ON THE EXISTENCE OF NORMAL FORMS FOR LOGICS THAT CAPTURE COMPLEXITY CLASSES

By

Argimiro A. Arratia-Quesada

A dissertation submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

(Mathematics)

at the

UNIVERSITY OF WISCONSIN – MADISON

1997

Abstract

We show that the logic HEX*[FO$_s$], which is first-order logic (with successor) extended with a generalized quantifier (HEX) corresponding to the **PSPACE**–complete problem Generalized Hex, has a projective normal form. This gives as a corollary that HEX*[FO$_s$] captures **PSPACE** and that the problem Generalized Hex is complete via first-order projections. We define a variation of HEX, namely WHEX, and prove that as a problem (class of finite structures) it is also complete for **PSPACE** via logspace reducibility. Considering WHEX as a generalized quantifier, we form the logic WHEX*[FO$_s$] and show that it has similar properties as HEX*[FO$_s$]; that is, it has a projective normal form and captures **PSPACE**; hence WHEX is complete for **PSPACE** via first-order projections. Furthermore, we show that WHEX*[FO$_s$] is contained in the infinitary logic $L^{\omega}_{\infty\omega}$.

We then study the existence of normal forms for the logics HEX*[FO] and WHEX*[FO], where the successor relation is not present, and conclude that they have no projective normal forms. In fact, we prove a general theorem that states the non-existence of a projective normal form for various extensions of first-order logic (without successor) of the form Ω^*[FO] when Ω is a generalized quantifier that corresponds to a firmly monotone problem. The tool we employ for obtaining our negative results is Ehrenfeucht-Fraïssé games for logics with generalized quantifiers.

We continue our explorations in the world of unordered structures with a revision of a theorem by Grädel, which states the existence of a hierarchy of logics inside Transitive Closure logic, or TC*[FO]. We translate this result to the framework of program schemes, and obtain a hierarchy of certain classes of program schemes. Our Hierarchy Theorem for program schemes generalizes Grädel's Hierarchy Theorem for TC*[FO], since, as we show, his theorem can be obtained as a

corollary from ours, and similar hierarchies inside other logics of the form Ω^*[FO], where Ω is a generalized quantifier of complexity within **NL**, are also consequences of our theorem.

Acknowledgements

As an undergraduate I found, in a treatise by Blaise Pascal, the following line:

> The best books are those whose readers think they could have written them.[1]

This thought occurred to me back, sometime in the spring of 1994, after I read for the first time *Using the Hamiltonian Path Operator to Capture NP* (J. of Comp. and System Sci., 45, 1992), and so I decided to study Descriptive Complexity and to learn to write mathematics in the same way as the author of that paper: Iain Stewart.

The fulfillment of my goal owes much to Deborah Joseph, who guided my initial steps into the worlds of Computational Complexity and Finite Model Theory (both fundamental for Descriptive Complexity), advised me on the how-to and what-to-look-for in the scientific literature, and, as my understanding of the field of my interest grew, encouraged me to continue and deepen my learning of the subject with the very same author whose work was the source of my original inspiration. In the spring and summer of 1996, I was invited by Iain Stewart to visit him at the University of Wales–Swansea first, and then at the University of Leicester. I am grateful to all the people in both institutions, in particular to Savita Chauhan and Anuj Dawar, for kind support and friendship throughout my stay. It was a delightful and enlightening semester. Most of the ideas presented in this thesis were developed in that period through continuous and intensive meetings with Iain Stewart. All of the ideas in this thesis have his stamp, which I have tried to grasp first from his papers and afterward through my personal contact with him. I am greatly indebted to Iain Stewart for all I know about Descriptive Complexity.

[1] B. Pascal, *On Geometrical Demonstration*, in Great Books of the Western World, vol. 33, Encyclopaedia Britannica

A special word of thanks goes to Terry Millar for proofreading this thesis, and, most of all, for all his out–of–classroom help and support throughout my years in Wisconsin.

Thanks to all the people that read and made comments on previous drafts of this thesis whom I have not mentioned yet: Mirna Dzamonja, Eric Bach, Jerry Keisler, Michael Barnwell, and my dearest friend and colleague Carlos Ortiz.

Warm thanks to Orlando Cabrera for giving me room and board in his house, together with endless nights of stress-releasing sessions of vodka and conversation while I was typing this work. Thanks to Mary Klehr and Allen Cross, Raquel Gonzáles and Francisco López, Sue Byram and Aicardo Roa, for taking care of my belongings while I was overseas, for lending a hand through the writing period, and most of all for their friendship.

My heart goes out to Eurídice, Tita, and Alejandro (sister, mother, and father), and the rest of my family and friends in Caracas for their continued and unconditional support.

My graduate education at Wisconsin was possible thanks, mainly, to a loan that supported me through 1993, made by the Government of Venezuela by means of its non–profit foundation Fundación Gran Mariscal de Ayacucho, which is administered by LASPAU (Latin American Scholarship Program of American Universities). Thanks to all the staff in these institutions for being attentive to my economic and visa affairs.

Heartfelt thanks to the Consejo de Desarrollo Científico y Humanístico of the Universidad Central de Venezuela, my alma mater, for a plane ticket and a greatly appreciated monthly allowance for books.

Thanks to the Computer Sciences Department and the Mathematics Department of the University of Wisconsin for their Teaching Assistantship appointments, which helped me get by from 1993 to the end of my graduate program.

Thanks to the Graduate School of the University of Wisconsin, for granting a fellowship that allowed me the free time needed to write this thesis.

Finally, thanks and apologies to whoever is firmly convinced I should have explicitly mentioned here but have neglected to include due to the failure of my recollections.

List of Figures

Contents

Chapter 1

Introduction

Descriptive Complexity attempts to describe the computational complexity of problems using logical formalism. Its origins date to a result by Fagin from 1974 [Fag74], which says that the complexity class **NP** is "captured" by existential second-order logic (or Σ_1^1). By this it is meant that any problem in the computational complexity class corresponds to a class of models of a sentence of the logic, and any set of structures that satisfies a sentence of the logic corresponds to a problem in the given class. In recent years, logics that capture other known complexity classes have been discovered by different authors.

An interesting way of constructing logics that capture computational complexity classes is as follows. Start with first-order logic with equality and a successor relation (denoted FO_s), and boost its expressive power by adding a formula-building operator, or generalized quantifier, which corresponds to a problem complete under some known Turing–machine–based reducibility (e.g., polynomial-time, logspace), in the computational complexity class that we seek to capture. For example, in the class of directed graphs (digraphs) whose vertices are ordered, let TC be the property "there exists a path in the graph from the vertex indexed 0 (the smallest element in the order) to the vertex indexed max (the largest element in the order)". Then, a formula such as

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

where ψ is a formula in the variables x and y, holds in a finite ordered structure

$\mathcal{A}$, if the graph with set of vertices $|\mathcal{A}|$ and edge relation

$$\{(a,b) \in |\mathcal{A}|^2 \mid \mathcal{A} \models \psi(a,b)\},$$

has the property TC. Note that it is logically reasonable to consider ψ itself to be a formula with the operator TC (e.g., $\psi(x,y) := \mathrm{TC}[\lambda\, v\, u\, \theta](x,y)$); and a further generalization is to consider k-tuples of variables, $\overline{x} := (x_1, \ldots, x_k)$ and $\overline{y} := (y_1, \ldots, y_k)$, for some positive integer k, instead of single variables; the interpretation being that the appropriate graph has k-tuples of elements as its vertices. Let $\mathrm{TC}^*[\mathrm{FO}_s]$ be the set of formulas thus formed (with all possible finite vectorization and nesting of TC). The problem of deciding if a graph has the TC property is logspace complete for **NL**, and it has been shown by Immerman, in [Imm87], that the logic $\mathrm{TC}^*[\mathrm{FO}_s]$ captures **NL**.

The logic $\mathrm{TC}^*[\mathrm{FO}_s]$ has an interesting "collapsing" property: any formula in $\mathrm{TC}^*[\mathrm{FO}_s]$ is equivalent to a formula of the form $\mathrm{TC}[\lambda\, \overline{x}\, \overline{y}\, \psi](\overline{0}, \overline{max})$, where $\overline{x}$ and $\overline{y}$ are k-tuples of distinct variables (for some positive integer k), and ψ is a quantifier-free first-order *projective* formula (basically a formula in disjunctive normal form where each disjunct has a very simple structure). One then says of $\mathrm{TC}^*[\mathrm{FO}_s]$ that it has a *projective normal form*. Other problems of computational complexity in the classes **L**, **P**, and **NP**, have been discovered by various authors to generate logics with similar properties as $\mathrm{TC}^*[\mathrm{FO}_s]$; that is, they capture the corresponding class to which the problem belongs, and satisfy a projective normal form. However, as far as we know, no example that has both of these properties has been found for **PSPACE**. One contribution of this thesis is to give two such examples. We show

Normal Form Theorem: *Every formula* $\phi \in \mathrm{HEX}^*[\mathrm{FO}_s]$ *is equivalent to a formula of the form* $\mathrm{HEX}[\lambda\, \overline{x}\, \overline{y}\, \psi](\overline{0}, \overline{max})$, *where* $\psi \in \mathrm{FO}_s$, ψ *projective and over the distinct* k*-tuples of variables* $\overline{x}$ *and* $\overline{y}$, *for some* $k \geq 1$, *and where* $\overline{0}$ *(resp.* $\overline{max}$*) is the constant symbol* 0 *(resp.* max*) repeated* k *times.*

The logic HEX*[FO$_s$] is obtained by adding to FO$_s$ the generalized quantifier corresponding to the graph problem Generalized Hex (or simply HEX), which was shown to be complete for **PSPACE** via logspace reducibility in [ET76]. We will give the definition of HEX and prove the theorem mentioned above in chapter 3. We will also define in that chapter a variation of the HEX-problem, which we named WHEX. To dispel any doubts that we have indeed found a new problem, we show that WHEX $\neq$ HEX as classes of graphs. Next, since we have never seen the WHEX problem treated in the literature, we prove that WHEX is complete for **PSPACE** via a logspace reduction; thus contributing with another problem to the list compiled in [GJ79]. Then we prove that the logic WHEX*[FO$_s$] also captures **PSPACE** and has a projective normal form.

There are some important consequences of these type of results that are worth mentioning:

- It provides a classification of problems within the same computational complexity class. For example, in [Ste94b], Stewart shows that the extension of FO$_s$ with the operator 3COL, which corresponds to the **NP**–complete problem of deciding if a graph is 3-colorable, cannot have a projective normal form unless **NP** = **coNP**. On the other hand, the same author shows, in [Ste91], that the logic HP*[FO$_s$], which is FO$_s$ with the operator HP corresponding to the **NP**–complete problem of deciding if a graph has a Hamiltonian path, captures **NP** and has a projective normal form. Thus, in terms of expressibility, the logics HP*[FO$_s$] and 3COL*[FO$_s$] are different (relative to the **NP** = **coNP** question).

- It shows that the problem is complete via a reduction weaker than the usual polynomial–time or logspace, namely, *first-order projections (or f.o.p.)*. Since first-order projective formulas are very restrictive, it is not so immediate to conclude that a problem which is complete via polynomial–time or logspace

reductions, is also complete via f.o.p. In fact, it is shown in [ABI93] that there are problems in **NP**, complete via polynomial–time or logspace reductions, which are not complete via f.o.p. This is in contrast with the general belief that all problems complete via polynomial–time reduction are also complete via logspace reduction. Thus, showing that a problem is (is not) complete via f.o.p. may lead to lower bound results within a complexity class (see [Imm87]). (Let us point out that a problem can be complete via f.o.p. but its associated logic may not have a normal form, e.g., 3COL.)

Thus, both HEX and WHEX are complete for **PSPACE** via f.o.p. So far, then, HEX and WHEX are similar with respect to ordered structures. Hence, the next natural question to ask is: are they similar with respect to arbitrary structures? So we remove the successor relation from our set of basic logical symbols and study the expressibility of the logics HEX*[FO] and WHEX*[FO]. (Note that we write FO instead of FO_s to stress that we are working on structures that are not necessarily ordered.) We show that a universally quantified sentence of the form

$$\forall \overline{z}\ \mathrm{HEX}[\lambda\, \overline{x}\, \overline{y}\, \psi(\overline{x}, \overline{y}, \overline{z})](\overline{0}, \overline{max}) \tag{1.1}$$

with $\psi \in \mathrm{FO}$, cannot be expressed as a sentence with a single application of the quantifier HEX to a *successor free* first-order formula. The same happens for WHEX. In fact, we prove a general result that says that it is not possible to simplify an equation like (1.1) when the generalized quantifier is *firmly monotone* (e.g., a graph property is firmly monotone if it is invariant under the addition of new vertices and edges to the graphs with the property). Both HEX and WHEX are firmly monotone, but so also are many problems in the classes **L**, **NL**, **P**, and **NP**; hence, our inexpressibility result applies to problems in these classes as well. We pause to mention a further application of this result: If one can find, for example, some **NP**–complete problem Ω which is firmly monotone and its associated logic

Ω^1[FO] captures **NP** (where Ω^1 denotes that we only allow a single application of the operator Ω), then our inexpressibility result yields that **NP** $\neq$ **PSPACE**, solving an outstanding problem of Complexity Theory.

In order to prove our last theorem we employ a version of Ehrenfeucht-Fraïssé games for logics of the type Ω^*[FO]. These games were defined in [Ste96] for firmly monotone problems, and, although those are sufficient for our applications, we define a variation of these games and prove the corresponding Ehrenfeucht-Fraïssé type of theorem for problems that are not necessarily firmly monotone; thus, giving a small contribution to the toolbox of Descriptive Complexity.

Returning to the question of how similar in expressive power are HEX*[FO] and WHEX*[FO], we study their possible definability in the infinitary logic $L^{\omega}_{\infty\omega}$. We prove that the problem WHEX is definable in $L^{\omega}_{\infty\omega}$ by showing that it is definable in **PFP**, where **PFP** is the Partial Fixed Point logic and it is known that **PFP** $\leq L^{\omega}_{\infty\omega}$. Thus WHEX*[FO] $\leq$ **PFP** $\leq L^{\omega}_{\infty\omega}$. As for HEX, the problem of its definability in $L^{\omega}_{\infty\omega}$ is still open.

Now to the second part of this thesis. Motivated by the inexpressibility result of (1.1), we wondered about the possibility of showing something similar for more complex formulas. The proof of our Normal Form Theorem (Theorem 3.1.1) shows that the only place we need the successor relation, in order to construct the desired projective formula, is where we are trying to simplify universally quantified formulas as (1.1). This suggests that some combination of the quantifier $\forall$ with $\exists$, or the quantifier HEX, might yield more complex formulas that do not simplify to just one application of HEX to a projective formula.

In [Grä91], Grädel exhibits a strict hierarchy of logics inside the logic TC*[FO], which he obtained by interleaving the TC operator with the quantifier $\forall$. We found Grädel's techniques difficult to generalize because they are too intrinsically

related to the operator TC: his main tool is a TC-game, which is an Ehrenfeucht-Fraïssé type of game with the TC property built in it. So we sought a better framework and found it in *program schemes*; in particular, in the class of programs NPS. An NPS program scheme is, essentially, a nondeterministic while program with quantifier-free tests which take as input some finite structure over some fixed vocabulary. It is shown in [Ste88] that, over ordered structures, a program scheme has the same computational power as a nondeterministic Turing machine using logarithmic work-space. That is, NPS($succ$) = **NL**, where NPS($succ$) denotes the class of Nondeterministic Program Schemes with successor as a built-in relation. The advantage of program schemes over the more common model of computation of Turing machines is that they can be best treated as logical formulas, hence being more suitable for the descriptive analysis of computational classes.

Thus, starting with the notion of a program scheme from NPS (without any built-in relation), we define a series of classes of program schemes by allowing first-order formulas and other program schemes as tests. We show that the containment is proper at all levels, using infinitary pebble games to simulate the computations of our program schemes in a combinatorial way. We then show the relation between our classes of program schemes and logics like TC*[FO], and show that our hierarchy of program schemes implies that of Grädel's for TC. But, furthermore, we can also show that there exists hierarchies of logics, similar to Grädel's, inside any logic of the form Ω^*[FO], where Ω is a class of structures accepted by a program scheme of the kind we define. These problems Ω are of a complexity not higher than **NL**; so, our generalization of Grädel's work applies so far to problems in the complexity class **NL** and below.

Chapter 2

Basic Background

The purpose of this chapter is to present the basic concepts and to fix the notation used throughout this thesis. We are doing Descriptive Complexity, which is partly Computational Complexity and mostly Finite Model Theory. The former is a well developed subject with many textbooks written on it (e.g., [Pap94], [HU79]), and so we start by giving only a brief description of some of the important objects of Computational Complexity that are often used in this thesis. Then, we describe in detail the logical concepts. Most of our notations and definitions are based on [Ste94b], [EF95], and [Pap94].

2.1 Computational Complexity

Let $\{0,1\}^*$ be the set of all finite strings over $\{0,1\}$. For $w \in \{0,1\}^*$, $|w|$ denotes the length of the string w. In Computational Complexity, a decision problem is usually considered as a set of strings $S \subseteq \{0,1\}^*$. The typical model of computation is the Turing machine model that takes as inputs elements from $\{0,1\}^*$, operates in modes deterministic (for each instruction there is at most one next instruction) or nondeterministic (for each instruction there could be more than one next instruction), and has bounded resources. The resources that are generally measured are time and space: Given a Turing machine M and a function $f : \mathbb{N} \to \mathbb{R}^+$, we say that M has time bounded by f if for all $w \in \{0,1\}^*$, accepted by M, there is an accepting computation of M that begins with w and has at most $f(|w|)$ steps. M has space bounded by f if for all $w \in \{0,1\}^*$, accepted by M,

there is an accepting computation of M which uses at most $f(|w|)$ cells of its work tape(s).

In order to give a measure of how hard it is to compute a set of strings with respect to another different set of strings, various concepts of reducibility among sets have been introduced. The most common ones are *logspace* and *polynomial time* reducibility: Let S and T be two sets of strings in $\{0,1\}^*$. S is logspace (resp. polynomial time) reducible to T if and only if there exists a function $g : \{0,1\}^* \to \{0,1\}^*$ which is computable by a deterministic Turing machine with logarithmic space (resp. polynomial time) bound, and is such that, for all $w \in \{0,1\}^*$,

$$w \in S \text{ iff } g(w) \in T\,.$$

The complexity classes we will be referring to are:

- **L**, or Logarithmic space (resp. **NL**, or Nondeterministic Logarithmic space), the class of sets $S \subseteq \{0,1\}^*$ such that S is accepted by a deterministic (resp. nondeterministic) Turing machine with space bounded by a logarithmic function (with domain in $\mathbb{N}$).

- **P**, or Polynomial time (resp. **NP**, or Nondeterministic Polynomial time), the class of sets $S \subseteq \{0,1\}^*$ such that S is accepted by a deterministic (resp. nondeterministic) Turing machine with time bounded by a polynomial.

- **PSPACE**, or Polynomial Space (resp. **NPSPACE**, or Nondeterministic Polynomial Space), the class of sets $S \subseteq \{0,1\}^*$ such that S is accepted by a deterministic (resp. nondeterministic) Turing machine with space bounded by a polynomial.

For any complexity class $\mathcal{C}$, co$\mathcal{C}$ denotes its complement. From the definitions it follows that $\mathcal{C} = \text{co}\mathcal{C}$ for any deterministic class $\mathcal{C}$ (e.g., **L**, **P**, and **PSPACE**).

It has been shown that **NL** = **coNL** [Imm88, Sze87]; but it is currently unknown whether **NP** = **coNP** or **NP** $\neq$ **coNP**.

Another well established relation among the complexity classes listed above is the following:

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE}.$$

It has been long known that **L** is a proper subset of **PSPACE** (see [HLS65]); therefore, some of the above four inclusions must be proper. However, it is still unknown whether any of the above inclusions is proper, and this constitute a major open problem in Computational Complexity. It is the hope of many researchers (and ours as well) that by applying the tools of Finite Model Theory to Computational Complexity that, and several other open problems, will be solved.

2.2 Problems, generalized quantifiers, and logics

We are dealing with finite vocabularies and finite structures. Our vocabularies consists of relation and constant symbols (no function symbols), and we reserve the letters τ and σ (with subscripts) to denote them. So, a vocabulary τ has the form $\tau = \{R_1, \ldots, R_r, C_1, \ldots, C_c\}$, where each R_i is a relation symbol of arity a_i and each C_j is a constant symbol ($1 \leq i \leq r$, $1 \leq j \leq c$). A τ-structure is a tuple

$$\mathcal{A} := \langle \{0, \ldots, n-1\}, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, C_1^{\mathcal{A}}, \ldots, C_c^{\mathcal{A}} \rangle$$

consisting of a *universe* $|\mathcal{A}| = \{0, \ldots, n-1\}$; relations $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$, for each relation symbol R_i ($i = 1, \ldots, r$); and constants $C_j^{\mathcal{A}} \in |\mathcal{A}|$, for each constant symbol C_j ($j = 1, \ldots, c$). The *size* of $\mathcal{A}$ is n, and we denote it by $||\mathcal{A}||$. We assume that our structures have universe of size at least 2. This is a standard assumption and it is done to avoid unnecessary work on special cases relating to structures of size 1.

The set of all finite τ-structures is denoted by STRUCT(τ).

A *(computational) problem* Ω, over a vocabulary τ, is a subset of STRUCT(τ) closed under isomorphisms, i.e.,

$$\mathcal{A} \in \Omega \text{ and } \mathcal{A} \cong \mathcal{B} \text{ implies } \mathcal{B} \in \Omega.$$

We will be dealing mostly with the following type of problems:

Definition 2.2.1 (Firmly monotone problem) A problem Ω over the vocabulary τ is *monotone* (resp. *firmly monotone*) if for every pair of τ-structures, $\mathcal{A}$ and $\mathcal{B}$, such that:

(i) $|\mathcal{A}| = |\mathcal{B}|$ (resp. $|\mathcal{A}| \leq |\mathcal{B}|$),

(ii) for every relation symbol $R \in \tau$, $R^{\mathcal{A}} \subseteq R^{\mathcal{B}}$, and

(iii) for every constant symbol $C \in \tau$, $C^{\mathcal{A}} = C^{\mathcal{B}}$,

we have that $\mathcal{A} \in \Omega$ implies $\mathcal{B} \in \Omega$. ∎

For example, a graph problem, that is, a class of structures over the vocabulary $\tau_2 = \{E\}$ where E is a binary relation symbol, is monotone if the property that characterizes the problem is invariant under the addition of edges (to any graph in the class), and it is firmly monotone if the property is invariant under the addition of edges and vertices.

For any vocabulary τ, $\mathrm{FO}_s(\tau)$ denotes first-order logic over the vocabulary τ. This is the set of well-formed formulae, built using the symbols of τ, the binary relation symbols s and $=$, the constant symbols 0 and max, the usual logical connectives $\wedge$, $\vee$ and $\neg$, quantifiers $\forall$ and $\exists$, and variables x, y, z, ..., etc. We assume that $=$, s, 0, and max don't occur in any vocabulary. We denote tuples of variables $x_1, \ldots, x_k$ as $\overline{x}$, and its length k as $|\overline{x}|$ which we will always try to make clear from context.

For a formula $\phi(\overline{x}) \in \mathrm{FO}_s(\tau)$ in the free variables $\overline{x} = x_1, \ldots, x_k$, a τ-structure $\mathcal{A}$ of size n, and $\overline{a} \in |\mathcal{A}|^k$, the interpretation of ϕ in $\langle \mathcal{A}, \overline{a} \rangle$ is denoted by $\phi^{\mathcal{A}}(\overline{a})$ and is defined as usual, except that the binary relation symbol $=$ is interpreted as equality, the binary relation symbol s is interpreted as the successor relation on $|\mathcal{A}|$, the constant symbol 0 is interpreted as $0 \in |\mathcal{A}|$, and the constant symbol max is interpreted as $n-1 \in |\mathcal{A}|$.

$\mathrm{FO}_s = \bigcup\{\mathrm{FO}_s(\tau) \mid \tau \text{ some vocabulary }\}$, and FO is the sublogic of FO_s in which the successor relation s is absent.

Definition 2.2.2 (First order translations) Let $\tau = \{R_1, \ldots, R_r, C_1, \ldots, C_c\}$, where each R_i is a relation symbol of arity a_i and each C_j is a constant symbol. Let σ be some other vocabulary, and let k be a positive integer. A k-ary τ-translation of $\mathrm{STRUCT}(\sigma)$ to $\mathrm{STRUCT}(\tau)$ is determined by k and a set Σ of σ-formulas, $\{\phi_1(\overline{x}_1, \overline{z}), \ldots, \phi_r(\overline{x}_r, \overline{z}), \psi_1(\overline{y}_1), \ldots, \psi_c(\overline{y}_c)\}$, where each ϕ_i is over the ka_i distinct variables $\overline{x}_i$ plus possibly other free variables from some tuple $\overline{z}$, and each ψ_j is over the k distinct variables $\overline{y}_j$, and is such that for $\mathcal{A} \in \mathrm{STRUCT}(\sigma)$ its k-ary τ-translation with respect to Σ is the τ-structure $\mathcal{A}_\Sigma$ with universe $|\mathcal{A}|^k$; relations

$$R_i^{\mathcal{A}_\Sigma} = \{\overline{u} \in |\mathcal{A}|^{ka_i} \mid \langle \mathcal{A}, \overline{u}, \overline{v} \rangle \models \phi_i(\overline{x}_i, \overline{z})\},$$

where $\overline{v}$ is some tuple in $|\mathcal{A}|^{|\overline{z}|}$; and constants determined by

$$C_j^{\mathcal{A}_\Sigma} = \overline{u} \text{ iff } \mathcal{A} \models \psi_j(\overline{u}), \text{ for } \overline{u} \in |\mathcal{A}_\Sigma|.$$

The successor relation on $\mathcal{A}_\Sigma$ is defined using the lexicographical ordering on k-tuples. Thus, $\overline{0}^{\mathcal{A}_\Sigma} = (0^{\mathcal{A}}, \ldots, 0^{\mathcal{A}})$ and $\overline{max}^{\mathcal{A}_\Sigma} = (max^{\mathcal{A}}, \ldots, max^{\mathcal{A}})$. We called the elements of Σ *τ-descriptive formulas of arity k*; we use sometimes the shorthand $\phi_i^{\mathcal{A}}$ for $R_i^{\mathcal{A}_\Sigma}$, and $\psi_j^{\mathcal{A}}$ for $C_j^{\mathcal{A}_\Sigma}$. ∎

Example 2.2.1 Let σ_0 be the empty vocabulary; $\tau_2 = \{E\}$ where E is a binary relation symbol, and consider the following σ_0-formula:

$$\begin{aligned}\phi((x_1,y_1),(x_2,y_2)) \;\; := \;\; & s(x_1,x_2) \wedge [(y_1 = 0 \wedge y_2 = max) \\ & \vee \; (y_1 = max \wedge y_2 = 0)].\end{aligned}$$

For any σ_0-structure $\mathcal{A}$ (i.e., a set $\mathcal{A} = \langle |\mathcal{A}| \rangle$), its 2-ary τ_2-translation with respect to $\Sigma = \{\phi\}$ is a graph, $\mathcal{A}_\Sigma$, with set of vertices $|\mathcal{A}_\Sigma| = |\mathcal{A}| \times |\mathcal{A}|$ and edge relation

$$\begin{aligned}E^{\mathcal{A}_\Sigma} \;\; = \;\; & \{((a_1,a_2),(b_1,b_2)) \in |\mathcal{A}_\Sigma|^2 \mid \\ & \langle \mathcal{A},(a_1,a_2),(b_1,b_2)\rangle \models \phi((x_1,y_1),(x_2,y_2))\}.\end{aligned}$$

■

We now describe the kind of logics that we work with in this thesis. These are extensions of FO_s (or FO) by means of *generalized quantifiers*. The idea of augmenting the expressive power of first-order logic in this manner goes back to a paper by Lindström [Lin66].

