in D_1$ (resp. $u_2 \in D_2$) so that the conditions of Definition 5.19 are fulfilled in each case. That is:

- For every $i, r_i[d] = i$;
- for every $i \in |D_1|, r_i[c] = u_1$; and
- for every $i \in |D_2|, r_i[c] = u_2$.

Define $R = R_1 \cup R_2$, and define the k-tuple t by setting

$$t[i] := \begin{cases} u_1 & \text{if } p[i] = D_1 \\ u_2 & \text{if } p[i] = D_2 \end{cases}$$

for each $i \leq k$. Clearly, t has pattern p, and is thus one of the vertices of $G_p(R)$. Let t' be any other vertex of $G_p(R)$. We will show that t and t' are connected by an edge. Intuitively, we show that $G_p(R)$ contains as a subgraph the star graph whose central vertex is t.

So choose such a t', and consider the sequence $s_1, s_2, \ldots, s_k$ of tuples from R defined for every $i \leq k$ by

$$s_i := r_{t'[i]}.$$

Then for every $i \leq k$, it is the case both that $s_i[d] = r_{t'[i]}[d] = t'[i]$ and

$$\begin{aligned} s_i[c] &= r_{t'[i]}[c] \\ &= \begin{cases} u_1 & \text{if } t'[i] \in |D_1| \\ u_2 & \text{if } t'[i] \in |D_2| \end{cases} \\ &= \begin{cases} u_1 & \text{if } p[i] = D_1 \\ u_2 & \text{if } p[i] = D_2 \end{cases} \\ &= t[i]. \end{aligned}$$

Therefore, the tuples $s_1, s_2, \ldots, s_k$ are witnesses to the existence of an edge between vertices t and t' in $G_p(R)$. So $G_p(R)$ is connected (indeed, has diameter at most 2). □

Corollary 5.22 Let Γ_1 and Γ_2 be sets of relations over disjoint domains D_1 and D_2 respectively. Suppose that there is some $R_1 \in \Gamma_1$ and $R_2 \in \Gamma_2$, along with $c, d \in \mathbb{N}$ so that R_1 and R_2 are both of the same arity, and both have the (c, d)-star property. Then for every $\varphi_p \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_p$ of arity k (with $p \in \{D_1, D_2\}^k$) it is the case that either $range(\varphi_p) \subseteq D_1$ or $range(\varphi_p) \subseteq D_2$.

Proof Immediate from Proposition 5.18 and Lemma 5.21. □

Example 5.23 Let us turn our attention once again to the relations R_1, R_2, and R' of Example 5.5. Observe that whilst each of these relations has the (c, d)-star property for some c and d, R_1 has the $(1, 2)$-star property, whereas R_2 and R' each have the $(2, 1)$-star property. Thus Corollary 5.22 tells us that every $\psi \in Pol(R_2 \dot{\bowtie} R')$ of arity k must be such that for any pattern $p \in \{D, D'\}^k$, either $range(\psi_p) \subseteq D$ or $range(\psi_p) \subseteq D'$. However, if we examine the operation φ defined in the example, we notice that $range(\varphi_{\langle D,D'\rangle}) = \{0, 3\}$. Thus, φ cannot be a member of $Pol(R_2 \dot{\bowtie} R')$. This is what we had discovered by hand.

The next step we take towards restricting the possible polymorphisms of $\Gamma_1 \dot{\bowtie} \Gamma_2$ is motivated by the following example.

Example 5.24 Consider R_1 over the domain $D_1 = \{0, 1\}$ and R_2 over the domain $D_2 = \{2, 3\}$ defined by

$$\begin{aligned} R_1 &:= \{\langle 0,1\rangle, \langle 1,1\rangle\}, \text{ and} \\ R_2 &:= \{\langle 2,3\rangle, \langle 3,3\rangle\}. \end{aligned}$$

It is easy to see that R_1 and R_2 both have the $(2, 1)$-star property, and so the range of any polymorphism of $R_1 \dot{\bowtie} R_2$, when its arguments are restricted to having a specific pattern, is a subset either of $\{0, 1\}$ or of $\{2, 3\}$. Consider one such polymorphism, φ, defined on pattern $\langle D_2, D_1\rangle$ by

$$\begin{aligned} \varphi_{\langle D_2,D_1\rangle}(2,0) &= 0 \\ \varphi_{\langle D_2,D_1\rangle}(3,0) &= 1 \\ \varphi_{\langle D_2,D_1\rangle}(2,1) &= 1 \\ \varphi_{\langle D_2,D_1\rangle}(3,1) &= 1 \end{aligned}$$

Such a φ exists by Lemma 5.9. Note that $\varphi_{\langle D_2,D_1\rangle}$ depends on both of its arguments: $\varphi_{\langle D_2,D_1\rangle}(2,1) \neq \varphi_{\langle D_2,D_1\rangle}(2,0) \neq \varphi_{\langle D_2,D_1\rangle}(3,0)$. From this, we can immediately conclude that $\varphi \notin Syn(Pol(R_1) \cup Pol(R_2))$. For if it were, then the restriction of φ to any pattern $p \in \{D_1, D_2\}^2$ could depend only on those of its arguments which came from the same domain as did $range(\varphi_p)$. In particular, $\varphi_{\langle D_2, D_1 \rangle}$ could depend only on its second argument.

The following definition and lemma will give a condition which will enable us to assert that the polymorphisms of $\Gamma_1 \ltimes\!\rtimes \Gamma_2$ are precisely those operations contained within $Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2))$.

