Logic, Computation and Constraint Satisfaction

Barnaby D. Martin

University of Leicester

Submitted for the degree of Doctor of Philosophy

November 2005

Abstract

We study a class of non-deterministic program schemes with while loops: firstly, augmented with a priority queue for memory; secondly, augmented with universal quantification; and, thirdly, augmented with universal quantification and a stack for memory. We try to relate these respective classes of program schemes to well-known complexity classes and logics.

We study classes of structure on which path system logic coincides with polynomial time P.

We examine the complexity of generalisations of non-uniform boolean constraint satisfaction problems, where the inputs may have a bounded number of quantifier alternations (as opposed to the purely existential quantification of the CSP). We prove, for all bounded-alternation prefixes that have some universal quantifiers to the outside of some existential quantifiers (*i.e.* Π_2 and above), that this generalisation of boolean CSP respects the same dichotomy as that for the non-uniform boolean quantified constraint satisfaction problem.

We study the non-uniform QCSP, especially on digraghs, through a combinatorial analog – the alternating-homomorphism problem – that sits in relation to the QCSP exactly as the homomorphism problem sits with the CSP. We establish a trichotomy theorem for the non-uniform QCSP when the template is restricted to antireflexive, undirected graphs with at most one cycle. Specifically, such templates give rise to QCSPs that are either tractable, NP-complete or Pspace-complete.

We study closure properties on templates that respect QCSP hardness or QCSP equality. Our investigation leads us to examine the properties of first-order logic when deprived of the equality relation.

We study the non-uniform QCSP on tournament templates, deriving sufficient conditions for tractability, NP-completeness and Pspace-completeness. In particular, we prove that those tournament templates that give rise to tractable CSP also give rise to tractable QCSP.

Acknowledgements

I would like to thank my global and local supervisors, Iain Stewart and Rick Thomas, respectively, for their help, guidance and advice throughout the last four years. I would like to thank my mentor Florent Madelaine for his assistance in the field of constraint satisfaction problems. I would also like to thank Florent and his wife Sukhi for their hospitality in Newcastle on many occasions.

I would like to thank my examiners, Dave Cohen and Vincent Schmitt, for undertaking the arduous task of reading this thesis, and for their many helpful suggestions.

I am very grateful to the Engineering and Physical Sciences Research Council (EPSRC) for funding this research.

Contents

1 Introduction 5

2 Program Schemes 9

2.1 Structures and Logic . . . 9

2.1.1 Graphs and Transitive Closure . . . 10

2.1.2 Alternating graphs and Alternating Reachability . . . 11

2.1.3 Hypergraphs and Path Systems . . . 12

2.1.4 Least Fixed Point logic . . . 13

2.1.5 Stratified Fixed Point logic . . . 14

2.2 Program Schemes . . . 14

2.2.1 Introducing NPS . . . 15

2.2.2 Shorthands . . . 18

2.2.3 Shorthands on successor structures . . . 19

2.2.4 Adding a stack: introducing NPSS . . . 20

2.3 Turing Machines . . . 21

3 Adding a Priority Queue 22

3.1 A single weight: $k = 0$ . . . 25

3.2 The Hamilton Path problem is in NPSPQ . . . 27

3.3 $\mathrm{NPSPQ}_s^b \subseteq \mathsf{NPspace}$ . . . 29

3.4 Expanding alphabets . . . 30

3.5 Pushing and Popping Numbers in NPSPQ_s^b / NPSPQ_s^u. . . 31

3.5.1 $\mathrm{NPSPQ}_s^{u+}(k) = \mathrm{NPSPQ}_s^u(k)$. . . 32

3.5.2 $\mathrm{NPSPQ}_s^{b+} = \mathrm{NPSPQ}_s^b$. . . 32

3.6 $\mathrm{NPSPQ}^b(k) \subseteq \mathrm{NPSPQ}^u(k)$ . . . 33

3.7 $\mathrm{NPSPQ}_s^u(k) \subseteq \mathrm{NPSPQ}^u(k+2)$. . . 34

3.7.1 Start-routine: τ_{start} . . . 35

3.7.2 Simulation of ρ: τ_ρ. . . 35

3.7.3 End-routine: τ_{end} . . . 36

3.8 $\mathsf{NPspace} \subseteq \mathrm{NPSPQ}_s^u$ . . . 38

3.8.1 Preparation: $\tau_{bin(\mathcal{A})}$ . . . 39

3.8.2 Simulation: τ_{sim} 41
3.8.3 Verification: τ_{end} 42
3.9 A polynomial-time restriction of NPSPQ^u. 44

4 Adding Universal Quantification 46
4.1 Introducing $\mathrm{APS}(1)$ 46
4.2 Introducing $\mathrm{APSS}(1)$ 49
4.3 The ACCEPT instruction. 51
4.4 $\mathrm{APSS}_s(1) = \mathrm{APSS}(1)$ 52
4.4.1 Simulation of ρ; τ_ρ 53
4.5 $\mathsf{NPspace} \subseteq \mathrm{APSS}_s(1)$ 54
4.6 Summary 58

5 Classes of Structure on which $\mathsf{P} = \pm\mathbf{PS}^k[\mathbf{FO}]$ 60
5.1 Finitely generated sets 61
5.1.1 F contains a single k-ary partial function f_0. 62
5.1.2 F contains multiple partial functions. 64
5.1.3 An application: finitely generated groups 65
5.2 Hamiltonian Outerplanar graphs 65

6 Dichotomies in Boolean Constraint Satisfaction 68
6.1 Introduction 68
6.2 Technical results 72
6.3 A dichotomy theorem for Π_2-SAT$_{\mathrm{NC}}$ 75
6.3.1 Case 1 : $R^{\mathcal{A}}$ is 0-valid and not 1-valid. 75
6.3.2 Case 2 : $R^{\mathcal{A}}$ is 1-valid and not 0-valid. 76
6.3.3 Case 3 : $R^{\mathcal{A}}$ is 0-valid and 1-valid, but not complimentative. 76
6.3.4 Case 4 : $R^{\mathcal{A}}$ is 0-valid and 1-valid, and complimentative. . 77

7 Quantified Constraints on Graphs 79
7.1 Introduction 79
7.2 Preliminaries 81
7.2.1 Structures and Logic. 81
7.2.2 Alternating-homomorphism problems. 82
7.2.3 QCSP versus ALT-HOM. 85
7.2.4 Alternating-homomorphisms as winning strategies. 86
7.2.5 Graphs 87
7.3 Basic graph results 88
7.3.1 Restricting partitions 88
7.3.2 Basic results. 90
7.3.3 Non-connected templates. 93

7.4 Quantified $\mathcal{H}$-colouring 94
7.4.1 Bipartite templates. 94
7.4.2 Odd Catherine Wheels 95
7.4.3 A trichotomy theorem 101
7.5 Closure properties 101
7.5.1 Indicator construction 102
7.5.2 Adding a vertex to a core. 104
7.5.3 A sufficient condition for ALT-HOM problem equivalence. 107
7.5.4 Why that condition is not necessary: equivalence in fragments of **FO**. 108
7.6 Results concerning tournament templates 113
7.6.1 Template is a directed cycle. 113
7.6.2 Template is a digraph with source and sink. 114
7.6.3 Tractable tournament ALT-HOM problems. 115
7.6.4 NP-complete tournament ALT-HOM problems. 118
7.6.5 Pspace-complete tournament ALT-HOM problems. 118

8 Conclusions and Further Work 121
8.1 Program Schemes 121
8.2 Classes of Structure on which $\mathsf{P} = \pm\mathrm{PS}^k[\mathbf{FO}]$ 121
8.3 Quantified Constraints on Graphs 122

Chapter 1

Introduction

Structural Complexity is that part of the study of Computational Complexity that concerns itself with the intrinsic computational difficulty of decidable problems. Perhaps its main thrust is an attempt to classify problems into *complexity classes* by various upper and lower bounds on their computational complexity. Since its inception, logic has impinged on Computational Complexity in a variety of ways: in the first instance, many of the problems that are amongst the hardest of many natural complexity classes have been problems in logic. These problems, known as *complete* for the given complexity class, include the following problems in the Propositional Calculus: Circuit Value, complete for P; Satisfiability, complete for NP; and Quantified Satisfiablity, complete for Pspace.

Another intersection between logic and complexity is in the field of Descriptive Complexity – in which finding an algorithm for a computational problem is seen as a question of expression – where complexity classes translate to classes of expressions, *i.e.* logics. Indeed, the complexity classes, defined through Turing Machines, may be seen as logics already: for example, if P is defined as those Turing Machines T, such that there exists k, such that T accepts an input x of size n iff T accepts x within time n^k, then this class of Turing Machines is a logic of sorts. However, it is a logic not of a form that lends itself to study by what are usually considered the tools of logic: it has a cumbersome syntax and is interpreted over strings and not first-order structures. We may remedy the second of theses problems by considering certain standard binary string encodings of a structure. Since, for us, a decision problem is always a subset of finite structures, we consider a problem to be in P iff the language of all binary encodings of these structures is in P, as defined previously. In this way, we can talk of conventional logics capturing Turing complexity classes: a logic captures a complexity class iff the set of problems expressible in each coincides. It is in the translation between computational problems over strings and expression problems over finite structures that Descriptive Complexity is concerned. As such, it is very much a part of

Finite Model Theory.

Perhaps the greatest hope for Descriptive Complexity was that known methods for separating logics might be brought to bear on complexity classes; that hard questions on the (non)-equivalence of complexity classes might become easier questions on the separation of logics. Among the major results of Descriptive Complexity are the proven equivalence of: existential second-order logic $\exists$**SO** and NP (Fagin, 1974 [17]); least fixed-point logic LFP (with successor) and P (Immerman/Vardi 1982 [28, 44]); partial fixed point logic PFP (with successor) and Pspace (Vardi 1982 [44]); and transitive closure logic TC (with successor) and NL (Immerman 1983/1988 [28, 30]). Despite these results, few advances have been made in the use of techniques such as Ehrenfeucht-Fraisse games to separate these logics, and, consequently, their complexity classes[1]. Partly, this can be explained by the somewhat artificial inclusion of the successor function into many of these logics (not $\exists$**SO**), since Ehrenfeucht-Fraisse games are notoriously hard to win on structures with successor. However, the equivalence of LFP and PFP, both with successor, is known to be consequent on their equivalence without successor [1], yet still a proof resists that $\mathsf{P} \neq \mathsf{Pspace}$. One ray of sunshine in this field was Immerman's proof, through Transitive Closure logic, that NL =co-NL [30].

The syntax of a logic is exactly its set of well-formed formulae; a logic is said to have recursive syntax iff its syntax is decidable. The inclusion of successor generally precludes the possibility that the resultant logic has recursive (or even recursively enumerable) syntax [24], a property which is certainly desirable, and is thought by some authors (*e.g.* Gurevich [24], Otto [36]) to be necessary, if the name 'logic' is to be bestowed.

In the first part of this thesis we study various classes of non-deterministic Program Schemes with while loops (based on those in [2, 41]), which are logics in Gurevich's sense, but which appear well-suited for computation. We attempt to relate these logics to standard complexity classes, preferably in the absence of that built-in successor.

Chapter 2 introduces these program schemes, and discusses some known results involving them [2]. The situation where a stack is available for memory [2] is considered.

In Chapter 3, we introduce some new work investigating the addition of a priority queue as a memory device available to these program schemes. We prove that the priority queue is sufficiently powerful to simulate a successor function: thus we define two 'logics' with recursive syntax that subsume NPspace[2] and NP,

[1]Indeed, it is not known that NL,P,NP and Pspace are not equivalent.

[2]We remind the reader that NPspace and Pspace coincide (*e.g.* [37]). Even so, we use both classes in this thesis, depending on which appears the more natural in a given situation.

respectively. These logics actually have identical syntax, and differ only in their semantics.

In Chapter 4, we introduce some new work examining the explicit introduction of universal quantification to our program schemes (since they are non-deterministic, existential quantification is already insinuated), both in the absence of any memory and with the benefit of a stack. With no added memory, we tie the ensuing logic to least fixed-point logic LFP. With the addition of a stack we, once again, are able to simulate a successor: this enhanced logic subsumes NPspace.