Definition 2.2.3 (The logic $\pm\Omega^*[\mathbf{FO}_s]$) Let τ and σ be as in the preceding definition, and let Ω be a problem over τ. The *first-order extension of* $\mathrm{FO}_s(\sigma)$ *by* Ω, denoted $\pm\Omega^*[\mathrm{FO}_s(\sigma)]$, is the smallest set of formulas such that:

(i) $\mathrm{FO}_s(\sigma) \subseteq \pm\Omega^*[\mathrm{FO}_s]$;

(ii) if ψ, $\phi \in \pm\Omega^*[\mathrm{FO}_s]$ then $\psi \vee \phi$, $\neg\psi$, and $\exists z\psi$ are also in $\pm\Omega^*[\mathrm{FO}_s]$. The interpretation of each one of these formulas in a σ-structure $\mathcal{A}$ is as follows:

Assume that the free variables of ψ and ϕ are among the variables in the tuple $\overline{x}$. If $\overline{u} \in |\mathcal{A}|^{|\overline{x}|}$ then

$\langle \mathcal{A},\overline{u}\rangle \models (\psi \vee \phi)(\overline{x})$ iff $\langle \mathcal{A},\overline{u}\rangle \models \psi(\overline{x})$ or $\langle \mathcal{A},\overline{u}\rangle \models \phi(\overline{x})$;

$\langle \mathcal{A},\overline{u}\rangle \models \neg\psi(\overline{x})$ iff it is not the case that $\langle \mathcal{A},\overline{u}\rangle \models \psi(\overline{x})$;

if z appears in $\overline{x}$ then

$\langle \mathcal{A}, \overline{u} \rangle \models \exists z \psi(\overline{x})$ iff $\langle \mathcal{A}, \overline{u}', v \rangle \models \psi(\overline{x})$ for some $v \in |\mathcal{A}|$ such that v is in the tuple $\overline{u}$, v is the value assign to z, and $\overline{u}'$ is $\overline{u}$ with v removed;

if z does not appear in $\overline{x}$, then

$\langle \mathcal{A}, \overline{u} \rangle \models \exists z \psi(\overline{x})$ iff $\langle \mathcal{A}, \overline{u} \rangle \models \psi(\overline{x})$;

(iii) if $\Sigma = \{\phi_1, \ldots, \phi_r, \psi_1, \ldots, \psi_c\} \subseteq \pm\Omega^*[\mathrm{FO}_s(\sigma)]$ is a set of τ-descriptive formulas of arity k and t is some positive integer, then

$$\Phi(\overline{x}) := \Omega[\lambda\, \overline{x}_1 \phi_1, \ldots, \overline{x}_r \phi_r, \overline{y}_1 \psi_1, \ldots, \overline{y}_c \psi_c](\overline{z}_1, \ldots, \overline{z}_t)$$

is a formula in $\pm\Omega^*[\mathrm{FO}_s(\sigma)]$, where each $\overline{z}_h$ is a tuple of variables or constant symbols, and the variables of each $\overline{z}_h$ do not occur in any formula ϕ_i or ψ_j; furthermore, the variables of each tuple $\overline{x}_i$ and $\overline{y}_j$ are bound in Φ (as indicated by the symbol λ), and hence, the free variables of Φ (represented by the tuple $\overline{x}$) are the variables of each $\overline{z}_h$ and the variables of each ϕ_i distinct from $\overline{x}_i$.

The interpretation of Φ is as follows: If $\mathcal{A} \in \mathrm{STRUCT}(\sigma)$ and $\overline{u}$ is a tuple of elements of $|\mathcal{A}|$ such that $|\overline{u}| = |\overline{x}|$, then $\langle \mathcal{A}, \overline{u} \rangle \models \Phi(\overline{x})$ if and only if the k-ary τ-translation of $\mathcal{A}$ with respect to Σ, that is, the structure $\mathcal{A}_\Sigma \in \mathrm{STRUCT}(\tau)$, is such that $\langle \mathcal{A}_\Sigma, \overline{u}_1, \ldots, \overline{u}_t \rangle \in \Omega$, where $|\overline{u}_i| = |\overline{z}_i|$ for $i = 1, \ldots, t$ and takes values from $\overline{u}$.

We also use the connective $\wedge$ and the quantifier $\forall$, which can be regarded as abbreviations: $\psi \wedge \phi$ stands for $\neg(\neg\psi \vee \neg\phi)$, and $\forall z \psi$ stands for $\neg\exists z \neg\psi$.

$\pm\Omega^*[\mathrm{FO}_s] = \bigcup\{\pm\Omega^*[\mathrm{FO}_s(\sigma)] \mid \sigma \text{ some vocabulary } \}$.

Let $\Omega^*[\mathrm{FO}_s]$ denote the sublogic of $\pm\Omega^*[\mathrm{FO}_s]$ where only positive applications of the quantifier Ω are allowed, i.e., Ω does not appear within the scope of an odd number of negation signs in any formula.

Let $\pm\Omega^k[\mathrm{FO}_s]$ (resp. $\Omega^k[\mathrm{FO}_s]$) denote the sublogic of $\pm\Omega^*[\mathrm{FO}_s]$ (resp. $\Omega^*[\mathrm{FO}_s]$) where at most k applications of the quantifier Ω are nested.

Let $\pm\Omega^*[\mathrm{FO}]$ denote the sublogic of $\pm\Omega^*[\mathrm{FO}_s]$ where the successor relation is absent, and define similar sublogics to this one as above. ∎

Notice that following common practice we employ the same symbol Ω to denote a problem and the quantifier corresponding to that problem. From now on, whenever we write the word logic we mean FO_s, FO, or some set of formulas constructed as in Definition 2.2.3. We use the letter $\mathcal{L}$ to denote any of these logics, and $\mathcal{L}(\tau)$ to specify that these $\mathcal{L}$-formulas are over the vocabulary τ. Therefore, $\mathcal{L} = \bigcup\{\mathcal{L}(\tau) \mid \tau \text{ some vocabulary}\}$.

Example 2.2.2 Let $\tau_2 = \{E\}$ where E is a binary relation symbol. $\pm\mathrm{TC}^*[\mathrm{FO}_s]$ is known as *Transitive Closure logic*, and is the extension of FO_s with the generalized quantifier TC corresponding to the problem, over the vocabulary τ_2,

$$\begin{aligned}\mathrm{TC} = \{ \langle \mathcal{A}, u, v\rangle \in \mathrm{STRUCT}(\tau_2) \mid &\text{ there is a path in the} \\ &\text{digraph } \mathcal{A} \text{ from vertex } u \text{ to vertex } v\}.\end{aligned}$$

A related logic is $\pm\mathrm{DTC}^*[\mathrm{FO}_s]$, or *Deterministic Transitive Closure logic*, which is obtained by extending FO_s with the generalized quantifier DTC corresponding to the problem

$$\begin{aligned}\mathrm{DTC} = \{ \langle \mathcal{A}, u, v\rangle \in \mathrm{STRUCT}(\tau_2) \mid &\text{ there is a path in the digraph } \mathcal{A} \\ &\text{from vertex } u \text{ to vertex } v \text{ such that each vertex on} \\ &\text{the path, except for possibly vertex } v, \text{ has out-degree 1 }\}.\end{aligned}$$

∎

We will be interested in counting the number of quantifiers occurring in a formula.

Definition 2.2.4 The quantifier rank of a formula θ in $\pm\Omega^*[\mathrm{FO}_s]$ is denoted by $qr(\theta)$ and is defined, by induction on the complexity of θ, as follows:

$$qr(\theta) = \begin{cases} 0 & \text{if } \theta \text{ is atomic} \\ qr(\phi) & \text{if } \theta := \neg\phi \\ max[qr(\phi),\ qr(\psi)] & \text{if } \theta := \phi \vee \psi \\ qr(\phi)+1 & \text{if } \theta := \exists x\phi \end{cases}$$

and if $\theta := \Omega[\lambda \overline{x}_1\phi_1, \ldots, \overline{x}_r\phi_r, \overline{y}_1\psi_1, \ldots, \overline{y}_c\psi_c](\overline{z})$ where, for each $i = 1, \ldots, r$, $|\overline{x}_i| = ka_i$, for each $j = 1, \ldots, c$, $|\overline{y}_j| = k$, and $\overline{z}$ is a tuple of variables or constant symbols, then

$$qr(\theta) := max[ka_i + qr(\phi_i),\ k + qr(\psi_j)\ :\ 1 \leq i \leq r\ ;\ 1 \leq j \leq c].$$

■

For any sentence ϕ, over some vocabulary τ, $MOD(\phi)$ denotes the class of finite models of ϕ, that is,

$$MOD(\phi) = \{\mathcal{A} \in \mathrm{STRUCT}(\tau) \mid \mathcal{A} \models \phi\}.$$

2.3 Descriptive Complexity meets Computational Complexity

In the context of Descriptive Complexity a computational problem is, as seen in the previous section, a set of finite structures and not a set of strings over some finite set of symbols, as is usually considered in Computational Complexity. However, one of the goals of Descriptive Complexity is to construct logics that syntactically characterize the usual Turing–machine–based notion of a computational complexity class. For that we need a bridge that would allow us to go back and forth between these two different ways of expressing the same thing.

Let $\tau = \{R_1, \ldots, R_r, C_1, \ldots, C_s\}$ be some vocabulary where each R_i is a relation symbol of arity a_i and each C_j is a constant symbol. Let $\mathcal{A}$ be a τ-structure of size n. The encoding of $\mathcal{A}$, $e_\tau(\mathcal{A})$, over the set $\{0,1\}$ is defined as follows.

For each $i = 1,\ldots,r$ let w_i be the string in $\{0,1\}^*$ of length n^{a_i} such that if $(b_1,\ldots,b_{a_i})$ is the k-th member of $|\mathcal{A}|^{a_i}$ in lexicographical order, then the k-th digit of w_i is 1 if $R_i^{\mathcal{A}}(b_1,\ldots,b_{a_i})$ holds; 0 otherwise, for all $1 \leq k \leq n^{a_i}$.

Let c_j be the binary representation of $C_j^{\mathcal{A}}$ for $j = 1,\ldots,s$.

Then, $e_\tau(\mathcal{A})$ is the string formed by concatenating $c_1, \ldots, c_s, w_1, \ldots, w_r$, as follows:

$$e_\tau(\mathcal{A}) := c_1 \cdots c_s w_1 \cdots w_r.$$

If Ω is some problem over τ, then $e_\tau(\Omega) := \{e_\tau(\mathcal{A}) \mid \mathcal{A} \in \Omega\} \subseteq \{0,1\}^*$. This is the set of strings over $\{0,1\}$ corresponding to the problem Ω.

Conversely, given a set of strings $S \in \{0,1\}^*$, there might be many problems Ω, over different vocabularies τ, such that $e_\tau(\Omega) = S$; there is at least one: to each $w \in S$ associate the structure $\mathcal{A}_w = \langle\{1, \ldots, |w|\}, U^{\mathcal{A}_w}\rangle$ over the vocabulary $\sigma = \{U\}$, where U is a unary relation symbol and is such that

$$U^{\mathcal{A}_w}(i) \text{ holds iff the } i\text{-th letter of } w \text{ is a 1.}$$

Let Ω be the problem over σ consisting of the structures $\mathcal{A}_w$, for each $w \in S$. Then $e_\sigma(\Omega) = S$.

Definition 2.3.1 A complexity class $\mathcal{C}$ is captured by a logic $\mathcal{L}$ if and only if

(i) for each set of strings $S \in \mathcal{C}$, there is some vocabulary τ and a sentence ϕ in $\mathcal{L}(\tau)$, such that $S = e_\tau(MOD(\phi))$, and

(ii) for each vocabulary τ and each sentence ϕ in $\mathcal{L}(\tau)$, $e_\tau(MOD(\phi)) \in \mathcal{C}$.

If class $\mathcal{C}$ is captured by logic $\mathcal{L}$ (or $\mathcal{L}$ captures $\mathcal{C}$), we write $\mathcal{C} = \mathcal{L}$. Moreover, $\mathcal{C} \leq \mathcal{L}$ denotes that item (i) above is satisfied, and $\mathcal{L} \leq \mathcal{C}$ denotes that item (ii) is satisfied. So, $\mathcal{C} = \mathcal{L}$ if and only if $\mathcal{C} \leq \mathcal{L}$ and $\mathcal{L} \leq \mathcal{C}$. ∎

We also use the symbol $=$ instead of the more common $\equiv$ to denote equivalence of logics. Thus, if $\mathcal{L}_1$ and $\mathcal{L}_2$ are two logics then $\mathcal{L}_1 = \mathcal{L}_2$ means that, for all vocabulary τ, every $\mathcal{L}_1(\tau)$-sentence ϕ_1 is equivalent to an $\mathcal{L}_2(\tau)$-sentence ϕ_2 (i.e., $\models \phi_1 \iff \phi_2$), and vice-versa.

We close this section with an important result, due to Immerman, concerning the logics $\pm\mathrm{TC}^*[\mathrm{FO}_s]$ and $\pm\mathrm{DTC}^*[\mathrm{FO}_s]$ presented in Example 2.2.2.

Theorem 2.3.2 [Imm87, Imm88].

(i) $\pm\mathrm{TC}^*[\mathrm{FO}_s] = \mathrm{TC}^*[\mathrm{FO}_s] = \mathrm{TC}^1[\mathrm{FO}_s] = \mathbf{NL}$.

(ii) $\pm\mathrm{DTC}^*[\mathrm{FO}_s] = \mathrm{DTC}^*[\mathrm{FO}_s] = \mathrm{DTC}^1[\mathrm{FO}_s] = \mathbf{L}$.

∎

We remark that the order, given by the underlying built-in relation s, is crucial for the above result to hold. For example, it is shown in [Grä91] that without order the second equality in (i) is not true. The question of whether results in Descriptive Complexity that hold for ordered structures also hold for arbitrary structures (ordered and unordered) is an important one. In this thesis we address this question with respect to a property that we will introduce in the next section. Due to the difference in results, according if the successor relation is present or not, we make the following proviso: unless stated otherwise we are always working on ordered structures.

2.4 Normal forms, completeness, and capturing

A central notion in Descriptive Complexity is that of a logical reduction.

Definition 2.4.1 Given two problems, $\Omega_1 \subseteq \text{STRUCT}(\sigma)$ and $\Omega_2 \subseteq \text{STRUCT}(\tau)$, and a logic $\mathcal{L}$, it is said that Ω_1 *$\mathcal{L}$-reduces to* Ω_2 (and denoted $\Omega_1 \leq_{\mathcal{L}} \Omega_2$) if there exists a $k > 0$ and a set of τ-descriptive formulas $\Sigma \subseteq \mathcal{L}(\sigma)$, that translates σ-structures into τ-structures, and is such that, for all $\mathcal{A} \in \text{STRUCT}(\sigma)$:

$$\mathcal{A} \in \Omega_1 \text{ iff } \mathcal{A}_\Sigma \in \Omega_2.$$

■

As in Computational Complexity, we are interested in finding the weakest possible logical reduction among two problems.

Definition 2.4.2 Let Ω_1 and Ω_2 be as in the previous definition, and suppose Ω_1 FO_s-reduces to Ω_2 via the set of τ-descriptive formulas Σ. If each formula ϕ in Σ satisfy:

$$\phi := \bigvee \{\alpha_l \wedge \beta_l \mid l \in I\}$$

for some finite index set I, where

(i) each α_l is a conjunction of the logical atomic relations (e.g., s, $=$), and their negations;

(ii) each β_l is atomic or negated atomic;

(iii) if $i \neq j$ then α_i and α_j are mutually exclusive,

then Σ is a *first-order projection* (abbreviated f.o.p.), and each ϕ is called a *projective* formula. If each β_l is atomic, then Σ is a first-order *monotone* projection, and each ϕ is then called a *monotone projective* formula. ■

Example 2.4.1 Let σ_0 be the empty vocabulary, and consider the following problem contained in STRUCT(σ_0):

$$\text{EVEN} := \{\mathcal{A} \in \text{STRUCT}(\sigma_0) \mid ||\mathcal{A}|| \text{ is even } \}.$$

Let $\phi := \phi((x_1, y_1), (x_2, y_2))$ be the formula defined in Example 2.2.1, let $\Sigma = \{\phi\}$, and let TC be the problem Transitive Closure defined in Example 2.2.2. Then, for all $\mathcal{A} \in \text{STRUCT}(\sigma_0)$,

$$\mathcal{A} \in \text{EVEN iff } \langle \mathcal{A}_\Sigma, (0,0), (max, max) \rangle \in \text{TC}.$$

Note that ϕ is a first-order projective formula; therefore, the problem EVEN reduces to the problem TC via f.o.p. ∎

Now, suppose $\Omega_1 \leq_{\mathcal{L}} \Omega_2$ via the set $\Sigma = \{\phi_1(\overline{x}_1), \ldots, \phi_r(\overline{x}_r), \psi_1(\overline{y}_1), \ldots, \psi_c(\overline{y}_c)\} \subseteq \mathcal{L}(\sigma)$, with $|\overline{x}_i| = ka_i$ and $|\overline{y}_j| = k$. Let $\mathcal{A} \in \text{STRUCT}(\sigma)$. Then

$$\begin{aligned} \mathcal{A} \in \Omega_1 \quad &\text{iff} \quad \mathcal{A}_\Sigma = \langle |\mathcal{A}|^k, \phi_1^{\mathcal{A}}, \ldots, \phi_r^{\mathcal{A}}, \psi_1^{\mathcal{A}}, \ldots, \psi_c^{\mathcal{A}} \rangle \in \Omega_2 \\ &\text{iff} \quad \mathcal{A} \models \Omega_2[\lambda\, \overline{x}_1\phi_1, \ldots, \overline{x}_r\phi_r, \overline{y}_1\psi_1, \ldots, \overline{y}_c\psi_c]. \end{aligned}$$

This proves the following useful relation between logical reducibility and expressibility in the logic $\pm\Omega^*[\text{FO}_s]$:

Proposition 2.4.3 *Let $\mathcal{L}$ be a logic, $\Omega_1 \subseteq \text{STRUCT}(\sigma)$, and $\Omega_2 \subseteq \text{STRUCT}(\tau)$. Then $\Omega_1 \leq_{\mathcal{L}} \Omega_2$ if and only if $\Omega_1 = MOD(\Phi)$ for some sentence $\Phi \in \Omega_2^1[\mathcal{L}(\sigma)]$.* ∎

An immediate consequence of Proposition 2.4.3 is that if $\mathcal{L}_1$ and $\mathcal{L}_2$ are two logics, such that $\mathcal{L}_1 \leq \mathcal{L}_2$ and $\Omega_1 \leq_{\mathcal{L}_1} \Omega_2$, then $\Omega_1 \leq_{\mathcal{L}_2} \Omega_2$. We use this fact later.

The following important result that links the usual Turing–machine–based reducibility and the logical reducibility has been proven in [Ste91] (cf. [EF95, Proposition 10.3.22]).

Proposition 2.4.4 *Let* $\Omega_1 \subseteq \mathrm{STRUCT}(\sigma)$ *and* $\Omega_2 \subseteq \mathrm{STRUCT}(\tau)$ *be problems. Then* $e_\sigma(\Omega_1)$ *is logspace reducible to* $e_\tau(\Omega_2)$ *if and only if* $\Omega_1 \leq_{DTC^1[FO_s]} \Omega_2$. ∎

We now show our main tool for capturing complexity classes with our logics. Our result is a generalization of Corollary 3.1 of [Ste91].

Corollary 2.4.5 *Let* $\mathcal{C}$ *be a complexity class above* **L** *and closed under logspace reducibility. Let* $\Omega \subseteq \mathrm{STRUCT}(\tau)$ *be a problem such that*

(i) $\mathrm{DTC}^1[\mathrm{FO}_s] \leq \Omega^*[\mathrm{FO}_s]$, *and*

(ii) $e_\tau(\Omega)$ *is in* $\mathcal{C}$ *and is complete via logspace reducibility.*

Then,

$$\Omega^1[\mathrm{FO}_s] \leq \mathcal{C} \leq \Omega^*[\mathrm{FO}_s].$$

Proof: $\Omega^1[\mathrm{FO}_s] \leq \mathcal{C}$: Let $\Phi \in \Omega^1[\mathrm{FO}_s(\sigma)]$ for some vocabulary σ. By Proposition 2.4.3 $MOD(\Phi) \leq_{FO_s} \Omega$. By Proposition 2.4.4 and the observation following Proposition 2.4.3, $e_\sigma(MOD(\Phi))$ is logspace reducible to $e_\tau(\Omega)$, and since $\mathcal{C}$ is closed under logspace reducibility $e_\sigma(MOD(\Phi)) \in \mathcal{C}$.
$\mathcal{C} \leq \Omega^*[\mathrm{FO}_s]$: Let $S \in \mathcal{C}$ be some set of strings. By hypothesis (ii), S is logspace reducible to $e_\tau(\Omega)$. On the other hand, there is some vocabulary σ and a problem $\Omega' \subseteq \mathrm{STRUCT}(\sigma)$ such that $S = e_\sigma(\Omega')$. By Proposition 2.4.4 $\Omega' \leq_{DTC^1[FO_s]} \Omega$. By Proposition 2.4.3 $\Omega' = MOD(\Phi')$ for some sentence $\Phi' \in \Omega^1[\mathrm{DTC}^1[\mathrm{FO}_s]]$. By hypothesis (i), $\Omega^1[\mathrm{DTC}^1[\mathrm{FO}_s]] \leq \Omega^*[\mathrm{FO}_s]$, therefore, there is a $\Phi \in \Omega^*[\mathrm{FO}_s]$ equivalent to Φ'. Hence, $\Omega' = MOD(\Phi)$ with $\Phi \in \Omega^*[\mathrm{FO}_s]$. ∎

Corollary 2.4.5 above gives an easy way to sandwich a complexity class $\mathcal{C}$ between the logic $\Omega^*[\mathrm{FO}_s]$ and its sublogic $\Omega^1[\mathrm{FO}_s]$. For $\Omega^*[\mathrm{FO}_s]$ to capture $\mathcal{C}$ then, all we have to show is that it collapses to $\Omega^1[\mathrm{FO}_s]$; in other words, that the repeated application of the quantifier Ω adds no extra expressive power to $\Omega^1[\mathrm{FO}_s]$,

and hence, $\Omega^*[\mathrm{FO}_s] = \Omega^1[\mathrm{FO}_s]$. If this happens, we say of the logic $\Omega^*[\mathrm{FO}_s]$ that it has a *normal form*. This concept is the central subject of study in this thesis, and, hence, it deserves to be given as a definition.

Definition 2.4.6 Let Ω be a problem over the vocabulary $\tau = \{R_1, \ldots, R_r, C_1, \ldots, C_c\}$, where each R_i is a relation symbol of arity a_i and each C_j is a constant symbol. The logic $\Omega^*[\mathrm{FO}_s]$ has a *normal form* if, for every vocabulary σ, every sentence $\Phi \in \Omega^*[\mathrm{FO}_s(\sigma)]$ is equivalent to a sentence of the form $\Omega[\lambda\, \overline{x}_1\, \phi_1, \ldots, \overline{x}_r\, \phi_r, \overline{y}_1\, \psi_1, \ldots, \overline{y}_c\, \psi_c](\overline{C})$, with $|\overline{x}_i| = ka_i$, $|\overline{y}_j| = k$ (for some $k > 0$), $\phi_i, \psi_j \in \mathrm{FO}_s(\sigma)$, and $\overline{C}$ some tuple of constant symbols (possibly empty). If, in addition, each ϕ_i and each ψ_j $(1 \leq i \leq r,\ 1 \leq j \leq c)$ is a projective formula then we have a *projective normal form* for $\Omega^*[\mathrm{FO}_s]$. We can define, in a similar manner, a normal form property for logics as $\pm\Omega^*[\mathrm{FO}_s]$, $\pm\Omega^*[\mathrm{FO}]$, and $\Omega^*[\mathrm{FO}]$. ∎

We have already given examples of logics with the property described above: $\pm\mathrm{TC}^*[\mathrm{FO}_s]$ and $\pm\mathrm{DTC}^*[\mathrm{FO}_s]$ have normal forms and, in fact, Immerman showed that these are projective. For another set of examples consider the vocabularies $\tau_2 = \{E\}$, $\tau = \{E, C_1, C_2\}$, and $\tau_3 = \{R, C_1, C_2\}$, where E is a binary relation symbol, R is a ternary relation symbol, C_1 and C_2 are constant symbols. Define the following problems:

PS $= \{\mathcal{A} \in \mathrm{STRUCT}(\tau_3) \mid \mathcal{A}$ is a path system where the sink $C_2^{\mathcal{A}}$ is accessible from the source $C_1^{\mathcal{A}}\}$.

HP $= \{\mathcal{A} \in \mathrm{STRUCT}(\tau) \mid$ there is a Hamiltonian path in the digraph $\mathcal{A}$ from vertex $C_1^{\mathcal{A}}$ to vertex $C_2^{\mathcal{A}}\}$.

3COL $= \{\mathcal{A} \in \mathrm{STRUCT}(\tau_2) \mid$ the graph $\mathcal{A}$ is 3-colorable $\}$.

(A path system is a set of vertices V, a relation $R \subseteq V \times V \times V$, a source $s \in V$, and a sink $t \in V$, and a vertex is accessible if it is the source s, or if x and y

are accessible (with possibly $x = y$) and $R(x, y, z)$ holds then z is accessible. A Hamiltonian path is a path that touches all the vertices of the graph only once. A graph is 3-colorable if all its vertices can be colored with 3 colors such that no two adjacent vertices have the same color.)

Theorem 2.4.7 [Ste91, Ste92, Ste94a].

(i) $\mathbf{P} = \pm\mathrm{PS}^*[\mathrm{FO}_s] = \mathrm{PS}^*[\mathrm{FO}_s] = \mathrm{PS}^1[\mathrm{FO}_s]$ *and the normal form is projective.*

(ii) $\mathbf{NP} = \mathrm{HP}^*[\mathrm{FO}_s] = \mathrm{HP}^1[\mathrm{FO}_s]$ *and the normal form is projective.*

(iii) $\pm\mathrm{3COL}^*[\mathrm{FO}_s] = \mathrm{3COL}^*[\mathrm{FO}_s]$ *and if* $\mathrm{3COL}^*[\mathrm{FO}_s] = \mathrm{3COL}^1[\mathrm{FO}_s]$ *then* $\mathbf{NP} = \mathbf{coNP}$.

(iv) $\mathbf{NP} = \mathrm{3COL}^1[\mathrm{FO}_s]$ *and* 3COL *is complete for* **NP** *via f.o.p.*

■

Parts (ii) and (iii) of Theorem 2.4.7 are the results we alluded to in the introduction: we have two well–known problems, complete for **NP** via logspace reducibility [GJ79], which, as generalized quantifiers, produce extensions of FO_s of different expressive power (assuming $\mathbf{NP} \neq \mathbf{coNP}$). Also, part (iv) tell us that a problem may be complete via f.o.p. even though the logic, obtained by extending FO_s with a generalized quantifier corresponding to that problem, may not have a normal form. However, knowing that a projective normal form exists for the logic $\Omega^*[\mathrm{FO}_s]$ yields that the problem Ω is complete via f.o.p., which is an important fact as explained earlier. Thus, DTC, TC, PS, and HP are problems complete via f.o.p. for the classes **L**, **NL**, **P**, and **NP**, respectively.

We have given examples of logics $\Omega^*[\mathrm{FO}_s]$ with the normal form property for problems Ω within all the important complexity classes but **PSPACE**. The reason being that, as far as we know, there are no such examples. One contribution of this thesis is to fill that gap by giving two examples in the next chapter.

Chapter 3

Normal Forms for Two Logics that Capture PSPACE

3.1 The HEX logic

The Generalized Hex problem (HEX). An instance of this problem is an undirected graph $G = (V, E)$ together with a source $s \in V$ and a sink $t \in V$, and we want to know if Player 1 has a winning strategy in the following game played on G: two players, Player 1 and Player 2, alternate choosing a vertex from $V - \{s, t\}$ with Player 1 making the first move. Player 1 colors his chosen vertices blue and Player 2 colors his red, and the game ends when all the vertices (except s and t) have been colored. Player 1 wins if and only if there is a path from s to t in G that passes only through blue vertices. We refer to this game as the HEX-game on (G, s, t), and we refer to the Generalized Hex problem as the HEX problem, for short. Thus, the HEX problem is to determine if Player 1 has a winning strategy in the HEX-game on (G, s, t). Note that a winning strategy for a player is a sequence of moves that makes the player win no matter how the opponent plays. The HEX problem is known to be complete for **PSPACE** via logspace reduction [ET76], [GJ79].

Let E be a binary relation symbol and let $\tau_2 = \{E\}$. We can think of structures over τ_2 as undirected graphs as well as directed graphs (digraphs). Since we are interested in undirected graphs we treat $E(x, y)$ as a symmetric relation; that is,

$E(x,y) \longleftrightarrow E(y,x)$ holds for all x and y. We encode HEX as a class of structures over the vocabulary τ_2 as follows:

$$\begin{aligned}\text{HEX} = \{\, \langle \mathcal{A}, u, v\rangle \in \text{STRUCT}(\tau_2) \mid & \text{ Player 1 has a winning strategy} \\ & \text{for the HEX-game played on } (\mathcal{A}, u, v), \text{ with } u \text{ and } v \text{ in } |\mathcal{A}| \,\}.\end{aligned}$$

Remark 3.1.1 We could have encoded HEX over the vocabulary $\tau := \tau_2 \cup \{C_1, C_2\}$, where C_1 and C_2 are two constant symbols, as the class of structures

$$\begin{aligned}\text{HEX}(C_1, C_2) = \{\mathcal{A} \in \text{STRUCT}(\tau) \mid & \text{ Player 1 has a winning strategy} \\ & \text{for the HEX-game played on } (\mathcal{A}, C_1^{\mathcal{A}}, C_2^{\mathcal{A}}) \,\}.\end{aligned}$$

We think of this as a *localization* of the problem HEX to the constants C_1 and C_2, and although it might be a more precise way of encoding HEX, it adds the extra burden of having to deal with two extra τ-descriptive formulas, corresponding to C_1 and C_2, in our logical analysis. In fact, as a consequence of our Normal Form Theorem below (Theorem 3.1.1), we have that all possible localizations can be thought of as being over the already given constant symbols 0 and max. Nonetheless, we might sometimes refer to the problem HEX in a localized manner, e.g., as $\text{HEX}(0, max)$, again for convenience. ∎

We first note that HEX is firmly monotone: if Player 1 has a winning strategy in a game played on a graph G, then adding more edges and vertices to G would still give Player 1 a win with the original strategy.