Definition 5.25 Let D be a domain. Then the *equality relation of arity a on D* is defined to be the relation

$$=_a^D := \{\langle x, x, \ldots, x\rangle : x \in D\}$$

where the number of occurrences of x in each tuple is a.

Lemma 5.26 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively. Suppose that there is $a \in \mathbb{N}$ such that each Γ_i contains the relation $=_a^{D_i}$, and suppose further that there is some $R \in \Gamma_1 \ltimes\!\rtimes \Gamma_2$, also of arity a, with the property that $G_p(R)$ is connected for every pattern $p \in \{D_1, D_2\}^k$ of every arity. Fix some such p. Then for every $\varphi \in Pol(\Gamma_1 \ltimes\!\rtimes \Gamma_2)$, it is the case that φ_p depends only on those of its arguments which come from the same domain as does $range(\varphi_p)$.

Proof Suppose without loss of generality that $range(\varphi_p) \subseteq D_1$. If $|D_2| = 1$ then there is nothing to prove: φ_p cannot depend on those of its arguments which come from D_2, since these arguments cannot vary. So assume that $|D_2| \geq 2$, and let $u_1, u_2, \ldots, u_s$ be those indexes (in ascending order) such that $p[u_i] = D_2$.

Now $R = R_1 \cup R_2$, where $R_1 \in \Gamma_1$ and $R_2 \in \Gamma_2$. Define $R' \in \Gamma_1 \ltimes\!\rtimes \Gamma_2$ by $R' := (=_a^{D_1}) \cup R_2$. By definition, R' is closed under φ_p. Also, $G_{\langle D_2, D_2, \ldots, D_2\rangle}(R')$ is a connected graph, where the number of occurrences of D_2 in that expression is s.

Let $t_1, t_2 \in (D_1 \cup D_2)^k$ be tuples of pattern p with the properties that:

(i) whenever $p[i] = D_1$, it is the case that $t_1[i] = t_2[i]$; and

(ii) there is an edge in $G_{\langle D_2, D_2, \ldots, D_2 \rangle}(R')$ between $\langle t_1[u_1], t_1[u_2], \ldots, t_1[u_s] \rangle$ and $\langle t_2[u_1], t_2[u_2], \ldots, t_2[u_s] \rangle$.

We now show that $\varphi_p(t_1) = \varphi_p(t_2)$, whereupon the connectivity of $G_{\langle D_2, D_2, \ldots, D_2 \rangle}(R')$ will mean that we are done (for we will have shown that varying those arguments of φ_p which come from D_2 does not change the value of $\varphi_p(t_1)$).

The second of the properties above implies that there exist tuples $r_{u_1}, r_{u_2}, \ldots, r_{u_s}$ from R, along with $c, d \in \mathbb{N}$ with the property that for each $i \leq s$, $r_{u_i}[c] = t_1[u_i]$ and $r_{u_i}[d] = t_2[u_i]$

Define the sequence v_1, v_2, …, v_k of tuples from R by:

$$v_i := \begin{cases} \langle t_2[i], t_2[i], \ldots, t_2[i] \rangle & \text{if } p[i] = D_1 \\ r_i & \text{if } p[i] = D_2 \end{cases}$$

Given this definition, it is not hard to see that $\langle v_1[c], v_2[c], \ldots, v_k[c] \rangle = t_1$, and that $\langle v_1[d], v_2[d], \ldots, v_k[d] \rangle = t_2$. Now, $\varphi_p(v_1, v_2, \ldots, v_k)$ is defined, and by assumption is equal to some $r \in =_a^{D_1}$. But by the definition of equality, $r[c] = r[d]$, and hence $\varphi_p(t_1) = \varphi_p(t_2)$, as required. □

Corollary 5.27 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively. Suppose that there is some $a \in \mathbb{N}$ such that each Γ_i contains the relation $=_a^{D_i}$, and suppose further that there is some $R \in \Gamma_1 \bowtie \Gamma_2$, also of arity a, with the property that $G_p(R)$ is connected for every pattern $p \in \{D_1, D_2\}^*$. Then

$$Pol(\Gamma_1 \bowtie \Gamma_2) = Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2)).$$

Proof That the synthesisable operations are in $Pol(\Gamma_1 \bowtie \Gamma_2)$ follows immediately from Proposition 5.13. It is the converse that we prove here. In particular, we will show that the converse holds when we take Ψ' in the definition of synthesis to be the whole of $Pol(\Gamma_1) \cup Pol(\Gamma_2)$.

Clearly, every operation from Ψ' is total (on its own domain), and the union of all the domains of the operations in Ψ' is $D_1 \cup D_2$, as required.

Now for any k, $\{D_\psi : \psi \in \Psi'\}^k$ is just $\{D_1, D_2\}^k$. So choose some $\varphi \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ of arity k, and let p be some pattern from $\{D_1, D_2\}^k$. Assume without loss of generality that $range(\varphi_p) \subseteq D_1$. Let $u_1, u_2, \ldots, u_r$ be those values, in ascending order, for which $p[u_i] = D_1$, and consider any r-ary operation φ' over D_1, which has been obtained from φ_p by arbitrarily fixing those of its arguments which come from D_2. Since every relation in $\Gamma_1 \dot{\bowtie} \Gamma_2$ is closed under φ_p, it is clear that every relation in Γ_1 is closed under φ'; that is, that $\varphi' \in Pol(\Gamma_1) \subseteq \Psi'$. Now, by Lemma 5.26, the same operation φ' results regardless of how the arguments from D_2 are fixed. Thus,

$$\varphi_p(x_1, x_2, \ldots, x_k) = \varphi'(x_{u_1}, x_{u_2}, \ldots, x_{u_r}).$$

Since this holds for any pattern p, it follows that $\varphi \in Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2))$. □

The following corollary is immediate from Lemma 5.21.