Chapter 5 uses some known results utilising program schemes [42] to study various classes of structure on which the infinite hierarchy which constitutes Path System logic collapses and captures P. We give a brief overview of known results and introduce some new ones.

A further confluence of logic and complexity, and, indeed, combinatorics, is in the study of the Constraint Satisfaction Problem, CSP, and its generalisations. In terms of complexity classification, much has been made of the conjectured dichotomy of the non-uniform CSP on finite templates [1]: it seems as though, for any $\mathcal{T}$, $\mathrm{CSP}(\mathcal{T})$ is either tractable or NP-complete. This is remarkable given the breadth of CSP problems, together with Ladner's result [35] that such a dichotomy will not hold over all NP (unless we actually have $\mathsf{P} = \mathsf{NP}$). The non-uniform CSPs, and their generalisations, lend themselves to dual interpretations: one in which they are model-checking problems over restricted logics; and one in which they are combinatorial problems between two structures. In particular, the non-uniform CSP may be seen as both a model-checking problem in existential positive conjunctive first-order logic and as the homomorphism problem. This duality is perhaps at its most obvious on graphs, and it is on these that we dwell most.

Chapter 6 concerns itself with the dichotomy of alternation-bounded generalisations of the non-uniform CSP on boolean templates. (These results have been obtained independently in [27] and, to a lesser extent, in [18].) We prove, for all bounded-alternation prefixes that have some universal quantifiers to the outside of some existential quantifiers (*i.e.* Π_2 and above), that our generalisation of boolean CSP respects the same dichotomy as that for boolean quantified constraint satisfaction problems.

Chapter 7 examines the non-uniform Quantified Constraint Satisfaction Problem, QCSP, on graph templates. We study the QCSP through a combinatorial analog, the so-called Alternating-Homomorphism problem ALT-HOM. We study a variety of graph templates that give rise to tractable, NP-complete or Pspace-complete QCSPs, culminating in a complete classification to those classes – a trichotomy theorem – when the template ranges over undirected antireflexive graphs with at most one cycle.

We consider two problems to be equal exactly when the respective subsets of structures that they define coincide. It is well-known that the two problems $\mathrm{CSP}(\mathcal{T})$ and $\mathrm{CSP}(\mathcal{T}')$ are equal iff the templates $\mathcal{T}$ and $\mathcal{T}'$ are homomorphically equivalent (which is exactly the condition that they have isomorphic cores). We study a similar condition on templates $\mathcal{T}, \mathcal{T}'$ that is sufficient to guarantee the equality of $\mathrm{QCSP}(\mathcal{T})$ and $\mathrm{QCSP}(\mathcal{T}')$. However, we find that this condition is not necessary, and that is has a closer relationship with first-order logic without equality than the logic we associate with QCSP (positive-conjunctive first-order logic).

Finally, we study the non-uniform QCSP on tournament templates, deriving sufficient conditions for tractability, NP-completeness and Pspace-completeness. In particular, we prove that those tournament templates that give rise to tractable CSP also give rise to tractable QCSP.

Chapter 2

Program Schemes

2.1 Structures and Logic

We will only consider finite relational structures, of at least two elements, over a given signature σ. We denote this set $\mathrm{STRUC}(\sigma)$. If $\mathcal{A}$ is a structure, then $|\mathcal{A}|$ is the universe, or domain, of the structure, and $||\mathcal{A}||$ is the cardinality of that domain. If R is a relation symbol of σ then $R^{\mathcal{A}}$ is the interpretation of R over $\mathcal{A}$. When the structure $\mathcal{A}$ is clear, we may abuse notation by dropping it as the superscript, thus identifying R with both the relation *symbol* and the relation *actual*.

We will also consider the situation where a successor is available to us, built-in to the signature σ. We consider a successor to be a binary relation *succ*, whose realisation as a graph is a directed path, together with two constants *min* and *max*, whose interpretations are the first and last vertices of that path. This is equivalent to considering the restricted class of structures over $\sigma \uplus \{succ, min, max\}$ in which the interpretations of *succ*, *min* and *max* satisfy the properties given. Throughout, when we consider the restriction to structures that have a successor relation, we add the subscript s to our logic or class, for example $\mathbf{FO}_s$. When we consider logics in which we have a successor, we will insist on a further *semantic* restriction on their formulae, namely, that a formula may only be in that logic if its truth is independent of the actual successor function used. For example, consider the formula $E(min, max)$, ostensibly of $\mathbf{FO}_s$, interpreted on the directed 3-path – the graph with vertices $\{0,1,2\}$ and edge set $\{(0,1),(1,2)\}$. The truth of this formula is not independent of the ordering we choose on the graph – if the successor is $\{(0,2),(2,1)\}$, it is true; if the successor is $\{(0,1),(1,2)\}$, it is false. We conclude that $E(min, max)$ is not a formula of $\mathbf{FO}_s$. Given such a formula, ostensibly of $\mathbf{FO}_s$, establishing whether it has this property of order-independence is, in general, undecidable [24].

2.1.1 Graphs and Transitive Closure

A graph, or digraph, $\mathcal{G}$ is a structure over signature $\sigma_2 = \langle E \rangle$, where E is a binary relation symbol. There is a (directed) path in $\mathcal{G}$ from vertex x to vertex y iff either: $x = y$, $E(x,y)$, or there is a sequence of vertices $z_1, \dots, z_r$ such that $E(x,z_1)$, $E(z_i, z_{i+1})$, for $1 \leq i < r$, and $E(z_r, y)$. This is equivalent to the inductive definition that there is a path from x to y iff:

- $x = y$, or
- there is a z such that $E(x,z)$, and there is a path from z to y.

Definition. Define TC to be the global binary relation on graphs expressing reachability. Specifically:

$$\text{TC} = \{\ \{(x,y) : \text{ there is a path in } \mathcal{G} \text{ from } x \text{ to } y\ \} : \mathcal{G} \in \text{STRUC}(\sigma_2)\}$$

Let ψ be some formula whose only free variables are among those of the j-tuples $\overline{x}$ and $\overline{y}$. A formula $\text{TC}[\lambda \overline{x}, \overline{y} \psi](\overline{u}, \overline{v})$ is interpreted as true on a structure $\mathcal{A}$ in the case that, in the graph of $||\mathcal{A}||^j$ vertices with edge set specified by $\psi(\overline{x}, \overline{y})$, there is a path from vertex $\overline{u}$ to vertex $\overline{v}$. (It is usual to allow additional free variables, *i.e.* other than $\overline{x}$ and $\overline{y}$, in ψ. However, this does not increase our expressive power, since i such variables can be moved so they only appear free in the end-point $(j+i)$-tuples $\overline{\overline{u}}'$ and $\overline{\overline{v}}'$ of some new Transitive Closure formula over a graph of size $||\mathcal{A}||^{j+i}$ (see [16]). We forbid such additional free variables for the sake of a simpler exposition.)

In the fashion described, the global relation TC has given us a uniform sequence of vectorised quantifiers of the same name. This sequence is derived from the arity of $\overline{x}$ and $\overline{y}$. The first quantifier in the sequence corresponds to the arity of $\overline{x} = (x_1)$ and $\overline{y} = (y_1)$ being 1, and binds the 2 variables, x_1 and y_1. The ith quantifier in this sequence corresponds to the arity of $\overline{x} = (x_1, \dots, x_i)$ and $\overline{y} = (y_1, \dots, y_i)$ being i, and binds the $2i$ variables $x_1, \dots, x_i$ and $y_1, \dots, y_i$. This sequence of quantifiers is uniform in semantics and syntax, and is an example of a sequence of Lindström quantifiers (see, *e.g.*, [16]).

Definition. Let $\overline{x}, \overline{y}$ be j-tuples of variables. Let $j', j'' \leq j$ and $u_1, \dots, u_{j'}, v_1, \dots, v_{j''}$ be variables, and $u_{j'+1}, \dots, u_j, v_{j''+1}, \dots, v_j$ be variables or constant symbols. Define:

- $\pm\text{TC}^1[\mathbf{FO}]$ to be the set of formulae of the form

$$\exists u_1 \dots u_{j'} \exists v_1 \dots v_{j''}\ \text{TC}[\lambda \overline{x}, \overline{y} \psi](\overline{u}, \overline{v})$$

  where ψ is quantifier-free, and

- $\pm\mathrm{TC}^{m+1}[\mathbf{FO}]$ to be the set of formulae of the form

  $$\exists u_1 \ldots u_{j'} \exists v_1 \ldots v_{j''}\ \mathrm{TC}[\lambda\bar{x},\bar{y}\psi](\bar{u},\bar{v})$$

  where ψ is in the closure under boolean operators of formulae in $\pm\mathrm{TC}^m[\mathbf{FO}]$.

In the presence of two distinct constants, any formula in $\mathrm{TC}^{m+1}[\mathbf{FO}]$ is equivalent to some formula of the form $\mathrm{TC}[\lambda\bar{x},\bar{y}\psi](\bar{u},\bar{v})$, with $\psi \in \mathrm{TC}^m[\mathbf{FO}]$, *i.e.* without the need for existential quantification outside the TC operator [21]. However, we do not wish to restrict ourselves only to structures with such constants. We define $\pm\mathrm{TC}^*[\mathbf{FO}]$ to be $\bigcup_{i\in\omega} \pm\mathrm{TC}^i[\mathbf{FO}]$.

Recall that the subscript s denotes a built-in successor. The following gives us an idea as to the power of Transitive Closure logic.

Proposition 1 (Immerman 1983/1988, [29, 30])**.** $\mathrm{TC}^1_s[\mathbf{FO}] = \mathrm{TC}^*_s[\mathbf{FO}] = \mathsf{NL}$.

Remark. It may be noticed that we are rather liberal with notation such as TC, allowing it to denote a global relation, an operator and a logic. Hopefully, the meaning should be clear by context.

2.1.2 Alternating graphs and Alternating Reachability

An *alternating graph* $\mathcal{A}$ is a structure over the signature $\sigma_{21} = \langle E, U\rangle$, where the relation symbols E and U are binary and unary, respectively. In an alternating graph the unary relation U partitions the vertices into those that are existential $(\neg U)$, and those that are universal (U). There is an alternating path in an alternating graph $\mathcal{A}$, from vertex x to vertex y, iff:

- $x = y$, or
- x is existential, and there is a z such that $E(x,z)$, and there is an alternating path from z to y, or
- x is universal, and, for all z such that $E(x,z)$, there is an alternating path from z to y.

Definition. Define AR to be the global binary relation on graphs expressing alternating reachability. Specifically:

$$\mathrm{AR} = \{\{(x,y) : \text{there is an alternating path in } \mathcal{A} \text{ from } x \text{ to } y\} : \mathcal{A} \in \mathrm{STRUC}(\sigma_{21})\}$$

Let ψ be some formula whose only free variables are among those of $\bar{x}$ and $\bar{y}$. A formula $\mathrm{AR}[\lambda\bar{x},\bar{y}\psi](\bar{u},\bar{v})$ is interpreted as true in the case that, in the graph specified by $\psi(\bar{x},\bar{y})$, there is an alternating path from $\bar{u}$ to $\bar{v}$. (Again, it is customary to permit additional free variables in ψ. Again, it is unnecessary for the same reason as given for TC.)

Definition. Let $\overline{x}, \overline{y}$, be j-tuples of variables. Let $j', j'' \leq j$ and $u_1, \ldots, u_{j'}, v_1, \ldots, v_{j''}$ be variables, and $u_{j'+1}, \ldots, u_j, v_{j''+1}, \ldots, v_j$ be variables or constant symbols. Define:

- $\pm\mathrm{AR}^1[\mathbf{FO}]$ to be the set of formulae of the form:

$$\exists u_1 \ldots u_{j'} \exists v_1 \ldots v_{j''} \ \mathrm{AR}[\lambda \overline{x}, \overline{y} \psi](\overline{u}, \overline{v})$$

  where ψ is quantifier-free, and

- $\pm\mathrm{AR}^{m+1}[\mathbf{FO}]$ to be the set of formulae of the form

$$\exists u_1 \ldots u_{j'} \exists v_1 \ldots v_{j''} \ \mathrm{AR}[\lambda \overline{x}, \overline{y} \psi](\overline{u}, \overline{v})$$

  where ψ is in the closure under boolean operators of formulae in $\pm\mathrm{AR}^m[\mathbf{FO}]$.