We consider HEX as a generalized quantifier and add it to FO_s to form the logic $\pm\text{HEX}^*[\text{FO}_s]$, as described in Definition 2.2.3. We restrict our work to the positive fraction $\text{HEX}^*[\text{FO}_s]$ and show that this logic has a projective normal form.

Theorem 3.1.1 *Let τ be some vocabulary. Every formula $\phi \in \text{HEX}^*[\text{FO}_s(\tau)]$ is equivalent to a formula of the form $\text{HEX}[\lambda \overline{x}\,\overline{y}\,\psi](\overline{0}, \overline{max})$, where $\psi \in \text{FO}_s(\tau)$, ψ projective and over the distinct k-tuples of variables $\overline{x}$ and $\overline{y}$, for some $k \geq 1$, and where $\overline{0}$ (resp. $\overline{max}$) is the constant symbol 0 (resp. max) repeated k times.*

Proof: We proceed by induction on the complexity of ϕ.

Case 1. $\phi \in \mathrm{FO}_s(\tau)$ is atomic or the negation of an atomic formula.

Let x_1, x_2, y_1, and y_2 be four distinct variables not occurring in ϕ. Then

$$\models \phi \iff \mathrm{HEX}[\lambda\,(x_1, y_1)(x_2, y_2)\,\phi]((0,0),(max, max)).$$

Indeed, let $\theta := \mathrm{HEX}[\lambda\,(x_1, y_1)(x_2, y_2)\,\phi]((0,0),(max, max))$, $\mathcal{A}$ be any τ-structure, and $\mathcal{A}_\Sigma$ be the τ-translation of $\mathcal{A}$ with respect to $\Sigma = \{\phi((x_1, y_1),(x_2, y_2))\}$ (i.e., we treat ϕ as if it were defined over the variables (x_1, y_1) and (x_2, y_2)). Note that the universe of $\mathcal{A}_\Sigma$ is $|\mathcal{A}|^2$, and so this structure has size at least 4. We then have:

$$\begin{aligned}
&\mathcal{A} \models \phi \;\Rightarrow\; \mathcal{A}_\Sigma \text{ is a complete graph (i.e., every possible edge is defined)} \\
&\Rightarrow\; \langle \mathcal{A}_\Sigma, (0,0), (max, max)\rangle \;\in \mathrm{HEX} \;\Rightarrow\; \mathcal{A} \models \theta\,.
\end{aligned}$$

Conversely,

$$\begin{aligned}
&\mathcal{A} \models \neg\phi \;\Rightarrow\; \mathcal{A}_\Sigma \text{ is a graph with no edges} \\
&\Rightarrow\; \langle \mathcal{A}_\Sigma, (0,0), (max, max)\rangle \;\notin \mathrm{HEX} \;\Rightarrow\; \mathcal{A} \models \neg\theta\,.
\end{aligned}$$

The remaining cases will be proven with constructions that resemble those made by Immerman in his proof of the existence of a projective normal form for $\mathrm{TC}^*[\mathrm{FO}_s]$ in [Imm87, Theorem 3.3]. The difference with our proof is in the following gadget, which will be at the heart of the solution of each case of the induction.

Let G be an undirected graph with source s and sink t. Let $X(G)$ be a new graph obtained from 8 copies of G, namely G_1, G_2, ..., G_8, as follows:

the sources of G_1, G_2, G_3, and G_4 are combined into one vertex s_1;

the sources of G_5, G_6, G_7, and G_8 are combined into one vertex s_2;

the sinks of G_1, G_2, G_5, and G_6 are combined into one vertex t_1;

the sinks of G_3, G_4, G_7, and G_8 are combined into one vertex t_2.

We say that $X(G)$ has two sources, s_1 and s_2, and two sinks, t_1 and t_2. The graph $X(G)$ is shown in Figure 1, where each graph G_i is represented by a line labelled with the number i ($i = 1, \ldots, 8$). Note that $X(G)$ is undirected also.

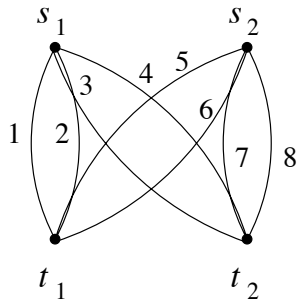

Figure 1: The graph $X(G)$.

Now, by the *HEX-game on* $X(G)$ we mean a slightly different game than the HEX-game on (G, s, t). In an HEX-game on $X(G)$ either player can start the game. Both players alternate coloring vertices of $X(G)$ as usual, but they can also color the two sources and the two sinks. Player 1 wins if and only if there is a path from one of the sources to one of the sinks of vertices colored blue (Player 1's color) including the source and the sink.

We show next that Player 1 has a winning strategy in the HEX-game on (G, s, t) if and only if Player 1 has a winning strategy in the HEX-game on $X(G)$.

Lemma 3.1.2 (i) *If Player 1 has a winning strategy in the HEX-game on* (G, s, t) *then Player 1 has a winning strategy in the HEX-game on* $X(G)$.

(ii) *If Player 2 has a winning strategy in the HEX-game on* (G, s, t) *then Player 2 has a winning strategy in the HEX-game on* $X(G)$.

Proof: (i) There are two cases to consider depending on who makes the first move in the HEX-game on $X(G)$:

(1) *Player 1 starts the HEX-game on* $X(G)$: Then Player 1 wins the HEX-game on $X(G)$ by playing as follows. Player 1 starts by coloring the source s_1 blue. If Player 2 colors a vertex of G_1 or G_2 red, then Player 1 colors the sink t_2 blue; otherwise, Player 1 colors the sink t_1 blue. Assume w.l.o.g. that Player 1 ends up coloring the sink t_1 blue (and so the vertices of G_1 and G_2 are free of colors). If Player 2 plays outside of G_1 then Player 1 plays in G_1, and continues playing there until the end of the game, according to his winning strategy for the HEX-game on (G, s, t); this strategy is effective since Player 1 is the first one to color a vertex in G_1 different from the source s_1 and the sink t_1. The argument is similar for the case when Player 2 plays outside of G_2.

(2) *Player 2 starts the HEX-game on* $X(G)$:

(a) Player 2 colors a source (resp. sink) red in his first move. Then Player 1 replies by coloring the other source (resp. sink) blue. W.l.o.g. suppose that Player 2 has colored s_1 red and Player 1 has colored s_2 blue. If Player 2 colors a vertex of G_5 or G_6 red then Player 1 replies by coloring the sink t_2 blue; otherwise, Player 1 colors t_1 blue. W.l.o.g. we may assume that s_2 and t_2 have been colored blue, and no other vertex of G_7 and G_8 has been colored. If Player 2 colors a vertex of G_7 red then Player 1 plays in G_8 (and continues playing in this copy of G until the end of the game) according to his winning strategy in the HEX-game on (G, s, t); otherwise, Player 1 plays in G_7, according to his winning strategy in the HEX-game on (G, s, t).

(b) Player 2 colors a vertex different from the sources and the sinks in his first move. W.l.o.g. assume Player 2 colors a vertex in G_1 red. Then Player 1 colors the source s_2 blue. If Player 2 replies by coloring a vertex in G_7 or G_8 red then Player 1 colors t_1 blue; otherwise, Player 1 colors t_2 blue. The rest of Player 1's strategy is as in (a).

(ii) We need to describe a sequence of moves for Player 2 such that no matter

how Player 1 plays on $X(G)$, Player 1 can not build a path of blue vertices from some source to some sink. As in part (i) there are two cases to consider:

(1) *Player 1 starts the HEX-game on* $X(G)$:

(a) Suppose that Player 1 begins by coloring some source or some sink blue. W.l.o.g. say Player 1 colors s_1 blue. Then Player 2 colors t_1 red and, hence, Player 1 is forced to color t_2 blue in order to secure a blue a path from s_1 to t_2. Player 2 colors s_2 red and, therefore, Player 1 can build a blue path only by playing on either G_3 or G_4. Wherever Player 1 plays, Player 2 replies according to his winning strategy in the HEX-game on (G, s, t) and effectively blocks all paths that Player 1 tries to construct in $X(G)$.

(b) Suppose that Player 1 begins by coloring a vertex different from the sources and the sinks blue. Then, in whichever copy of G Player 1 plays, Player 2 replies according to his winning strategy for the HEX-game on (G, s, t), and when Player 1 plays on some source or some sink, Player 2 proceeds as explained in (a).

(2) *Player 2 starts the HEX-game on* $X(G)$: Then, Player 2 has the extra advantage of restricting further the set of vertices that Player 1 can color to make a path and, so, Player 2 colors any vertex red, and continues playing as explained in (1) depending on which vertex Player 1 colors. ∎

To describe the construction of $X(G)$ from a given graph G, we introduce the following formulas on the variables x_4, x_3, x_2, and x_1. We use abbreviations as $x = y = z$ for $x = y \wedge y = z \wedge x = z$.

$$
\begin{aligned}
\mathbf{0}(x_4, x_3, x_2, x_1) &:= x_4 = x_3 = x_2 = x_1 = 0 \\
\mathbf{1}(x_4, x_3, x_2, x_1) &:= x_4 = x_3 = x_2 = 0 \wedge x_1 = max \\
\mathbf{2}(x_4, x_3, x_2, x_1) &:= x_4 = x_3 = x_1 = 0 \wedge x_2 = max \\
\mathbf{3}(x_4, x_3, x_2, x_1) &:= x_4 = x_3 = 0 \wedge x_2 = x_1 = max \\
\mathbf{4}(x_4, x_3, x_2, x_1) &:= x_4 = x_2 = x_1 = 0 \wedge x_3 = max
\end{aligned}
$$

$$\begin{aligned}
\mathbf{5}(x_4,x_3,x_2,x_1) &:= x_4 = x_2 = 0 \wedge x_3 = x_1 = max \\
\mathbf{6}(x_4,x_3,x_2,x_1) &:= x_4 = x_1 = 0 \wedge x_3 = x_2 = max \\
\mathbf{7}(x_4,x_3,x_2,x_1) &:= x_4 = 0 \wedge x_3 = x_2 = x_1 = max \\
\mathbf{8}(x_4,x_3,x_2,x_1) &:= x_4 = max \wedge x_3 = x_2 = x_1 = 0 \\
\mathbf{m}(x_4,x_3,x_2,x_1) &:= x_4 = x_3 = x_2 = x_1 = max
\end{aligned}$$

(Think of max as 1. Then the above formulas, but the last, spell out the binary encoding of the natural number which are they named after. We use these formulas to codify the elements of each one of the eight copies of the graph G in Figure 1.)

Case 2. $\phi := \mathrm{HEX}[\lambda\,\overline{u}\,\overline{v}\,\theta](\overline{q},\overline{r})$, where $\overline{q}$ and $\overline{r}$ are k-tuples of constant symbols, and θ is projective. We wish to replace $\overline{q}$ and $\overline{r}$ by $\overline{0}$ and $\overline{max}$.

For $i = 1,\ldots,24$, we define the following formulas α_i on the variables $\overline{u}$, x_4, x_3, x_2, x_1, $\overline{v}$, y_4, y_3, y_2, and y_1. To simplify notation let $\overline{x} = (x_4,x_3,x_2,x_1)$ and $\overline{y} = (y_4,y_3,y_2,y_1)$.

$$\begin{aligned}
\alpha_1 &:= \overline{u} = \overline{0} \wedge \mathbf{0}(\overline{x}) \wedge \overline{v} = \overline{q} \wedge \mathbf{1}(\overline{y}) \\
\alpha_2 &:= \overline{u} = \overline{0} \wedge \mathbf{0}(\overline{x}) \wedge \overline{v} = \overline{q} \wedge \mathbf{8}(\overline{y}) \\
\alpha_3 &:= \mathbf{1}(\overline{x}) \wedge \mathbf{1}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_4 &:= \mathbf{2}(\overline{x}) \wedge \mathbf{2}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_5 &:= \mathbf{3}(\overline{x}) \wedge \mathbf{3}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_6 &:= \mathbf{4}(\overline{x}) \wedge \mathbf{4}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_7 &:= \mathbf{5}(\overline{x}) \wedge \mathbf{5}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_8 &:= \mathbf{6}(\overline{x}) \wedge \mathbf{6}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_9 &:= \mathbf{7}(\overline{x}) \wedge \mathbf{7}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_{10} &:= \mathbf{8}(\overline{x}) \wedge \mathbf{8}(\overline{y}) \wedge \theta(\overline{u},\overline{v}) \\
\alpha_{11} &:= \overline{u} = \overline{q} \wedge \mathbf{1}(\overline{x}) \wedge \mathbf{2}(\overline{y}) \wedge \theta(\overline{u},\overline{v})
\end{aligned}$$

$$
\begin{aligned}
\alpha_{12} &:= \overline{u} = \overline{q} \wedge \mathbf{1}(\overline{x}) \wedge \mathbf{3}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{13} &:= \overline{u} = \overline{q} \wedge \mathbf{1}(\overline{x}) \wedge \mathbf{4}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{14} &:= \mathbf{2}(\overline{x}) \wedge \overline{v} = \overline{r} \wedge \mathbf{1}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{15} &:= \mathbf{5}(\overline{x}) \wedge \overline{v} = \overline{r} \wedge \mathbf{1}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{16} &:= \mathbf{6}(\overline{x}) \wedge \overline{v} = \overline{r} \wedge \mathbf{1}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{17} &:= \overline{u} = \overline{q} \wedge \mathbf{8}(\overline{x}) \wedge \mathbf{5}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{18} &:= \overline{u} = \overline{q} \wedge \mathbf{8}(\overline{x}) \wedge \mathbf{6}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{19} &:= \overline{u} = \overline{q} \wedge \mathbf{8}(\overline{x}) \wedge \mathbf{7}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{20} &:= \mathbf{3}(\overline{x}) \wedge \overline{v} = \overline{r} \wedge \mathbf{8}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{21} &:= \mathbf{4}(\overline{x}) \wedge \overline{v} = \overline{r} \wedge \mathbf{8}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{22} &:= \mathbf{7}(\overline{x}) \wedge \overline{v} = \overline{r} \wedge \mathbf{8}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}) \\
\alpha_{23} &:= \overline{u} = \overline{r} \wedge \mathbf{1}(\overline{x}) \wedge \overline{v} = \overline{max} \wedge \mathbf{m}(\overline{y}) \\
\alpha_{24} &:= \overline{u} = \overline{r} \wedge \mathbf{8}(\overline{x}) \wedge \overline{v} = \overline{max} \wedge \mathbf{m}(\overline{y})
\end{aligned}
$$

Let

$$\psi(\overline{u}, x_4, x_3, x_2, x_1, \overline{v}, y_4, y_3, y_2, y_1) := \bigvee_{i=1}^{24} \alpha_i.$$

Then ψ is a projective formula, and, for any appropriate τ-structure $\mathcal{A}$, the τ-translation of $\mathcal{A}$ with respect to ψ is a graph, namely G_ψ, with vertex set contained in $|\mathcal{A}|^{k+4}$ and edge relation described by $\psi^{\mathcal{A}}$. Let G_θ be the graph with vertex set $|\mathcal{A}|^k$ and edge relation described by $\theta^{\mathcal{A}}$. Then G_ψ is, in fact, $X(G_\theta)$ with the sources s_1 and s_2 being $(\overline{q}, 0, 0, 0, max)^{\mathcal{A}}$ and $(\overline{q}, max, 0, 0, 0)^{\mathcal{A}}$ respectively, joined to the $(k+4)$-tuple $\mathbf{0} := (\overline{0}, 0, 0, 0, 0)^{\mathcal{A}}$, and the sinks t_1 and t_2 being $(\overline{r}, 0, 0, 0, max)^{\mathcal{A}}$ and $(\overline{r}, max, 0, 0, 0)^{\mathcal{A}}$ respectively, joined to the $(k+4)$-tuple $\mathbf{max} := (\overline{max}, max, max, max, max)^{\mathcal{A}}$. Note that G_ψ is an undirected graph because θ is symmetric, by assumption, and the logical connective $\wedge$ is commutative. By Lemma 3.1.2, it follows that Player 1 has a winning strategy in the HEX-game

on $(G_\theta, \overline{q}^{\mathcal{A}}, \overline{r}^{\mathcal{A}})$ if and only if Player 1 has a winning strategy in the HEX-game on $(G_\psi, \mathbf{0}, \mathbf{max})$. Therefore,

$$\models \phi \iff \mathrm{HEX}[\lambda\,(\overline{u}, x_4, x_3, x_2, x_1)\,(\overline{v}, y_4, y_3, y_2, y_1)\,\psi](\mathbf{0}, \mathbf{max}).$$

Case 3. $\phi := \exists z\,\mathrm{HEX}[\lambda\,\overline{u}\,\overline{v}\,\theta(\overline{u}, \overline{v}, z)](\overline{0}, \overline{max})$, where z is not bound in θ and is different from any variable of the k-tuples $\overline{u}$ and $\overline{v}$, and θ is projective.

We define the following formulas β_i (for $1 \leq i \leq 24$) on the variables $\overline{u}$, $\overline{v}$, x_5, x_4, x_3, x_2, x_1, y_5, y_4, y_3, y_2, and y_1. As before, $\overline{x} = (x_4, x_3, x_2, x_1)$ and $\overline{y} = (y_4, y_3, y_2, y_1)$.

$$\begin{aligned}
\beta_1 &:= \overline{u} = \overline{0} \wedge x_5 = 0 \wedge \mathbf{0}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{1}(\overline{y}) \\
\beta_2 &:= \overline{u} = \overline{0} \wedge x_5 = 0 \wedge \mathbf{0}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{8}(\overline{y}) \\
\beta_3 &:= \mathbf{1}(\overline{x}) \wedge \mathbf{1}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_4 &:= \mathbf{2}(\overline{x}) \wedge \mathbf{2}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_5 &:= \mathbf{3}(\overline{x}) \wedge \mathbf{3}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_6 &:= \mathbf{4}(\overline{x}) \wedge \mathbf{4}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_7 &:= \mathbf{5}(\overline{x}) \wedge \mathbf{5}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_8 &:= \mathbf{6}(\overline{x}) \wedge \mathbf{6}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_9 &:= \mathbf{7}(\overline{x}) \wedge \mathbf{7}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{10} &:= \mathbf{8}(\overline{x}) \wedge \mathbf{8}(\overline{y}) \wedge x_5 = y_5 \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{11} &:= \overline{u} = \overline{0} \wedge \mathbf{1}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{2}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{12} &:= \overline{u} = \overline{0} \wedge \mathbf{1}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{3}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{13} &:= \overline{u} = \overline{0} \wedge \mathbf{1}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{4}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{14} &:= \overline{v} = \overline{max} \wedge \mathbf{2}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{1}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{15} &:= \overline{v} = \overline{max} \wedge \mathbf{5}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{1}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5)
\end{aligned}$$

$$\begin{aligned}
\beta_{16} &:= \overline{v} = \overline{max} \wedge \mathbf{6}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{1}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{17} &:= \overline{u} = \overline{0} \wedge \mathbf{8}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{5}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{18} &:= \overline{u} = \overline{0} \wedge \mathbf{8}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{6}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{19} &:= \overline{u} = \overline{0} \wedge \mathbf{8}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{7}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{20} &:= \overline{v} = \overline{max} \wedge \mathbf{3}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{8}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{21} &:= \overline{v} = \overline{max} \wedge \mathbf{4}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{8}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{22} &:= \overline{v} = \overline{max} \wedge \mathbf{7}(\overline{x}) \wedge x_5 = y_5 \wedge \mathbf{8}(\overline{y}) \wedge \theta(\overline{u}, \overline{v}, x_5) \\
\beta_{23} &:= \overline{u} = \overline{max} \wedge \mathbf{1}(\overline{x}) \wedge \overline{v} = \overline{max} \wedge y_5 = max \wedge \mathbf{m}(\overline{y}) \\
\beta_{24} &:= \overline{u} = \overline{max} \wedge \mathbf{8}(\overline{x}) \wedge \overline{v} = \overline{max} \wedge y_5 = max \wedge \mathbf{m}(\overline{y})
\end{aligned}$$

Let

$$\psi(\overline{u}, x_5, x_4, x_3, x_2, x_1, \overline{v}, y_5, y_4, y_3, y_2, y_1) := \bigvee_{i=1}^{24} \beta_i.$$

Then ψ is a projective formula, and, for any appropriate τ-structure $\mathcal{A}$, the τ-translation of $\mathcal{A}$ with respect to ψ is the (undirected) graph G_ψ pictured in Figure 2, which is constructed as follows: for each $z \in |\mathcal{A}|$ let G_z be the (undirected) graph described by $\theta(\cdot, \cdot, z)^{\mathcal{A}}$, with the vertex $\overline{0}^{\mathcal{A}}$ as the source and the vertex $\overline{max}^{\mathcal{A}}$ as the sink; build the graphs $X(G_z)$ for each $z \in |\mathcal{A}|$ and join both sources (resp. sinks) of each $X(G_z)$ to the vertex labelled $\mathbf{0} := (\overline{0}, 0, 0, 0, 0, 0)^{\mathcal{A}}$ (resp. $\mathbf{max} := (\overline{max}, max, max, max, max, max)^{\mathcal{A}}$), which is the source (resp. sink) of G_ψ.

Suppose that Player 1 has a winning strategy in the HEX-game on G_z, for some $z \in |\mathcal{A}|$. By Lemma 3.1.2, Player 1 has a winning strategy in the HEX-game on $X(G_z)$. Hence, in the HEX-game on G_ψ, Player 1's winning strategy is to play on $X(G_z)$ according to his winning strategy for that graph.

Conversely, suppose that Player 2 has a winning strategy in the HEX-game on G_z, for every $z \in |\mathcal{A}|$. By Lemma 3.1.2, Player 2 has a winning strategy in the

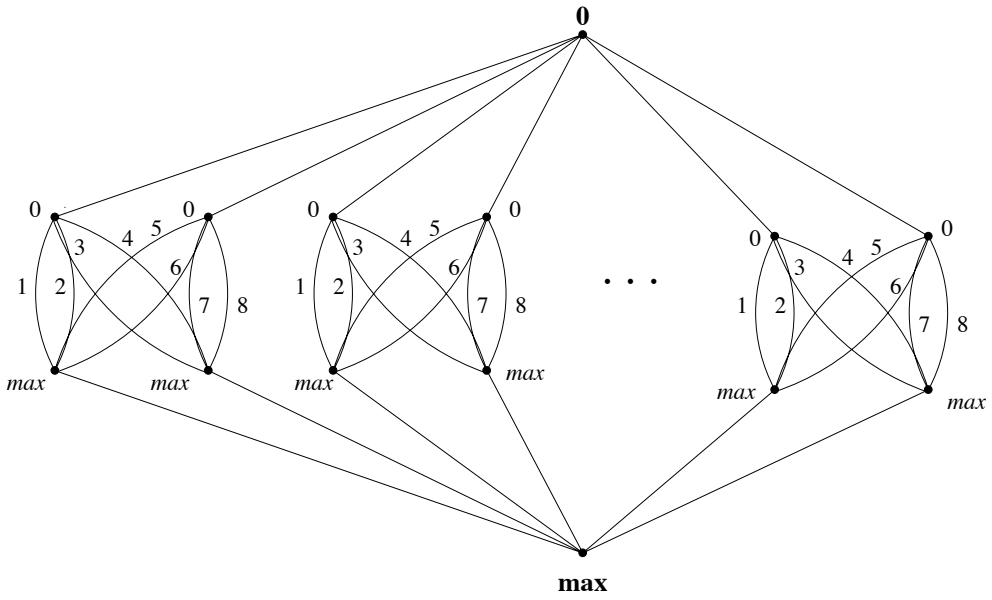

Figure 2: The existential case.

HEX-game on $X(G_z)$, for every $z \in |\mathcal{A}|$. Then, on G_ψ, wherever Player 1 plays, Player 2 replies according to his winning strategies for the HEX-game on each graph $X(G_z)$, thus blocking every possible path that Player 1 tries to construct in G_ψ from the source **0** to the sink **max**. Hence,

$$\models \phi \iff \mathrm{HEX}[\lambda\,(\overline{u}, x_5, x_4, x_3, x_2, x_1)\,(\overline{v}, y_5, y_4, y_3, y_2, y_1)\,\psi](\mathbf{0}, \mathbf{max}).$$

Case 4. $\phi := \forall z\,\mathrm{HEX}[\lambda\,\overline{u}\,\overline{v}\,\theta(\overline{u}, \overline{v}, z)](\overline{0}, \overline{max})$, where z is not bound in θ and is different from any variable of the k-tuples $\overline{u}$ and $\overline{v}$, and θ is projective.

We define the following formulas γ_i (for $1 \leq i \leq 28$) on the variables $\overline{u}$, $\overline{v}$, x_5, x_4, x_3, x_2, x_1, y_5, y_4, y_3, y_2, and y_1 (again we set $\overline{x} = (x_4, x_3, x_2, x_1)$ and $\overline{y} = (y_4, y_3, y_2, y_1)$).

$$\gamma_1 \;:=\; \beta_1 \wedge y_5 = 0$$

$$
\begin{aligned}
\gamma_2 &:= \beta_2 \wedge y_5 = 0 \\
\gamma_i &:= \beta_i \quad \text{for } i \text{ such that } 3 \leq i \leq 22 \\
\gamma_{23} &:= \beta_{23} \wedge x_5 = max \\
\gamma_{24} &:= \beta_{24} \wedge x_5 = max \\
\gamma_{25} &:= s(x_5, y_5) \wedge \overline{u} = \overline{max} \wedge \mathbf{1}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{1}(\overline{y}) \\
\gamma_{26} &:= s(x_5, y_5) \wedge \overline{u} = \overline{max} \wedge \mathbf{1}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{8}(\overline{y}) \\
\gamma_{27} &:= s(x_5, y_5) \wedge \overline{u} = \overline{max} \wedge \mathbf{8}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{1}(\overline{y}) \\
\gamma_{28} &:= s(x_5, y_5) \wedge \overline{u} = \overline{max} \wedge \mathbf{8}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{8}(\overline{y})
\end{aligned}
$$

Let

$$\psi(\overline{u}, x_5, x_4, x_3, x_2, x_1, \overline{v}, y_5, y_4, y_3, y_2, y_1) := \bigvee_{i=1}^{28} \gamma_i.$$

Then ψ is a projective formula, and, for any appropriate τ-structure $\mathcal{A}$, the τ-translation of $\mathcal{A}$ with respect to ψ is the (undirected) graph G_ψ pictured in Figure 3. As in Case 3, for each $z \in |\mathcal{A}|$, let G_z be the graph described by $\theta(\cdot, \cdot, z)^{\mathcal{A}}$. Then, for each z such that $0^{\mathcal{A}} \leq z < max^{\mathcal{A}}$, the graph $X(G_z)$ is joined to $X(G_{z+1})$ by edges from each sink of $X(G_z)$ to each source of $X(G_{z+1})$. The sources of $X(G_0)$ are joined to a vertex labelled $\mathbf{0} := (\overline{0}, 0, 0, 0, 0, 0)^{\mathcal{A}}$ (the source of G_ψ), and the sinks of $X(G_{max})$ are joined to a vertex labelled $\mathbf{max} := (\overline{max}, max, max, max, max, max)^{\mathcal{A}}$ (the sink of G_ψ).

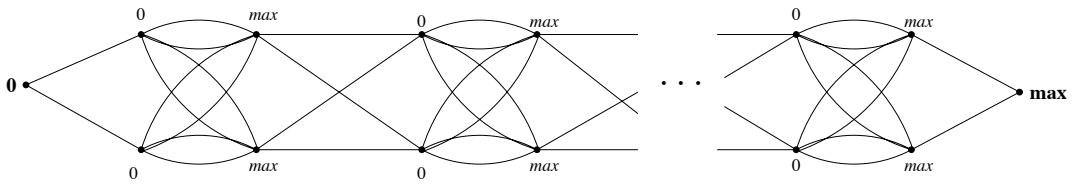

Figure 3: The universal case.