Corollary 5.28 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively. Suppose that there is some $R_1 \in \Gamma_1$ and $R_2 \in \Gamma_2$, with R_1 and R_2 both being of arity a, and each having the (c, d)-star property for some c and d. Suppose too that each Γ_i contains the relation $=_a^{D_i}$. Then

$$Pol(\Gamma_1 \dot{\bowtie} \Gamma_2) = Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2)).$$

Corollary 5.29 Let Γ_1 and Γ_2 be relational clones over the disjoint domains D_1 and D_2 respectively. Then

$$Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2)) = Pol(\Gamma_1 \dot{\bowtie} \Gamma_2).$$

Proof Fix $a = 2$. Then the two relations $=_2^{D_1}$ and $D_1 \times D_1$ must both be present in Γ_1, since it is easy to see that adding them to any set of relations over D_1 does not change the polymorphisms of that set of relations. Similarly, the two relations $=_2^{D_2}$ and $D_2 \times D_2$ must both be present in Γ_2. Now, $D_1 \times D_1$ and $D_2 \times D_2$ both have the $(1, 2)$-star property. The result follows from Corollary 5.28. □

5.3 Combining Tractable Sets of Relations

5.3.1 Generating Novel Tractable Classes

We are now in a position to determine whether or not the multiple relational union operator is capable of generating genuinely novel tractable classes. The following proposition shows that the classes it generates are not always novel.

Proposition 5.30 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively. Then the following statements all hold.

1. If Γ_1 is closed under a constant function φ, then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is closed under that constant function.

2. If Γ_1 and Γ_2 are closed under ACI operations φ_1 and φ_2 respectively, then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is closed under an ACI operation.

3. If Γ_1 and Γ_2 are closed under majority operations φ_1 and φ_2 respectively, then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is closed under a majority operation.

Proof 1. We may assume without loss of generality that φ is an operation of arity 0. Then Proposition 5.13 tells us that $\varphi_{\langle\rangle}$ is a member of $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)_{\langle\rangle}$. Since $\varphi_{\langle\rangle} \equiv \varphi$, it follows that $\Gamma_1 \dot{\bowtie} \Gamma_2$ is closed under φ.

2. Define the binary operation ψ by:

$$\psi_p(x,y) := \begin{cases} \varphi_1(x,y) & \text{if } p = \langle D_1, D_1 \rangle \\ x & \text{if } p = \langle D_1, D_2 \rangle \\ y & \text{if } p = \langle D_2, D_1 \rangle \\ \varphi_2(x,y) & \text{if } p = \langle D_2, D_2 \rangle \end{cases}$$

Since ψ is built from φ_1, φ_2, and various projections, Proposition 5.13 tells us that $\psi \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$. It remains to show that ψ is an ACI operation. ψ is clearly idempotent, because each φ_i is. That $\psi(x,y) = \psi(y,x)$ follows from the commutativity of each φ_i (if x and y come from the same domain) and from the symmetry of the rest of the definition (if not). That $\psi(\psi(x,y),z) = \psi(x,\psi(y,z))$ is a little less obvious, but can be demonstrated by performing a case analysis on the possible domains of x, y, and z:

x	y	z	$\psi(\psi(x,y),z)$	$\psi(x,\psi(y,z))$
D_1	D_1	D_1	$\varphi_1(\varphi_1(x,y),z)$	$\varphi_1(x,\varphi_1(y,z))$
D_1	D_1	D_2	$\varphi_1(x,y)$	$\varphi_1(x,y)$
D_1	D_2	D_1	$\varphi_1(x,z)$	$\varphi_1(x,z)$
D_1	D_2	D_2	x	x
D_2	D_1	D_1	$\varphi_1(y,z)$	$\varphi_1(y,z)$
D_2	D_1	D_2	y	y
D_2	D_2	D_1	z	z
D_2	D_2	D_2	$\varphi_2(\varphi_2(x,y),z)$	$\varphi_2(x,\varphi_2(y,z))$

Since φ_1 and φ_2 are associative, the table shows that the whole of ψ is associative also.

3. Define the ternary operation ψ by:

$$\psi_p(x,y,z) := \begin{cases} \varphi_1(x,y,z) & \text{if } p = \langle D_1, D_1, D_1\rangle \\ x & \text{if } p = \langle D_1, D_1, D_2\rangle \text{ or } \langle D_2, D_2, D_1\rangle \\ y & \text{if } p = \langle D_1, D_2, D_2\rangle \text{ or } \langle D_2, D_1, D_1\rangle \\ z & \text{if } p = \langle D_1, D_2, D_1\rangle \text{ or } \langle D_2, D_1, D_2\rangle \\ \varphi_2(x,y,z) & \text{if } p = \langle D_2, D_2, D_2\rangle \end{cases}$$

Once again, Proposition 5.13 guarantees that $\psi \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$; we must show that ψ is a majority operation. Consider $\psi(x,y,z)$. If x, y, and z all come from the same domain, then the result follows from the fact that φ_1 and φ_2 are majority operations. Suppose not, and suppose that (say) x and z are equal. Then x and z must clearly come from the same domain, and (by assumption) that must be a different domain from y. Thus, by definition, $\psi(x, y, z) = z$. All other cases are similar.