We define $\pm\mathrm{AR}^*[\mathbf{FO}]$ to be $\bigcup_{i \in \omega} \pm\mathrm{AR}^i[\mathbf{FO}]$. In the presence of two distinct constants, $\pm\mathrm{AR}^{m+1}[\mathbf{FO}]$ collapses to the class of formulae of the form $\mathrm{AR}[\lambda \overline{x}, \overline{y} \psi](\overline{u}, \overline{v})$ for $\psi \in \pm\mathrm{AR}^m[\mathbf{FO}]$ (proof similar to that for TC).

Recall that the subscript s denotes a built-in successor. The following gives us an idea as to the power of Alternating Reachability logic.

Proposition 2 (Immerman 1983, [29])**.** $\mathrm{AR}^1_s[\mathbf{FO}] = \mathrm{AR}^*_s[\mathbf{FO}] = \mathsf{P}$.

2.1.3 Hypergraphs and Path Systems

We consider a *hypergraph*[1] $\mathcal{H}$ to be a structure over the signature $\sigma_3 = \langle R \rangle$, where R is a ternary relation symbol. A vertex y is said to be *R-accessible* (or just *accessible*) from a vertex x iff:

- $x = y$, or
- there exist z_1, z_2, both accessible from x, such that $R(z_1, z_2, y)$.

A hypergraph $\mathcal{H}$ is said to be *commutative* exactly when, for all x, y, z, we have $R(x,y,z)$ iff $R(y,x,z)$. It is said to be *deterministic* iff, for all x, y, there exists at most one z such that $R(x,y,z)$.

Definition. Define PS to be the global binary relation on commutative hypergraphs expressing accessibility. Specifically:

$$\mathrm{PS} = \{\{(x,y) : \ y \text{ is accessible in } \mathcal{H}, \text{ from } x\ \} : \mathcal{H} \text{ is a commutatative hypergraph}\}$$

[1]What we refer to as a hypergraph would perhaps be better described as a *directed* 3-*uniform hypergraph*, taking into consideration more standard definitions.

Let ψ be some formula whose only free variables are among those of $\overline{x}$, $\overline{y}$ and $\overline{z}$. A formula $\mathrm{PS}[\lambda\overline{x},\overline{y},\overline{z}\psi](\overline{u},\overline{v})$ is interpreted as true in the case that, in the commutative hypergraph specified by $\psi(\overline{x},\overline{y},\overline{z})$, $\overline{v}$ is accessible from $\overline{u}$. (Again, it is customary to permit additional free variables in ψ. Again, it is unnecessary for the same reason as given for TC.)

Definition. Let $\overline{x},\overline{y}$, be j-tuples of variables. Let $j', j'' \leq j$ and $u_1,\ldots,u_{j'},v_1,\ldots,v_{j''}$ be variables, and $u_{j'+1},\ldots,u_j,v_{j''+1},\ldots,v_j$ be variables or constant symbols. Define:

- $\pm\mathrm{PS}^1[\mathbf{FO}]$ to be the set of formulae of the form
$$\exists u_1\ldots u_{j'}\exists v_1\ldots v_{j''}\ \mathrm{PS}[\lambda\overline{x},\overline{y},\overline{z}\psi](\overline{u},\overline{v})$$
where ψ is quantifier-free, and
- $\pm\mathrm{PS}^{m+1}[\mathbf{FO}]$ to be the set of formulae of the form
$$\exists u_1\ldots u_{j'}\exists v_1\ldots v_{j''}\ \mathrm{PS}[\lambda\overline{x},\overline{y},\overline{z}\psi](\overline{u},\overline{v})$$
where ψ is in the closure under boolean operators of formulae in $\pm\mathrm{PS}^m[\mathbf{FO}]$.

We define $\pm\mathrm{PS}^*[\mathbf{FO}]$ to be $\bigcup_{i\in\omega}\pm\mathrm{PS}^i[\mathbf{FO}]$. In the presence of two distinct constants, $\pm\mathrm{PS}^{m+1}[\mathbf{FO}]$ collapses to the class of formulae of the form $\mathrm{PS}[\lambda\overline{x},\overline{y}\psi](\overline{u},\overline{v})$ for $\psi\in\pm\mathrm{PS}^m[\mathbf{FO}]$ (proof similar to that for TC).

Recall that the subscript s denotes a built-in successor. The following gives us an idea as to the power of Path System logic.

Proposition 3 ([40])**.** $\mathrm{PS}^1_s[\mathbf{FO}] = \mathrm{PS}^*_s[\mathbf{FO}] = \mathsf{P}$.

2.1.4 Least Fixed Point logic

Let $\psi(P,\overline{x})$ be a first-order formula with free j-ary relation symbol P whose only free variables are those of the j-tuple $\overline{x}$. Then, over a structure $\mathcal{A}$, ψ may be seen as a function $f_{\mathcal{A}} : \mathcal{P}(|\mathcal{A}|^j) \to \mathcal{P}(|\mathcal{A}|^j)$ defined by:

$$f_{\mathcal{A}}(R) = \{\overline{x}\ :\ \mathcal{A} \models \psi(R,\overline{x})\}$$

If ψ does not contain negated instances of the free relation symbol P (*i.e.* is P-positive), then the function $f_{\mathcal{A}}$ is monotone, satisfying $R \subseteq f_{\mathcal{A}}(R)$. Given a P-positive ψ, we define inductively: $\psi^0_{\mathcal{A}} = \phi$, and thereafter $\psi^{k+1}_{\mathcal{A}} = f_{\mathcal{A}}(\psi^k_{\mathcal{A}})$. Since f is monotone and $\mathcal{A}$ is finite, we are guaranteed that this sequence of relations must reach a fixed-point K where $\psi^K_{\mathcal{A}} = \psi^i_{\mathcal{A}}$ (for all $i \geq K$). This relation is denoted $\psi^\infty_{\mathcal{A}}$.

Definition. Given a formula $\psi(P,\overline{x})$ with free j-ary relation symbol P and including free variables of the j-tuple $\overline{x}$, and another j-tuple of variables or constants $\overline{u}$, we may apply the Least Fixed Point operator LFP to generate the formula $\mathrm{LFP}[\lambda P\overline{x}\psi](\overline{u})$. This formula's free variables are those free in ψ that are not in $\overline{x}$, and those of $\overline{u}$. The formula is interpreted as true on a structure $\mathcal{A}$ (under some valuation of its free variables) exactly when $\overline{u} \in \psi^{\infty}_{\mathcal{A}}$.

Least Fixed Point logic $\mathrm{LFP}[\mathbf{FO}]$ is the closure of **FO** under the Least Fixed Point operator.

Remark. It may be noted that we have allowed free variables in ψ that are not among the variables of $\overline{x}$, in contrast with the situation with the Lindström logics of the previous sections. It seems particularly unnatural to specify LFP with such free variables forbidden, moreover, we will make use of them in later chapters. It suffices to say that these additional variables could be forbidden by being forced into the outer tuple $\overline{u}$, as described for Transitive Closure logic.

2.1.5 Stratified Fixed Point logic

Definition. Let R be a free j-ary relation, and $\overline{x}$ be a j-tuple of variables. Let $j' \leq j$ and $u_1,\ldots,u_{j'}$ be variables, and $u_{j'+1},\ldots,u_j$ be variables or constant symbols. Define:

- $\exists\mathrm{LFP}^1[\mathbf{FO}]$ to be the set of formulae of the form $\exists u_1 \ldots u_{j'} \mathrm{LFP}[\lambda R\overline{x}\psi](\overline{u})$ where ψ is first-order with no universal quantifiers and with negation only of atomic formulae, and
- $\exists\mathrm{LFP}^{m+1}[\mathbf{FO}]$ to be the set of formulae of the form $\exists u_1 \ldots u_{j'} \mathrm{LFP}[\lambda R\overline{x}\psi](\overline{u})$ where ψ is first-order with no universal quantifiers but may contain positive or negative occurrences of formulae of $\exists\mathrm{LFP}^m[\mathbf{FO}]$ that do not contain R.

We naturally define $\exists\mathrm{LFP}^*[\mathbf{FO}]$ to be $\bigcup_{i\in\omega}\exists\mathrm{LFP}^i[\mathbf{FO}]$. In the presence of two distinct constants, $\exists\mathrm{LFP}^{m+1}[\mathbf{FO}]$ collapses to the class of formulae of the form $\exists\mathrm{LFP}[\lambda R\overline{x}\psi](\overline{u},\overline{v})$ for $\psi \in \exists\mathrm{LFP}^m[\mathbf{FO}]$ [22].

We note that $\exists\mathrm{LFP}^*[\mathbf{FO}]$ is often known as *Stratified Fixed Point logic* SFP. The following gives us an idea as to the power of $\exists$LFP.

Lemma 4 (Grohe 1997, [22])**.** $\pm\mathrm{PS}^m[\mathbf{FO}] = \exists\mathrm{LFP}^m[\mathbf{FO}]$.

2.2 Program Schemes

We will examine several classes of non-deterministic program schemes with while loops, originally seen in [2]. These program schemes were born of an attempt to imbue logic with the tools of computation, whilst keeping that logic well-behaved, *e.g.*, with recursive syntax. Unlike Turing Machines, which compute on strings encoding some structure, these schemes compute on a structure, in a similar manner to a formula of logic being interpreted on that structure. However, the syntax of computation is often more easily followed, and this may have advantages in simplifying proofs. For example, the recursion of while loops may be considered more natural than that of fixed point logics. Such advantages are largely cosmetic, but, in studying objects of computation, forms of memory can be added that would be most bizarre added directly to conventional logic. In doing this, new logics can be defined without obvious parallel in conventional logic. However, that which is not obvious is not necessarily untrue, and several results are known tying these new logics with their better known, conventional counterparts.

2.2.1 Introducing NPS

Definition (Syntax of NPS [2])**.** Each program scheme $\rho \in \mathrm{NPS}(1)$, over signature σ, involves a set of input-output variables V_{io}, a set of free variables V_f, and a finite sequence of $|\rho|$ instructions, where each instruction, other than the first and last, is of one of the following forms:

- an assignment instruction of the form '$v := q$' , where $v \in V_{io}$ and $q \in V_{io} \cup V_f \cup \{c : c \text{ is a constant symbol of } \sigma\}$, or
- a guess instruction of the form 'GUESS v' , where $v \in V_{io}$, or
- while loops of the form 'WHILE t DO τ OD' , where t is quantifier-free **FO**(σ) with free variables among $V_{io} \cup V_f$, and where τ is a sequence of instructions of one of the forms listed.

The first instruction is INPUT(V_{io}), and the last OUTPUT(V_{io}). All instructions begin a new line, and all, except while loops, take up only one line. While loops take up $1 + |\tau|$ lines, where $|\tau|$ is the number of lines in τ, in the obvious way. We consider sub-routines τ to be sequences of instructions of the types in the list, *i.e.* program schemes without an input instruction at the beginning and an output instruction at the end.

The program schemes $\rho \in \mathrm{NPS}(m+1)$ are defined exactly as the schemes of $\mathrm{NPS}(1)$, except that schemes $\rho' \in \mathrm{NPS}(m')$ (for $m' \leq m$) may take the place of extensional relations in the tests in while loops. We define NPS to be $\bigcup_{i \in \omega} \mathrm{NPS}(i)$.

In terms of semantics, the assignment instructions and while loops behave in the obvious way, and the guess instruction non-deterministically assigns an element of the universe to the variable in question. At the start of computation, the input-output variables are GUESSed, as just specified. A computation is deemed accepting if, and only if, it reaches the (final) OUTPUT line. It follows that non-accepting computations are forever trapped in while loops. Suppose a program scheme $\rho \in \text{NPS}(m)$ involves precisely i free variables $z_1, \ldots, z_i$. Then, computing on a structure $\mathcal{A}$, we write $(\mathcal{A}, a_1 \ldots, a_i) \models \rho$, or $\mathcal{A} \models \rho(a_1, \ldots, a_i)$, iff ρ makes it to the output instruction when computing on $\mathcal{A}$ under the free-variable assignment $(z_1, \ldots, z_i) := (a_1, \ldots, a_i) \in |\mathcal{A}|^i$.