Suppose that Player 1 has a winning strategy in the HEX-game on G_z, for each $z \in |\mathcal{A}|$. By Lemma 3.1.2, Player 1 has a winning strategy in the HEX-game on $X(G_z)$, for each $z \in |\mathcal{A}|$. Then, in the HEX-game on G_ψ, Player 1's winning strategy is to play on each $X(G_z)$ according to his winning strategy for the HEX-game on that graph. No matter what Player 2 does, Player 1 will be able to construct a blue path in each $X(G_z)$ from some of its sources to some of its sinks (with these source and sink being colored blue also), and, consequently, Player 1 will have a blue path in G_ψ from the source $\mathbf{0}$ to the sink $\mathbf{max}$.

Conversely, suppose that Player 2 has a winning strategy in the HEX-game on G_z, for some $z \in |\mathcal{A}|$. By Lemma 3.1.2, Player 2 has a winning strategy in the HEX-game on $X(G_z)$. Then, in the HEX-game on G_ψ, Player 2 plays in the graph $X(G_z)$ according to his winning strategy, effectively blocking all possible blue paths that Player 1 tries to construct in that graph and, consequently, blocking all possible blue paths in G_ψ from the source $\mathbf{0}$ to the sink $\mathbf{max}$.

Thus,

$$\models \phi \iff \mathrm{HEX}[\lambda\,(\overline{u}, x_5, x_4, x_3, x_2, x_1)\,(\overline{v}, x_5, x_4, y_3, y_2, y_1)\,\psi](\mathbf{0}, \mathbf{max}).$$

Note that here is where we use successor.

Case 5. $\phi := \mathrm{HEX}[\lambda\,\overline{x}\,\overline{y}\,\mathrm{HEX}[\lambda\,\overline{u}\,\overline{v}\,\theta(\overline{u}, \overline{v}, \overline{x}, \overline{y})](\overline{0}, \overline{max})](\overline{0}, \overline{max})$, where $\overline{x}$ and $\overline{y}$ are k-tuples, $\overline{u}$ and $\overline{v}$ are k'-tuples, and θ is projective.

We define the following formulas δ_i (for $1 \leq i \leq 28$) on the variables $\overline{u}$, $\overline{v}$, $\overline{x}_6$, $\overline{x}_5$, x_4, x_3, x_2, x_1, $\overline{y}_6$, $\overline{y}_5$, y_4, y_3, y_2, and y_1 ($\overline{x}_6$, $\overline{x}_5$, $\overline{y}_6$, $\overline{y}_5$ are k'-tuples of variables and, as before $\overline{x} = (x_4, x_3, x_2, x_1)$ and $\overline{y} = (y_4, y_3, y_2, y_1)$).

$$\begin{aligned}
\delta_1 &:= \overline{u} = \overline{v} = \overline{0} \wedge \overline{x}_5 = \overline{x}_6 = \overline{0} \wedge \mathbf{0}(\overline{x}) \wedge \overline{y}_5 = \overline{0} \wedge \mathbf{1}(\overline{y}) \\
\delta_2 &:= \overline{u} = \overline{v} = \overline{0} \wedge \overline{x}_5 = \overline{x}_6 = \overline{0} \wedge \mathbf{0}(\overline{x}) \wedge \overline{y}_5 = \overline{0} \wedge \mathbf{8}(\overline{y})
\end{aligned}$$

The formulas δ_3 to δ_{22} are similar to the formulas β_3 to β_{22} of Case 3, but instead of $x_5 = y_5$ write $\overline{x}_5 = \overline{y}_5 \wedge \overline{x}_6 = \overline{y}_6$, and instead of $\theta(\overline{u}, \overline{v}, \overline{x}_5)$ write $\theta(\overline{u}, \overline{v}, \overline{x}_5, \overline{x}_6)$

$$\begin{aligned}
\delta_{23} &:= \overline{u} = \overline{v} = \overline{max} \wedge \overline{x}_5 = \overline{max} \wedge \mathbf{1}(\overline{x}) \wedge \overline{y}_5 = \overline{y}_6 = \overline{max} \wedge \mathbf{m}(\overline{y}) \\
\delta_{24} &:= \overline{u} = \overline{v} = \overline{max} \wedge \overline{x}_5 = \overline{max} \wedge \mathbf{8}(\overline{x}) \wedge \overline{y}_5 = \overline{y}_6 = \overline{max} \wedge \mathbf{m}(\overline{y}) \\
\delta_{25} &:= \overline{u} = \overline{max} \wedge \mathbf{1}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{1}(\overline{y}) \wedge \overline{x}_6 = \overline{y}_5 \\
\delta_{26} &:= \overline{u} = \overline{max} \wedge \mathbf{1}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{8}(\overline{y}) \wedge \overline{x}_6 = \overline{y}_5 \\
\delta_{27} &:= \overline{u} = \overline{max} \wedge \mathbf{8}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{1}(\overline{y}) \wedge \overline{x}_6 = \overline{y}_5 \\
\delta_{28} &:= \overline{u} = \overline{max} \wedge \mathbf{8}(\overline{x}) \wedge \overline{v} = \overline{0} \wedge \mathbf{8}(\overline{y}) \wedge \overline{x}_6 = \overline{y}_5
\end{aligned}$$

Let

$$\psi(\overline{u}, \overline{x}_6, \overline{x}_5, x_4, x_3, x_2, x_1, \overline{v}, \overline{y}_6, \overline{y}_5, y_4, y_3, y_2, y_1) := \bigvee_{i=1}^{28} \delta_i.$$

Then ψ is a projective formula, and, for any appropriate τ-structure $\mathcal{A}$ of size m, the τ-translation of $\mathcal{A}$ with respect to ψ is the (undirected) graph G_ψ pictured in Figure 4: It is $m \times m$ subgraphs $X(G_{i,j})$, for each pair $(\overline{x}_i, \overline{y}_j) \in |\mathcal{A}|^{2k}$ (where $G_{i,j}$ is the graph described by $\theta(\cdot, \cdot, \overline{x}_i, \overline{y}_i)^{\mathcal{A}}$), string together as follows. For each i and j in $\{0, \ldots, m-2\}$ and for all $h \in \{0, \ldots, m-1\}$, the sinks of $X(G_{i,j})$ have edges to the sources of $X(G_{j,h})$; for all $h \in \{0, \ldots, m-1\}$, the sources of $X(G_{0,h})$ have edges to a new vertex labelled $\mathbf{0} := (\overline{0}, \overline{0}, \overline{0}, 0, 0, 0, 0, 0)^{\mathcal{A}}$ (the source of G_ψ), and the sinks of $X(G_{m-1,h})$ have edges to a new vertex labelled $\mathbf{max} := (\overline{max}, \overline{max}, \overline{max}, max, max, max, max)^{\mathcal{A}}$ (the sink of G_ψ).

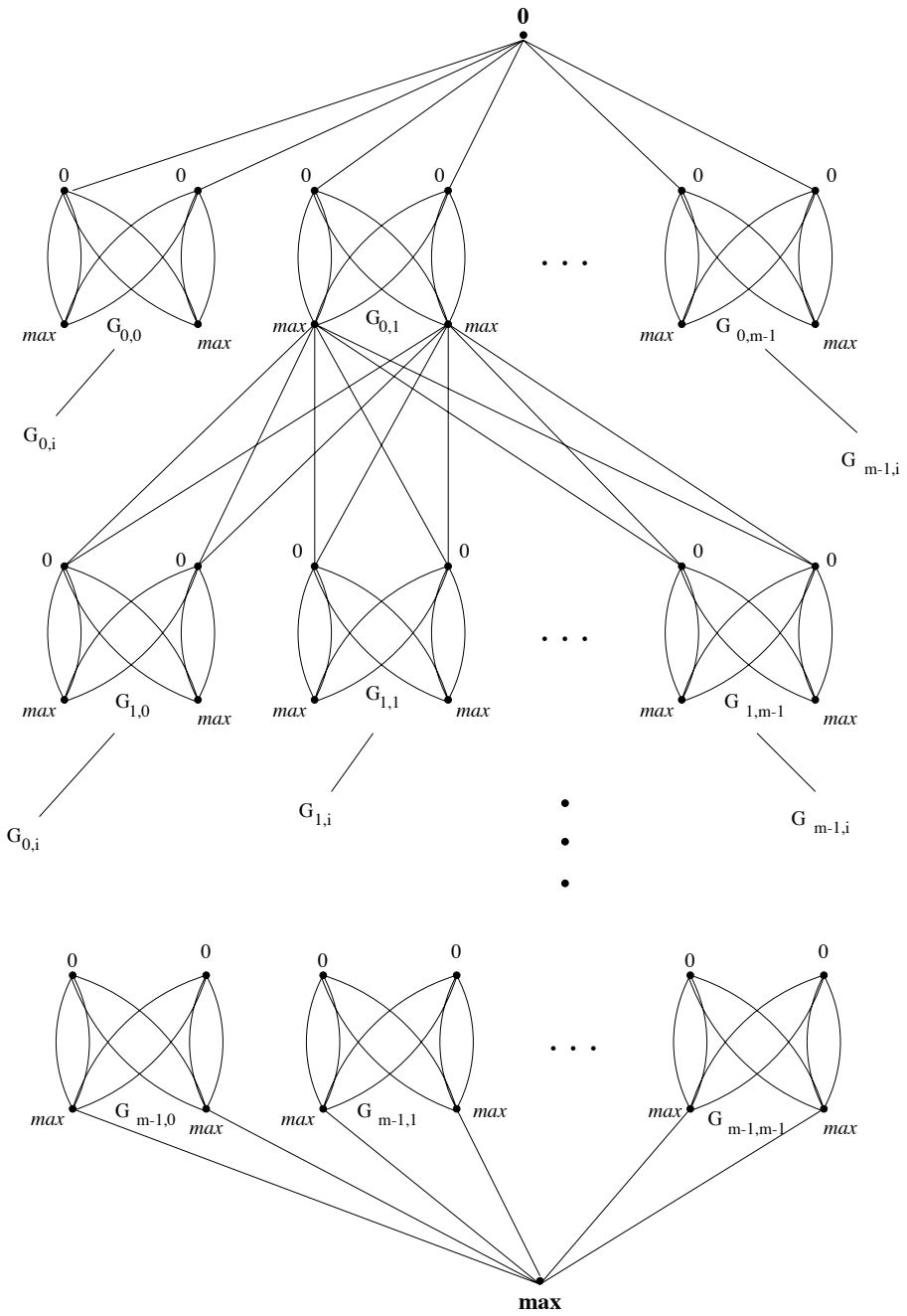

Figure 4: The nested case.

Using Lemma 3.1.2, and arguing as in the previous cases, one can see that Player 1 has a winning strategy in the HEX-game on $(G_\psi, \mathbf{0}, \mathbf{max})$ if and only if there exists a sequence of m pairs $(\overline{x}_0, \overline{y}_{j_0})$, $(\overline{x}_1, \overline{y}_{j_1})$, ..., $(\overline{x}_{m-1}, \overline{y}_{j_{m-1}})$, in $|\mathcal{A}|^{2k}$, such that Player 1 has a winning strategy in the HEX-game on each graph determined by $\theta(\cdot, \cdot, \overline{x}_h, \overline{y}_{j_h})^{\mathcal{A}}$ ($j_h \in \{0, \ldots, m-1\}$, $0 \leq h \leq m-1$), with source $0^{\mathcal{A}}$ and sink $max^{\mathcal{A}}$ (that is, on $(G_{h,j_h}, 0^{\mathcal{A}}, max^{\mathcal{A}})$).

Thus

$$\models \phi \iff \mathrm{HEX}[\lambda\,(\overline{u}, \overline{x}_6, \overline{x}_5, x_4, x_3, x_2, x_1)\,(\overline{v}, \overline{y}_6, \overline{y}_5, y_4, y_3, y_2, y_1)\,\psi](\mathbf{0}, \mathbf{max}).$$

All other form that ϕ may have can be easily reduced to one of the five cases above. For example, say

$$\phi := \mathrm{HEX}[\lambda\,\overline{x}\,\overline{y}\,\theta](\overline{0}, \overline{max}) \vee \mathrm{HEX}[\lambda\,\overline{u}\,\overline{v}\,\chi](\overline{0}, \overline{max}),$$

where θ and χ are projective formulas, $\overline{x}$ and $\overline{y}$ are k-tuples, $\overline{u}$ and $\overline{v}$ are k'-tuples, and $k \geq k'$. We first add to $\overline{u}$ (resp. $\overline{v}$) a $(k-k')$-tuple $\overline{u}'$ (resp. $\overline{v}'$) of variables not showing in any of the formulas θ or χ to form the k-tuple $(\overline{u}, \overline{u}')$ (resp. $(\overline{v}, \overline{v}')$), and consider

$$\chi'((\overline{u}, \overline{u}'), (\overline{v}, \overline{v}')) := \chi(\overline{u}, \overline{v}).$$

Since HEX is firmly monotone we get

$$\models \mathrm{HEX}[\lambda\,\overline{u}\,\overline{v}\,\chi](\overline{0}, \overline{max}) \iff \mathrm{HEX}[\lambda\,(\overline{u}, \overline{u}')\,(\overline{v}, \overline{v}')\,\chi'](\overline{0}, \overline{max});$$

therefore,

$$\models \phi \iff \mathrm{HEX}[\lambda\,\overline{x}\,\overline{y}\,\theta](\overline{0}, \overline{max}) \vee \mathrm{HEX}[\lambda\,(\overline{u}, \overline{u}')\,(\overline{v}, \overline{v}')\,\chi'](\overline{0}, \overline{max}),$$

and all the tuples of variables involved in the formula above are of length k; we can now proceed as in Case 3. The conjunction of two sentences is treated similarly, and as in Case 4.

Cases like $\phi := \exists z \mathrm{HEX}[\lambda\, x\, y\, \theta](z, max)$ are resolved with a standard technique of Model Theory: add a new constant C to the vocabulary τ_2 and use it as a witness for z. That is, in the extended vocabulary $\tau_2 \cup \{C\}$, ϕ is equivalent to $\mathrm{HEX}[\lambda\, x\, y\, \theta](C, max)$ and, by Case 2, this sentence is equivalent to one of the form $\mathrm{HEX}[\lambda\, \overline{u}\, \overline{v}\, \theta'](\overline{0}, \overline{max})$ with θ' a projective formula.

This completes the proof. ∎

Now that we know that the logic $\mathrm{HEX}^*[\mathrm{FO}_s]$ has a projective normal form we would like to use this fact, together with Corollary 2.4.5, to show that it captures **PSPACE**. For that we need first the following result.

Theorem 3.1.3 $\mathrm{DTC}^1[\mathrm{FO}_s] \leq \mathrm{HEX}^*[\mathrm{FO}_s]$.

Proof: Consider the τ_2-sentence $\psi := \mathrm{TC}[\lambda\, \overline{x}\, \overline{y}\, \theta(\overline{x}, \overline{y})](\overline{0}, \overline{max})$, where $\overline{x}$ and $\overline{y}$ are k-tuples of distinct variables, for some $k > 0$, and $\theta \in \mathrm{FO}_s$, and consider the following formula ϕ on the variables $\overline{x}$, u_1, u_2, $\overline{y}$, v_1, and v_2:

$$
\begin{aligned}
\phi \;:=\; & (\overline{x} = \overline{0} \wedge u_1 = u_2 = 0 \wedge \overline{y} = \overline{0} \\
& \wedge\; ((v_1 = max \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = max))) \\
\vee\; & (\overline{x} = \overline{max} \wedge ((u_1 = max \wedge u_2 = 0) \vee (u_1 = 0 \wedge u_2 = max)) \\
& \wedge\; \overline{y} = \overline{max} \wedge v_1 = max \wedge v_2 = max) \\
\vee\; & (u_1 = v_1 \wedge u_2 = v_2 \wedge \theta(\overline{x}, \overline{y})) \\
\vee\; & (u_1 = 0 \wedge u_2 = max \wedge v_1 = max \wedge v_2 = 0 \wedge \theta(\overline{x}, \overline{y})) \\
\vee\; & (u_1 = max \wedge u_2 = 0 \wedge v_1 = 0 \wedge v_2 = max \wedge \theta(\overline{x}, \overline{y})).
\end{aligned}
$$

Then

$$
\models \psi \;\longleftrightarrow\; \mathrm{HEX}[\lambda\, (\overline{x}, u_1, u_2)(\overline{y}, v_1, v_2)\phi](\overline{0}, \overline{max}).
$$

(Player 1's winning strategy is simple: play first in $(\overline{0}, 0, max)$ or $(\overline{0}, max, 0)$, and thereafter play opposite vertex to Player 2's choice, i.e., if Player 2 colors $(\overline{x}, u_1, u_2)$ then Player 1 colors $(\overline{x}, v_1, v_2)$ for $u_i, v_i \in \{0, max\}$, $v_i \neq u_i$, and $i = 1, 2$.)

Since

$$\begin{aligned} \models \ & \mathrm{DTC}[\lambda\, \overline{x}\, \overline{y}\, \theta(\overline{x}, \overline{y})](\overline{0}, \overline{max}) \ \longleftrightarrow \\ & \mathrm{TC}[\lambda\, \overline{x}\, \overline{y}(\theta(\overline{x}, \overline{y}) \wedge \forall \overline{z}(\theta(\overline{x}, \overline{z}) \ \longrightarrow \ \overline{z} = \overline{y}))](\overline{0}, \overline{max}), \end{aligned}$$

it follows that

$$\begin{aligned} \models \ & \mathrm{DTC}[\lambda\, \overline{x}\, \overline{y}\, \theta(\overline{x}, \overline{y})](\overline{0}, \overline{max}) \ \longleftrightarrow \\ & \mathrm{HEX}[\lambda\, \overline{x}'\, \overline{y}'\, \phi'(\overline{x}', \overline{y}')](\overline{0}, \overline{max}) \end{aligned}$$

for some ϕ' in FO_s. ∎

Note that we don't use successor to define the formulas in the above proof; hence, Theorem 3.1.3 holds on arbitrary structures:

Corollary 3.1.4 $\mathrm{DTC}^1[\mathrm{FO}] \leq \mathrm{HEX}^*[\mathrm{FO}]$. ∎

Corollary 3.1.5 $\mathbf{PSPACE} = \pm\mathrm{HEX}^*[\mathrm{FO}_s] = \mathrm{HEX}^*[\mathrm{FO}_s] = \mathrm{HEX}^1[\mathrm{FO}_s]$ *and* HEX *is complete for* **PSPACE** *via first-order projections.*

Proof: HEX is complete for **PSPACE** via logspace reducibility (see [ET76]), and **PSPACE** is closed under complement. These two facts together with Corollary 2.4.5, Theorem 3.1.1, and Theorem 3.1.3, give the desired conclusion. ∎

The equality $\mathbf{PSPACE} = \pm\mathrm{HEX}^*[\mathrm{FO}_s]$ was first shown by Makowsky and Pnueli in [MP92] using techniques different to ours: theirs consist on relating the logic $\mathrm{HEX}^*[\mathrm{DTC}^*[\mathrm{FO}_s]]$ with the complexity class $\mathbf{L}$ with an oracle in **PSPACE**, that is, $\mathbf{L}^{\mathbf{PSPACE}}$. However, one cannot deduce a normal form for $\mathrm{HEX}^*[\mathrm{FO}_s]$

with those methods, neither that HEX is complete via first-order projections. Thus, our results complements [MP92] in that regard.

For a logic such as HEX*[FO$_s$], we define the sublogic HEX*[FO$_s^+$] as the set of formulas in HEX*[FO$_s$] where any relation or constant symbol appears positively. By noting that the projective formula found in each case of the proof of Theorem 3.1.1 belongs to HEX*[FO$_s^+$], we have the following stronger result about HEX:

Corollary 3.1.6 HEX *is complete for* HEX*[FO$_s^+$] *via* monotone *first-order projections.* ∎

3.2 The WHEX logic

Consider the following variation of the game of HEX: given an undirected graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$, Player 1 must start the game by coloring a vertex b_1 for which there is an edge to s blue; Player 2 must respond by coloring a vertex r_1 for which there is an edge to b_1 but no edge to s (i.e., the equation $E(b_1, r_1) \wedge \neg E(s, r_1)$ holds) red; Player 1 continues (if he can) by coloring a vertex b_2, different from r_1, and for which there is an edge to b_1 but no edge to s blue; Player 2 responds by coloring a vertex r_2 for which there is an edge to b_2 but no edge to b_1 red; and so on. The goal of Player 1 is to reach t in this step-by-step construction of a path of vertices colored blue (except s and t), while Player 2 tries to prevent this, with both players making no triangles with their chosen vertex and two consecutive vertices on the path constructed so far. We regard this game as a weaker version of the game of HEX and, hence, named it *Weak Hex*, or WHEX, for short. Then, the WHEX problem is to determine if Player 1 has a winning strategy in the WHEX-game on G, s, and t.

Surprisingly enough, WHEX is not weaker than HEX in the usual sense of their computational complexity: we show that WHEX is **PSPACE**-complete via

a logspace reduction; our reduction is inspired by that of QBF to GEOGRAPHY in [Sch78]. We then view WHEX as a class of τ_2-structures (where $\tau_2 = \{E\}$) and form the logic WHEX*[FO$_s$]. We show that this logic has a projective normal form, it captures **PSPACE**, and, as a consequence, we obtain that WHEX is complete for **PSPACE** via first-order projections. Finally, we show that, over arbitrary structures, the logic WHEX*[FO] is contained in the logic $\mathrm{L}^{\omega}_{\infty\omega}$, which is defined as follows (cf. [EF95]): for a fixed $s \geq 1$, $\mathrm{L}^{s}_{\infty\omega}$ denotes the fragment of $\mathrm{L}_{\infty\omega}$, consisting of formulas with at most s free and bound variables, where $\mathrm{L}_{\infty\omega}$ is first-order logic with possibly infinite disjunctions and conjunctions; then, $\mathrm{L}^{\omega}_{\infty\omega} = \bigcup_{s\geq 1} \mathrm{L}^{s}_{\infty\omega}$.

3.2.1 WHEX is PSPACE-complete

QBF, or the problem of determining if a quantified boolean formula in conjunctive normal form is true, is the classical example of a **PSPACE**-complete problem [GJ79], and can be regarded as a game as follows [Sch78]: given

$$\Phi := \exists x_1 \forall x_2 \ldots Q_{n-1} x_{n-1} Q_n x_n \, \phi,$$

where ϕ is a boolean formula in conjunctive normal form involving the variables $x_1, \ldots, x_n$, and the quantifiers $Q_i \in \{\forall, \exists\}$ alternate starting with $\exists$ [1]; two players, called $\exists$ and $\forall$, take turns assigning, in the i-th move, a value of 0 (false) or 1 (true) to x_i, with player $\exists$ making the first move. Player $\exists$ wins the game if and only if ϕ is true after the n-th move.

To show that WHEX is **PSPACE**-complete, we describe below how to construct, using polynomial space, an instance, (G, s, t), of WHEX from an instance, Φ, of QBF, and show that *Player 1 has a winning strategy for the WHEX-game on* (G, s, t) *if and only if* $\exists$ *has a winning strategy for the QBF-game on* Φ.

[1] This is no loss of generality, since we can always add clauses of the form $x \vee \neg x$ without altering the truth value of ϕ.

So, let $\Phi := \exists x_1 \forall x_2 \ldots Q_n x_n (C_1 \wedge C_2 \wedge \ldots \wedge C_m)$, where each clause C_i is a conjunction of literals. Define the graph $G = (V, E)$ as follows:

$$
\begin{aligned}
V &= \{s, t, y\} \cup \{x_i, \ddot{x}_i, u_i, \ddot{u}_i, v_i \,|\, 1 \leq i \leq n\} \cup \\
&\quad \{w_{2i} \,|\, 1 \leq i \leq \lceil n/2 \rceil\} \cup \{c_i, z_i \,|\, 1 \leq i \leq m\}
\end{aligned}
$$

$$
\begin{aligned}
E &= \{(s, x_1), (s, \ddot{x}_1), (v_n, c_1), (v_n, z_1), (z_m, t), (y, t)\} \cup \\
&\quad \{(x_i, u_i), (\ddot{x}_i, \ddot{u}_i), (u_i, t), (\ddot{u}_i, t), (x_i, v_i), (\ddot{x}_i, v_i) \,|\, 1 \leq i \leq n\} \cup \\
&\quad \{(v_{2i}, w_{2i}), (w_{2i}, t) \,|\, 1 \leq i \leq \lceil n/2 \rceil\} \cup \\
&\quad \{(c_i, y) \,|\, 1 \leq i \leq m\} \cup \\
&\quad \{(c_i, u_j) \,|\, \text{ the literal } \neg x_j \text{ is in clause } C_i\} \cup \\
&\quad \{(c_i, \ddot{u}_j) \,|\, \text{ the literal } x_j \text{ is in clause } C_i\}.
\end{aligned}
$$

In Figure 5, we illustrate the graph corresponding to $\exists x_1 \forall x_2 \exists x_3 [(\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_1) \wedge (x_2 \vee \neg x_3)]$.

First note that the moves of Player 1 and Player 2 on the graph G corresponds to moves by $\exists$ and $\forall$ on Φ: Player 1 starts by coloring either vertex x_1 or $\ddot{x}_1$ blue. This corresponds to $\exists$ giving value 1 or 0 to x_1, respectively. Then, Player 2 is forced to color vertex u_1 or $\ddot{u}_1$ red in order to block Player 1 from reaching t; Player 1 has no choice but to color v_1 blue, and then it's Player 2's turn to color either vertex x_2 or $\ddot{x}_2$ red, which corresponds to $\forall$ choosing the truth value of x_2. According to the selection of Player 2, Player 1 must continue by coloring the opposite vertex; then Player 2 is forced to color either u_2 or $\ddot{u}_2$ –whichever one that will block Player 1's passage to t. Then Player 1's only option is v_2, and then, once again, Player 2 must color w_2. Hence, in his next move, Player 1 has the choice of coloring vertex x_3 or $\ddot{x}_3$ blue, and so on.

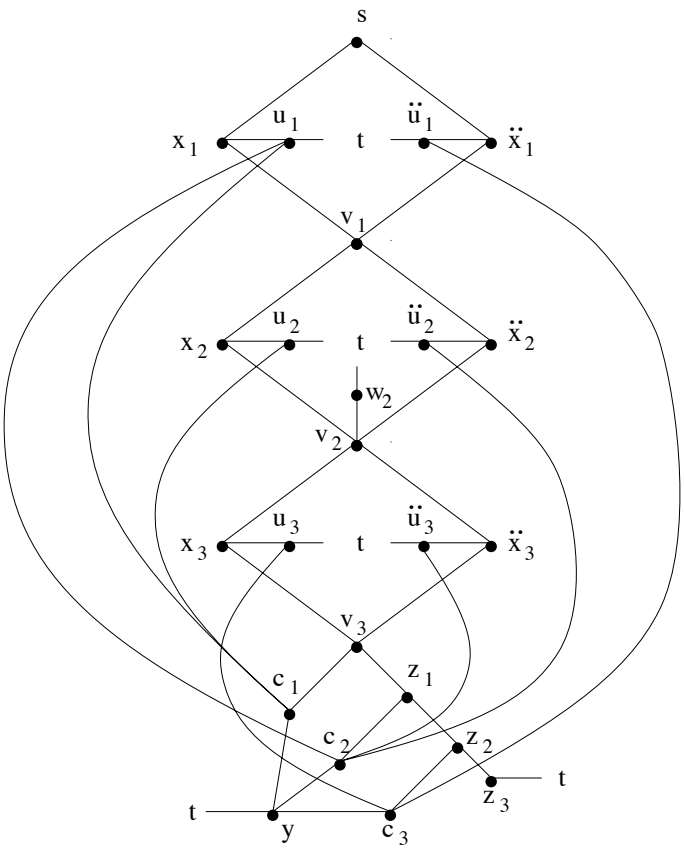

Figure 5: An instance of WHEX corresponding to $\exists x_1 \forall x_2 \exists x_3 [(\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_1) \wedge (x_2 \vee \neg x_3)]$.

Now, suppose $\exists$ has a winning strategy in the QBF-game. Then Player 1 has the following winning strategy in the WHEX-game: if $\exists$ gives value 1 (resp. 0) to variable x_{2i-1} then Player 1 colors vertex x_{2i-1} (resp. $\ddot{x}_{2i-1}$) blue. If, when Player 1 reaches vertex v_n, Player 2 subsequently colors all vertices c_i red then Player 1 will be left with z_m to color and will end up winning; otherwise, Player 1 will be able to color some c_i blue. Since all clauses are satisfiable, some literal in C_i is true; therefore, there is an edge from c_i to some unvisited vertex u_j or $\ddot{u}_j$, besides the edge to y, so Player 1 colors the one left free by Player 2 and wins.