□

Note that the analogous result to Proposition 5.30 fails to hold when we turn our attention to affine operations. That is, if Γ_1 and Γ_2 are both closed under an affine operation then the same is not in general true of $\Gamma_1 \dot{\bowtie} \Gamma_2$. This is because any relation R which is closed under an affine operation defined via some Abelian group G, must be such that $|G|$ is an exact multiple of $|R|$. But $|R_1 \cup R_2|$ is not, in general, a divisor of $|G|$.

Proposition 5.31 Suppose that Γ_1 and Γ_2 are relational clones over the disjoint domains D_1 and D_2 respectively. Let Φ be some property of operations which can be defined by identities, and suppose that there is no $\varphi \in Pol(\Gamma_1)$ which has the property Φ. Then there is no non-constant $\varphi \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ which has property Φ.

Proof Let $\varphi \in Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ be a non-constant operation. Since Γ_1 and Γ_2 are relational clones, we can deduce from Corollary 5.29 that $\varphi \in Syn(Pol(\Gamma_1) \cup Pol(\Gamma_2))$. Now, φ has a strictly positive arity, and so the restriction of φ to the domain D_1 is a member of $Pol(\Gamma_1)$, and thus does not have property Φ. Since Φ is defined by identities, it follows *a fortiori* that φ itself does not have property Φ. □

Corollary 5.32 Suppose that Γ_1 and Γ_2 are relational clones over the disjoint domains D_1 and D_2 respectively, and that $Pol(\Gamma_1)$ contains no binary ACI operations (resp. no majority operations, resp. no affine operations). Then $Pol(\Gamma_1 \dot{\bowtie} \Gamma_2)$ contains no such operations either.

Proof Follows immediately from Proposition 5.31 and the fact that constant operations are never binary ACI (resp. majority, resp. affine) operations on domains of size two or more. □

Recall that four different classes of tractable sets of relations were defined in [53]. Of these, one consists of those sets of relations which are closed under a constant operation, and is therefore relatively uninteresting. The immediate consequence of Corollary 5.32 however, is that if Γ_1 and Γ_2 are maximal tractable sets of relations from two of the other three classes, then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is a tractable set of relations which doesn't fall into any of the classes from [53].

We can go further however, and observe that since all known tractable sets of relations can be defined by closure under some operation defined by identities, it follows that the $\dot{\bowtie}$ operation is capable of generating genuinely new tractable sets of relations. For let Φ be *any* class of non-constant operations which is defined by identities and which guarantees tractability (this class may be one of the four classes from [53] or some other class: perhaps even a class not yet identified). We may then construct two sets of relations. The first, Γ_1, is closed under an operation from Φ, but not under any other operation known to give tractability. The second, Γ_2, is not closed under any operation from Φ, but is closed under some other non-constant operation known to give tractability: a binary ACI operation, for example, or a majority operation.

Let us assume, without loss of generality, that Γ_1 and Γ_2 are relational clones. Then their multiple relation union will be a tractable set of relations. The reason for its tractabilty cannot be because of closure under an operation not from Φ, because Γ_1 is not closed under any such operation. Neither can the reason for its tractability be because of closure under an operation from Φ, because Γ_2 is not closed under such an operation. Consequently, $\Gamma_1 \dot{\bowtie} \Gamma_2$ is tractable for a hitherto unknown reason.

5.3.2 The Full Tractable Class

In the previous subsection we saw how combining two tractable sets of relations over disjoint domains could lead to novel tractable sets of relations. The question then arises as to whether there are any other relations, apart from those in $\Gamma_1 \dot{\bowtie} \Gamma_2$, whose tractability we can now deduce. Theorem 2.21 says that in fact the whole of $\mathit{Inv}(\mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2))$ is tractable, so our question reduces to the following: What relations are present in $\mathit{Inv}(\mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2))$ which are not present in $\Gamma_1 \dot{\bowtie} \Gamma_2$?

In some sense this is a trivial question: if we assume that equality is present, then the relations are precisely those which can be constructed from $\Gamma_1 \dot{\bowtie} \Gamma_2$ by applying a sequence of relational join and project operations [51]. Nevertheless, in this subsection we shall give some examples of relations from $\mathit{Inv}(\mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2)) \setminus (\Gamma_1 \dot{\bowtie} \Gamma_2)$, and show how they lead to tractable classes of relations whose tractability is not given directly by Theorem 5.2. The point is that our algebraic framework allows us to take one new tractable class of relations, and from it to generate yet more new tractable classes.

Consider, for example, the Cartesian product operator, $\times$, defined on pairs of relations by:

$$R^1 \times R^2 := \{r_1 r_2 : r_1 \in R^1 \wedge r_2 \in R^2\}$$

(where $r_1 r_2$ denotes the concatenation of the two tuples).

The following lemma comes essentially from [51].

Lemma 5.33 Let Γ_1 and Γ_2 be sets of relations over the disjoint domains D_1 and D_2 respectively, and let $R^1, R^2 \in \mathit{Inv}(\mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2))$ be two relations. Then $R^1 \times R^2 \in \mathit{Inv}(\mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2))$.