Note that free variables may not be 'used' during computation, in that they can not have values assigned to them. However, input-output variables of schemes in $\text{NPS}(m)$ may appear as free variables in schemes of strictly lower strata that appear in tests in their while loops. In this manner, program schemes of $\text{NPS}(m)$ are evaluated 'top-down', entering sub-routines to evaluate any required tests involving such schemes of $\text{NPS}(m')$ (where $m' < m$).

Definition ([2])**.** Let the lines on $\bar{\bar{z}}$ denote an i-tuple, and the line on $\bar{v}$ denote a j-tuple. Suppose the program scheme $\rho \in \text{NPS}(1)$ involves i free variables $\bar{\bar{z}}$ and j input-output variables $\bar{v}$. Then a configuration of ρ, computing on a structure $\mathcal{A}$, is an $(i+1+j)$-tuple $(\bar{\bar{z}}, l, \bar{v})$ giving the values of the free variables, the number of the line just executed, and the values of the input-output variables.

Each such program scheme $\rho \in \text{NPS}(1)$, computing over a structure $\mathcal{A}$ of size n, gives rise to a graph:

- whose vertices are the $|\rho|.n^{(i+j)}$ possible configurations,
- and in which there is an edge (c, c') iff ρ, executing a single instruction, can move from configuration c to configuration c'.

It may be asking too much to specify this graph in quantifier-free **FO**, especially since $\mathcal{A}$ may not have $|\rho|$ distinct constants to play the part of the line numbers. We will actually specify a variant of it, namely the graph $\mathcal{G}^{\rho}_{\mathcal{A}}$, with $n^{i+|\rho|+j}$ vertices, where the $|\rho|$-sub-tuple $\widehat{w} = (w_1, \ldots, w_{|\rho|})$ represents a certain line according to the following scheme:

- if $w_1 = w_2$ then $\widehat{w}$ represents line 1,
- if $w_1 \neq w_2$ but $w_2 = w_3$ then $\widehat{w}$ represents line 2,
- if $w_1 \neq w_2$, $w_2 \neq w_3$ but $w_3 = w_4$ then $\widehat{w}$ represents line 3,
- $\vdots$
- if $w_1 \neq w_2$, $\ldots$, $w_{|\rho|-2} \neq w_{|\rho|-1}$ but $w_{|\rho|-1} = w_{|\rho|}$ then $\widehat{w}$ represents line $|\rho| - 1$, and

- if $w_1 \neq w_2, \ldots, w_{|\rho|-1} \neq w_{|\rho|}$ then $\widehat{w}$ represents line $|\rho|$.

Let the the lines on $\overline{\overline{z}}$ denote an i-tuple, the line on $\overline{u}, \overline{v}$ denote j-tuples, the hat on $\widehat{w}$ denote a $|\rho|$-tuple, and the line on $\vec{x}, \vec{y}$ denote $(i + |\rho| + j)$-tuples.

Proposition 5 ([2])**.** *Suppose* $\rho \in \mathrm{NPS}(1)$ *is as in the definition, and that* $\psi(\vec{x}, \vec{y})$ *is a quantifier-free first order formula expressing the edge relation of* $\mathcal{G}^{\rho}_{\mathcal{A}}$*. The following are equivalent:*

- $\mathcal{A} \models \rho(\overline{\overline{z}})$
-
$$\begin{aligned}\mathcal{A} \models\ & \exists w_1, \ldots, w_{|\rho|}\ w_1 = w_2 \wedge \\ & \exists w'_1, \ldots, w'_{|\rho|}\ w'_1 \neq w'_2 \wedge \ldots \wedge w'_{|\rho|-1} \neq w'_{|\rho|} \wedge \\ & \qquad \exists \overline{u}, \overline{v}\ \ \mathrm{TC}[\lambda \vec{x}, \vec{y} \psi]((\overline{\overline{z}}, w_1, \ldots, w_{|\rho|}, \overline{u}), (\overline{\overline{z}}, w'_1, \ldots, w'_{|\rho|}, \overline{v}))\end{aligned}$$

Proof. Follows immediately from the definition of $\mathcal{G}^{\rho}_{\mathcal{A}}$, together with the existential semantics of $\mathrm{NPS}(1)$. Note that the bizarre constraints on the ws are simply our means of encoding the first and last lines. As can be seen, we are not too interested in what the input-output variables are at the start and end of the computation, *i.e.* $\overline{u}$ and $\overline{v}$, respectively. □

Recall that the class $\mathrm{NPS}_s(m)$ is as $\mathrm{NPS}(m)$, but with a built-in successor available. The following gives us an idea as to the power of NPS.

Theorem 6 ([2])**.** *For* $m \geq 1$, $\mathrm{TC}^m[\mathbf{FO}] = \mathrm{NPS}(m)$*, and, consequently,* $\mathrm{TC}^*[\mathbf{FO}] = \mathrm{NPS}$*. Furthermore, for* $m \geq 1$, $\mathrm{TC}^m_s[\mathbf{FO}] = \mathrm{NPS}_s(m) = \mathrm{TC}^*_s[\mathbf{FO}] = \mathrm{NPS}_s = \mathsf{NL}$.

Proof. The first part follows from the fact that there is a program scheme $\rho_{\mathrm{TC}} \in \mathrm{NPS}(1)$ that expresses the relation TC, combined with the previous proposition, by induction. The second part follows from the NL-completeness of the Transitive Closure problem. □

Remark. The class NPS_s appears to be devoid of any memory, and it may seem surprising, in that light, that $\mathrm{NPS}_s = \mathsf{NL}$. However, NPS has memory, in the form of the constant number of input-output variables. Moreover, this constant number of variables may collectively attain $n^{|V_{io}|}$ values, computing on a structure $\mathcal{A}$ of size n. This is of similar order to the number of different tape configurations on an NL-Turing Machine, which is $\log n.|Q|.|\Sigma|^{\log n}$, where Q is the set of states and Σ the alphabet. The NL-Turing Machine has constant alphabet and logarithmically-bounded number of tape squares, while the class NPS_s has linearly-sized alphabet and constant number of memory-variables.

2.2.2 Shorthands

We can build other useful instructions from those that we have, possibly requiring the introduction of additional new variables. Specifically:

- If $\overline{p}, \overline{q}$ are j-tuples of variables or constants, then consider $\overline{p} = \overline{q}$ to be shorthand for $p_1 = q_1 \wedge \ \dots \ \wedge p_j = q_j$.
- If $\overline{v}$ is a j-tuple of variables and $\overline{q}$ is a j-tuple of variables or constants, then consider $\overline{v} := \overline{q}$ to be shorthand for $v_1 := q_1 \ ; \ \dots \ ; \ v_j := q_j$.
- Consider LOOP FOREVER to be shorthand for:

```
WHILE $v_1 = v_1$ DO OD.
```

- Consider IF t THEN DO τ FI to be shorthand for:

```
GUESS $v_1, v_2$
WHILE $v_1 = v_2$ DO LOOP FOREVER OD
WHILE $v_1 \neq v_2 \wedge t$ DO $\tau$ ; $v_1 := v_2$ OD
```

Of course, it may come to pass that a computation entering an IF statement gets trapped in an endless loop. This may seem undesirable, but it does not affect us: owing to our existential semantics, we only require that *some* path leads through the conditional.

- Consider $v' :\neq v$ (where v, v' are distinct variables) to be shorthand for:

```
GUESS $v'$
IF $v = v'$ THEN DO LOOP FOREVER FI
```

Sometimes we will want the computation to evaluate the disjunction of a fixed collection of possibilities. It may not be possible to write these directly in quantifier-free tests in WHILE loops. In the following, the labels word1, ..., word$j$ act as local dummy 'variables'.

- Let word1, ..., word$j$ be words representing certain possibilities. Consider:

```
EITHER(word1, ..., word$j$)
IF word1 THEN DO $\tau_1$ FI
  ⋮
IF word$j$ THEN DO $\tau_j$ FI
```

  to be shorthand for:

```
GUESS $v_1, \ldots, v_j$
IF $v_1 = v_2$ THEN DO $\tau_1$ FI
IF $v_1 \neq v_2 \wedge v_2 = v_3$ THEN DO $\tau_2$ FI
  $\vdots$
IF $v_1 \neq v_2 \wedge \ldots \wedge v_{j-2} \neq v_{j-1} \wedge v_{j-1} = v_j$ THEN DO $\tau_{j-1}$ FI
IF $v_1 \neq v_2 \wedge \ldots \wedge v_{j-1} \neq v_j$ THEN DO $\tau_j$ FI
```

The EITHER construction allows us to choose between any finite number of possibilities. Note that, in the EITHER shorthand, we have no need for an 'Else' construction, since all possibilities for the antecedent are covered. In all use of shorthands we will require that the variables we introduce in the longhand do not appear elsewhere in the program schemes involving those shorthands, lest we lose their information. This may ultimately require the introduction of new variables to our program schemes. We only need a fixed number of new variables for this, and we will usually be sloppy, omitting these variables when writing out program schemes involving shorthand.

2.2.3 Shorthands on successor structures

In the presence of a successor relation *succ*, we will use the following shorthands:

- $v' := cyc.succ(v)$ to be shorthand for:

```
IF $v = max$ THEN DO $v' = min$ OD
IF $v \neq max$ THEN DO
  GUESS $v'$
  IF $v' \neq succ(v)$ THEN DO LOOP FOREVER FI FI
```

In contrast to *succ*, which is a partial function, *cyc.succ* is a total function. Moreover, it is a bijection.

- $v' := inv.cyc.succ(v)$ to be shorthand for:

```
GUESS $v'$
IF $v \neq cyc.succ(v')$ THEN DO LOOP FOREVER FI
```

- For variable j-tuples $\overline{v}, \overline{v}'$, consider $\overline{v}' := cyc.succ(\overline{v})$ to be shorthand for:

```
IF $v_j \neq max$ THEN DO
  $(v'_1, \ldots, v'_{j-1}) := (v_1, \ldots, v_{j-1})$
  $v'_j := cyc.succ(v_j)$ FI
```

```
IF $(v_j = max) \wedge (v_{j-1} \neq max)$ THEN DO
  $(v'_1, \ldots, v'_{j-2}) := (v_1, \ldots, v_{j-2})$
  $v'_{j-1} := cyc.succ(v_{j-1})$
  $v'_j := min$ FI
⋮
⋮
IF $(v_j = max) \wedge \ldots \wedge (v_1 = max)$ THEN DO
  $(v'_1, \ldots, v'_j) := (min, \ldots, min)$ FI
```

- For variable j-tuples $\overline{v}, \overline{v}'$, consider $\overline{v}' := inv.cyc.succ(\overline{v})$ to be shorthand for:

```
IF $v_j \neq min$ THEN DO
  $(v'_1, \ldots, v'_{j-1}) := (v_1, \ldots, v_{j-1})$
  $v'_j := inv.cyc.succ(v_j)$ FI
IF $(v_j = min) \wedge (v_{j-1} \neq min)$ THEN DO
  $(v'_1, \ldots, v'_{j-2}) := (v_1, \ldots, v_{j-2})$
  $v'_{j-1} := inv.cyc.succ(v_{j-1})$
  $v'_j := max$ FI
⋮
⋮
IF $(v_j = min) \wedge \ldots \wedge (v_1 = min)$ THEN DO
  $(v'_1, \ldots, v'_j) := (max, \ldots, max)$ FI
```

2.2.4 Adding a stack: introducing NPSS

We can increase the power of our program schemes by introducing certain types of memory. In [2], the authors considered adding a stack.

Definition (Syntax of NPSS[2])**.** The syntax of NPSS(1) is as that of NPS(1), with the addition of two new instructions:

- a push instruction 'PUSH v' , where

$$v \in V_{io} \cup V_f \cup \{c : c \text{ is a constant symbol of } \sigma\} \text{ , and}$$

- a pop instruction '$v :=$POP' , where $v \in V_{io}$.