Conversely, suppose $\forall$ has a winning strategy in the QBF-game. Then Player

2 wins the WHEX-game in such a way: if $\forall$ gives value 1 (resp. 0) to variable x_{2i}, then Player 2 colors vertex $\ddot{x}_{2i}$ (resp. x_{2i}) red; this forces Player 1 to color x_{2i} (resp. $\ddot{x}_{2i}$) blue, and Player 2 to color u_{2i} (resp. $\ddot{u}_{2i}$) red, afterwards. When Player 1 reaches v_n, there is one clause C_i that is false, and, therefore, Player 2 forces Player 1 to color vertex c_i by coloring vertex z_i red. After Player 1 has colored vertex c_i blue, Player 2 colors y red, thus succeeding in blocking Player 1, since all other edges lead to vertices u_j already colored by Player 2.

Finally, we note that the instance (G, s, t) of WHEX, constructed above, is computable from the instance Φ of QBF using logarithmic space: an appropriate Turing machine M writes on its work tape the set of vertices V, which is obtained from the input of n variables and m clauses (each clause can have up to n literals) using binary notation, and, hence, taking $O(log(nm))$ storage space for this. Then the head(s) of M scan(s) the work tape and output(s) the edges. For this operation M only needs to write on the work tape all the (c_i, u_j) pairs ($1 \leq i \leq n$, $1 \leq j \leq m$), which, written in binary, take $O(log(nm))$ storage cells; it then checks if x_j (resp. $\neg x_j$) is in C_i, and, if that is the case, outputs the corresponding pair. ∎

3.2.2 A Normal Form Theorem for WHEX

We encode WHEX as a class of τ_2-structures as follows:

$$\text{WHEX} = \{\ \langle \mathcal{A}, u, v \rangle\ \in \text{STRUCT}(\tau_2) \mid \text{Player 1 has a winning strategy for the WHEX-game played on } (\mathcal{A}, u, v), \text{ with } u \text{ and } v \text{ in } |\mathcal{A}|\ \}.$$

Remark 3.2.1 The remark made on other possible encodings of HEX (Remark 3.1.1) also applies to WHEX. Also, it is easy to see that WHEX is firmly monotone. ∎

As classes of structures, WHEX $\neq$ HEX. To see this, consider the graph in Figure 6. Player 1 can always win the HEX-game from **s** to **t** on this graph but

looses the WHEX-game. For the HEX-game, Player 1 plays as follows: on his first move Player 1 colors either a_1 or b_1; let's say he colors a_1. If Player 2 doesn't color b_4, Player 1 colors b_4 and wins; otherwise, Player 1 colors a_4. Then, after Player 2's move, Player 1 colors b_2 or b_3, whichever is free, and wins. For the WHEX-game, after Player 1 has colored, say, a_1, Player 2 colors b_4, and on his next turn, Player 2 colors a_4; thus, blocking all possible paths of Player 1 from $\mathbf{s}$ to $\mathbf{t}$.

On the other hand, the graph in Figure 5 is an example where Player 1 always win the WHEX-game (because it corresponds to a satisfiable sentence), but looses the HEX-game: Player 2's strategy consists on coloring u_i (resp. $\ddot{u}_i$) if Player 1 colors x_i (resp. $\ddot{x}_i$); to color x_i (resp. $\ddot{x}_i$) if Player 1 colors u_i (resp. $\ddot{u}_i$); to color w_2 (resp. v_2) if Player 1 colors v_2 (resp. w_2); to color v_1 (resp. v_3) if Player 1 colors v_3 (resp. v_1), and anywhere else that Player 1 plays, Player 2 plays on a free v_i, a free u_i, or a free $\ddot{u}_i$. ∎

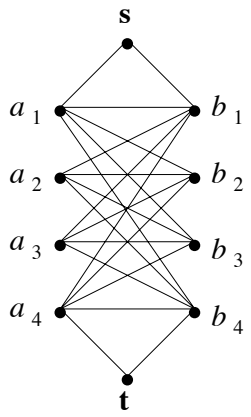

Figure 6: A yes-instance of HEX but not of WHEX.

We now show that the logic WHEX*[FO$_s$] has a projective normal form.

Theorem 3.2.1 *Let τ be some vocabulary. Every formula $\phi \in$ WHEX*[FO$_s(\tau)$] is equivalent to a formula of the form* WHEX$[\lambda\, \overline{x}\, \overline{y}\, \psi](\overline{0}, \overline{max})$*, where $\psi \in$ FO$_s(\tau)$,*

ψ projective and over the distinct k-tuples of variables $\overline{x}$ and $\overline{y}$, for some $k \geq 1$, and where $\overline{0}$ (resp. $\overline{max}$) is the constant symbol 0 (resp. max) repeated k times.

Proof: (Sketch) The proof runs along the same line as the proof of Theorem 3.1.1. We proceed by induction on the complexity of ϕ, beginning with ϕ as an atomic formula (positive or negative). As in Case 1 of Theorem 3.1.1, we introduce four distinct variables x_1, x_2, y_1, and y_2, not occurring in ϕ, and by similar argument we have that

$$\models \phi \iff \text{WHEX}[\lambda\,(x_1, y_1)(x_2, y_2)\,\phi]((0,0),(max, max)).$$

For the four other main cases we also need a gadget, namely $W(G)$, corresponding to an undirected graph $G = (V, E)$ with source s and sink t, to guarantee a winning strategy for Player 1 in a modified WHEX-game (played on all the vertices of G, including s and t), provided that Player 1 has a winning strategy in the WHEX-game on (G, s, t).

The required gadget $W(G)$ is constructed as follows: we draw a copy of G with two sources, s_1 and s_2, in place of s, and two sinks, t_1 and t_2, in place of t; s_1 (resp. s_2) is connected to each vertex that forms an edge with s in the original graph G; similarly, t_1 (resp. t_2) is connected to each vertex that forms an edge with t in G. We add a new vertex w, distinct from all the vertices in G, and draw edges between w and s_1, s_2, t_1, and t_2, respectively. The graph $W(G)$ is shown in Figure 7. Note that $W(G)$ is undirected.

By the WHEX-game on $W(G)$ we mean the following: either player can start the game, and the first move is to color either s_1 or s_2. If Player 2 starts, and let's say he colors s_1 red, then Player 1 colors s_2 blue and the game continues as in the usual WHEX-game on G; that is, Player 2 colors a vertex in an edge with s_2 (possibly w) red; Player 1 colors a different vertex in an edge with s_2 blue, etc. If Player 1 starts, and let's say he colors s_1 blue, then Player 2 continues as in the

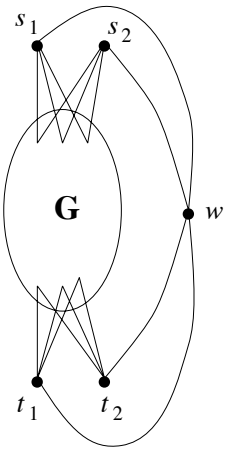

Figure 7: The graph $W(G)$.

usual WHEX-game on G; that is, he does not color s_2 but a vertex in an edge with s_1. Player 1 wins if he can reach, and color blue, one of the sinks t_1 or t_2.

In practice, it is obvious that Player 2 has no advantage by starting the WHEX-game on $W(G)$, so we may assume that it is Player 1 who always starts the WHEX-game on $W(G)$. With this remark in mind, we prove the following key lemma.

Lemma 3.2.2 *Player 1 has a winning strategy in the WHEX-game on (G, s, t) if and only if Player 1 has a winning strategy in the WHEX-game on $W(G)$.*

Proof: [$\Rightarrow$]: Player 1's strategy consists on first coloring s_1 blue. If Player 2 doesn't color w, Player 1 colors that vertex and wins in his next move; otherwise, Player 1 plays in G according to his winning strategy for the WHEX-game on (G, s, t), which is effective since he is the first one to color a vertex in $V - \{s, t\}$. Player 1 will then be able to construct a path of blue vertices through the copy of G up to a vertex with an edge to t_1 and an edge to t_2; hence, winning the WHEX-game on $W(G)$ on his next move.

[$\Leftarrow$]: Suppose Player 2 has a winning strategy in the WHEX-game on (G, s, t). Then, Player 2 wins the WHEX-game on $W(G)$ by playing as follows: after Player

1 has colored s_1 or s_2 blue, Player 2 colors w red. Then Player 1 has to color a vertex in the copy of G in an edge with s_1 (and s_2), and Player 2 continues coloring according to his winning strategy in the WHEX-game on (G, s, t), effectively blocking all possible paths that Player 1 tries to construct to either t_1 or t_2. ∎

The rest of the proof consists in stringing together copies of the gadget $W(G)$, as in cases 2 to 5 in the proof of Theorem 3.1.1, and writing down the projective formulas that describes each situation. We feel that it is unnecessary to write these formulas, and, so, we omit them, and only outline the construction for the three most important cases of the induction, namely the existential, the universal, and the nested case.

Existential case: $\phi := \exists z\, \mathrm{WHEX}[\lambda\, \overline{u}\, \overline{v}\, \theta(\overline{u}, \overline{v}, z)](\overline{0}, \overline{max})$, and we want to eliminate the $\exists$ quantifier, that is, to find a projective formula ψ such that

$$\models \phi \iff \mathrm{WHEX}[\lambda\, \overline{x}\, \overline{y}\, \psi](\overline{0}, \overline{max}).$$

The required formula ψ is the one that describes the edge relation of the graph pictured in Figure 8, which we have named G_ψ for further reference. For an arbitrary τ-structure $\mathcal{A}$ of size m, G_ψ consists of m subgraphs $W(G_1), \ldots, W(G_m)$, where each G_i is a graph described by $\theta(\cdot, \cdot, i-1)^{\mathcal{A}}$ $(i = 1, \ldots, m)$, and both sources (resp. sinks) in each $W(G_i)$ are joined to vertex $\mathbf{0}$ (resp. $\mathbf{max}$).

Suppose Player 1 has a winning strategy in the WHEX-game on $(G_i, 0, max)$, for some $i \in \{1, \ldots, m\}$. Then, by Lemma 3.2.2, Player 1 has a winning strategy in the WHEX-game on $W(G_i)$, and, using this strategy, Player 1 wins the WHEX-game on $(G_\psi, \mathbf{0}, \mathbf{max})$.

Conversely, suppose Player 2 has a winning strategy in the WHEX-game on $(G_i,\ 0,\ max)$, for each $i \in \{1, \ldots, m\}$. Then, by Lemma 3.2.2, Player 2 has a winning strategy in the WHEX-game on $W(G_i)$, for each $i \in \{1, \ldots, m\}$. Hence, in the WHEX-game on $(G_\psi, \mathbf{0}, \mathbf{max})$, Player 2 uses these strategies to effectively

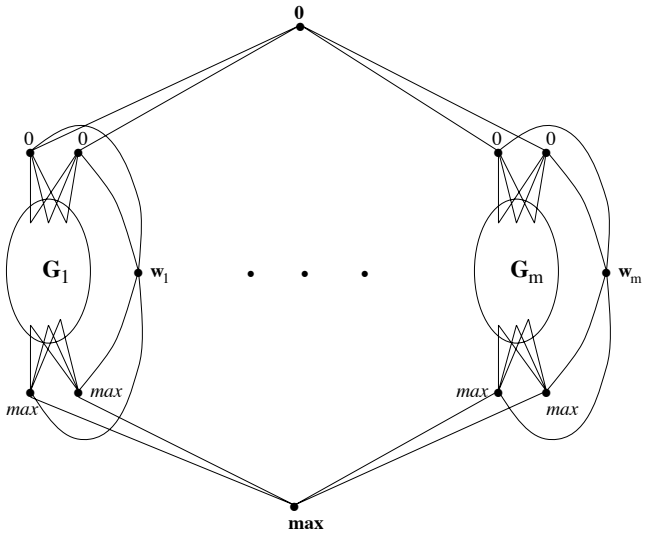

Figure 8: The existential case.

block Player 1 in any subgraph $W(G_i)$ and, so, Player 2 has a winning strategy in the WHEX-game on $(G_\psi, \mathbf{0}, \mathbf{max})$.

Universal case: $\phi := \forall z\, \mathrm{WHEX}[\lambda\, \overline{u}\, \overline{v}\, \theta(\overline{u}, \overline{v}, z)](\overline{0}, \overline{max})$. This time the required graph G_ψ is a series of subgraphs $W(G_1)$, ..., $W(G_m)$ linked as shown in Figure 9.

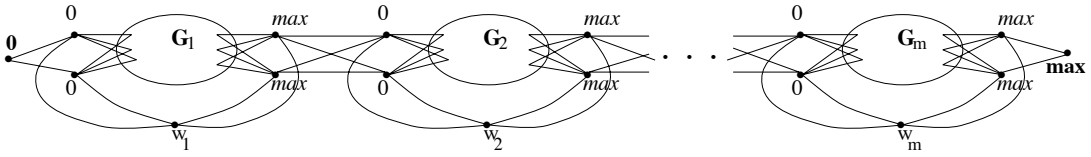

Figure 9: The universal case.

Once again, it follows from Lemma 3.2.2 that Player 1 has a winning strategy in the WHEX-game on $(G_i, 0, max)$, for all $i \in \{1, \ldots, m\}$, if and only if Player 1 has winning strategy in the WHEX-game on $(G_\psi, \mathbf{0}, \mathbf{max})$.

The nested case: we want to simplify a formula of the form

$$\mathrm{WHEX}[\lambda\,\overline{x}\,\overline{y}\,\mathrm{WHEX}[\lambda\,\overline{u}\,\overline{v}\,\theta(\overline{u},\overline{v},\overline{x},\overline{y})](\overline{0},\overline{max})](\overline{0},\overline{max});$$

therefore, for a structure $\mathcal{A}$ of size m, we arrange $m \times m$ graphs $W(G_{i,j})$ corresponding to graphs defined by $\theta(\cdot,\cdot,\overline{x}_i,\overline{y}_j)^{\mathcal{A}}$, for $i,j \in \{0,\ldots,m-1\}$, strung together as in the nested case of Theorem 3.1.1 (cf. Figure 4). Then the required formula ψ is the one that describes this construction, and one can show, using Lemma 3.2.2, that Player 1 has a winning strategy in the WHEX-game on the graph $\psi^{\mathcal{A}}$, with source $\mathbf{0}^{\mathcal{A}}$ and sink $\mathbf{max}^{\mathcal{A}}$, if and only if there exists a sequence of m pairs $(\overline{x}_0,\overline{y}_{j_0})$, $(\overline{x}_1,\overline{y}_{j_1})$, $\ldots$, $(\overline{x}_{m-1},\overline{y}_{j_{m-1}})$, in $|\mathcal{A}|^{2k}$, such that Player 1 has a winning strategy in the WHEX-game on each graph determined by $\theta(\cdot,\cdot,\overline{x}_h,\overline{y}_{j_h})^{\mathcal{A}}$ ($j_h \in \{0,\ldots,m-1\}$, $0 \le h \le m-1$), with source $0^{\mathcal{A}}$ and sink $max^{\mathcal{A}}$. ∎

Corollary 3.2.3 $\mathbf{PSPACE} = \pm\mathrm{WHEX}^*[\mathrm{FO}_s] = \mathrm{WHEX}^*[\mathrm{FO}_s] = \mathrm{WHEX}^1[\mathrm{FO}_s]$ *and* WHEX *is complete for* **PSPACE** *via first-order projections.*

Proof: We have shown that WHEX is complete for **PSPACE** via logspace reducibility. The containment $\mathrm{DTC}^1[\mathrm{FO}_s] \le \mathrm{WHEX}^*[\mathrm{FO}_s]$ can be proven using the same formula ϕ in the proof of Theorem 3.1.3. Then we apply Corollary 2.4.5 and Theorem 3.2.1. ∎

We also have a result analogous to Corollary 3.1.6.

Corollary 3.2.4 WHEX *is complete for* $\mathrm{WHEX}^*[\mathrm{FO}_s^+]$ *via* monotone *first-order projections.* ∎

3.2.3 WHEX and $\mathbf{L}^{\omega}_{\infty\omega}$

In this section we show that $\mathrm{WHEX}^*[\mathrm{FO}] \le \mathbf{PFP}$, where **PFP** is the closure of FO under the operation of taking partial fixed points. Since $\mathbf{PFP} \le \mathrm{L}^{\omega}_{\infty\omega}$, we

conclude that WHEX*[FO] $\leq L^{\omega}_{\infty\omega}$ also. Note that we have removed the successor relation s, so we are working on unordered structures.

Partial Fixed Point logic has been widely studied and many of its important properties are nicely presented in [EF95], so we refer the interested reader to that book for details and only sketch here the few results regarding **PFP** that we need. We start by reviewing its semantics:

If R is a k-ary relation, σ some vocabulary, $\phi(\overline{x}, X)$ a $\sigma \cup \{R\}$-formula (with $|\overline{x}| = k$ and X a k-ary relational variable), and $\mathcal{A}$ a σ-structure, then

$$\mathcal{A} \models PFP[\lambda\, \overline{x}, X\, \phi](\overline{u}) \text{ means } \overline{u}^{\mathcal{A}} \in X^{\mathcal{A}}_{\infty},$$

where $X^{\mathcal{A}}_{\infty}$ is the fixed point (if it exists) of the following sequence of subsets of $|\mathcal{A}|^k$ (also known as *stage sets*):

$$\begin{aligned} X^{\mathcal{A}}_1 &= \emptyset, \\ X^{\mathcal{A}}_{i+1} &= \{\overline{a} \in |\mathcal{A}|^k \mid \langle \mathcal{A}, R^{\mathcal{A}}, \overline{a} \rangle \models \phi(\overline{x}, X^{\mathcal{A}}_i)\} \text{ for } i \geq 1. \end{aligned}$$

The canonical example is the definition of the transitive closure of the binary relation E; we give it here not for lack of other examples but because we do need it later.

Example 3.2.1 The formula TC$[\lambda\, u\, v\, E(u,v)](a,b)$ is equivalent to the following **PFP**-formula:

$$PFP[\lambda\, u\, v, R\ (u = v \vee \exists z(E(u,z) \wedge R(z,v)))](a,b).$$

■

Theorem 3.2.5 WHEX*[FO] $\leq$ **PFP**.

Proof: Let C_1 and C_2 be two constant symbols. It is sufficient to show that there exists a (successor-free) $\tau_2 \cup \{C_1, C_2\}$-sentence $\Phi \in$ **PFP** such that, as a class of finite structures, WHEX$(C_1, C_2) = MOD(\Phi)$. Let

$$\begin{aligned}\Phi \;:=\; & PFP[\lambda\, x, X\,((E(C_1,x) \vee \exists y \in X \exists z \notin X(E(y,x) \wedge E(y,z) \wedge x \neq z \wedge \\ & z \neq C_2) \vee\ \exists y \in X(E(y,x) \wedge x = C_2)) \wedge \\ & \mathrm{TC}[\lambda\, u\, v\, E(u,v)](x, C_2))](C_2).\end{aligned}$$

By the preceding example, Φ is a **PFP**-formula. Let $\mathcal{A} \in \mathrm{STRUCT}(\tau_2 \cup \{C_1, C_2\})$; we must show

$$\mathcal{A} \in \mathrm{WHEX}(C_1, C_2) \ \text{ iff } \ \mathcal{A} \models \Phi.$$

Suppose $\mathcal{A} \in \mathrm{WHEX}(C_1, C_2)$. Then Player 1 has a winning strategy in the WHEX-game on $(\mathcal{A}, C_1, C_2)$ and, hence, at the end of the game there will be a path $\mathcal{P}$ from $C_1^{\mathcal{A}}$ to $C_2^{\mathcal{A}}$ in $\mathcal{A}$ such that:

- every vertex in $\mathcal{P}$ has Player 1's color;
- there is a vertex a in $\mathcal{P}$ with an edge to $C_1^{\mathcal{A}}$;
- for each a in $\mathcal{P}$, with $a \neq C_1^{\mathcal{A}}$ and $(\neg E(a, C_2))^{\mathcal{A}}$, there is a b of Player 2's color such that $\mathcal{A} \models E(a,b)$ and $\mathcal{A} \models \neg E(c,b)$ for c in $\mathcal{P}$ with an edge to a (this is by the "no triangle" rule).

On the other hand, we can assume, without loss of generality, that each vertex colored by the players in the course of a game is on a path to $C_2^{\mathcal{A}}$. This is because, by the nature of the WHEX-game, either player gains nothing by coloring nodes outside a path to $C_2^{\mathcal{A}}$. Thus, the formula $\mathrm{TC}[\lambda\, u\, v\, E(u,v)](x, C_2)$ holds for all the vertices colored by the players.

All of the above give: the stage set $X_1^{\mathcal{A}} \neq \emptyset$; successively, since the formula $\exists y \in X \exists z \notin X(E(y,x) \wedge E(y,z) \wedge x \neq z \wedge z \neq C_2)$ holds in $\mathcal{P}$, we have $X_2^{\mathcal{A}} \neq \emptyset$, $X_3^{\mathcal{A}} \neq \emptyset, \ldots$, until $C_2^{\mathcal{A}}$ is reached in $\mathcal{P}$ then the formula $\exists y \in X_{i_0}(E(y,x) \wedge x = C_2)$ holds for some $i_0 > 1$, and $C_2^{\mathcal{A}} \in X_{i_0+1}$. It is possible that we could carry on

defining further stages, but that we eventually reach an index j such that

$$X_j^{\mathcal{A}} = X_{j+1}^{\mathcal{A}} = \cdots = X_\infty^{\mathcal{A}} = \{C_2^{\mathcal{A}}\}$$

follows from the fact that each element in each stage set $X_i^{\mathcal{A}}$ is on a path to $C_2^{\mathcal{A}}$. Thus $\mathcal{A} \models \Phi$.

Suppose $\mathcal{A} \notin \mathrm{WHEX}(C_1, C_2)$. Then Player 2 has a winning strategy in the WHEX-game on $(\mathcal{A}, C_1, C_2)$. This means that any path, starting at $C_1^{\mathcal{A}}$, that Player 1 tries to constructs will eventually be blocked by Player 2 before reaching $C_2^{\mathcal{A}}$. Hence:

- $(\neg E(C_1, C_2))^{\mathcal{A}}$ (no trivial case), therefore $C_2^{\mathcal{A}} \notin X_1^{\mathcal{A}}$;
- for all $i > 0$ $(\forall y \in X_i \forall x(E(y,x) \longrightarrow x \neq C_2))^{\mathcal{A}}$ ($C_2^{\mathcal{A}}$ can't be reached from any vertex colored by Player 1);
- there is an i_0 such that for all $i \geq i_0$,
  $(\forall y \in X_i \exists z \notin X_i(E(y,z) \wedge z \neq C_2 \wedge \forall x(E(y,x) \longrightarrow x = z)))^{\mathcal{A}}$.

Therefore, for all $i \geq i_0$, $C_2^{\mathcal{A}} \notin X_i^{\mathcal{A}}$, so $C_2^{\mathcal{A}} \notin X_\infty^{\mathcal{A}}$, and $\mathcal{A} \not\models \Phi$. ∎

3.3 Normal forms in the absence of order

We now study the normal form property for our logics without the successor relation. We will show that a projective normal form does not exists in this case, and to do that we employ an extended version of the Ehrenfeucht-Fraïssé games for first-order logic. In this section τ denotes a vocabulary $\{R_1, \ldots, R_r, C_1, \ldots, C_c\}$, where each R_i is a relation symbol of arity a_i and each C_j is a constant symbol.

3.3.1 Generalized Ehrenfeucht-Fraïssé games

We give a variation of the Ehrenfeucht-Fraïssé games presented in [Ste96], suitable for logics defined by generalized quantifiers that are not necessarily monotone.

Definition 3.3.1 Let Ω be a problem over the vocabulary τ. Let $\mathcal{A}$ and $\mathcal{B}$ be two structures over the same vocabulary. The Ω-game of n moves on $\mathcal{A}$ and $\mathcal{B}$ is played by the usual suspects, Spoiler and Duplicator, who alternate in pebbling n elements in each of these structures, according to the following rules (Spoiler is always the first to move).

- $\exists$-move: Spoiler plays on $\mathcal{A}$. He does so by placing an unused pebble u_i on an element of $\mathcal{A}$. Duplicator responds by placing a pebble v_i on an element of $\mathcal{B}$.

- $\forall$-move: Similar to the $\exists$-move, but Spoiler plays on $\mathcal{B}$ and Duplicator plays on $\mathcal{A}$.

- Ω_k-move: Spoiler begins by playing on $\mathcal{A}$. He selects a τ-structure $S_{\mathcal{A}} \in \Omega$ with $|S_{\mathcal{A}}| = |\mathcal{A}|^k$, the tuples from $|\mathcal{A}|^k$ determining the constants $C_1^{S_{\mathcal{A}}}, \ldots, C_c^{S_{\mathcal{A}}}$ consists only of previously pebbled elements of $|\mathcal{A}|$ or constants of $\mathcal{A}$. Duplicator replies in $\mathcal{B}$ by selecting $S_{\mathcal{B}} \in \Omega$ with $|S_{\mathcal{B}}| = |\mathcal{B}|^k$, the tuples from $|\mathcal{B}|^k$ determining the constants $C_1^{S_{\mathcal{B}}}, \ldots, C_c^{S_{\mathcal{B}}}$ of $S_{\mathcal{B}}$ consists of tuples corresponding to those selected by Spoiler. Then Spoiler chooses some $i \in \{1, \ldots, r\}$ and places ka_i new pebbles on some tuple $\overline{u}_i \in |\mathcal{B}|^{ka_i}$. Duplicator replies by placing ka_i pebbles on some tuple $\overline{v}_i \in |\mathcal{A}|^{ka_i}$ such that $\overline{v}_i \in R_i^{S_{\mathcal{A}}}$ iff $\overline{u}_i \in R_i^{S_{\mathcal{B}}}$.

- $\neg\Omega_k$-move: Similar to the Ω_k-move but with the roles of $\mathcal{A}$ and $\mathcal{B}$ reversed.

As usual, Duplicator wins if and only if, at the end of the game (i.e., after the nth move), the elements in $\mathcal{A}$ and in $\mathcal{B}$, where pebbles were placed on, determine a partial isomorphism of $\mathcal{A}$ into $\mathcal{B}$. ∎

In general, the goal of games such as Ehrenfeucht-Fraïssé's is to give a combinatorial characterization of elementary equivalence. Therefore, the *length* of a game (i.e.,

the number of moves) is determined by the quantifier rank of a formula, and the type of move is determined by the quantifier that we want to "eliminate" in the formula, beginning with the outermost quantifier.

The games presented here are only sufficient for describing elementary equivalence in $\pm\Omega^*$[FO] for arbitrary Ω, but that is really all we need for the applications described in this thesis. Although we are now working on unordered structures, the presence or absence of successor relation is irrelevant for the proof below.

Theorem 3.3.2 *Let Ω be some non empty problem over τ, n some positive integer and let $\mathcal{A}$ and $\mathcal{B}$ be two σ-structures for some vocabulary σ . If Duplicator has a winning strategy in the Ω-game of n moves on $\mathcal{A}$ and $\mathcal{B}$ then, for any σ-sentence $\theta \in \pm\Omega^*$[FO] with $qr(\theta) \leq n$, $\mathcal{A} \models \theta$ implies $\mathcal{B} \models \theta$.*

Proof: Suppose that there exists a σ-sentence $\theta \in \pm\Omega^*$[FO] with $qr(\theta) \leq n$, such that $\mathcal{A} \models \theta$ and $\mathcal{B} \models \neg\theta$. We describe a winning strategy for Spoiler in the Ω-game of n moves on $\mathcal{A}$ and $\mathcal{B}$. We do this by induction on n. The cases in which θ is quantifier-free (i.e., $n = 0$) or existential or universal are resolved easily and identically as is done, for example, in [Ste96]. We tackle the case when $\theta := \Omega[\lambda\,\overline{x}_1\phi_1, \ldots, \overline{x}_r\phi_r, \overline{y}_1\psi_1, \ldots, \overline{y}_c\psi_c](\overline{C})$ (here $\overline{C}$ is a tuple of constant symbols).