Proof Choose any $\varphi \in \mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2)$, and suppose φ has arity k. We shall show that $R^1 \times R^2$ is closed under φ. For let $r_1, r_2, \ldots, r_k$ be tuples from $R^1 \times R^2$. Each r_i is the concatenation of two tuples; one from R^1, and one from R^2. Let these two tuples be r_i^1 and r_i^2 respectively. Now, $\varphi(r_1^1, r_2^1, \ldots, r_k^1)$ and $\varphi(r_1^2, r_2^2, \ldots, r_k^2)$ are both defined, and equal to some $r^1 \in R^1$ and $r^2 \in R^2$ respectively. Thus $\varphi(r_1, r_2, \ldots, r_k) = r^1 r^2 \in (R^1 \times R^2)$. □

Example 5.34 Let $R_1 = \{r_1\}$ be a relation of size 1 over some domain D_1, and let R_2 be a tractable relation of the same arity over some disjoint domain D_2. Suppose that neither R_1 nor R_2 contains a constant tuple. Let $\Gamma_1 = \mathit{Inv}(\mathit{Pol}(R_1))$ and $\Gamma_2 = \mathit{Inv}(\mathit{Pol}(R_2))$. Then $R_1 \in \Gamma_1 \dot{\bowtie} \Gamma_2$ and $R_2 \in \Gamma_1 \dot{\bowtie} \Gamma_2$. Now, R_1 must be tractable; consequently Γ_1 and hence $\mathit{Inv}(\mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2))$ is tractable also. So by Lemma 5.33, $R_1 \times R_2$ is a tractable relation. But $R_1 \times R_2$ is just the concatenation of r_1 onto the front of every tuple in R_2. That is,

$$R_1 \times R_2 = \{r_1 r_2 : r_2 \in R_2\}.$$

This is obviously not a relation present in $\Gamma_1 \dot{\bowtie} \Gamma_2$.

Example 5.35 Suppose that

$$\begin{array}{rcl@{\quad}c@{\quad}rcl} R_1 & = & \{\langle 0,0,1,0\rangle, & \text{and} & R_2 & = & \{\langle 3,2,2,3\rangle, \\ & & \langle 0,1,0,0\rangle, & & & & \langle 2,3,2,2\rangle\}. \\ & & \langle 1,0,0,0\rangle, & & & & \\ & & \langle 1,1,1,0\rangle\} & & & & \end{array}$$

Let $\Gamma_1 = \mathit{Inv}(\mathit{Pol}(R_1))$ and $\Gamma_2 = \mathit{Inv}(\mathit{Pol}(R_2))$. It is easy to check that R_1 is closed under an affine operation, but not under any other operation known to yield tractability. Similarly, R_2 is closed under both majority and affine operations, but not under anything else significant. By Proposition 5.31 therefore, $\Gamma_1 \dot{\bowtie} \Gamma_2$ is a tractable set of relations which does not fall into any previously known tractable class (it is simple to verify that it is not closed under any affine operations). Moreover, $\mathit{Inv}(\mathit{Pol}(\Gamma_1 \dot{\bowtie} \Gamma_2))$ contains further relations whose tractability could not have been previously deduced. It can be verified that amongst these is not only the relation $R_1 \times R_2$ itself, but also the relation R of Example 5.3. For it is easy to see that R is a permutation of $(R_1 \cup R_2) \times \{\langle 2\rangle\}$, where $\{\langle 2\rangle\}$, being a projection of R_2, is a member of $\Gamma_1 \dot{\bowtie} \Gamma_2$.

The point of these examples is to demonstrate that the algebraic framework developed above, and in [51, 52, 54], allows us to immediately deduce that far more sets of relations are tractable than those that are explicitly given by Theorem 5.2 above. Put another way: although the algorithm given in Theorem 5.2 is a straightforward one, it gives rise at once to a collection of more complicated algorithms, each of which can be used in more general situations than can the original algorithm itself.

5.4 Multiple Relational Unions on Binary Domains

In this section we investigate how the results of Section 5.2 can be used to analyse the tractability of sets of relations over pairs of disjoint 2-valued domains.

Note that a pair of binary domains is the smallest interesting case to which the results of section 5.2 can be applied. If either of the two domains contained just one element – if $D_1 = \{e\}$ for example – then the only relations which could be built over D_1 would be the empty relation (of each arity), along with $\{\langle\rangle\}$, $\{\langle e\rangle\}$, $\{\langle e, e\rangle\}$, $\{\langle e, e, e\rangle\}$, *etc.* In particular, each of these, except for the empty relation, is closed under a constant operation, and so combining them with other sets of relations is not very interesting (in particular, it yields a set of relations which is closed under a constant operation, by Proposition 5.30). The empty relation itself is closed under every non-nullary operation, and combining it with other sets of relations turns out once again to be an uninteresting exercise. The proof of this is left to the reader.

Throughout this section therefore, we will take D_1 to be the set $\{0, 1\}$, and D_2 to be the set $\{2, 3\}$.

Post showed that there are exactly seven distinct minimal clones over a binary domain [67]. Each of them may be defined as the smallest possible clone containing some (non-trivial) operation. Specifically, the seven minimal clones on $\{0, 1\}$ are:

1. $\mathbf{0}$, containing the constant-0 operation;

2. $\mathbf{1}$, containing the constant-1 operation;

3. $\neg$, containing the negation operation;

4. $\wedge$, containing the binary conjunction operation;

5. $\vee$, containing the binary disjunction operation;

6. **maj**, containing the ternary majority operation; and

7. **par**, containing the ternary parity operation.

Given any set Γ of relations over $\{0, 1\}$, it may be the case that $Pol(\Gamma)$ contains one or more of the above minimal clones. If it contains any of **0**, **1**, $\wedge$, $\vee$, **maj**, or **par** then Γ is tractable [51, 68]; if it contains only $\neg$ (or contains none of the minimal clones) then it is **NP**-complete. Thus the tractability (or otherwise, assuming **PTIME** $\neq$ **NP**) of Γ can be determined in an efficient manner for all Γ.