Again, the program schemes of NPSS$(m+1)$ are those whose tests in while loops may include schemes from strictly lower strata as extensional relations.

For semantics, the push instruction should be viewed as pushing the value of the given variable (or constant) to the stack, and the pop instruction should be viewed as an assignment instruction removing the current top element of the stack. If the stack is empty, the pop instruction leaves its variable unchanged.

The following gives us an idea as to the power of NPSS.

Theorem 7 ([2]). *For* $m \geq 1$, $\mathrm{PS}^m[\mathbf{FO}] = \mathrm{NPSS}(m)$, *and* $\mathrm{PS}^*[\mathbf{FO}] = \mathrm{NPSS}$. *Furthermore,* $\mathrm{PS}^m_s[\mathbf{FO}] = \mathrm{NPSS}_s(m) = \mathrm{PS}^*_s[\mathbf{FO}] = \mathrm{NPSS}_s = \mathsf{P}$.

2.3 Turing Machines

Turing Machines compute on strings and not structures. In order that we can consider the Turing complexity of problems $\Omega \subseteq \mathrm{STRUC}(\sigma)$, we will need to have a standard encoding of structures over a signature σ. Let $\sigma = \langle R_1, \dots R_j, c_1, \dots c_{j'} \rangle$, where the arities of $R_1, \dots, R_j$ are $a_1, \dots, a_j$ respectively.

Over an ordered structure $\mathcal{A} \in \mathrm{STRUC}(\sigma)$ of size n, we will code each R_i by a string $bin(R_i)$ over $\{0,1\}$ of length n^{a_i}. For a number $0 \leq r \leq n^{a_i} - 1$, let $\overline{r}$ be the a_i-tuple that represents r in n-ary. Since $\mathcal{A}$ is ordered, this $\overline{r}$ represents an a_i-tuple $\overline{r_{\mathcal{A}}}$ over $\mathcal{A}$. Let the rth[2] entry of $bin(R_i)$ be a 1 if $\overline{r_A} \in R_i$, and a 0 otherwise. We code each c_i by a string $bin(c_i)$ of length n, as if c_i were a unary relation with one member. Finally, we consider $bin(\mathcal{A})$ to be the concatenation $bin(R_i) \dots bin(R_j) bin(c_1) \dots bin(c_{j'})$.

We consider Turing Machines to have a one-way infinite tape, finite state set Q and uniform alphabet $\Sigma = \{zero, one, blank\}$. The read/write head is initially over square 1. Given a (non-deterministic) Turing Machine T and a string $w \in \{0,1\}^*$, we write $T \downarrow w$, iff T enters the accept state, at some point in its computation, when it is given input w over squares 1 to $|w|$ with all other squares blank.

We say that a (non-deterministic) Turing Machine T accepts a problem $\Omega \subseteq \mathrm{STRUC}(\sigma)$ iff, for all structures $\mathcal{A} \in \mathrm{STRUC}(\sigma)$, and for all orderings of $\mathcal{A}$, we have:

$$T \downarrow bin(\mathcal{A}) \;\Leftrightarrow\; \mathcal{A} \in \Omega$$

[2]This should really be $r+1$, since otherwise we would be considering the first entry of $bin(\mathcal{R}_i)$ to be indexed by the number zero. This is an occupational hazard of variously considering the set $\mathbb{Z}_n$ to be $\{1, \dots, n\}$ or $\{0, \dots, n-1\}$. We largely use the former for the chapters on program schemes, and the latter for the chapters on constraint satisfaction.

Chapter 3

Adding a Priority Queue

We now consider the situation where we have a priority queue for memory. A priority queue allows us to send elements to memory tagged with a numerical weight. We are free to choose from a range of weights polynomially-bounded in the size of the structure on which we are computing, but we may only retrieve from the maximal (non-empty) weight. We will consider a variety of semantic variations, and will, therefore, be no more specific at this point as to the properties of the priority queue. However, we are in a very different situation from before, because now we deal with both elements of structures and numbers. Such is the power of the inclusion of numbers, that we will find ourselves dealing with Turing Machines and complexity classes directly, as opposed to Lindström logics that capture complexity classes only on ordered structures. We needed free variables in NPS and NPSS in order to build the stratification within those hierarchies. We do not need that variety of stratification here. Therefore, since we are only concerned with decision problems, we will have no need for free variables here, and we dispense with them for the sake of a simpler exposition.

Since we will deal in a range of queue weights that is polynomially-bounded in the size of the structure $\mathcal{A}$ on which we are computing, we will have interest in the numbers $1,\ldots,n$, where $n = ||\mathcal{A}||$. We allow ourselves the first and last of these, 1 and n, as constants that we may refer to by name.

Definition. For each $k \geq 0$, the program schemes of $\mathrm{NPSPQ}(k)$, over a signature σ, involve two finite sets of variables, a set V of element variables and a set N of numeric variables. A program scheme $\rho \in \mathrm{NPSPQ}(k)$ consists of a finite sequence of instructions, where each instruction, other than the first and last, is of one of the following forms:

- an assignment instruction of the form '$p := q$' , where $p \in V$ and $q \in V \cup \{c : c$ is a constant symbol of $\sigma\}$ <u>or</u> $p \in N$ and $q \in N \cup \{1, n\}$
- a guess instruction of the form 'GUESS v' , where $v \in V$

- an increase (numeric successor) instruction '$\textsc{Incr}\ m$' , where $m \in N$
- a push instruction '$\textsc{Push}\ v, m_{i1}, \ldots, m_{ik}$'[1] where, $v \in V$ and $m_{i1}, \ldots, m_{ik} \in N$
- a pop instruction '$v := \textsc{Pop}$' where $v \in V$.
- while loops of the form '$\textsc{While}\ t\ \textsc{Do}\ \tau\ \ \textsc{Od}$' , where t is quantifier-free $\mathbf{FO}(\sigma)$ with free variables among V <u>or</u> quantifier-free $\mathbf{FO}(\langle 1, n\rangle)$ whose free variables are among N, and where τ is a sequence of instructions of one of the forms listed.

The first instruction is $\textsc{Input}(V,N)$, and the last $\textsc{Output}(V,N)$. We further define NPSPQ to be $\cup_{k\in\omega}\text{NPSPQ}(k)$.

As hinted at before, the stratification here – dimension k of the queue – is quite different from the stratification we have seen thus far, which was based on nestings of negation. With a priority queue, we will have sufficient computational power [at what would have been the first level of that nesting] to not require stratification.

The assignment and guess instructions, and the while loops, behave as before. Observe that in each case there are two modes of use: one relates to elements, the other to numbers. We do not allow the guessing of numeric variables simply because it is unlikely to be useful. The instruction $\textsc{Incr}\ m$ increases the number m by one, under the convention that $\textsc{Incr}\ n$ is 1. This ensures that $\textsc{Incr}$ is a function, like *cyc.succ*, and in contrast to *succ*. The push instruction sends the element in question to the priority queue tagged with weight k-tuple $(m_{i1}, \ldots, m_{ik})$, *i.e.* the current value of those numeric variables. It is for this reason that k is considered the dimension of the queue. We will consider a number of alternative semantics for the pop instruction:

u The pop removes, deterministically, the last element to be sent to the queue at whatever is the maximal non-empty weight. This semantics leads to a potentially unbounded queue size, and hence will be referred to as semantics '*u*'.

b The pop removes, deterministically, the last element to be sent to the queue at whatever is the maximal non-empty weight, and then scrubs all other entries at that weight. This is equivalent to the condition that the queue has only one space at each weight, *i.e.* new pushes would overwrite. This semantics leads to a (polynomially-)bounded queue size, and hence will be referred to as semantics '*b*'.

[1] When $k = 0$ there are no m's.

$u+$ As with 'u', but the maximal weight is also returned. This requires pop syntax '$p, m_{i1}, \ldots, m_{ik} =:$ POP'. We refer to this as semantics '$u+$'.

$b+$ As with 'b', but the maximal weight is also returned. This also requires pop syntax '$p, m_{i1}, \ldots, m_{ik} =:$ POP'. We refer to this as semantics '$b+$'.

As before, the pop instruction leaves its variable unchanged if the queue is empty. Also as before, the INPUT instruction non-deterministically assigns elements of the structure to V. The numeric variables N are set initially to 1. Again, we consider an accepting computation of a program scheme ρ on a structure $\mathcal{A}$ to be any one that reaches OUTPUT, and we denote this $\mathcal{A} \models \rho$. We refer to each of the four alternative semantics above specifically by superscript, e.g. NPSPQ$^{b+}(k)$. Again, we will refer to the classes endowed with successor with the subscript s, *e.g.* NPSPQ$_s^{b+}(k)$.

We will use the following shorthands, specific to schemes of NPSPQ:

- Consider FOR $m = m'$ TO m''DO τ NEXT to be shorthand for:

```
$m := m'$
WHILE $m \neq m''$ DO
  $\tau$
  INCR $m$ OD
$\tau$.
```

- Consider DECR m to be shorthand for:

```
$m' := 1; m'' := m'$
INCR $m'$
WHILE $m' \neq m$ DO
  $m'' := m'$
  INCR $m'$ OD
$m := m''$.
```

- Consider FOR $m = m'$ DOWNTO m'' DO τ NEXT to be shorthand for:

```
$m := m'$
WHILE $m \neq m''$ DO
  $\tau$
  DECR $m$ OD
$\tau$.
```

Note that FOR loops are inclusive with respect to their limits.

- Consider $m := m' + m''$ to be shorthand for:

```
        $m := m'$
        FOR $m''' = 1$ TO $m''$ DO INCR $m$ OD.
```

- Consider $m := m' - m''$ to be shorthand for:

```
        $m := 1$
        DECR $m$; DECR $m$
        FOR $m''' = m'$ DOWNTO $m''$ DO INCR $m$ OD.
```

Henceforth, we will feel free to put arithmetic terms such as $m' - m''$ as limits in FOR loops.

Let $\overline{m}$ be a j-tuple $(m_1, \ldots, m_j)$ of numeric variables. Consider INCR $\overline{m}$ to be shorthand for:

```
    IF $(m_j = n) \wedge \ldots \wedge (m_2 = n)$ THEN DO INCR $m_1; \ldots;$ INCR $m_j$ OD
    IF $(m_j = n) \wedge \ldots \wedge (m_3 = n) \wedge (m_2 \neq n)$THEN DO INCR $m_2; \ldots;$ INCR $m_j$ OD
      $\vdots$
    IF $m_j \neq n$ THEN DO INCR $m_j$ OD
```

Let 1^j and n^j be the j-tuples of 1s and ns, respectively. We have that INCR $\overline{m}$ returns the lexicographic next number, subject to the convention that INCR $(n^j) = (1^j)$. Define DECR $\overline{m}$ analogously. Sums and differences of j-tuples $\overline{m}$ and $\overline{m}'$ are defined in the natural way. However, we will insist that we never attempt the sum or difference of a j-tuple and j'-tuple when $j \neq j'$.

Computing over a structure $\mathcal{A}$, with $||\mathcal{A}|| = n$, we find we have been granted basic modulo n arithmetic. This is ostensibly weaker than an ordering of the elements of $\mathcal{A}$, but it will ultimately allow us to build such an order.

Remark. For each j, 1^j represents the number 1, in modulo n^j arithmetic, and n^j represents the additive identity (zero). So, for example, INCR 1^j = $1^j + 1^j$ = $(1, \ldots, 1, 2)$, where 2 is INCR 1.

3.1 A single weight: $k = 0$

The bottom level in our apparent hierarchy merits brief attention. In the presence of a single weight, it is apparent that $b+$ (respectively, $u+$) is no stronger than b (respectively, u).

Lemma 8. $\mathrm{NPSPQ}_s^b(0) = \mathrm{NPS}_s(1) = \mathsf{NL}$

Proof. We already have the second equality; we prove the first.

($\mathrm{NPS}_s(1) \subseteq \mathrm{NPSPQ}_s^b(0)$). Trivially, we will have for any $\rho \in \mathrm{NPS}_s(1)$, that also $\rho \in \mathrm{NPSPQ}_s^b(0)$.