Since $\mathcal{A} \models \theta$ the $(\phi_1, \ldots, \phi_r, \psi_1, \ldots, \psi_c)$-translation $T_{\mathcal{A}}$ of $\mathcal{A}$ is a τ-structure of Ω. Spoiler chooses $S_{\mathcal{A}} = T_{\mathcal{A}}$. Since $\mathcal{B} \models \neg\theta$ whichever τ-structure $S_{\mathcal{B}}$ (with universe $|\mathcal{B}|^k$) Duplicator selects in Ω, its $(\phi_1, \ldots, \phi_r, \psi_1, \ldots, \psi_c)$-translation $T_{\mathcal{B}}$ is not in Ω. Therefore $S_{\mathcal{B}} \ncong T_{\mathcal{B}}$. Hence, there is some $i \in \{1, \ldots, r\}$ and some $\overline{u}_i \in |\mathcal{B}|^{ka_i}$ such that

$$\overline{u}_i \in R_i^{S_{\mathcal{B}}} \text{ and } \overline{u}_i \notin R_i^{T_{\mathcal{B}}},$$

or

$$\overline{u}_i \notin R_i^{S_{\mathcal{B}}} \text{ and } \overline{u}_i \in R_i^{T_{\mathcal{B}}}.$$

Spoiler pebbles the tuple $\overline{u}_i$. Now, regardless of whichever tuple $\overline{v}_i \in |\mathcal{A}|^{ka_i}$ Duplicator selects such that

$$\overline{v}_i \in R_i^{T_\mathcal{A}} \text{ iff } \overline{u}_i \in R_i^{S_\mathcal{B}},$$

we can not have

$$\overline{v}_i \in R_i^{T_\mathcal{A}} \text{ iff } \overline{u}_i \in R_i^{T_\mathcal{B}}.$$

Hence $\langle \mathcal{A}, \overline{v}_i \rangle \models \phi_i(\overline{x}_i)$ and $\langle \mathcal{B}, \overline{u}_i \rangle \models \neg\phi_i(\overline{x}_i)$. By induction, Spoiler has a winning strategy in the remainder of the game. ∎

3.3.2 A separation theorem

Definition 3.3.3 Let Ω be a problem over τ. A universal (or Π_1) hierarchy inside the logic Ω^*[FO] is defined inductively as follows:

- $\Omega(0)$ is the set of formulas of the form $\Omega[\lambda \overline{x}_1\phi_1, \ldots, \overline{x}_r\phi_r, \overline{y}_1\psi_1, \ldots, \overline{y}_c\psi_c](\overline{z})$, where each ϕ_i and each ψ_j are in FO;
- $\forall\Omega(m)$ is the universal closure of $\Omega(m)$, i.e., the set of formulas of the form $\forall x\psi(x)$ where $\psi \in \Omega(m)$;
- $\Omega(m+1)$ is the set of formulas of the form $\Omega[\lambda \overline{x}_1\phi_1, \ldots, \overline{x}_r\phi_r, \overline{y}_1\psi_1, \ldots, \overline{y}_c\psi_c](\overline{z})$, where each ϕ_i and each ψ_j are in $\forall\Omega(m)$.

∎

Remark 3.3.1 The corresponding generalized Ehrenfeucht-Fraïssé game for these logics $\Omega(m)$ (resp. $\forall\Omega(m)$), consists of $m+1$ (positive) Ω-moves that alternate with m (resp. $m+1$) $\forall$-moves, beginning with a Ω-move (resp. $\forall$-move), and then followed by a usual first-order game. We refer to these games as $\Omega(m)$-games (resp. $\forall\Omega(m)$-games).

∎

We prove that if Ω is firmly monotone then a single Ω_l-move is not enough for Spoiler to distinguish between certain structures that are already indistinguishable with respect to first-order formulas of bounded quantifier rank.

Theorem 3.3.4 *Let Ω be a firmly monotone problem over τ and let σ be some other vocabulary. If there exists families of σ-structures, $\{\mathcal{A}_k\}_{k\geq 0}$ and $\{\mathcal{B}_k\}_{k\geq 0}$, such that:*

(i) $||\mathcal{A}_k|| = f(k)$ *where f is a non decreasing function on positive integers;*

(ii) *for each k, $\mathcal{A}_k$ is isomorphic to a substructure of $\mathcal{B}_k$, and*

(iii) *for all sentence $\theta \in \mathrm{FO}(\sigma)$ and k such that $f(k) \geq qr(\theta)$*

$$\mathcal{A}_k \models \theta \text{ implies } \mathcal{B}_k \models \theta$$

then, for all σ-sentence $\theta \in \Omega(0)$ and k such that $f(k) \geq qr(\theta)$,

$$\mathcal{A}_k \models \theta \text{ implies } \mathcal{B}_k \models \theta\,.$$

Proof: Let $\theta := \Omega[\lambda \overline{x}_1\phi_1, \ldots, \overline{x}_r\phi_r, \overline{y}_1\psi_1, \ldots, \overline{y}_c\psi_c](\overline{C})$, where, for some $l > 0$, each $|\overline{x}_i| = la_i$, each $|\overline{y}_j| = l$, $\overline{C}$ is a tuple of constant symbols, and $\phi_1, \ldots, \phi_r$, ψ_1, $\ldots, \psi_c$, are first-order formulas. Recall that $qr(\theta) := max[la_i + qr(\phi_i),\ l + qr(\psi_j)\ :\ 1 \leq i \leq r\ ;\ 1 \leq j \leq c]$. Let k such that $f(k) \geq qr(\theta)$; let $\mathcal{A} := \mathcal{A}_k$ and $\mathcal{B} := \mathcal{B}_k$, and assume $\mathcal{A} \models \theta$. We have to show $\mathcal{B} \models \theta$. For that it is sufficient to show that Duplicator has a winning strategy in the Ω-game of $qr(\theta)$ many moves consisting of *one* Ω_l-move followed by $qr(\theta) - 1$ $\exists$- and $\forall$-moves. By hypothesis (item (iii)), Duplicator has a winning strategy in all but the first move of the game (the Ω_l-move), so we need only to describe a winning strategy for this part of the game. This would be achieved if we can show that, for each $S_{\mathcal{A}} \in \Omega$ with $|S_{\mathcal{A}}| = |\mathcal{A}|^l$, there exists $S_{\mathcal{B}} \in \Omega$ with $|S_{\mathcal{B}}| = |\mathcal{B}|^l$ such that for each $i \in \{1, \ldots, r\}$ and each tuple $\overline{u}_i \in |\mathcal{B}|^{la_i}$, there exists a tuple $\overline{v}_i \in |\mathcal{A}|^{la_i}$ such that $\overline{u}_i \in R_i^{S_{\mathcal{B}}}$ iff $\overline{v}_i \in R_i^{S_{\mathcal{A}}}$.

So, let $S_{\mathcal{A}} \in \Omega$ with $|S_{\mathcal{A}}| = |\mathcal{A}|^l$ and $C_1^{S_{\mathcal{A}}}, \ldots, C_c^{S_{\mathcal{A}}}$ be the interpretations, in $S_{\mathcal{A}}$, of the constant symbols. Let $\pi : \mathcal{A} \hookrightarrow \mathcal{B}$ be the isomorphism of $\mathcal{A}$ onto a

substructure of $\mathcal{B}$. Let $S_{\pi(\mathcal{A})}$ be the isomorphic copy of $S_{\mathcal{A}}$ through the isomorphism π (so, $|S_{\pi(\mathcal{A})}| = \pi(|S_{\mathcal{A}}|) \subseteq |\mathcal{B}|^l$, $R_i^{S_{\pi(\mathcal{A})}} = \pi(R_i^{S_{\mathcal{A}}})$, and $C_j^{S_{\pi(\mathcal{A})}} = \pi(C_j^{S_{\mathcal{A}}})$). Then $S_{\pi(\mathcal{A})}$ is in Ω, since Ω is closed under isomorphisms.

For each $i \in \{1, \ldots, r\}$ and each $\pi^{-1}(\overline{u}) \in |\mathcal{A}|^{la_i}$ such that $R_i^{S_{\pi(\mathcal{A})}}(\overline{u})$ holds, define

$$\Phi_i(\overline{u}) := \{\phi(\overline{x}_i) \mid \phi \in \mathrm{FO}(\sigma),\ |\overline{x}_i| = la_i,\ qr(\phi) \leq qr(\phi_i),\ \ \langle \mathcal{A}, \pi^{-1}(\overline{u})\rangle \ \models \phi(\overline{x}_i)\}.$$

Let

$$\alpha_i(\overline{x}_i) := \bigvee \{ \bigwedge \Phi_i(\overline{u}) \mid \pi^{-1}(\overline{u}) \in |\mathcal{A}|^{la_i} \ \text{ and } R_i^{S_{\pi(\mathcal{A})}}(\overline{u}) \text{ holds } \}.$$

Note that $\alpha_i \in \mathrm{FO}(\sigma)$ and

$$\mathcal{A} \models \exists \overline{x}_i \ \alpha_i(\overline{x}_i).$$

Also that

$$qr(\exists \overline{x}_i \, \alpha_i) = la_i + qr(\alpha_i) \leq la_i + qr(\phi_i) \leq f(k);$$

hence, by hypothesis

$$\mathcal{B} \models \exists \overline{x}_i \ \alpha_i(\overline{x}_i).$$

Let

$$S_{\mathcal{B}} = \ \langle |S_{\mathcal{B}}|, R_1^{S_{\mathcal{B}}}, \ldots, R_r^{S_{\mathcal{B}}}, C_1^{S_{\mathcal{B}}}, \ldots, C_c^{S_{\mathcal{B}}} \rangle \ ,$$

where

$$|S_{\mathcal{B}}| = |\mathcal{B}|^l \ , \ C_i^{S_{\mathcal{B}}} = C_i^{S_{\pi(\mathcal{A})}} \text{ and}$$

$$R_i^{S_{\mathcal{B}}} = \{\overline{v} \in |S_{\mathcal{B}}|^{a_i} \mid \ \langle \mathcal{B}, \overline{v}\rangle \ \models \alpha_i(\overline{x}_i)\}.$$

- $S_{\mathcal{B}} \in \Omega$:

  $|S_{\pi(\mathcal{A})}| \subseteq |\mathcal{B}|^l$, and for each $\overline{u} \in |S_{\pi(\mathcal{A})}|^{la_i}$ such that $R_i^{S_{\pi(\mathcal{A})}}(\overline{u})$ holds, we have that $\langle \mathcal{A}, \pi^{-1}(\overline{u})\rangle \ \models \alpha_i(\overline{x}_i)$. Then $\langle \pi(\mathcal{A}), \overline{u}\rangle \ \models \alpha_i(\overline{x}_i)$, where $\pi(\mathcal{A})$ is the substructure of $\mathcal{B}$ isomorphic to $\mathcal{A}$. On the other hand, $\mathcal{B} \models \exists \overline{x}_i \alpha_i(\overline{x}_i)$ since $\mathcal{A} \models \exists \overline{x}_i \alpha_i(\overline{x}_i)$. Therefore $\langle \mathcal{B}, \overline{u}\rangle \ \models \alpha_i(\overline{x}_i)$, so $R_i^{S_{\mathcal{B}}}(\overline{u})$ holds.

  Since $S_{\pi(\mathcal{A})} \in \Omega$ and Ω is firmly monotone, we get the claim.

- $S_{\mathcal{B}}$ is the required structure for Duplicator to win:
  for each $i \in \{1, \ldots, r\}$ and tuple $\overline{u}_i \in |\mathcal{B}|^{la_i}$ we have

  $\overline{u}_i \in R_i^{S_{\mathcal{B}}} \iff \mathcal{B} \models \alpha_i(\overline{u}_i) \iff \mathcal{A} \models \exists \overline{x}_i \ \alpha_i(\overline{x}_i) \iff$ for some $\overline{v} \in |S_{\pi(\mathcal{A})}|^{a_i}$, $\mathcal{A} \models \bigwedge \Phi_i(\pi^{-1}(\overline{v})) \iff \overline{v} \in R_i^{S_{\pi(\mathcal{A})}} \iff \pi^{-1}(\overline{v}) \in R_i^{S_{\mathcal{A}}}$ (and $\pi^{-1}(\overline{v}) \in |\mathcal{A}|^{la_i}$).

This completes the proof. ∎

Theorem 3.3.5 *Let Ω be a firmly monotone problem over τ and let*

$$\Phi := \forall u \forall v \, \mathrm{DTC}[\lambda x \, y \, E(x, y)](u, v) \, .$$

If there exists some sentence Θ in $\forall\Omega(0)$ logically equivalent to Φ then

$$\Omega(0) \subsetneq \forall\Omega(0).$$

Proof: For each positive integer k consider the following τ_2-structures: $\mathcal{A}_k$ a cycle of length 2^{k+2} with interpretations for the two constant symbols 0 and max, and $\mathcal{B}_k$ the disjoint union of $\mathcal{A}_k$ with another cycle of length 2^{k+2} without constants. Then $\mathcal{A}_k$ and $\mathcal{B}_k$ satisfy the hypothesis of Theorem 3.3.4, and since Ω is firmly monotone, we have that these structures satisfy the same τ_2-sentences in $\Omega(0)$. However,

$$\mathcal{A}_k \models \Phi \text{ and } \mathcal{B}_k \models \neg\Phi$$

and, by hypothesis,

$$\mathcal{A}_k \models \Theta \text{ and } \mathcal{B}_k \models \neg\Theta.$$

From this it follows that $\Omega(0)$ is properly contained in $\forall\Omega(0)$. ∎

Corollary 3.3.6 $\mathrm{HEX}(0) \subsetneq \forall\mathrm{HEX}(0)$.

Proof: HEX is firmly monotone. On the other hand, by Corollary 3.1.4, we have

$$\begin{aligned} \models \quad & \mathrm{DTC}[\lambda\,\overline{x}\,\overline{y}\,\theta(\overline{x},\overline{y})](\overline{0},\overline{max}) \iff \\ & \mathrm{HEX}[\lambda\,\overline{x}'\,\overline{y}'\,\phi(\overline{x}',\overline{y}')](\overline{0},\overline{max}), \end{aligned}$$

for some ϕ in FO. Hence

$$\begin{aligned} \models \quad & \forall\overline{u}\forall\overline{v}\,\mathrm{DTC}[\lambda\,\overline{x}\,\overline{y}\,\theta(\overline{x},\overline{y})](\overline{u},\overline{v}) \iff \\ & \forall\overline{u}\forall\overline{v}\,\mathrm{HEX}[\lambda\,\overline{x}'\,\overline{y}'\,\phi(\overline{x}',\overline{y}')](\overline{u},\overline{v}). \end{aligned}$$

We have fulfilled all the hypothesis of Theorem 3.3.5, and hence, $\mathrm{HEX}(0) \subsetneq \forall\mathrm{HEX}(0)$.

■

Corollary 3.3.7 $\mathrm{WHEX}(0) \subsetneq \forall\mathrm{WHEX}(0)$.

Proof: The proof is analogous to that of Corollary 3.3.6, and uses the fact that $\mathrm{DTC}^1[\mathrm{FO}] \leq \mathrm{WHEX}^*[\mathrm{FO}]$, which is proven with the same formula ϕ in the proof of Theorem 3.1.3. ■

The results of this section were inspired by the work of Grädel in [Grä91], where he shows the existence of a strict hierarchy of sublogics of $\mathrm{TC}^*[\mathrm{FO}]$, obtained by interleaving the $\forall$ quantifier with the TC quantifier (our Definition 3.3.3 is a generalization of Grädel's definition of these TC-logics). Grädel's main tool is a TC-game which consists of $\exists$- and $\forall$-moves, as in Definition 3.3.1, and a TC-move defined as follows:

> For some $l \geq 0$, Spoiler selects a sequence $\overline{a}_0, \overline{a}_1, \ldots, \overline{a}_m$ of l-tuples in $|\mathcal{A}|$ such that $\overline{a}_0$ and $\overline{a}_m$ consist only of constants and already pebbled elements. Duplicator replies in $\mathcal{B}$ by selecting a sequence $\overline{b}_0, \overline{b}_1, \ldots,$

> $\overline{b}_n$ of l-tuples in $|\mathcal{B}|$, with $\overline{b}_0$ and $\overline{b}_n$ being in correspondence with $\overline{a}_0$ and $\overline{a}_m$ respectively, and possibly $n \neq m$. Then Spoiler pebbles two consecutive l-tuples $\overline{b}_i$ and $\overline{b}_{i+1}$, for some $0 \leq i < n$, and Duplicator responds by pebbling two consecutive l-tuples $\overline{a}_j$ and $\overline{a}_{j+1}$, for some $0 \leq j < m$.

However, we noted, that at the heart of the proof of the separation result in [Grä91], there is used a rather nice property of locality that few problems except TC enjoys: namely, that in order to verify that a given sequence $\overline{a}_0$, $\overline{a}_1$, ..., $\overline{a}_m$ does indeed define a path from $\overline{a}_0$ to $\overline{a}_m$, we only have to check that every two consecutive elements of the sequence are in the edge relation E, *without taking into account any other element of the sequence.* Note that, for example, this is not true of the Hamiltonian path problem HP. Having realized this, it seemed to us that the TC-game (and its generalization, the Generalized Ehrenfeucht-Fraïssé game) is too powerful as a tool, for it allows the players to construct a full path in one move, when they could really do this in a stepwise manner through a sequence of "local" moves, i.e., by defining just one or two tuples of elements at one time. Based on these observations, we sought to modify Grädel's technique in order to explore its possible generalization to other logics, and ultimately to our logics HEX*[FO] and WHEX*[FO]. We summarize our findings in the next chapter.

Chapter 4

Program Schemes and Hierarchies of Logics

We present a model of computation, namely, program schemes, that, although not new, has not played such a popular role as the Turing machine model has played in Complexity Theory. However, for the purpose of analyzing a computational problem in terms of its expressibility in certain logics, this model is better suited, because instead of requiring that its input be encoded as a string of, say, 0's and 1's, it treats it as a finite structure over some vocabulary (see [Ste93] and the references therein for further discussion on this model). On the other hand, this model has a syntax very much like a Pascal pseudo-code, making it much easier to read and write than a Turing machine program. Thus, program schemes seem to present a nice link between Computational Complexity and Descriptive Complexity, and we illustrate this by translating a result, due to Grädel [Grä91], on the existence of a strict hierarchy of logics inside TC*[FO] to a theorem that states the existence of a strict hierarchy of certain classes of program schemes that we define. We then show how our Hierarchy Theorem for program schemes gives the TC-hierarchy in [Grä91] and in similar hierarchies inside other logics as, for example, DTC*[FO].

Throughout this chapter we work on unordered structures. Our notation and basic definition of program schemes is derived from [Ste93].

4.1 Program schemes

4.1.1 Syntax and semantics

Let τ be some vocabulary. The class NPS(τ) (i.e., Nondeterministic Program Schemes over τ), has as its elements finite sequences of instructions, where the first is an *input instruction*, the last is an *output instruction*, and the rest are *assignment instructions* and *test instructions* built up from *atoms*, the relations of τ, and using the logical connectives $\neg$, $\vee$, and $\wedge$ (no quantifiers). More specifically:

(i) an atom of NPS(τ) is any variable or constant symbol in $\tau \cup \{0, max\}$;

(ii) an *assignment instruction* of NPS(τ) has the form:

$$var := atom, \quad \text{or}$$

$$guess(var),$$

where var denotes some variable and $atom$ denotes some atom;

(iii) the *test instructions* of NPS(τ) are defined, inductively, as follows:
First, let the $while(0)$ instructions be of the form

WHILE t DO I OD

where I is a finite sequence of assignment instructions and t is an *atomic test*, that is, a conjunction or disjunction of *simple tests* or their negations. The simple tests are of two types:
(equality test): $atom = atom$, and
(relational test): $R(y_1, \ldots, y_a)$, where R is an a-ary relation symbol of τ and $y_1, \ldots, y_a$ are atoms.
(We usually denote $\neg(atom = atom)$ by $atom \neq atom$.)

Next, for $i \geq 0$, let the $while(i+1)$ instructions be of the form

WHILE t DO I OD

where I is a finite sequence of assignment instructions and $while(k)$ instructions, for any $k \leq i$, and t is an atomic test.

A test instruction is, then, an element of $\bigcup_{i \geq 0} while(i)$. (Thus, a $while(i)$ test instruction has at most i nested WHILE ... DO ... OD instructions.)

(iv) the *input and output instructions* of NPS(τ) are of the form $input(x_1, \ldots, x_m)$ and $output(x_1, \ldots, x_m)$, where $x_1, \ldots, x_m$ are the input/output variables, and are all the variables that appear in any program scheme. Also, in any program scheme, there is exactly one input and one output instruction.

NPS $= \{\rho \in \text{NPS}(\tau) \mid \tau \text{ is some vocabulary}\}$.

Thus, an NPS program scheme ρ is some finite sequence of instructions of the type (ii)–(iv) above, defined with respect to some fixed vocabulary τ, and where the test in any test instruction (if there is any) is a quantifier-free first-order formula. We will write program schemes following standard conventions for writing, say, Pascal programs; that is, by ordering the sequence of instructions in a top to bottom fashion, with one instruction per line, and using appropriate indentation.

We will use in our program schemes instructions of the form

IF t THEN I_1 ELSE I_2 FI

where I_1 and I_2 are valid sequences of instructions and t is an atomic test. Formally, this is an abbreviation for the following piece of code:

$x := 0$

WHILE $t \wedge x = 0$ DO

I_1

$x := max$

OD

WHILE $x = 0$ DO

I_2

$x := max$

OD

For a program scheme ρ, we denote by X_ρ the set of all the variables that show in ρ (the input/output variables). A program scheme ρ with input and output instructions of the form $input(x_1, \ldots, x_m)$ and $output(x_1, \ldots, x_m)$ is said to be of *arity* m.

The interpretation of a program scheme $\rho \in \mathrm{NPS}(\tau)$ in some finite τ-structure $\mathcal{A}$ of size n, denoted $\rho^{\mathcal{A}}$, is defined in the usual logical way on each one of its instructions and its constituent variables, constants and relations, with the following exceptions: we assume that the initial values (taken from $|\mathcal{A}|$) of the input/output variables are given; that $0^{\mathcal{A}}$ (resp. $max^{\mathcal{A}}$) is the $0 \in |\mathcal{A}|$ (resp. $n-1 \in |\mathcal{A}|$); that $(var := atom)^{\mathcal{A}}$ means to assign the value of $atom^{\mathcal{A}}$ to var, and that $guess(x_i)^{\mathcal{A}}$ means to nondeterministically assign some value from $|\mathcal{A}|$ to the variable x_i.

Moreover, given $\rho \in \mathrm{NPS}(\tau)$, with $X_\rho = \{x_1, \ldots, x_m\}$, $\mathcal{A} \in \mathrm{STRUCT}(\tau)$, and $(a_1, \ldots, a_m) \in |\mathcal{A}|^m$, a *computation of* $\rho^{\mathcal{A}}$ *with input values* $(a_1, \ldots, a_m)$ consists of first assigning the value a_i to the variable $x_i \in X_\rho$, for each $i \in \{1, \ldots, m\}$, and then executing each instruction in $\rho^{\mathcal{A}}$, after the input instruction, in the order in which they appear (i.e, top to bottom). A WHILE t DO I OD instruction is executed as follows. We first check the validity of t in $\mathcal{A}$: if t is false in $\mathcal{A}$, control is passed to the first instruction after the OD; if t is true in $\mathcal{A}$ then the set of instructions I is executed, the validity of t in $\mathcal{A}$ is checked again, and the whole process is repeated. Note that, due to the nondeterminism introduced by the $guess(\ )$ instruction, there could be more than one computation of $\rho^{\mathcal{A}}$ with input values $(a_1, \ldots, a_m)$.

We say that $\rho^{\mathcal{A}}$ *halts* for input values $(a_1, \ldots, a_m)$ and output values $(b_1, \ldots, b_m)$, for some $(a_1, \ldots, a_m), (b_1, \ldots, b_m) \in |\mathcal{A}|^m$, and write

$$[x_1/a_1, \ldots, x_m/a_m]\rho^{\mathcal{A}}[x_1/b_1, \ldots, x_m/b_m]$$

if for some computation of $\rho^{\mathcal{A}}$ with input values $(a_1, \ldots, a_m)$, there is a (possibly empty) sequence of $guess(x_i)$ instructions which nondeterministically assign values to the variables of $\rho^{\mathcal{A}}$, causing the program to halt with all its variables $(x_1, \ldots, x_m)$ set to $(b_1, \ldots, b_m)$. (When the variables and the order of the assignment is clear from the context we just write $[a_1, \ldots, a_m]\rho^{\mathcal{A}}[b_1, \ldots, b_m]$.)

A program scheme $\rho \in \mathrm{NPS}(\tau)$ accepts a τ-structure $\mathcal{A}$, in symbols $\mathcal{A} \models \rho$, if and only if $[\overline{0}]\rho^{\mathcal{A}}[\overline{max}]$ (i.e., there exists a computation of $\rho^{\mathcal{A}}$ that, starting with all input/output variables set to $0^{\mathcal{A}}$, halts with all input/output variables set to $max^{\mathcal{A}}$).

4.1.2 A digression on loop programs

The reader who is more familiar with the recursive function theoretical approach to Computability Theory will probably recognize a similarity between program schemes and *loop programs* (see [BL74] or [DW83]). Loop programs consists of assignment instructions of the form

$$var := atom \quad \text{or} \quad var := var + 1$$

(where $var + 1$ is interpreted as the successor of var), a

$$\text{GOTO } atom$$

instruction that transfers control to the instruction labelled with $atom$ (and labels are placed as the first word of the labelled instruction), and a

$$\text{LOOP } atom\ I\ \text{END}$$

instruction, which produces the repeatedly execution of the set of instructions I a number of times equal to the value assigned to $atom$. There are no $guess(\ )$

instructions.

One can show that the GOTO and LOOP instructions can be defined with WHILE instructions and +1 (successor), and conversely, a WHILE instruction can be defined with GOTO and LOOP (see [BL74, section 2.2]). Thus, syntactically, loop programs are, essentially, program schemes over the empty vocabulary with built–in successor and no *guess*( ) instruction (namely, the class DPS($\{\ \}, +1$) of deterministic program schemes with successor, over the empty vocabulary). Loop programs, however, as they are considered in [BL74, DW83], are interpreted over arbitrary structures, as opposed to finite structures, and with that semantic freedom loop programs (or program schemes in DPS($\{\ \}, +1$)) compute all primitive recursive functions (a result due to Meyer and Ritchie dated 1967; see [BL74] for further references). Also, one can define the class that functions computable by loop programs belong to, by determining how deep the nesting of LOOP instructions is in these programs, and show that there are functions computable by loop programs with $i+1$ nested LOOP instructions that are not computable by loop programs with i nested LOOP instructions (see [BL74, chapter 10] or [DW83, chapter 13]). In the terminology introduced in our definition of program schemes this says that, over arbitrary structures, the class $while(i)$ is properly contained in the class $while(i+1)$.

The result that we are about to present in this chapter can be seen as an analog to this loop hierarchy, over finite and unordered structures, although it was originally motivated by a hierarchy result for the logic TC*[FO]. We define in the next section classes of program schemes obtained by allowing program schemes to be taken as tests, and further, by nesting these "test" programs. Then, we show in the following sections that the containment between these classes is strict.

4.1.3 A class of program schemes

We are interested in allowing first-order formulas and other program schemes as tests; therefore, we will define a class of program schemes with such tests, namely PS, extending NPS. First, we introduce some notation. For $\rho \in \mathrm{NPS}$, we denote by $\rho - \mathrm{I/O}$ the sequence of instructions in ρ except the input and the output instructions. Fix some vocabulary τ and let $\mathrm{NPS}_0(\tau)$ be the class of all $\rho - \mathrm{I/O}$, where $\rho \in \mathrm{NPS}(\tau)$. We define $\mathrm{PS}_0(\tau)$ as the smallest class such that (i)–(iii) below hold:

(i) If $\rho \in \mathrm{NPS}_0(\tau)$ then $\rho \in \mathrm{PS}_0(\tau)$.

(ii) If $\rho \in \mathrm{PS}_0(\tau)$, and z is a variable of ρ, then $\rho[z] \in \mathrm{PS}_0(\tau)$. We interpret $\rho[z]$ in such a way: For a finite τ-structure $\mathcal{A}$ and for $a \in |\mathcal{A}|$,

$$\langle \mathcal{A}, a \rangle \models \rho[z] \text{ iff } [x_1/0, \ldots, x_n/0, z/a]\rho^{\mathcal{A}}[x_1/max, \ldots, x_n/max, z/max],$$

where $x_1, \ldots, x_n$, and z, are the input variables of ρ. In words, $\langle \mathcal{A}, a \rangle \models \rho[z]$ means that, initializing all variables but z to $0^{\mathcal{A}}$, and initializing z to a, a computation of $\rho^{\mathcal{A}}$ halts with all variables set to $max^{\mathcal{A}}$.

(iii) If $\rho_1, \rho_2 \in \mathrm{PS}_0(\tau)$ and θ is a first-order τ-formula, then

$$\text{WHILE } \theta \wedge \rho_1 \text{ DO } \rho_2 \text{ OD}$$

belongs to $\mathrm{PS}_0(\tau)$. For a finite τ-structure $\mathcal{A}$, this instruction is executed as follows: First, check if both $\mathcal{A} \models \theta$ and $[\overline{0}]\rho_1^{\mathcal{A}}[\overline{max}]$ hold (in case $\rho_1 = \rho_1[z]$, check if for some $a \in |\mathcal{A}|$ $[\overline{0}, z/a]\rho_1^{\mathcal{A}}[\overline{max}]$ hold). If that is not the case, skip execution of $\rho_2^{\mathcal{A}}$, passing control to the first instruction after OD; otherwise, execute $\rho_2^{\mathcal{A}}$, and, then, repeat the process.

Let $\mathrm{PS}_0 = \{\rho \in \mathrm{PS}_0(\tau) \mid \tau \text{ is some vocabulary}\}$. Then, in PS_0 there are program schemes with WHILE instructions with first-order formulas or program schemes as

tests. To deal with negations and quantifications of program schemes, we regard them as the following abbreviations.