Each of the above clones may be alternatively characterised as the set of polymorphisms of the corresponding relational clone (*Inv*(**0**), *Inv*(**1**), *etc.*). This follows from the fact [83] that for any clone Φ, $Pol(Inv(\Phi)) = \Phi$. Therefore, in order to determine the complexity of $\Gamma_1 \dot{\bowtie} \Gamma_2$ for arbitrary Γ_1 and Γ_2, it suffices to consider the cases where Γ_1 and Γ_2 are the invariants of minimal clones over their respective domains.

Since there are seven minimal clones, there are 49 possibilities for the two sets of relations. By symmetry however, we can immediately discard 21 of these combinations, leaving 28 which need to be examined in more detail. It is easy to see that **0** and **1** are identical up to renaming the domain elements; the same goes for $\wedge$ and $\vee$ (conjunction and disjunction are the same function, but with the concepts of Truth and Falsity reversed). So we need not consider either *Inv*(**1**) or *Inv*($\vee$) as being worthy of attention.

It follows from Proposition 5.30 that if $\Gamma_1 = Inv(\mathbf{0})$, or if $\Gamma_1 = \Gamma_2 = Inv(\wedge)$, or if $\Gamma_1 = \Gamma_2 = Inv(\mathbf{maj})$, then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is tractable for known reasons. There is nothing more to say in these cases. Similarly, if $\Gamma_1 = Inv(\neg)$ then $\Gamma_1 \dot{\bowtie} \Gamma_2$ is **NP**-complete by Proposition 5.4. This leaves us with just four essentially different cases to consider, all of which can be seen to be tractable by Theorem 5.2:

1. $\Gamma_1 = Inv(\wedge)$ and $\Gamma_2 = Inv(\mathbf{maj})$;

2. $\Gamma_1 = Inv(\wedge)$ and $\Gamma_2 = Inv(\mathbf{par})$;

3. $\Gamma_1 = Inv(\mathbf{maj})$ and $\Gamma_2 = Inv(\mathbf{par})$;

4. $\Gamma_1 = Inv(\mathbf{par})$ and $\Gamma_2 = Inv(\mathbf{par})$.

By Proposition 5.31, none of the corresponding classes $\Gamma_1 \dot{\bowtie} \Gamma_2$ have previously been known to be tractable.

Example 5.36 Consider once again the relations R_1 and R_2 from Example 5.35. As we implicitly observed in that example, $Pol(\{R_1\})$ is the minimal clone **par**. Similarly, if we take R_3 to be defined by:

$$\begin{aligned} R_3 &= \{\langle 3,2,2\rangle \\ &\quad\ \langle 2,3,2\rangle \\ &\quad\ \langle 2,2,3\rangle \\ &\quad\ \langle 2,2,2\rangle\} \end{aligned}$$

then $Pol(\{R_2, R_3\})$ is just **maj**. Finally, if we define

$$\begin{aligned} R_4 &= \{\langle 5,5,4\rangle, & \text{and} \quad R_5 &= \{\langle 4,4,4\rangle, \\ &\quad\ \langle 4,5,5\rangle, & &\quad\ \langle 4,5,4\rangle, \\ &\quad\ \langle 4,5,4\rangle\} & &\quad\ \langle 5,4,4\rangle, \\ & & &\quad\ \langle 5,5,5\rangle\} \end{aligned}$$

then $Pol(\{R_4, R_5\}) = \wedge$.

Some of these relations are of arity 3, and some are of arity 4. Consequently, if we are not careful then their multiple relational union might be empty. We can circumvent this problem however by replacing each arity-3 relation R over the domain D by the arity-4 relation $R' := R \times D$. R' clearly has the same polymorphisms as does R.

Once this transformation has been performed, we may take the disjoint union of any pair of these sets of relations, and end up with a new tractable class of relations.

5.5 Conclusions

In this chapter we have shown how combining tractable sets of constraint relations on disjoint domains can yield new, hitherto unknown, tractable sets of relations. The algorithm which establishes the tractability of this combination is particularly simple, yet we have been able to exploit algebraic properties of the new sets of tractable relations to show that even more sets of relations are tractable than those which can be directly solved by the algorithm. This result clearly demonstrates once again the power of the algebraic approach to analysing constraint satisfaction problems.

This work is related to that of [15]. In that paper, the authors considered the effect of combining tractable sets of relations on a *single* domain. One question which naturally arises is to ask what happens when there are two domains which are partially disjoint, but which overlap in one or more places. Could there be a continuum, of which the results presented here and those in [15] form the ends?

We have also shown how the polymorphisms of a set of relations can sometimes be presented in an extremely compact way. In the case where $Pol(\Gamma_1)$ is the clone $\wedge$ on $\{0, 1\}$ for example, and $Pol(\Gamma_2)$ is the clone **maj** on $\{2, 3\}$, there are a total of 48384 possible ternary polymorphisms of $\Gamma_1 \dot{\bowtie} \Gamma_2$, yet these can be precisely described by giving just the 11 operations from $Pol(\Gamma_1) \cup Pol(\Gamma_2)$. Such a reduction means that computer programs which seek to calculate polymorphisms as part of their operation can be speeded up by an exponential factor under appropriate circumstances.

Finally, the results presented in this chapter take us one step closer to the ultimate goal of classifying the complexity of all possible sets of relations over domains of arbitrary finite size.

Bibliography

[1] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. ACM Symp. on Theory of Computing*, pages 209–219, 1991.

[2] M. Ajtai. Σ_1^1 formulae on finite structures. *Ann. of Pure and Applied Logic*, 24:1–48, 1983.