($\mathrm{NPSPQ}_s^b(0) \subseteq \mathrm{NPS}_s(1)$). The priority queue may hold only one element at any time, and, as such, behaves like an extra element variable. Furthermore, the ability to count in $\mathrm{NPSPQ}_s^b(0)$ may be simulated by the successor relation of $\mathrm{NPS}_s(1)$. Specifically, if $\rho \in \mathrm{NPSPQ}_s^b(0)$ involves $|V|$ element variables and $|N|$ numeric variables, then we construct $\rho' \in \mathrm{NPS}_s(1)$ with $|V|+|N|+3$ variables. Our simulation is made somewhat more complicated by our convention that popping from an empty queue leaves the variable unchanged: this is why we need the extra variables v', v'' (we use $v' = v''$ to signify that the queue is non-empty). We construct ρ' thus:

```
INPUT$(v_1,\ldots,v_{|V|},v_{|V|+1},\ldots,v_{|V|+|N|},v_{queue},v',v'')$
$v' :\neq v''$
$\tau_{sim}$
OUTPUT$(v_1,\ldots,v_{|V|},v_{|V|+1},\ldots,v_{|V|+|N|},v_{queue},v',v'')$
```

Where τ_{sim} is the body of ρ (*i.e.* with the input and output lines removed), with the following substitutions:

- Convert all instances of variables m_i to variables $v_{|V|+i}$.
- Convert all instances of the numeric constant 1 (respectively, n) to the element constant *min* (respectively, *max*).
- Convert all instances of 'INCR m_i' to: '$v_{|V|+i} := cyc.succ(v_{|V|+i})$'.
- Convert all instances of 'PUSH v_i' to: '$v_{queue} := v_i$; $v' := v''$'.
- Convert all instances of '$v_i :=$ POP' to:

```
IF $v' = v''$ THEN DO
  $v_i := v_{queue}$ ; $v' :\neq v''$ FI.
```

It should be clear that we have, for all structures $\mathcal{A}$, $\mathcal{A} \models \rho$ iff $\mathcal{A} \models \rho'$. □

Lemma 9. $\mathrm{NPSPQ}_s^u(0) = \mathrm{NPSS}_s(1) = \mathsf{P}$

Proof. We already have the second equality; we prove the first.

($\mathrm{NPSS}_s(1) \subseteq \mathrm{NPSPQ}_s^u(0)$). Trivially, any $\rho \in \mathrm{NPSS}_s(1)$ is such that $\rho \in \mathrm{NPSPQ}_s^u(0)$.

($\mathrm{NPSPQ}_s^u(0) \subseteq \mathrm{NPSS}_s(1)$). The priority queue's single weight here acts as a stack. We may use a similar, though simpler, reduction to that of the previous lemma: we no longer need the variables v', v'' in any capacity, and we leave instances of 'PUSH v_i' and '$v_i :=$ POP' in ρ unchanged in ρ'. □

Remark. The previous lemmas are somewhat misleading. We had provision for free variables in NPS and NPSS, but we have none in NPSPQ. The previous results, therefore, can only authoritatively refer to *sentences* of NPS_s and NPSS_s, *i.e.* those schemes without free variables. The only reason we omit free variables from NPSPQ is to simplify our exposition. The previous lemmas would hold in generality, if we were to allow free variables in NPSPQ.

When we are deprived of the successor relation, we find $\mathrm{NPSPQ}^b(0) \nsubseteq \mathrm{NPS}(1)$ (respectively, $\mathrm{NPSPQ}^u(0) \nsubseteq \mathrm{NPSS}(1)$), since the parity problem may be expressed in the former, through counting, but not in the latter. Of course, we will have the inclusions

- $\mathrm{NPSPQ}^b(0) \subseteq \mathrm{NPSPQ}^b_s(0) = \mathrm{NPS}_s(1)$ and
- $\mathrm{NPSPQ}^u(0) \subseteq \mathrm{NPSPQ}^u_s(0) = \mathrm{NPSS}_s(1)$.

We conjecture that these inclusions are proper, and in particular that $\mathrm{NPSPQ}^u(0)$ is contained within $\mathrm{LFP} + \mathrm{COUNT}[\mathbf{FO}]$, which is known to be strictly contained in P [31].

3.2 The Hamilton Path problem is in NPSPQ

We proceed by examining the power of the program schemes of NPSPQ and, in particular, one of their number ρ_{HP} with the ability to accept the NP-complete Hamilton Path problem. The Hamilton Path problem[2] HP is exactly the class of digraphs that have a directed path containing each vertex exactly once. The following is part of the program scheme $\rho_{HP} \in \mathrm{NPSPQ}^u(2)$ that non-deterministically builds an order on such a structure.

```
1. INPUT($v_1, v_2, m_1, m_2, m_3$)
2. FOR $m_1 = 1$ TO $n$ DO
3.   GUESS $v_1$
4.   FOR $m_2 = 1$ TO $n$ DO
5.     PUSH $v_1, m_2, m_1$ NEXT NEXT
```

Our method is simple enough: we produce n copies of n guessed vertices, each copy occupying weights $(i,1)$ to (i,n) for $1 \leq i \leq n$. These could be genuine orders, but only if we haven't picked some element twice. After line 5, the queue

[2]In contrast to $\mathrm{TC}, \mathrm{AR}, \mathrm{PS}$ etc., which we initially defined as global relations, we define HP as a decision problem.

looks like this (entry followed by weight):

$$\begin{array}{cl} \overbrace{x_n} & (n,n) \\ \vdots & \\ \underbrace{x_1} & (n,1) \\ \vdots & \\ \vdots & \\ \overbrace{x_n} & (1,n) \\ \vdots & \\ \underbrace{x_1} & (1,1) \end{array}$$

At line 6, we proceed by consuming $n-1$ copies of our n guessed elements to see if some element is repeated. First we look at the last copy and last element, x_n, stored at weight (n,n), then we look through $(n,n-1)$ to $(n,1)$, elements $x_{n-1},\ldots,x_1$, to see if it is repeated. Next, lines 11-13, we remove the unneeded (already checked) element x_n at weight $(n-1,n)$ and repeat the process for $(n-1,n-1)$ to $(n-1,1)$. If we do this $n-1$ times, finding no element guessed twice, then we know we do indeed have a genuine order left in weights $(1,1)$ to $(1,n)$. (If the first element had been repeated we would have already discovered that; we only need $n-1$ iterations here.)

```
 6. FOR $m_1 = 1$ TO $n-1$ DO
 7.   $v_2$ := POP
 8.   FOR $m_2 = m_1$ TO $n-1$ DO
 9.     $v_1$ := POP
10.     IF $v_1 = v_2$ THEN DO LOOP FOREVER FI NEXT
11.   IF $m_1 \neq n-1$ THEN DO
12.     FOR $m_3 = 1$ TO $m_1$ DO
13.       $v_2$ := POP FI NEXT NEXT NEXT
```

For any computation that gets past line 13 the queue will look like

$$\begin{array}{cl} \overbrace{x_n} & (1,n) \\ \vdots & \\ \underbrace{x_1} & (1,1) \end{array}$$

where we know that $x_1, \ldots, x_n$ is an ordering of the vertices. We will now search along it for a Hamilton path:

```
14. $v_1 := \textsc{Pop}$
15. $\textsc{For}\ m_1 = 1\ \textsc{To}\ n-1\ \textsc{Do}$
16.   $v_2 := v_1$
17.   $v_1 := \textsc{Pop}$
18.   $\textsc{If}\ \neg E(v_1, v_2)\ \textsc{Then Do Loop Forever Fi Next}$
19. $\textsc{Output}(v_1, v_2, m_1, m_2, m_3)$
```

It is because we can non-deterministically guess all orderings that there will be an accepting computation if, and only if, the structure has a Hamilton path. For all digraphs $\mathcal{G}$, we will have $\mathcal{G} \models \rho_{HP}$ iff $\mathcal{G} \in HP$.

The scheme ρ_{HP} computes in such a way that, on all inputs $\mathcal{G}$, it only uses any weight at most once. Consequently, ρ_{HP} also accepts the Hamilton Path problem under semantics b. Clearly, ρ_{HP} can undergo minor syntactic changes to produce a program scheme that accepts HP for semantics $u+$ and $b+$, too.

3.3 $\mathbf{NPSPQ}_s^b \subseteq \mathsf{NPspace}$

With the polynomially-bounded memory of NPSPQ_s^b, the following is almost immediate.

Proposition 10. $\mathrm{NPSPQ}_s^b \subseteq \mathsf{NPspace}$.

Sketch Proof. The proof is by simulation. For $\rho \in \mathrm{NPSPQ}_s^b$ we will construct a non-deterministic Turing Machine T, together with an exhibited bound l, such that, for all structures $\mathcal{A}$ (of size n), and all orderings of $\mathcal{A}$, the following are equivalent:

- $\mathcal{A} \models \rho$.
- $T \downarrow bin(\mathcal{A})$.
- $T \downarrow bin(\mathcal{A})$ with the read/write head never leaving the first n^l squares.

Note that equivalence of the last two guarantees that T is an $\mathsf{NPspace}$ machine since there exists some l' (dependent on the maximum relation arity of the signature σ) s.t. $|bin(\mathcal{A})| = O(n^{l'})$.

If $\rho \in \mathrm{NPSPQ}_s^b(k)$ and involves j program scheme variables (element or numeric) then we need to record at most $n^k + j$ items, corresponding to the entries on the priority queue and the assignments of the variables of ρ, at any point of the simulation. Each of these $n^k + j$ items may take at most n possible values, so each of these items may be written on T's tape in $\log(n)$ squares. We do not give full details of T's simulation, but note that the amount of tape space required to hold all these $n^k + j$ items is $O((n^k + j)\log(n))$. It follows that we may take $l := k + j + 1$. □

Corollary. $\mathrm{NPSPQ}^b \subseteq \mathsf{NPspace}$.

Proof. The inclusion $\mathrm{NPSPQ}^b \subseteq \mathrm{NPSPQ}^b_s$ is trivial. □

3.4 Expanding alphabets

At present there are precisely n distinct symbols that we can send to the queue, namely the elements of the structure on which we are computing. However, we can expand this alphabet by always pushing and popping j-tuples, instead of single variables. In this way we potentially increase our working alphabet to n^j symbols.

Let $\overline{v}$ be a j-tuple of variables. We work in semantics u, but our results apply to semantics $u+$, and also to NPSS. A similar method may be used for semantics b and $b+$, although at the cost of more weights. The method by which these results for semantics u transfer to semantics b will be explored later.

- Consider '$\textsc{Push}\ \overline{v},\overline{m}$' to be shorthand for '$\textsc{Push}\ v_j,\overline{m}\ ;\ldots;\ \textsc{Push}\ v_1,\overline{m}$'.
- Consider '$\overline{v} := \textsc{Pop}$' to be shorthand for '$v_1 := \textsc{Pop}\ ;\ldots;\ v_j := \textsc{Pop}$' (note the reverse order).

By these methods, we can push and pop tuples as if they were single elements. We can now set up special symbols by the use of a certain convention. Suppose we want i special symbols $M_1,\ldots,M_i$, then we can achieve this, in a rather sloppy manner, by always pushing and popping $(i+1)$-tuples $(v_1,\ldots,v_{i+1})$, using the convention:

- $(v_1,\ldots,v_{i+1})$ where $v_1 = v_2$ is the element v_1.
- $(v_1,\ldots,v_{i+1})$ where $v_1 \neq v_2 \wedge v_2 = v_3$ is the symbol M_1.
- $\vdots$
- $(v_1,\ldots,v_{i+1})$ where $v_1 \neq v_2 \wedge \ldots \wedge v_{i-1} \neq v_i \wedge v_i = v_{i+1}$ is the symbol M_{i-1}.
- $(v_1,\ldots,v_{i+1})$ where $v_1 \neq v_2 \wedge \ldots \wedge v_i \neq v_{i+1}$ is the symbol M_i.

Note that all $(i+1)$-tuples are defined. Henceforth we will assume a finite set of special symbols at our disposal.