- For $\rho \in \mathrm{PS}_0$ with $x_1, \ldots, x_n$ as input variables, $\neg\rho$ stands for

  IF ρ THEN

  $x_1 := 0$

  $\vdots$

  $x_n := 0$

  ELSE

  $x_1 := max$

  $\vdots$

  $x_n := max$

  FI

  Note that this coincides with our logical notion of negation, because, for a finite structure $\mathcal{A}$ of the appropriate vocabulary, we have

  $$\mathcal{A} \models \neg\rho \text{ iff } \mathcal{A} \not\models \rho.$$

- For $\rho \in \mathrm{PS}_0$ and z an input variable of ρ, $\exists z\, \rho[z]$ abbreviates:

  WHILE $z \neq max$ DO

  $guess(z)$

  IF $\rho[z]$ THEN

  $z := max$

  ELSE

  $z := 0$

FI

OD

For a finite structure $\mathcal{A}$, we have

$$\mathcal{A} \models \exists z\, \rho[z] \text{ iff for some } a \in |\mathcal{A}| \ \langle \mathcal{A}, a \rangle \models \rho[z].$$

We will also consider $\forall z\, \rho[z]$ as an abbreviation for $\neg\exists z\, \neg\rho[z]$.

In the presence of successor we could syntactically characterize $\forall z\, \rho[z]$ with the following quantifier-free program scheme:

$z := 0$

WHILE $z \neq max$ DO

IF $\rho[z]$ THEN

$z := z + 1$

FI

OD

As a consequence of our results in section 4.3 we conclude that, indeed, we cannot escape the use of successor for simulating the $\forall$ quantifier as above.

Now, for each vocabulary τ, we define $\mathrm{PS}(\tau)$ to be the class of program schemes such that each $\rho \in \mathrm{PS}(\tau)$ is a finite sequence of instructions that belongs to $\mathrm{PS}_0(\tau)$, allowing the abbreviations $\neg\beta$, $\exists z\beta[z]$, and $\forall z\beta[z]$, as tests ($\beta \in \mathrm{PS}_0(\tau)$), and with an input and an output instructions. The input and output instruction have the form $input(x_1, \ldots, x_m)$ and $output(x_1, \ldots, x_m)$, they are unique, and are, respectively, the first and the last instruction of ρ.

Let $\mathrm{PS} = \{\rho \in \mathrm{PS}(\tau) \mid \tau \text{ is some vocabulary}\}$. Then, $\mathrm{NPS} \subseteq \mathrm{PS}$, and in PS we have program schemes with test instructions where the test could be a first-order formula, or a PS_0 (resp. quantified, negated) program scheme. We will define below a particular class of these program schemes, but, before doing that, we need to introduce more notation.

We will often use k-tuples of variables, for $k > 1$, in place of single variables in our program schemes. In those cases we regard instructions as $\overline{x} := \overline{y}$, where $\overline{x} = (x_1, \ldots, x_k)$ and $\overline{y} = (y_1, \ldots, y_k)$, as an abbreviation for the sequence of k instructions

$$x_1 := y_1,\ x_2 := y_2,\ \ldots,\ x_k := y_k;$$

and instructions as $guess(\overline{x})$ (again $\overline{x} = (x_1, \ldots, x_k)$), as an abbreviation for

$$guess(x_1),\ guess(x_2),\ \ldots,\ guess(x_k).$$

The test in a WHILE instruction is treated as is usually done in Logic with formulas that involves k-tuples of variables.

Thus, given an integer $k > 0$ and a program scheme ρ of arity m, we denote by ρ^k the program scheme of arity km obtained by substituting in ρ each one of its input variables by k-tuples of variables, and allowing its instructions to act on these k-tuples according to the above convention.

We now define the classes of program schemes that we are interested in studying. For the sake of clarity we restrict our vocabulary τ to have only one binary relation symbol E.

Let $\rho \in \mathrm{NPS}(\tau)$. We define inductively the following classes of program schemes (in PS) related to ρ:

$\boldsymbol{\rho}(0)$ is the class of program schemes β such that: for some vocabulary σ and for some $k > 0$, β is ρ^k with at most one of its relational tests having all occurrences of E substituted by a σ-formula of the form $\forall z_1 \ldots \forall z_l\, \psi(\overline{x}, \overline{y}, \overline{z})$, where ψ is a quantifier-free first-order formula in the variables $\overline{x} = (x_1, \ldots, x_k)$, $\overline{y} = (y_1, \ldots, y_k) \in X_{\rho^k}$, $\overline{z} = (z_1, \ldots, z_l) \notin X_{\rho^k}$ (l is some positive integer), and possibly some other variables. Furthermore, we require that the formula $\forall z_1 \ldots \forall z_l\ \psi(\overline{x}, \overline{y}, \overline{z})$ constitutes a k-ary τ-translation of STRUCT(σ) to STRUCT(τ) (cf. Definition 2.2.2).

$\boldsymbol{\rho}(m)$ is the class of program schemes β such that: β is ρ^k, for some $k > 0$, with exactly one of its tests being of the form $\forall z_1 \ldots \forall z_l \, \alpha$, where α is a program scheme in $\boldsymbol{\rho}(m-1)$ not necessarily of the same arity as β, without the input and output instructions, $l > 0$, $z_1, \ldots, z_l \in X_\alpha \setminus X_\beta$, and $X_\alpha \cap X_\beta$ is not necessarily empty, but any variable in X_β that shows in any assignment instruction of α is treated as a constant, i.e., it does not appears in a *guess*( ) instruction, and only on the right hand side of a $var := atom$ instruction of α.

(The general case is treated similarly: we consider a set Σ of τ-descriptive k-ary –universally quantified– σ-formulas, and in the definition of $\boldsymbol{\rho}(0)$ we ask that the relational test has all its relations substituted by their corresponding formulas of Σ.)

Example 4.1.1 Consider the following program scheme $\rho \in \mathrm{NPS}(\tau_2)$, where $\tau_2 = \{E\}$:

input(x, y)
WHILE $x \neq max$ DO
    guess(y)
    IF $E(x, y)$ THEN
        $x := y$
    FI
OD
output(x, y)

By definition of $\boldsymbol{\rho}(0)$, ρ is in $\boldsymbol{\rho}(0)$. ∎

Example 4.1.2 Consider ρ as in the previous example. Then the following program scheme β is in $\boldsymbol{\rho}(1)$:

$input(x, y)$

WHILE $x \neq max$ DO

$guess(y)$

IF $\forall u \forall v$ (WHILE $u \neq v$ DO; $guess(w)$; IF $E(u, w)$ THEN $u := w$; FI; OD) THEN

$y := max$

FI

$x := y$

OD

$output(x, y)$

■

Remark 4.1.1 We often use abbreviations as, for example,

$$\beta := \rho^k[input(\overline{x}_1, \ldots, \overline{x}_m) \ \ldots \ \forall \overline{z} \alpha[\ldots] \ \ldots],$$

to explicitly indicate some parts of a program scheme β to which we are referring in the course of our arguments. These abbreviations should be clear from the context. The example above says that β is a program scheme which has the structure of ρ^k, input instruction $input(\overline{x}_1, \ldots, \overline{x}_m)$, and a nested program scheme $\forall \overline{z} \alpha$. ■

We define a measure of complexity for a program scheme ρ based on the quantifier rank of its tests. Define the *rank of* ρ, $rank(\rho)$, as follows: Let $t_1, \ldots, t_m$ be all the tests that show in ρ, then

$$rank(\rho) = arity(\rho) + max[rank(t_i) \ : \ i = 1, \ldots, m]$$

where

$$rank(t_i) = \begin{cases} qr(t_i) & \text{if } t_i \text{ is a first-order formula} \\ max[qr(t), rank(\beta)] & \text{if } t_i = t \wedge \beta, \text{ where } t \text{ is a first-order formula} \\ & \text{and } \beta \text{ is a program scheme} \\ rank(\beta) + 1 & \text{if } t_i = \forall z \beta[z], \text{ where } \beta \text{ is a program scheme} \end{cases}$$

(qr is the quantifier rank of a first-order formula as defined in Definition 2.2.4).

4.2 k-pebble infinitary games

Definition 4.2.1 Let τ be some vocabulary, let $\mathcal{A}$ and $\mathcal{B}$ be two τ-structures, $k > 0$, and $l \geq 0$. The infinitary game of k-pebbles and $(P_0, \ldots, P_l)$-coloring on $(\mathcal{A}, \mathcal{B})$, consists of a set P of k pairs of pebbles and a board formed with the structures $\mathcal{A}$ and $\mathcal{B}$, and it is played by two players, Spoiler and Duplicator, as follows:

First, Duplicator gives a partition of P into $l+1$ subsets of $k_0, \ldots, k_l$ pairs of pebbles respectively, so that $k_0 + \cdots + k_l = k$ and, for each $i = 0, \ldots, l$, the k_i pairs of pebbles in each subset is colored P_i. We indicate each one of these subsets by $\{(p_{i,1}, q_{i,1}), \ldots, (p_{i,k_i}, q_{i,k_i})\}$.

Next, the players take turns placing pebbles on the structures $\mathcal{A}$ and $\mathcal{B}$, with Spoiler making the first move and using the pebbles colored P_0 first. There are four type of moves, which are described below, and Spoiler is free to choose which move to make.

Let $\mathcal{A}_0 := \mathcal{A}$ and $\mathcal{B}_0 := \mathcal{B}$, and for $i > 0$ $\mathcal{A}_i$ (resp. $\mathcal{B}_i$) is an extension of $\mathcal{A}$ (resp. $\mathcal{B}$) with at most $k_0 + \cdots + k_{i-1}$ new constants determined by elements upon which pebbles of colors $P_0, \ldots, P_{i-1}$ are placed. For $i \geq 0$ the allowable moves are:

$\forall_i$-move: For some $s \leq k_i$, Spoiler picks pebbles $q_{i,j_1}, q_{i,j_2}, \ldots, q_{i,j_s}$ colored P_i, which may or may not have been previously used, and places them on s elements of $|\mathcal{B}_i|$. Duplicator must respond by picking pebbles $p_{i,j_1}, p_{i,j_2}, \ldots, p_{i,j_s}$, and placing them on s elements of $|\mathcal{A}_i|$.

A_i-move: Similar to the $\forall_i$-move but Spoiler pebbles on $|\mathcal{A}_i|$ (with the $p_{i,j}$ pebbles) and Duplicator responds on $|\mathcal{B}_i|$ (with the $q_{i,j}$ pebbles).

$test_i$-move: Spoiler leaves fixed all his pebbles colored $P_0, \ldots, P_i$, that are already on the board $(\mathcal{A}, \mathcal{B})$, and continue playing with the set of k_{i+1} pairs of pebbles colored P_{i+1}. Duplicator does the same.

$reset_i$-move: Spoiler removes from the board all his pebbles colored P_{i+1}, and continue playing with the set of k_i pairs of pebbles colored P_i. Duplicator does the same.

Spoiler wins if at any point of the game the function defined by both mapping each element of $|\mathcal{A}|$, upon which a pebble currently rests, to the element of $|\mathcal{B}|$, upon which the pebble of same color and in its corresponding pair rests, and mapping each constant of $\mathcal{A}$ to the corresponding constant of $\mathcal{B}$, is not a partial isomorphism from $\mathcal{A}$ to $\mathcal{B}$, or is not well-defined. The game can go forever. We say that Duplicator has a winning strategy in the infinitary game of k-pebbles and $(P_0, \ldots, P_l)$-coloring on $(\mathcal{A}, \mathcal{B})$, if regardless of how Spoiler moves, Duplicator can prolong the game indefinitely. ∎

Theorem 4.2.2 *Let τ be some vocabulary, $\mathcal{A}$ and $\mathcal{B}$ be two τ-structures such that $\mathcal{A} \subseteq \mathcal{B}$, $l \geq 0$, and $\rho \in \mathrm{NPS}(\tau)$. If Duplicator has a winning strategy in the infinitary game of k-pebbles and $(P_0, \ldots, P_{l+1})$-coloring on the extended structures, $\langle \mathcal{A}, 0, max \rangle$ and $\langle \mathcal{B}, 0, max \rangle$ then, for every $\beta \in \boldsymbol{\rho}(l)$ with $rank(\beta) + 1 \leq k$, $\mathcal{A} \models \beta$ implies $\mathcal{B} \models \beta$.*

Proof: The proof is by induction on the complexity of the tests in β. Let P be the set of k pairs of pebbles. We first note that, for all cases, Duplicator's partition of P depends on the depth of the nesting of program schemes in β. In general, say

$$\beta := \rho^s[input(\overline{x}_1, \ldots, \overline{x}_m) \ \ldots \ \forall \overline{z}_1 \alpha_1[\ldots \ \forall \overline{z}_2 \alpha_2[\ldots \ \forall \overline{z}_{l+1} \psi \ \ldots] \ \ldots],$$

where $s > 0$, each $\overline{x}_i$ is an s-tuple, each $\alpha_i \in \boldsymbol{\rho}(l-i)$, and ψ is a quantifier-free first-order formula. Then Duplicator takes

$$\begin{aligned} k_0 &\geq arity(\beta) = sm \\ k_1 &\geq arity(\alpha_1) \\ &\vdots \\ k_l &\geq arity(\alpha_l) \\ k_{l+1} &\geq qr(\forall \overline{z}_{l+1} \psi) \end{aligned}$$

so that $k = k_0 + k_1 + \cdots + k_{l+1} \geq rank(\beta)$.

The first step of our induction is the case when all tests in β are quantifier-free first-order formulas, that is, $\beta \in \mathrm{NPS}(\tau)$. Then, according to the above note, Duplicator needs to take $k_0 \geq arity(\beta) = rank(\beta)$ and $k_1 \geq 1$, so the partition consists of two subsets colored P_0 and P_1 respectively, although only the pebbles colored P_0 will be used, since no test moves (and no reset moves) are needed. As it is noted in [CS96], each instruction of a $\beta \in \mathrm{NPS}$ determines a move of the pebbles in the input structure $\mathcal{A}$, or $\mathcal{B}$, as follows: since $arity(\beta) = sm$ and $k = k_0 \geq sm$, the players take the first sm pairs of pebbles $\{(p_{0,1}, q_{0,1}), \ldots, (p_{0,sm}, q_{0,sm})\}$ colored P_0, and make each pebble $p_{0,i}$ and $q_{0,i}$ correspond to the variable x_i. Initially, the pebbles are placed on the 0 of $|\mathcal{A}|$ (resp. $|\mathcal{B}|$), since what the players are trying to determine is

$$[\overline{0}]\beta^{\mathcal{A}}[\overline{max}] \text{ implies } [\overline{0}]\beta^{\mathcal{B}}[\overline{max}].$$

Then, to continue simulating the computation of $\beta^{\mathcal{A}}$ (resp. $\beta^{\mathcal{B}}$), each instruction of the form $x_i := atom$ corresponds to placing the i-th pebble on top of the element

upon which the j-th pebble resides, if $atom = x_j$, or on top of the constant element C, if $atom = C$; an instruction of the form $guess(x_i)$ corresponds to placing the i-th pebble on an arbitrary element; the tests are evaluated with respect to the extended structure $\langle \mathcal{A}, a_1, \ldots, a_{sm} \rangle$ (resp. $\langle \mathcal{B}, b_1, \ldots, b_{sm} \rangle$), where $a_1, \ldots, a_{sm}$ (resp. $b_1, \ldots, b_{sm}$) are the elements upon which the pebbles are currently placed.

Moving pebbles as described above one sees that $[\overline{0}]\beta^{\mathcal{A}}[\overline{max}]$ (resp. $[\overline{0}]\beta^{\mathcal{B}}[\overline{max}]$) if and only if there is a sequence of moves such that all pebbles end up placed on the element $max^{\mathcal{A}}$ (resp. $max^{\mathcal{B}}$). Assuming that there is such a sequence for $\beta^{\mathcal{A}}$, Spoiler wants to show that there is no such a sequence for $\beta^{\mathcal{B}}$. Therefore, Spoiler wants to do A_0-moves at all times in order to challenge Duplicator to find such convergent computation of $\beta^{\mathcal{B}}$. By hypothesis, Duplicator has a winning strategy in the general game of k-pebbles (not just restricted to the kind of moves described above) and all the tests are atomic, therefore, using his winning strategy, Duplicator can make the extended structures $\langle \mathcal{A}, a_1, \ldots, a_{sm} \rangle$ and $\langle \mathcal{B}, b_1, \ldots, b_{sm} \rangle$ indistinguishable through atomic formulas; hence, the outcome of the tests is the same for both $\beta^{\mathcal{A}}$ and $\beta^{\mathcal{B}}$; hence, the flow of control is the same, and, hence, the computations are the same. Thus, Duplicator wins in this case.

Inductively, let

$$\beta := \rho^s[input(\overline{x}_1, \ldots, \overline{x}_m) \ \ldots \ \forall \overline{z}\alpha[\ldots] \ \ldots],$$

where $\forall \overline{z}\alpha$ is the (only) non-atomic test that shows in β, and $\alpha \in \boldsymbol{\rho}(l-1)$. According to our remark, at the beginning of the proof, Duplicator's partition of P includes

$$\begin{aligned} k_0 &\geq arity(\beta) = sm, \text{ and} \\ k_1 &\geq arity(\alpha). \end{aligned}$$

Then, the game starts with the players moving at most k_0 pairs of pebbles colored P_0, and for all those moves that corresponds to instructions with atomic tests,

Duplicator applies his winning strategy as in the basic case. When the game arrives to the test $\forall\overline{z}\alpha$ the players do a $test_0$-move: they leave fixed their respective sm pebbles colored P_0, thus defining the extended structures

$$\mathcal{A}_1 := \langle \mathcal{A}, a_1, \ldots, a_{sm} \rangle \text{ and } \mathcal{B}_1 := \langle \mathcal{B}, b_1, \ldots, b_{sm} \rangle ,$$

and using the k_1 pairs of pebbles colored P_1 they need to determine if

$$\mathcal{A}_1 \models \forall\overline{z}\alpha \text{ iff } \mathcal{B}_1 \models \forall\overline{z}\alpha.$$

[$\Rightarrow$]: Suppose $\mathcal{A}_1 \models \forall\overline{z}\alpha$ and $\mathcal{B}_1 \not\models \forall\overline{z}\alpha$. Then Spoiler does a $\forall_1$-move: since there exists a tuple $\overline{b} \in |\mathcal{B}_1|^{|\overline{z}|}$ that would make the program $(\alpha[\overline{b}])^{\mathcal{B}_1}$ reject its input (i.e., it never halts or $output \neq \overline{max}$), Spoiler pebbles that tuple. Then, whatever $|\overline{z}|$-tuple $\overline{a}$ Duplicator pebbles in $\mathcal{A}_1$, we get

$$\langle \mathcal{A}_1, \overline{a} \rangle \models \alpha \text{ and } \langle \mathcal{B}_1, \overline{b} \rangle \not\models \alpha.$$

This yields a contradiction: $\alpha \in \boldsymbol{\rho}(l-1)$, so Duplicator only needs a partition of P into $(P_0, \ldots, P_l)$ coloring to play the game with respect to α. By hypothesis, Duplicator has a winning strategy for a game that involves partitions of P up to $l+2$ sets, and so he has a winning strategy for the game of k-pebbles and $(P_0, \ldots, P_l)$-coloring, and therefore, by inductive hypothesis, $\mathcal{A} \models \alpha$ implies $\mathcal{B} \models \alpha$, and hence, $\mathcal{A}_1 \models \alpha$ implies $\mathcal{B}_1 \models \alpha$, contradicting the equation above. Thus, it is true that if $\mathcal{A}_1 \models \forall\overline{z}\alpha$ then $\mathcal{B}_1 \models \forall\overline{z}\alpha$.

[$\Leftarrow$]: We show $\mathcal{A}_1 \not\models \forall\overline{z}\alpha$ implies $\mathcal{B}_1 \not\models \forall\overline{z}\alpha$ or, equivalently, $\mathcal{A}_1 \models \exists\overline{z}\neg\alpha$ implies $\mathcal{B}_1 \models \exists\overline{z}\neg\alpha$. In terms of games, we need to show that if Spoiler can find a tuple $\overline{a} \in |\mathcal{A}_1|^{|\overline{z}|}$ such that $\langle \mathcal{A}_1, \overline{a} \rangle \models \neg\alpha[\overline{z}]$, then Duplicator can find a corresponding tuple $\overline{b} \in |\mathcal{B}_1|^{|\overline{z}|}$ such that $\langle \mathcal{B}_1, \overline{b} \rangle \models \neg\alpha[\overline{z}]$. This follows from the hypothesis $\mathcal{A} \subseteq \mathcal{B}$: for then $\mathcal{A}_1 \subseteq \mathcal{B}_1$ and Duplicator can choose $\overline{b}$ isomorphic to $\overline{a}$ through the natural embedding of $\mathcal{A}_1$ into $\mathcal{B}_1$. Hence, $\langle \mathcal{A}_1, \overline{a} \rangle \models \neg\alpha[\overline{z}]$ implies $\langle \mathcal{B}_1, \overline{b} \rangle \models \neg\alpha[\overline{z}]$.

Thus, the outcome of the test is the same in both computations $\beta^{\mathcal{A}}$ and $\beta^{\mathcal{B}}$; hence, the next instruction is the same in both. The players do a $reset_0$-move and perform that next instruction with their pebbles colored P_0. Duplicator has won through the whole testing process, and continues playing according to his winning strategy for the simpler instructions. ∎

Note that for the proof of Theorem 4.2.2 we need only be concerned with those moves of the infinitary game that are dictated by the instructions in the program scheme β, in order to conclude that $\mathcal{A} \models \beta$ implies $\mathcal{B} \models \beta$. Thus, let the *infinitary game of k-pebbles and $(P_0, \ldots, P_l)$-coloring on $(\mathcal{A}, \mathcal{B})$ played according to (program scheme) β* be the game, as defined in Definition 4.2.1, where the players are only allowed to make moves that corresponds to instructions in β, as explained in the course of the proof of Theorem 4.2.2. That is, in this restricted game, Spoiler (resp. Duplicator) begins placing his pebbles on the 0 of $|\mathcal{A}|$ (resp. $|\mathcal{B}|$) and continues moving pebbles in the order in which the sequence of instructions in β appear, with each pebble move reflecting what the corresponding instruction in β says. If the third instruction in β is $guess(x_7)$, for example, and the game has been played up to this point with the pebbles colored P_0, then the third move of Spoiler should be the A_0-move of placing the pebble $p_{0,7}$ on an arbitrary element of $|\mathcal{A}|$. For this restricted infinitary game we have the following result, whose proof is contained in the proof of Theorem 4.2.2.

Theorem 4.2.3 *Let τ be some vocabulary, $\mathcal{A}$ and $\mathcal{B}$ be two τ-structures such that $\mathcal{A} \subseteq \mathcal{B}$, $l \geq 0$, and $\rho \in \mathrm{NPS}(\tau)$. Let $k > 0$ and let $\beta \in \boldsymbol{\rho}(l)$ with $rank(\beta) + 1 \leq k$. If Duplicator has a winning strategy in the infinitary game of k-pebbles and $(P_0, \ldots, P_{l+1})$-coloring on ($\langle \mathcal{A}, 0, max \rangle$, $\langle \mathcal{B}, 0, max \rangle$) played according to β, then $\mathcal{A} \models \beta$ implies $\mathcal{B} \models \beta$.*

4.3 A Hierarchy Theorem for program schemes

In this section τ is some vocabulary with at least a binary relation symbol.

For our separation results we employ the following structures, $\mathcal{A}_{k,m}$ and $\mathcal{B}_{k,m}$, that were defined in [Grä91].

Definition 4.3.1 (The m-extensions) Let $\{\mathcal{A}_k\}_{k\geq 0}$ and $\{\mathcal{B}_k\}_{k\geq 0}$ be two families of τ-structures. For each $m \geq 0$, we define the *m-extensions* $\{\mathcal{A}_{k,m}\}_{k\geq 0}$ *and* $\{\mathcal{B}_{k,m}\}_{k\geq 0}$ *of* $\{\mathcal{A}_k\}_{k\geq 0}$ *and* $\{\mathcal{B}_k\}_{k\geq 0}$, inductively as follows:
For $m = 0$ we extend τ with a new unary relation symbol U_0 and we define, for each $k \geq 0$, $\mathcal{A}_{k,0} := \langle \mathcal{A}_k, \{c_0\}\rangle$ and $\mathcal{B}_{k,0} := \langle \mathcal{B}_k, \{c_0\}\rangle$, and the following hold on these structures:

(i) c_0 is a new element different from all the elements in $\mathcal{A}_k$ (resp. $\mathcal{B}_k$).

(ii) c_0 is the unique element that satisfies U_0 in $\mathcal{A}_{k,0}$ (resp. $\mathcal{B}_{k,0}$).

(iii) c_0 is connected to all points. More formally, the sentence $\forall x\, E(c_0, x)$ holds in $\mathcal{A}_{k,0}$ (resp. $\mathcal{B}_{k,0}$).

c_0 is called the *root* of $\mathcal{A}_{k,0}$ (resp. $\mathcal{B}_{k,0}$).

For $m > 0$ assume $\mathcal{A}_{k,m}$ and $\mathcal{B}_{k,m}$ have been defined and τ has been extended with $m+1$ new unary relation symbols $U_0, \ldots, U_m$. Then, for each $k \geq 0$, $\mathcal{A}_{k,m+1}$ and $\mathcal{B}_{k,m+1}$ are define, according to the parity of m, as follows:

1. If m is even then $\mathcal{A}_{k,m+1}$ consists of the disjoint union of k copies of $\mathcal{B}_{k,m}$, one copy of $\mathcal{A}_{k,m}$, and a new element c_{m+1} (the root). $\mathcal{B}_{k,m+1}$ consists of the disjoint union of $k+1$ copies of $\mathcal{B}_{k,m}$ and the root c_{m+1}.

2. If m is odd then $\mathcal{A}_{k,m+1}$ consists of the disjoint union of $k+1$ copies of $\mathcal{A}_{k,m}$ and the root c_{m+1}. $\mathcal{B}_{k,m+1}$ consists of the disjoint union of k copies of $\mathcal{A}_{k,m}$, one copy of $\mathcal{B}_{k,m}$, and the root c_{m+1}.

For both cases the following hold on these structures:

(i) c_{m+1} is different from all the elements in $\mathcal{A}_{k,m+1}$ (resp. $\mathcal{B}_{k,m+1}$).

(ii) c_{m+1} is the unique element that satisfies U_{m+1} in $\mathcal{A}_{k,m+1}$ (resp. $\mathcal{B}_{k,m+1}$).

(iii) c_{m+1} is connected to the roots of each of the substructures $\mathcal{A}_{k,m}$ or $\mathcal{B}_{k,m}$ in $\mathcal{A}_{k,m+1}$ (resp. $\mathcal{B}_{k,m+1}$).

■

Figure 10 shows the m-extensions $\mathcal{A}_{k,m}$ and $\mathcal{B}_{k,m}$.

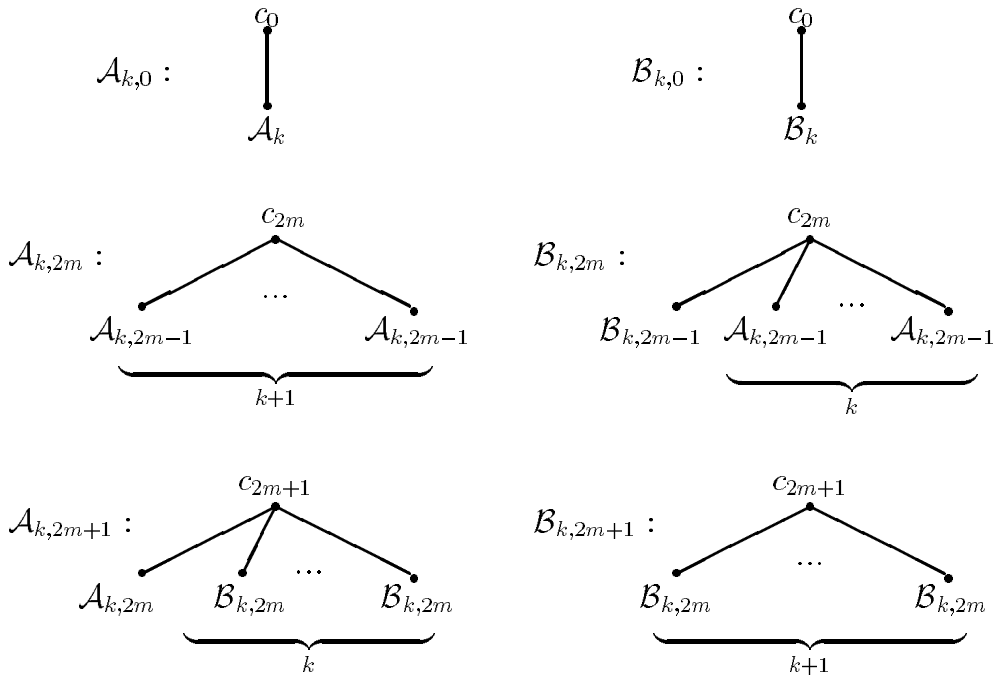

Figure 10: The m-extensions.