[3] M. Ajtai and R. Fagin. Reachability is harder for directed than for undirected graphs. *Journal of Symbolic Logic*, 55(1):1–48, March 1990.

[4] M. Ajtai, R. Fagin, and L. J. Stockmeyer. The closure of monadic NP (extended abstract). In *30th Annual ACM Symposium on Theory of Computing*, pages 309–318, 1998.

[5] A. A. Arratia-Quesada. *On the Existence of Normal Forms for Logics that Capture Complexity Classes.* PhD thesis, University of Wisconsin-Madison, 1997.

[6] A. A. Arratia-Quesada, S. R. Chauhan, and I. A. Stewart. Hierarchies in classes of program schemes. *Journal of Logic and Computation*, 9:915–957, 1999.

[7] A. A. Arratia-Quesada and I. A. Stewart. Generalized Hex and logical characterizations of polynomial space. *Inf. Proc. Letters*, 63(3):147–152, 1997.

[8] M. Bjäreland and P. Jonsson. Exploiting bipartiteness to identify yet another tractable subclass of CSP. In J. Jaffar, editor, *Principles and Practice of Con-*

straint Programming — CP'99, number 1713 in Lecture Notes in Computer Science, pages 118–128. Springer, 1999.

[9] A. Blass, Y. Gurevich, and S. Shelah. Choiceless Polynomial Time. *Annals of Pure and Applied Logic*, 100:141–187, 1999.

[10] J. Büchi. Weak second-order logic and finite automata. *Z.Math. Logik Grundlagen Math.*, 5:66–92, 1960.

[11] A. A. Bulatov, A. A. Krokhin, and P. Jeavons. Constraint satisfaction problems and finite algebras. Technical Report TR-4-99, Oxford University Computing Laboratory, 1999.

[12] A. A. Bulatov, A. A. Krokhin, and P. Jeavons. Constraints over a three-element domain: tractable maximal relational clones. Unpublished Manuscript, 1999.

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

[14] C. C. Chang and H. J. Keisler. *Model Theory.* North-Holland, 2nd edition, 1978.

[15] D. Cohen, P. Jeavons, and M. Koubarakis. Tractable disjunctive constraints. In *Proceedings 3rd International Conference on Constraint Programming — CP'96 (Linz, October 1997)*, volume 1330 of *Lecture Notes in Computer Science*, pages 478–490. Springer-Verlag, 1996.

[16] P. Cohn. *Universal Algebra.* Harper & Row, 1965.

[17] S. A. Cook. An observation of time-storage trade-off. *Journal of Computer and System Sciences*, 9:308–316, 1974.

[18] M. Cooper, D. Cohen, and P. Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65:347–361, 1994.

[19] S. S. Cosmadakis. Logical reducibility and monadic NP. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 52–61, 1993.

[20] B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. In *Proceedings of the Graph Minors Conference, Seattle.* Contemporary Mathematics, American Mathematical Society, 1991.

[21] B. Courcelle. The monadic second order logic of graphs VI: On several representations of graphs by relational structures. *Discrete Appl. Math.*, 54:117–149, 1994.

[22] B. Courcelle. The monadic second order logic of graphs VIII: orientations. *Ann. Pure and Appl. Logic*, 72:103–143, 1995.

[23] B. Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. *DIMACS Series in Discrete Mathematics and Theoretical Comp. Sci.*, 31:33–62, 1997.

[24] E. Dahlhaus. Reductions to NP-complete problems by interpretations. In E. B. et al., editor, *Logic and Machines: Decision Problems and Complexity, Lecture Notes in Computer Science 171*, pages 357–365. Springer-Verlag, 1983.

[25] V. Dalmau. A new tractable class of constraint satisfaction problems. In *6th International Symposium on Mathematics and Artificial Intelligence*, 2000.

[26] V. Dalmau and J. Pearson. Closure functions and width 1 problmes. In J. Jaffar, editor, *Principles and Practice of Constraint Programming — CP'99*, number 1713 in Lecture Notes in Computer Science, pages 159–173. Springer, 1999.

[27] A. Dawar. A restricted second-order logic for finite structures. In D. Leivant, editor, *Logic and Computational Complexity*, volume 960 of *LNCS*, pages 393–413. Springer, 1995.

[28] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[29] A. Durand, C. Lautemann, and T. Schwentick. Subclasses of Binary NP. Technischer Report 1/96, Institut für Informatik, Johannes Gutenberg-Universität Mainz, 1995.

[30] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 1995. ISBN 3-540-60149-X.

[31] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math.*, 49:129–141, 1961.

[32] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

[33] R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

[34] R. Fagin, L. Stockmeyer, and M. Y. Vardi. On monadic NP vs. monadic co-NP. *Information and Computation*, 120(1), July 1995.

[35] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Computing*, 28(1):57–104, 1998.

[36] R. Fraïssé. Sur quelques classifications des systèmes de relations. *Publications Scientifiques de l'Université d'Alger*, 1:35–182, 1954.

[37] E. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32:755–761, 1985.

[38] F. Gire and H. Hoang. An extension of fixpoint logic with a symmetry-based choice construct. *Information and Computation*, 144:40–65, 1998.

[39] E. Grädel. On transitive closure logic. In *CSL '91: 5th Workshop on Computer Science Logic*, volume 626 of *Lecture Notes in Computer Science*, pages 149–163. Springer-Verlag, 1991.

[40] E. Grädel. Capturing complexity classes by fragments of second order logic. *Theoretical Computer Science*, 101:35–57, 1992.