Given a program scheme in which we are always pushing and popping j-tuples, we may drop the line over the variables, and use that line only when referring to some j'-tuple of 'variables' each of which is actually a j-tuple of real variables. This should not cause too much confusion. This will result in our having variables v that can hold values that do not represent actual elements of the universe on which we are computing. Such special characters will constitute the symbol set Λ.

3.5 Pushing and Popping Numbers in $\mathbf{NPSPQ}_s^b$ / $\mathbf{NPSPQ}_s^u$.

For program schemes of NPSPQ_s^u or NPSPQ_s^b, consider '$v := element(m)$' to be shorthand for

```
$v := min$
FOR $m' = 1$ TO $m-1$ DO
GUESS $v'$
IF $v' \neq succ(v)$ THEN DO LOOP FOREVER FI
  $v := v'$ NEXT
```

and '$m := position(v)$' to be shorthand for

```
$v' := min$
$m := 1$
WHILE $v' \neq v$ DO
  GUESS $v''$
  IF $v'' \neq succ(v')$ THEN DO LOOP FOREVER FI
  INCR $m$
  $v' := v''$ OD
```

The instruction $v := element(m)$ assigns to v the mth element of the universe, conversely the instruction $m := position(v)$ assigns to m the position of the element v in that order.

For j-tuples $\overline{m} = (m_1, \ldots, m_j)$ and $\overline{v} = (v_1, \ldots, v_j)$:

- Consider $\overline{v} := element(\overline{m})$ to be shorthand for $v_1 := element(m_1)$; $\ldots$; $v_j := element(m_j)$.

- Consider $\overline{m} := position(\overline{v})$ to be shorthand for $m_1 := position(v_1)$; $\ldots$; $m_j := position(v_j)$.

3.5.1 $\mathbf{NPSPQ}_s^{u+}(k) = \mathbf{NPSPQ}_s^{u}(k)$.

Lemma 11. $\mathrm{NPSPQ}_s^{u+}(k) \subseteq \mathrm{NPSPQ}_s^{u}(k)$.

Proof. The proof is by simulation. For all $\rho \in \mathrm{NPSPQ}_s^{u+}(k)$ we construct a $\rho' \in \mathrm{NPSPQ}_s^{u}(k)$ such that, for all structures $\mathcal{A}$, we have $\mathcal{A} \models \rho$ iff $\mathcal{A} \models \rho'$.

The program scheme ρ' will involve all the variables of ρ together with a new k-tuple of element variables $\overline{v_m}$. Where ρ pushes and pops single variables, ρ' will always push and pop $(k+1)$-tuples of variables (of which the trailing k-tuple contains the weight).

- Convert all instances of '$\textsc{Push}\ v, \overline{m}$' , in ρ, to the following in ρ':

```
$\overline{v_m} := element(\overline{m})$
$\textsc{Push}\ (v, \overline{v_m}), \overline{m}$
```

- Convert all instances of '$v, \overline{m} := \textsc{Pop}$' , in ρ, to to the following in ρ':

```
$(v, \overline{v_m}) := \textsc{Pop}$
$\overline{m} := position(\overline{v_m})$
```

□

Corollary. $\mathrm{NPSPQ}_s^{u+}(k) = \mathrm{NPSPQ}_s^{u}(k)$

Proof. The converse inclusion $\mathrm{NPSPQ}_s^{u}(k) \subseteq \mathrm{NPSPQ}_s^{u+}(k)$ is trivial. □

3.5.2 $\mathbf{NPSPQ}_s^{b+} = \mathbf{NPSPQ}_s^{b}$.

Lemma 12. $\mathrm{NPSPQ}_s^{b+}(k) \subseteq \mathrm{NPSPQ}_s^{b}(2k)$.

Proof. The proof is broadly similar to that of the previous lemma, but we will require more than single weights to store the $(k+1)$-tuples of that proof. For all $\rho \in \mathrm{NPSPQ}_s^{b+}(k)$ we construct a $\rho' \in \mathrm{NPSPQ}_s^{b}(k)$ such that, for all structures $\mathcal{A}$, we have $\mathcal{A} \models \rho$ iff $\mathcal{A} \models \rho'$.

The program scheme ρ' will involve all the variables of ρ together with a new k-tuple of element variables $\overline{v_m} = (v_m^1, \ldots, v_m^k)$.

- Convert all instances of '$\textsc{Push}\ v, \overline{m}$' , in ρ, to the following in ρ':

```
$\textsc{Push}\ v, (\overline{m}, 1^k)$
$\textsc{Push}\ v_m^k, (\overline{m}, 1^{k-1}, n)$
  $\vdots$
$\textsc{Push}\ v_m^1, (\overline{m}, n, 1^{k-1})$
```

- Convert all instances of '$v, \overline{m} :=$ POP' , in ρ, to to the following in ρ':

```
$(v, \overline{v_m}) :=$ POP
$\overline{m} := position(\overline{v_m})$
```

□

Corollary. $\text{NPSPQ}_s^{b+} = \text{NPSPQ}_s^b$

Proof. The inclusion $\text{NPSPQ}_s^b(k) \subseteq \text{NPSPQ}_s^{b+}(k)$ is trivial. □

3.6 $\textbf{NPSPQ}^b(k) \subseteq \textbf{NPSPQ}^u(k)$

Intuitively, semantics u appears at least as strong as semantics b. It is relatively straightforward to prove this.

Lemma 13. $\text{NPSPQ}^b(k) \subseteq \text{NPSPQ}^u(k)$.

Proof. The proof is by simulation. For all $\rho \in \text{NPSPQ}^b(k)$ we construct a $\rho' \in \text{NPSPQ}^u(k)$ such that, for all structures $\mathcal{A}$, we have $\mathcal{A} \models \rho$ iff $\mathcal{A} \models \rho'$.

The program scheme ρ' will involve all the variables of ρ together with two new k-tuples of numeric variables $\overline{m}', \overline{m}''$ and a new element variable v'. Assume that M is a special marker symbol not used by ρ. Build ρ' from ρ by adding the following lines to the beginning (after INPUT):

```
FOR $\overline{m}' = 1^k$ TO $n^k$ DO
  PUSH $M, \overline{m}'$ NEXT
```

This sends a copy of the marker M to every weight on the queue. Finally, convert all instances of '$v, \overline{m} :=$ POP' , in ρ, to the following in ρ'.

```
$\overline{m}' := n^k$
$v' :=$ POP
WHILE $\overline{m}' \neq 1^k \wedge v' = M$ DO
  DECR $\overline{m}'$
  $v' :=$ POP OD
IF $v' \neq M$ THEN DO
  $v := v'$
  WHILE $v' \neq M$ DO
    $v' :=$ POP OD FI
FOR $\overline{m}'' = \overline{m}'$ TO $n^k$ DO
  PUSH $M, \overline{m}''$ NEXT
```

The given subroutine counts, top-down, the number of empty weights in the queue of ρ – these contain just M in the queue of ρ'. When it finds something other than an M, it stores this in v then removes everything else at that weight, *i.e.* until it reaches another M. The situation where the queue of ρ is empty is dealt with by the conditional $v' \neq M$ in the sixth line. Finally, an M is returned to each of the weights of the queue of ρ' above and including the weight of the retrieved element. □

Corollary.

- $\mathrm{NPSPQ}^b_s(k) \subseteq \mathrm{NPSPQ}^u_s(k)$
- $\mathrm{NPSPQ}^{b+}(k) \subseteq \mathrm{NPSPQ}^{u+}(k)$
- $\mathrm{NPSPQ}^{b+}_s(k) \subseteq \mathrm{NPSPQ}^{u+}_s(k)$

Proof. Our proof is equally valid for these statements. □

Remark. It may seem that our method is unnecessarily complicated. In simulating semantics b with semantics u, when popping from the queue at a certain weight, why do we not simply then pop everything else off at that same weight (foregoing any need for the marker M)? This method would generate very simple proofs for the last two statements of the corollary, and a relatively easy proof of the first statement of the corollary. However, it would not easily be applied in the case of the statement of the theorem.

3.7 $\mathbf{NPSPQ}^u_s(k) \subseteq \mathbf{NPSPQ}^u(k+2)$.

Lemma 14. *$NPSPQ^u_s(k) \subseteq \mathrm{NPSPQ}^u(k+2)$.*

Proof. We prove the inclusion by simulating the successor relation. We take any scheme $\rho \in \mathrm{NPSPQ}^u_s(k)$, and construct a scheme $\rho' \in \mathrm{NPSPQ}^u(k+2)$ such that, for all structures $\mathcal{A}$, $\mathcal{A} \models \rho$ iff $\mathcal{A} \models \rho'$.

Assume, without loss of generality, that ρ involves element variables V and numeric variables N, and that $v_1, v_2 \notin V$ and $m_1, m_2, m_3 \notin N$. Given ρ we will construct ρ' by adding a special start-routine, a special end-routine, and amending push and pop instructions, as well as successor tests in while loops.

Let $V' = V \cup \{v_1, v_2\}$ and $N' = N \cup \{m_1, m_2, m_3\}$. Then ρ' will be:

```
$\textsc{Input}(V', N')$
$\tau_{start}; \tau_\rho; \tau_{end}$
$\textsc{Output}(V', N')$
```

We will now meet the sub-routines $\tau_{start}, \tau_\rho, \tau_{end}$, and explain why each one performs the function that will be claimed of it.

3.7.1 Start-routine: τ_{start}

We will add a sub-routine τ_{start} to the start, that builds an order over $\mathcal{A}$'s n elements. We will simply guess an order, as we did in ρ_{HP}, and we will put this putative order in the weights $(1^k, 1, 1)$ to $(1^k, 1, n)$. τ_{start} will first send a special marker symbol M to each of these weights via:

```
FOR $m_1 = 1$ TO $n$ DO
  PUSH $M, (1^k, 1, m_1)$ NEXT
```

We then add lines 2-13 of the scheme ρ_{HP} that solved the Hamilton Path problem, with the proviso that weight tuples (m_1, m_2) in ρ_{HP} become $(1^k, m_1, m_2)$ in τ_{start}. Any computation that gets through τ_{start} will leave the queue looking like:

$$\begin{array}{cc} \overbrace{x_n} & (1^k, 1, n) \\ M & (1^k, 1, n) \\ \vdots & \\ x_1 & (1^k, 1, 1) \\ \underbrace{M} & (1^k, 1, 1) \end{array}$$

where $x_1, \ldots, x_n$ is an ordering of the elements of $\mathcal{A}$.

3.7.2 Simulation of ρ: τ_ρ.

The main body of ρ', the sub-routine τ_ρ, is that bit that actually simulates ρ. It will use higher weights of the form $(n, n, \overline{m})$, where the line on $\overline{m}$ specifies a k-tuple. Before we get to the main simulation, we will push a special marker symbol M' to weight $(n, n, 1^k)$ to ensure that we never stray into the lower weights, in which the order is contained, during the simulation. Thus:

```
PUSH $M', (n, n, 1^k)$
```

For the actual simulation:

- Convert all instances of 'PUSH $v, \overline{m}$' to: 'PUSH $v, (n, n, \overline{m})$'.
- Convert all instances of '$v :=$ POP' to the following in ρ':

```
$v_1 := v$ ; $v :=$ POP
IF $v = M'$ THEN DO PUSH $v, (n, m, 1^k)$ FI
$v := v_1$
```

The simulation of pop is rather complicated because the original program scheme ρ must leave a pop unchanged when the queue is empty. But when the queue associated with ρ is empty, the queue associated with ρ' still contains entries in the lower weights beneath M'.

We may also have to evaluate quantifier-free successor queries of the form $v' = succ(v)$, that might appear in a test for a while loop, immediately before the test of that while loop. Let Φ be a propositional formula that involves the atom $v' = succ(v)$:

- Convert all instances of WHILE $\Phi(v' = succ(v))$ DO τ OD to:

  τ_{succ} ; WHILE $\Phi(m_2 = n)$ DO $\tau; \tau_{succ}$ OD.