Lemma 4.3.2 *Let $\rho \in \mathrm{NPS}(\tau)$ with $arity(\rho) = n$. Let $\{\mathcal{A}_k\}_{k\geq 0}$ and $\{\mathcal{B}_k\}_{k\geq 0}$ be families of τ-structures such that, for each $k \geq 0$, $\mathcal{A}_k \subseteq \mathcal{B}_k$ and for all sentence of the form $\psi := \exists x_1 \ldots \exists x_{sn} \forall \overline{y} \phi(\overline{x}, \overline{y})$, with ϕ a quantifier-free first-order formula, $|\overline{y}| \leq r$ and $sn + r \leq k$, we have*

$$\mathcal{A}_k \models \psi \quad \textit{iff} \quad \mathcal{B}_k \models \psi.$$

Then, for all $m \geq 0$ and $\beta \in \boldsymbol{\rho}(m)$ with $rank(\beta) + 1 \leq k$,

$$\mathcal{A}_{k,2m} \models \beta \textit{ implies } \mathcal{B}_{k,2m} \models \beta.$$

Proof: By Theorem 4.2.3 it is sufficient to show that for all $m \geq 0$ and $\beta \in \boldsymbol{\rho}(m)$, with $rank(\beta) + 1 \leq k$, Duplicator has a winning strategy in the infinitary game of k-pebbles and $(P_0, \ldots, P_{m+1})$-coloring on ($\langle \mathcal{A}_{k,2m}, 0, max\rangle$, $\langle \mathcal{B}_{k,2m}, 0, max\rangle$) played according to β. The proof is by induction on m.

Case $m = 0$: Then β has the form

$$\beta := \rho^s[input(\overline{x}_1, \ldots, \overline{x}_n) \ \ldots \ (\forall y_1 \ldots \forall y_r \phi) \ \ldots],$$

where $r, s > 0$, each $\overline{x}_i$ is an s-tuple, ϕ is a quantifier-free first-order formula, and $\forall y_1 \ldots \forall y_r \phi$ is the non-atomic test that shows in β. To simplify notation we put $\mathcal{A} := \mathcal{A}_{k,0}$ and $\mathcal{B} := \mathcal{B}_{k,0}$. Note that $\mathcal{A}$ (resp. $\mathcal{B}$) is essentially $\mathcal{A}_k$ (resp. $\mathcal{B}_k$), with an extra element c_0, and so, whenever Spoiler pebbles the c_0 of one structure, Duplicator must pebble the c_0 of the opposite structure. Next, we describe how Duplicator must play for the rest of the elements. Duplicator's partition of the set of k pairs of pebbles P consists of a set of $k_0 \geq sn$ pairs of pebbles colored P_0, and a set of $k_1 \geq r$ pairs of pebbles colored P_1. The players start playing with the sn pairs of pebbles colored P_0. Since $\mathcal{A}_k \subseteq \mathcal{B}_k$, Duplicator's strategy through all the assignment and the *simple* test instructions[1] of β is obvious: Wherever Spoiler pebbles on $|\mathcal{A}|$, Duplicator replies with the isomorphic element on $|\mathcal{B}|$ given by the

[1] i.e., the test is at most atomic

natural embedding of $\mathcal{A} \subseteq \mathcal{B}$. When the game arrives to simulating the instruction with the test $\forall y_1 \ldots \forall y_r \phi$, the players have to check the validity of this test with respect to the extended structures $\langle \mathcal{A}, \overline{a} \rangle$ and $\langle \mathcal{B}, \overline{b} \rangle$, where $\overline{a}$ (resp. $\overline{b}$) is the sn-tuple of elements of $|\mathcal{A}|$ (resp. $|\mathcal{B}|$) upon which the pebbles reside. That is, the players must verify that

$$\mathcal{A} \models \exists x_1 \ldots \exists x_{sn} \forall \overline{y} \phi(\overline{x}, \overline{y}) \text{ iff } \mathcal{B} \models \exists x_1 \ldots \exists x_{sn} \forall \overline{y} \phi(\overline{x}, \overline{y}).$$

But this is true by hypothesis. Therefore, Duplicator can pebble so as to make the value of the test in $\beta^{\mathcal{B}}$ the same as that in $\beta^{\mathcal{A}}$ and, hence, the flow of control in both computations is identical, that is, the next instruction to be executed after the test is the same in $\beta^{\mathcal{A}}$ and $\beta^{\mathcal{B}}$. Thus, Duplicator wins on this part of the game, and thereafter the remaining instructions have, at most, simple tests for which Duplicator's strategy is the same as was explained earlier.

Case $m > 0$: Then β has the form

$$\beta := \rho^s[input(\overline{x}_1, \ldots, \overline{x}_n) \ \ldots \ (\forall y_1 \ldots \forall y_r \alpha) \ \ldots],$$

where $r, s > 0$, each $\overline{x}_i$ is an s-tuple, $y_1, \ldots, y_r \in X_\alpha$, $\alpha \in \boldsymbol{\rho}(m-1)$, and our inductive hypothesis is that

> Duplicator has a winning strategy in the infinitary game of k-pebbles and $(P_2, \ldots, P_{m+1})$-coloring on $\langle \mathcal{A}_{k,2m-2}, 0, max \rangle$ and $\langle \mathcal{B}_{k,2m-2}, 0, max \rangle$, which is played according to α.

We describe a winning strategy for Duplicator in the game played according to β. Throughout the game, to any pebbling of the roots c_i by Spoiler, Duplicator responds by pebbling the corresponding root in the opposite model. The players start moving the sn pairs of pebbles colored P_0 according to the assignment and simple test instructions of β. Thus, Spoiler begins placing pebbles on $\mathcal{A}_{k,2m}$. Since $arity(\beta) < k$ and there are $k+1$ copies of $\mathcal{A}_{k,2m-1}$ in $\mathcal{A}_{k,2m}$, one of these copies

remains free of pebbles once all the sn pebbles colored P_0 of Spoiler are on the board. Hence, Duplicator responds by pebbling isomorphic elements among the k copies of $\mathcal{A}_{k,2m-1}$, in $\mathcal{B}_{k,2m}$, which leaves the copy of $\mathcal{B}_{k,2m-1}$ free. When the game comes to a point where it simulates the instruction with the test $(\forall y_1 \ldots \forall y_r \alpha)$, the players do a $test_0$-move followed by a $\forall_1$-move: they leave their pebbles colored P_0 fixed on the board and Spoiler places r pebbles colored P_1 on $\mathcal{B}_{k,2m}$. Duplicator responds as follows: if Spoiler pebbles on a copy of $\mathcal{A}_{k,2m-1}$, Duplicator pebbles the isomorphic element on the corresponding copy of $\mathcal{A}_{k,2m-1}$ in $\mathcal{A}_{k,2m}$; if Spoiler pebbles on the (up to now free) copy of $\mathcal{B}_{k,2m-1}$, Duplicator should then place his pebbles on isomorphic elements among the k copies of $\mathcal{B}_{k,2m-2}$, in $\mathcal{A}_{k,2m-1}$, leaving the copy of $\mathcal{A}_{k,2m-2}$ free of pebbles. This is possible, since $r \leq k$. Also note that the fact that $r \leq k$ and that there are $k+1$ copies of $\mathcal{B}_{k,2m-2}$, in $\mathcal{B}_{k,2m-1}$, implies that one of these copies is not pebbled by Spoiler. The game continues with moves dictated by the instructions in α: if Spoiler pebbles the copies of $\mathcal{B}_{k,2m-2}$, in $\mathcal{A}_{k,2m-1}$, Duplicator pebbles isomorphic elements among the k copies of $\mathcal{B}_{k,2m-2}$, in $\mathcal{B}_{k,2m-1}$, and if Spoiler pebbles the free copy of $\mathcal{A}_{k,2m-2}$, in $\mathcal{A}_{k,2m-1}$, Duplicator pebbles in a free copy of $\mathcal{B}_{k,2m-2}$, in $\mathcal{B}_{k,2m-1}$, according to his winning strategy given by inductive hypothesis. Thus, Duplicator is always able to allocate his pebbles so as to keep partial isomorphism through all stages of the game dictated by the test $(\forall y_1 \ldots \forall y_r \alpha)$; so the flow of control in $\beta^{\mathcal{A}_{k,2m}}$ and $\beta^{\mathcal{B}_{k,2m}}$ is the same after the test. Then, once the players perform the necessary $reset$-moves in order to remove all but the pebbles colored P_0 from the board, the game resumes in the same instruction in both $\beta^{\mathcal{A}_{k,2m}}$ and $\beta^{\mathcal{B}_{k,2m}}$ and, from this point on, Duplicator's strategy is as it was described before reaching the test.

■

Lemma 4.3.3 *Let $\rho \in \mathrm{NPS}(\tau)$. Let $\{\mathcal{A}_k\}_{k\geq 0}$ and $\{\mathcal{B}_k\}_{k\geq 0}$ be families of τ-structures such that, for some $\beta \in \boldsymbol{\rho}(1)$ and all $k \geq 0$,*

$$\mathcal{A}_k \models \beta \text{ and } \mathcal{B}_k \not\models \beta.$$

Then, for all $m \geq 0$ and $k \geq 0$, there exists a program scheme $\beta_m \in \boldsymbol{\rho}(m+1)$ such that

$$\mathcal{A}_{k,2m} \models \beta_m \text{ and } \mathcal{B}_{k,2m} \not\models \beta_m.$$

Proof: Assume, without loss of generality, that the program scheme $\beta \in \boldsymbol{\rho}(1)$ given in the hypothesis has the form

β: $input(x_1, \ldots, x_n)$
$\vdots$
WHILE $\forall z_1 \ldots \forall z_s\, t[\ldots\ \alpha[z_1, \ldots, z_s, w_1, \ldots, w_l]\ \ldots]$ DO
$\vdots$
OD
$\vdots$
$output(x_1, \ldots, x_n)$

where $t[\ldots\ \alpha[z_1, \ldots, z_s, w_1, \ldots, w_l]\ \ldots]$ is a boolean combination of atomic tests and the (program scheme) test $\alpha[z_1, \ldots, z_s, w_1, \ldots, w_l] \in \boldsymbol{\rho}(0)$, with $X_\alpha = \{z_1, \ldots, z_s, w_1, \ldots, w_l\}$.

Extend τ with a new unary relation symbol U_0 and let $\tau_0 = \tau \cup \{U_0\}$. Let

β_0: $input(y_0, x_1, \ldots, x_n)$
WHILE $y_0 \neq max$ DO
$guess(y_0)$
IF $[U_0(y_0) \wedge \bigwedge_{1\leq i\leq n} (E(y_0, x_i) \vee E(x_i, y_0)) \wedge \beta|U_0]$ THEN
$y_0 := max$
ELSE

$y_0 := 0$

FI

OD

$output(y_0, x_1, \ldots, x_n)$

where $\beta|U_0$ is the program scheme β with its input and output instructions stripped away and the test, in the WHILE instruction, modified as follows:

$\beta|U_0$: % no input instruction

$\vdots$

WHILE $[\bigwedge_{1\leq i\leq l}(E(y_0, w_i) \vee E(w_i, y_0)) \wedge \forall z_1 \ldots \forall z_s [\bigwedge_{1\leq i\leq s}(E(y_0, z_i) \vee E(z_i, y_0))$
$\longrightarrow t[\ldots\ \alpha[z_1, \ldots, z_s, w_1, \ldots, w_l]\ \ldots]]]$ DO

$\vdots$

OD

$\vdots$

% no output instruction

First note that, technically, β_0 is not in $\boldsymbol{\rho}(1)$ even though β is. This is because, according to our definition of the classes $\boldsymbol{\rho}(m)$, β_0 should have the structure of ρ, and this is not so since we have defined it by adding extra instructions to β. So, formally, $\beta_0 \in \hat{\boldsymbol{\rho}}(1)$ where $\hat{\rho}$ is a program scheme that has the form

$input(y, x_1, \ldots, x_n)$

WHILE $y \neq max$ DO

$guess(y)$

IF ρ THEN

$y := max$

ELSE

$y := 0$

FI

OD

$output(y, x_1, \ldots, x_n)$

where $\{x_1, \ldots, x_n\} = X_\rho$.

One can see that for any $\mathcal{A} \in \mathrm{STRUCT}(\tau)$

$$\mathcal{A} \models \hat{\rho} \text{ iff } \mathcal{A} \models \rho$$

and defining the classes $\hat{\boldsymbol{\rho}}(m)$ as program schemes where the test ρ in $\hat{\rho}$ is substituted by an element of $\boldsymbol{\rho}(m)$, we see that for each $m \geq 0$ $\hat{\boldsymbol{\rho}}(m)$ is equivalent to $\boldsymbol{\rho}(m)$. Thus, showing our result for $\hat{\boldsymbol{\rho}}(m)$ gives implicitly the result for $\boldsymbol{\rho}(m)$.

In particular, $\mathcal{A}_{k,0} \models \beta_0$ and $\mathcal{B}_{k,0} \not\models \beta_0$, since $\mathcal{A}_k \models \beta$ and $\mathcal{B}_k \not\models \beta$ by hypothesis.

Inductively, for $m \geq 1$, let $\tau_m = \tau \cup \{U_0, \ldots, U_{2m}\}$ where $U_0, \ldots, U_{2m}$ are new unary relation symbols. Let

β_m: $input(y_{2m})$

WHILE $y_{2m} \neq max$ DO

$guess(y_{2m})$

IF $\forall y_{2m-1}[U_{2m}(y_{2m}) \wedge U_{2m-1}(y_{2m-1}) \wedge$

$(E(y_{2m}, y_{2m-1}) \vee E(y_{2m-1}, y_{2m})) \wedge \beta_{m-1}]$ THEN

$y_{2m} := max$

ELSE

$y_{2m} := 0$

FI

OD

$output(y_{2m})$

Then, $\beta_m \in \hat{\boldsymbol{\rho}}(m+1)$, and $\mathcal{A}_{k,2m} \models \beta_m$ and $\mathcal{B}_{k,2m} \not\models \beta_m$. This follows from the fact that, for the computations of β_m on $\mathcal{A}_{k,2m}$ and on $\mathcal{B}_{k,2m}$, where the guessed value of y_{2m} is the root c_{2m}, the test in the IF ... THEN instruction is true for

$\beta_m^{\mathcal{A}_{k,2m}}$ but false for $\beta_m^{\mathcal{B}_{k,2m}}$. The reason is as follows: c_{2m} is the unique value for y_{2m} that makes $U_{2m}(y_{2m})$ true, in both $\mathcal{A}_{k,2m}$ and $\mathcal{B}_{k,2m}$ (any other guessed value for y_{2m} makes both computations, $\beta_m^{\mathcal{A}_{k,2m}}$ and $\beta_m^{\mathcal{B}_{k,2m}}$, fail); however, in $\mathcal{A}_{k,2m}$, for all roots c_{2m-1} there is a root c_{2m-2} that can be the guessed value of the (input) variable y_{2m-2} of the program β_{m-1} and, by inductive hypothesis, $\mathcal{A}_{k,2m-2} \models \beta_{m-1}$; hence,

$$\mathcal{A}_{k,2m} \models \forall y_{2m-1}[U_{2m}(y_{2m}) \wedge U_{2m-1}(y_{2m-1}) \wedge (E(y_{2m}, y_{2m-1}) \vee E(y_{2m-1}, y_{2m})) \wedge \beta_{m-1}].$$

On the other hand, in $\mathcal{B}_{k,2m}$, there is a root c_{2m-1} such that all roots c_{2m-2}, with an edge to c_{2m-1}, are in copies of $\mathcal{B}_{k,2m-2}$ and, by inductive hypothesis, $\mathcal{B}_{k,2m-2} \not\models \beta_{m-1}$, so the test is false in $\beta_m^{\mathcal{B}_{k,2m}}$. ∎

Theorem 4.3.4 (Hierarchy Theorem for NPS) *Let $\rho \in \mathrm{NPS}(\tau)$. If there exists families of τ-structures, $\{\mathcal{A}_k\}_{k\geq 0}$ and $\{\mathcal{B}_k\}_{k\geq 0}$, satisfying the hypotheses of Lemmas 4.3.2 and 4.3.3, then, for all $m \geq 0$,*

$$\boldsymbol{\rho}(m) \subsetneq \boldsymbol{\rho}(m+1).$$

Proof: Immediate from Lemmas 4.3.2 and 4.3.3. ∎

4.4 Applications to logics with generalized quantifiers

4.4.1 A hierarchy for TC

We show how our results give Grädel's hierarchy for TC in [Grä91]. Consider the problem $\mathrm{TC}(0, max)$[2]. This is accepted by the program scheme ρ of Example 4.1.1.

[2] $\mathrm{TC}(0, max)$ is the localization of the problem TC to the constants 0 and max; cf. Remark 3.1.1

Let's denote it by ρ_{TC}. Define TC(0) as the class of formulas of the form[3]

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

where ψ is a quantifier-free first-order formula. Define the class of program schemes $\boldsymbol{\rho}_{TC}(0)$ according to the general definition of $\boldsymbol{\rho}(0)$. Then, one can see that, for every $\Phi \in \mathrm{TC}(0)$ there is a $\beta \in \boldsymbol{\rho}_{TC}(0)$ such that, for every $\mathcal{A} \in \mathrm{STRUCT}(\tau_2)$,

$$\mathcal{A} \models \Phi \;\text{ iff }\; [\overline{0}]\beta^{\mathcal{A}}[\overline{max}] \tag{4.2}$$

and conversely: for every $\beta \in \boldsymbol{\rho}_{TC}(0)$ there is a $\Phi \in \mathrm{TC}(0)$ such that (4.2) holds.

Inductively define, for $m > 0$, TC(m) as the class of formulas of the form

$$\mathrm{TC}[\lambda\,\overline{x}\,\overline{y}\,\forall\overline{z}\,\Psi(\overline{x},\overline{y},\overline{z})](\overline{0},\overline{max}),$$

where $\Psi \in \mathrm{TC}(m-1)$, and define, $\boldsymbol{\rho}_{TC}(m)$, as the class of program schemes β such that: β is ρ_{TC}^k, for some $k > 0$, and the test that shows in the IF ... THEN instruction, in ρ_{TC}, is of the form $\forall\overline{z}\,\alpha$ where $\alpha \in \boldsymbol{\rho}_{TC}(m-1)$, and similar proviso about X_α and X_β, as in the general definition of $\boldsymbol{\rho}(m)$, applies.

Then, again, one sees that for every $\Phi \in \mathrm{TC}(m)$ there is a $\beta \in \boldsymbol{\rho}_{TC}(m)$ such that, for every $\mathcal{A} \in \mathrm{STRUCT}(\tau_2)$, equation (4.2) above holds, and conversely.

Now, for each $k \geq 0$, take $\mathcal{A}_k$ a cycle of length 2^{k+2} with interpretations for the two constant symbols 0 and max, and $\mathcal{B}_k$ the disjoint union of $\mathcal{A}_k$ with another cycle of length 2^{k+2} without constants. Then, for all $k \geq 0$, the program scheme β in Example 4.1.2 accepts $\mathcal{A}_k$ and rejects $\mathcal{B}_k$. Thus, ρ_{TC}, $\mathcal{A}_k$, and $\mathcal{B}_k$ satisfy the hypotheses of Theorem 4.3.4 and we can apply that result to our particular sequence of classes of program schemes, $\{\boldsymbol{\rho}_{TC}(m)\}_{m\geq 0}$, to obtain

$$\boldsymbol{\rho}_{TC}(m) \subsetneq \boldsymbol{\rho}_{TC}(m+1), \text{ for all } m \geq 0.$$

[3]cf. Definition 3.3.3

This implies $\mathrm{TC}(m) < \mathrm{TC}(m+1)$ for all $m \geq 0$: Suppose that there is an m such that $\mathrm{TC}(m) = \mathrm{TC}(m+1)$. Since $\boldsymbol{\rho}_{TC}(m) \subsetneq \boldsymbol{\rho}_{TC}(m+1)$, there is a $\beta \in \boldsymbol{\rho}_{TC}(m+1)$ and two τ_2-structures, $\mathcal{B}$ and $\mathcal{C}$, such that

$$\mathcal{B} \models \beta \text{ and } \mathcal{C} \not\models \beta,$$

and

$$\mathcal{B} \models \alpha \text{ iff } \mathcal{C} \models \alpha, \text{ for all } \alpha \in \boldsymbol{\rho}_{TC}(m).$$

We have shown above that there is a $\Phi \in \mathrm{TC}(m+1)$ such that, for every $\mathcal{A} \in \mathrm{STRUCT}(\tau_2)$,

$$\mathcal{A} \models \Phi \text{ iff } [\overline{0}]\beta^{\mathcal{A}}[\overline{max}].$$

So, in particular, $\mathcal{B} \models \Phi$ and $\mathcal{C} \models \neg\Phi$. By our assumption there is a $\Phi' \in \mathrm{TC}(m)$ such that $\models \Phi \longleftrightarrow \Phi'$. On the other hand, there is a program scheme $\beta' \in \boldsymbol{\rho}_{TC}(m)$ such that, for every $\mathcal{A} \in \mathrm{STRUCT}(\tau_2)$,

$$\mathcal{A} \models \Phi' \text{ iff } [\overline{0}]\beta'^{\mathcal{A}}[\overline{max}].$$

In particular, $\mathcal{B} \models \Phi'$ and $\mathcal{C} \models \Phi'$, and since $\models \Phi \longleftrightarrow \Phi'$, we must have $\mathcal{C} \models \Phi$ also, which is a contradiction.

Addendum: For the sake of illustrating the construction described in the proof of Lemma 4.3.3, we give the program scheme in $\boldsymbol{\rho}_{TC}(2)$ that distinguish $\mathcal{A}_{k,2}$ from $\mathcal{B}_{k,2}$.

$input(x, y_2)$
WHILE $x \neq max$ DO
    $guess(y_2)$
    IF [$\forall y_1(U_2(y_2) \wedge U_1(y_1) \wedge E(y_2, y_1) \wedge \alpha_1[y_1, y_0])$] THEN
        $y_2 := max$
    FI

$x := y_2$

OD

$output(x, y_2)$

where α_1 is the following program scheme:

$y_0 := 0$

WHILE $y_0 \neq max$ DO

$guess(y_0)$

IF [$\forall z_1 \forall z_2 (U_0(y_0) \wedge E(y_1, y_0) \wedge \alpha_2[z_1, z_2, z_3])$] THEN

$y_0 := max$

FI

OD

and α_2 is the following program scheme:

WHILE $z_1 \neq z_2$ DO

$guess(z_3)$

IF [$\bigwedge_{i=1,2,3} \neg U_i(z_1) \wedge \bigwedge_{i=1,2,3} \neg U_i(z_3) \wedge E(z_1, z_3) \wedge E(y_0, z_1) \wedge E(y_0, z_3)$] THEN

$z_1 := z_3$

FI

OD

4.4.2 A hierarchy for DTC and relatives

The following program scheme, ρ_{DTC}, accepts the problem DTC(0,max):

$input(x, y)$

WHILE $x \neq max$ DO

$guess(y)$

IF $(E(x, y) \wedge \forall z (E(x, z) \longrightarrow z = y))$ THEN

$x := y$

FI

OD

$output(x, y)$

Define DTC(0) as the class of formulas of the form

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

where ψ is a quantifier-free first-order formula, and define the class $\boldsymbol{\rho}_{DTC}(0)$ according to the general definition of $\boldsymbol{\rho}(0)$. Then, one can see that, for every $\Phi \in$ DTC(0) there is a $\beta \in \boldsymbol{\rho}_{DTC}(0)$ such that, for every $\mathcal{A} \in \mathrm{STRUCT}(\tau_2)$,

$$\mathcal{A} \models \Phi \;\text{ iff }\; [\overline{0}]\beta^{\mathcal{A}}[\overline{max}]$$

and conversely. For $m > 0$, define DTC(m) and $\boldsymbol{\rho}_{DTC}(m)$ analogous to TC(m) and $\boldsymbol{\rho}_{TC}(m)$ in 4.4.1. Using the same families of τ_2-structures, $\{\mathcal{A}_k\}_{k\geq 0}$ and $\{\mathcal{B}_k\}_{k\geq 0}$, defined in 4.4.1, and the program scheme

$input(x, y)$

WHILE $x \neq max$ DO

$guess(y)$

IF $\forall u \forall v$ (WHILE $u \neq v$ DO; $guess(w)$; IF $(E(u,w) \wedge \forall z(E(u,z) \longrightarrow z = w))$ THEN $u := w$; FI; OD) THEN

$y := max$

FI

$x := y$

OD

$output(x, y)$

it is easy to see that the hypotheses of Theorem 4.3.4 are satisfied and, hence, we obtain that the hierarchy $\{\boldsymbol{\rho}_{DTC}(m)\}_{m\geq 0}$ is strict at all levels, which implies that the hierarchy $\{\mathrm{DTC}(m)\}_{m\geq 0}$ is also strict at all levels, by same arguments as in 4.4.1.

Another relative to the problem $\mathrm{TC}(0, max)$ is

$$\begin{aligned}\mathrm{STC}(0, max) = \{\mathcal{A} \in \mathrm{STRUCT}(\tau_2) \mid & \text{ there is a path in the (undirected)} \\ & \text{graph } \mathcal{A} \text{ from vertex } 0^{\mathcal{A}} \text{ to vertex } max^{\mathcal{A}}\}.\end{aligned}$$

This problem is complete for **NSYMLOG** (symmetric logspace) via first-order projections [Imm87], and it is accepted by a program scheme ρ, similar to that in Example 4.1.1, interpreting the edge relation $E(x, y)$ as a symmetric relation. Then, doing the same analysis as in 4.4.1, we obtain that the hierarchy of logics $\{\mathrm{STC}(m)\}_{m\geq 0}$ is strict at all levels.

Bibliography

[ABI93] E. Allender, J. Balcázar, and N. Immerman. A first order isomorphism theorem. In *Proc. 10th Symp. on Theoretical Aspects of Comp. Sci. STACS 93*, volume 665 of *LNCS*, pages 163–174. Springer-Verlag, 1993.

[BL74] W. S. Brainerd and L. H. Landweber. *Theory of Computation.* John Wiley & sons, 1974.

[CS96] S. R. Chauhan and I. A. Stewart. On the power of built-in relations in certain classes of program schemes. Technical report, CS-Department, University of Wales Swansea, 1996.

[DW83] M. Davis and E. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Sciences.* Academic Press, 1983.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 1995.

[ET76] S. Even and R. E. Tarjan. A combinatorial problem which is complete in polynomial space. *J. of the Assoc. for Comp. M.*, 23:710–719, 1976.

[Fag74] 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.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[Grä91] E. Grädel. On transitive closure logic. In *CSL '91: 5th Workshop on Computer Science Logic*, volume 626 of *LNCS*, pages 149–163. SV, 1991.

[HLS65] J. Hartmanis, P.L. Lewis, and R.E. Stearns. Hierarchies of memory–limited computations. In *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logic Design*, pages 179–190, 1965.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, August 1987.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM J. of Comput.*, 17:935–938, 1988.

[Lin66] P. Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

[MP92] J.A. Makowsky and Y.B. Pnueli. Computable quantifiers and logics capturing complexity classes. Technical report, CS-Department, Technion - Israel Institute of Technology, 1992. To appear in: *Quantifiers: Generalizations, Extensions and Variants of Elementary Logic* (M. Krynicki, M. Mostowski and L.W. Szczerba eds.), Kluwer Academic Publishers.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Sch78] T. J. Schaefer. On the complexity of some two-person perfect-information games. *J. of Comp. and System Sci.*, 16:185–225, 1978.

[Ste88] I. A. Stewart. Expressibility, complexity, and comparative schematology. Technical report, University of Newcastle upon Tyne, 1988.

[Ste91] I. A. Stewart. Comparing the expressibility of languages formed using NP-complete operators. *J. Logic Computat.*, 1(3):305–330, 1991.

[Ste92] I. A. Stewart. Using the Hamiltonian path operator to capture NP. *J. of Comp. and System Sci.*, 45:127–151, 1992.

[Ste93] I. A. Stewart. Logical and schematic characterization of complexity classes. *Acta Informatica*, 30:61–87, 1993.

[Ste94a] I. A. Stewart. Logical descriptions of monotone NP problems. *J. Logic Computat.*, 4:337–357, 1994.

[Ste94b] I. A. Stewart. On completeness for NP via projection translations. *Math. Systems Theory*, 27:125–157, 1994.

[Ste96] I. A. Stewart. Logics with zero-one laws that are not fragments of bounded-variable infinitary logic. Technical report, CS-Department, University of Wales Swansea, 1996. To appear in: *Math. Logic Quart.*

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Bull. of the EATCS*, 33:96–100, 1987.