[41] M. Grohe. Bounded arity hierarchies in fixed-point logics. In *Proceedings of the 7th workshop on computer science logic*, Lecture Notes in Computer Science 832, pages 150–164. Springer-Verlag, 1993.

[42] M. Grohe. Arity hierarchies. *Annals of Pure and Applied Logic*, 82:103–163, 1996.

[43] M. Grohe and L. Hella. A double arity hierarchy theorem for transitive closure logic. *Archive for Mathematical Logic*, 35:157–171, 1996.

[44] Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Current trends in theoretical computer science*, pages 1–57. Computer Science Press, 1988.

[45] Y. Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation*, pages 9–36. Oxford University Press, 1995.

[46] Y. Gurevich. May 1997 draft of the ASM guide. Technical report, EECS Department, University of Michigan, 1997.

[47] W. Hanf. Model-theoretic methods in the study of elementary logic. In J. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 132–145. North Holland, 1965.

[48] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[49] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16:760–778, August 1987.

[50] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, October 1988.

[51] P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

[52] P. Jeavons and D. Cohen. An algebraic characterization of tractable constraints. In *Computing and Combinatorics. First International Conference COCOON'95 (Xi'an,China,August 1995)*, volume 959 of *Lecture Notes in Computer Science*, pages 633–642. Springer-Verlag, 1995.

[53] P. Jeavons, D. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In *Proceedings 1st International Conference on Constraint Programming — CP'95 (Cassis, France, September 1995)*, volume 976 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 1995.

[54] P. Jeavons, D. Cohen, and M. Gyssens. A test for tractability. In *Proceedings 2nd International Conference on Constraint Programming — CP'96 (Boston, August 1996)*, volume 1118 of *Lecture Notes in Computer Science*, pages 267–281. Springer-Verlag, 1996.

[55] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.

[56] P. Jeavons and M. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.

[57] A. Kaldewaij. *Programming: The Derivation of Algorithms.* International Series In Computer Science (editor C. A. R. Hoare). Prentice Hall, 1990.

[58] L. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence*, 64:147–160, 1993.

[59] P. Ladkin and R. Maddux. On binary constraint problems. *Journal of the ACM*, 41:435–469, 1994.

[60] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[61] F. Madelaine and I. A. Stewart. Some problems not definable using structure homomorphisms. Technical Report 1999/18, Dept. of Mathematics and Computer Science, University of Leicester, 1999.

[62] R. McKenzie, G. McNulty, and W. Taylor. *Algebras, Lattices and Varieties*, volume I. Wadsworth and Brooks, California, 1987.

[63] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[64] C. Morgan. *Programming from Specifications.* International Series In Computer Science (editor C. A. R. Hoare). Prentice Hall, 1990.

[65] B. Nebel and H.-J. Burckert. Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42:43–66, 1995.

[66] F. Neven, M. Otto, J. Tyszkiewicz, and J. Van den Bussche. Adding for-loops to first-order logic. In C. Beeri and P. Buneman, editors, *Proceedings of International Conference on Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, pages 58–69. Springer-Verlag, 1999.

[67] E. L. Post. The two-valued iterative systems of mathematical logic. *Annals of Math. Studies*, 5, 1941.

[68] T. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

[69] T. Schwentick. On winning Ehrenfeucht games and monadic NP. Informatik-Bericht 3/95, Universität Mainz, 1995.

[70] I. A. Stewart. Logical definability versus logical complexity: another equivalence. Submitted for publication.

[71] I. A. Stewart. Using program schemes to logically capture polynomial-time on certain classes of structures. Submitted for publication.

[72] I. A. Stewart. Comparing the expressibility of languages formed using NP-complete operators. *J. Logic Computat.*, 1(3):305–330, 1991.

[73] I. A. Stewart. Complete problems involving Boolean labelled structures and projection translations. *J. of Logic and Computation*, 1:861–882, 1991.

[74] I. A. Stewart. Using the Hamiltonian path operator to capture NP. *Journal of Computer and System Sciences*, 45:127–151, 1992.

[75] I. A. Stewart. Logical and schematic characterization of complexity classes. *Acta Informatica*, 30:61–87, 1993.

[76] I. A. Stewart. Logical characterizations of bounded query classes I: logspace oracle machines. *Fundamenta Informaticae*, 18:65–92, 1993.

[77] I. A. Stewart. Logical characterizations of bounded query classes II: Polynomial-time oracle machines. *Fundamenta Informaticae*, 18:93–105, 1993.

[78] I. A. Stewart. Logical descriptions of monotone NP problems. *J. of Logic and Computation*, 4:337–357, 1994.

[79] I. A. Stewart. On completeness for NP via projection translations. *Math. Systems Theory*, 27:125–157, 1994.

[80] I. A. Stewart. Completeness of path-problems via logical reductions. *Information and Computation*, 121(1), August 1995.

[81] I. A. Stewart. Program schemes, arrays, Lindström quantifiers and zero-one laws. In J. Flum and M. Rodríguez-Artalejo, editors, *Proceedings of Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 374–388. Springer-Verlag, 1999.

[82] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Comp. Sci.*, 3:1–22, 1977.

[83] A. Szendrei. *Clones in Universal Algebra*, volume 99 of *Seminaires de Mathematiques Superieures*. University of Montreal, 1986.

[84] B. A. Trakhtenbrot. The impossibilty of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR*, 70:569–572, 1950.

[85] P. van Beek and R. Dechter. On the minimality and decomposability of row-convex constraint networks. *Journal of the ACM*, 42:543–561, 1995.