Where τ_{succ} is the sub-routine:

```
$m_1 := 1; m_2 := 1$
WHILE $m_1 \neq n$ DO
  GUESS $v_1$
  PUSH $v_1, (1^k, 1, m_1)$
  IF $v_1 = v$ THEN DO
    INCR $m_1$
    GUESS $v_1$
    PUSH $v_1, (1^k, 1, m_1)$
    IF $v_1 = v'$ THEN DO $m_2 := n$ FI FI
  INCR $m_1$ OD
```

Observe that v' and v are free in the sub-routine. What is happening in the while loop in the added sub-routine is that we are guessing what we hope to be an order. If it is the order that we guessed at the start, then $m_2 = n$ iff $v' = succ(v)$. We will check later that all these guessed 'orders' are not only genuine orders, but also the same as the first. In this manner, each instance of $v' = succ(v)$ in Φ becomes a test of $m_2 = n$. There may be any constant number of tests of the form $v' = succ(v)$, involving different variable pairs: each one of these will cause its own copy of the τ_{succ} sub-routine to appear before, and in, the while loop.

3.7.3 End-routine: τ_{end}

Once the simulation of ρ is accomplished we will want access to the lower weights to verify that these 'orders' we have been guessing are uniformly the same. We will want to pop everything on the queue down to, and including, the marker M'. τ_{end} will therefore begin:

```
WHILE $v_1 \neq M'$ DO $v_1 :=$ POP OD.
```

At this point the queue will look like:

$$
\begin{array}{cl}
\overbrace{y_{n,s}} & \\
\vdots & \\
y_{n,1} & (1^k, 1, n) \\
x_n & \\
\underbrace{M} & \\
\vdots & \\
\vdots & \\
\overbrace{y_{1,s}} & \\
\vdots & \\
y_{1,1} & (1^k, 1, 1) \\
x_1 & \\
\underbrace{M} &
\end{array}
$$

where s is the number of times that we needed to check successor queries in while loops. We already know that $x_1, \ldots, x_n$ is an order of the elements – what we must now check is that:

- $x_1 = y_{1,1} = \ldots = y_{1,s}$
- $\vdots$
- $x_n = y_{n,1} = \ldots = y_{n,s}$

so τ_{end} concludes:

```
FOR $m_1 = 1$ TO $n$
  $v_1 :=$ POP
  WHILE $v_1 \neq M$ DO
    $v_2 := v_1$
    $v_1 :=$ POP
    IF $v_1 \neq v_2$ THEN DO LOOP FOREVER FI OD NEXT
```

$\square$

Corollary. $\mathrm{NPSPQ}^{u+} = \mathrm{NPSPQ}_s^{u+} = \mathrm{NPSPQ}_s^{u} = \mathrm{NPSPQ}^{u}$.

Proof. $\mathrm{NPSPQ}^{u+} \subseteq \mathrm{NPSPQ}_s^{u+}$ is trivial; $\mathrm{NPSPQ}_s^{u+} \subseteq \mathrm{NPSPQ}_s^{u}$ was proved in Lemma 11; $\mathrm{NPSPQ}_s^{u} \subseteq \mathrm{NPSPQ}^{u}$ was proved in the previous lemma; and, $\mathrm{NPSPQ}^{u} \subseteq \mathrm{NPSPQ}^{u+}$ is trivial. $\square$

Remark. Whilst we have $\mathrm{NPSPQ}_s^u \subseteq \mathrm{NPSPQ}^u$, there is no reason to think that $\mathrm{NPSPQ}_s^b \subseteq \mathrm{NPSPQ}^b$. NPSPQ^b can simulate NPSPQ^u up to a point, as we will see, but if there is a super-polynomial number of successor calls in a scheme of NPSPQ_s^b, then we can not use our method to simulate in NPSPQ^b.

3.8 NPspace $\subseteq$ $\mathbf{NPSPQ}_s^u$

Let $\Omega \subseteq \mathrm{STRUC}(\sigma)$ be some problem in NPspace. Then there exists a positive integer k and a non-deterministic Turing machine T such that, for all structures $\mathcal{A}$ (of size n), and all orderings of $\mathcal{A}$, the following are equivalent:

- $T \downarrow bin(\mathcal{A})$.
- $T \downarrow bin(\mathcal{A})$ with the read/write head never leaving the first n^k squares.
- $\mathcal{A} \in \Omega$.

Let Q be the set of states of T, including start state q_s and accept state q_a.

In addition to variables ranging over the elements of $\mathcal{A}$, we will want to enlarge our alphabet such that we also have:

- The set of pairs $\Pi = \{(zero, q), (one, q), (blank, q) : q \in Q\}$.
- The special symbols L, R, and U. These will track the movement of T's read/write head.
- The marker symbol M.

Since Q is fixed this will not be a problem.

Let $\Gamma \subseteq \Pi^2$ be such that $((y_1, q_1), (y_2, q_2)) \in \Gamma$ iff $y_1 = y_2$. Γ appears to be a rather unusual set, but we will need to verify such pairs on the queue in our given simulation.

Let $\Delta \subseteq \Pi^2 \cup (\Pi \times \{L, R\})$ be such that:

- $((y_1, q_1), (y_2, q_2)) \in \Delta$ if there is a transition rule of T from (y_1, q_1) to (y_2, q_2).
- $((y, q), R) \in \Delta$ if there is a transition rule that moves the read/write head Right from (y, q).
- $((y, q), L) \in \Delta$ if there is a transition rule that moves the read/write head Left from (y, q).

Δ is, therefore, our visualisation of T's transition rules.

Theorem 15. $\mathsf{NPspace} \subseteq \mathrm{NPSPQ}^u_s$.

Proof. We aim to prove this by simulation. We will construct a program scheme $\rho_\Omega \in \mathrm{NPSPQ}^u_s(k+1)$ such that $\mathcal{A} \models \rho_\Omega$ if, and only if, $\mathcal{A} \in \Omega$. The n^k weights of the form $(1, \overline{m})$ will mimic the n^k squares of the Turing machine T. The line on $\overline{m}$ will always refer to a k-tuple. ρ_Ω will be:

```
INPUT($\overline{v}, v_q, v, v', v'', \overline{m}, \overline{m_{r/w}}$)
$\tau_{bin(\mathcal{A})}$; $\tau_{sim}$; $\tau_{end}$
OUTPUT($\overline{v}, v_q, v, v', v'', \overline{m}, \overline{m_{r/w}}$)
```

We will now meet the sub-routines $\tau_{bin(\mathcal{A})}$, τ_{sim}, and τ_{end}, and explain why each one performs the function claimed of it.

3.8.1 Preparation: $\tau_{bin(\mathcal{A})}$

First we will write the marker symbol M to the weights $(1, 1^k)$ to $(1, n^k)$. Before we can simulate the computation of T we must write $bin(\mathcal{A})$ to the queue. We will do this by randomly writing *zero*, *one* or *blank*, together with the start state q_s, simultaneously to the weight ranges $(1, 1^k)$ to $(1, n^k)$ and $(n, 1^k)$ to (n, n^k). The n^k entries in the range $(1, 1^k)$ to $(1, n^k)$ will represent the n^k squares of the Turing tape at the start of computation. We let the variable $\overline{m}$ range over these tape squares in the following.

```
FOR $\overline{m} = (1^k)$ TO $(n^k)$ DO
  PUSH $M, (1, \overline{m})$
  EITHER(Zero, One, Blank)
  IF Zero THEN DO PUSH $(zero, q_s), (1, \overline{m})$; PUSH $(zero, q_s), (n, \overline{m})$ FI
  IF One THEN DO PUSH $(one, q_s), (1, \overline{m})$; PUSH $(one, q_s), (n, \overline{m})$ FI
  IF Blank THEN DO PUSH $(blank, q_s), (1, \overline{m})$; PUSH $(blank, q_s), (n, \overline{m})$ FI
NEXT
```

This will leave the queue looking like:

$$\begin{array}{cc}
\overbrace{(y^0_{n^k},q_s)} & (n,n^k)\\
\vdots & \\
\underbrace{(y^0_{1^k},q_s)} & (n,1^k)\\
\overbrace{(y^0_{n^k},q_s)} & (1,n^k)\\
M & (1,n^k)\\
\vdots & \\
(y^0_{1^k},q_s) & (1,1^k)\\
\underbrace{M} & (1,1^k)
\end{array}$$

where each $y \in \{zero, one, blank\}$. We will consume the top copy in weights $(n,1^k)$ to (n,n^k) to check that $y^0_{1^k},\ldots,y^0_{n^k}$ is an encoding $bin(\mathcal{A})$.

If σ is a signature with relations $R_1,R_2,\ldots,R_j$, of arities $a_1,a_2,\ldots,a_j$ then the coding of R_1 will take the weights $(n,1^k)$ to $(n,1^k+n^{a_1})$, the coding of R_2 will take the weights $(n,1^k+n^{a_1}+1)$ to $(n,1^k+n^{a_1}+n^{a_2}+1)$ etc. For $i \leq k$, note that the i'th power of n is represented by the k-ary vector $\overline{n^i}$ that has a 1 in positions $1 \leq i' \leq k-i$ and an n in positions $k-i < i' \leq k$.

We will explicitly give the method when $\sigma = \sigma_2 = \langle E^2 \rangle$, *i.e.* on graphs. In the sequence $y^0_{1^k},\ldots,y^0_{n^k}$, we must ensure that all apart from the first n^2 entries are $blank$. We must then ensure that the first n^2 entries code the edge relation of the graph. Recall that $\overline{n^2} = (1,\ldots,1,n,n)$[3].

```
FOR $\overline{m} = n^k$ DOWNTO $1^k + \overline{n^2} + 1^k$ DO
  $v :=$ POP
  IF $v \neq blank$ THEN DO LOOP FOREVER FI NEXT
FOR $\overline{m} = \overline{n^2}$ DOWNTO $1^k$ DO
  $v :=$ POP
  IF $v = blank$ THEN DO LOOP FOREVER FI
  $(v_1,\ldots,v_k) := element(m_1,\ldots,m_k)$
  IF $E(v_{k-1},v_k) \wedge v = zero$ THEN DO LOOP FOREVER FI
  IF $\neg E(v_{k-1},v_k) \wedge v = one$ THEN DO LOOP FOREVER FI NEXT
```

[3]We retain the line on the k-ary $\overline{n^2}$ to distinguish it from the binary $n^2 = (n,n)$.

Any computation that gets through that will leave the queue looking like:

$$\begin{array}{cc} \overbrace{y^0_{n^k}} & (1,n^k) \\ M & (1,n^k) \\ \vdots & \\ y^0_{1^k} & (1,1^k) \\ \underbrace{M} & (1,1^k) \end{array}$$

where $y^0_{1^k},\ldots,y^0_{n^k}$ is necessarily a copy of $bin(\mathcal{A})$ – which we consider to be T's tape on input. The superscript 0 refers to time 0.

3.8.2 Simulation: τ_{sim}

Throughout the simulation we will keep track of the position of T's read/write head in a numeric variable k-tuple $\overline{m_{r/w}}$, and the state will be remembered in a single variable v_q.

In simulating the ith step of T we first guess what type of move T will perform at that stage. We will verify later that these were valid choices in the computation. There are two basic cases: either moving the read/write head; or changing the entry at the read/write head's current position. We can not move left from position 1^k and if we move right from position n^k we may assume we do not have an accepting computation.

In the first case we write the symbol R or L to the weight $(1,\overline{m_{r/w}})$, depending on whether the read/write head is to move right or left. We amend the position of the read/write head as stored in $\overline{m_{r/w}}$, either adding one, or subtracting one. Afterwards we guess what will be the entries of T's tape at time $i+1$ and write them, together with the current state stored in v_q, to all the weights $(1,1^k)$ to $(1,n^k)$. We will want the tape-entries we have guessed to be exactly the same as at time i, written beneath them on the queue (except for the introduction somewhere of a symbol R or L). We will only verify that this is the case at the end of the simulation.

In the second case we write the symbol U to the weight $(1,\overline{m_{r/w}})$. We then choose a new state to go into, amending v_q accordingly. We then guess the entries of T's tape at time $i+1$ and write them, together with the new state stored in v_q, to all the weights $(1,1^k)$ to $(1,n^k)$. In this case we will want the tape-entries to be the same as at time i except possibly for the weight $(1,\overline{m_{r/w}})$, *i.e.*, for entries split by a U symbol. We will verify this at the end of the computation.

This simulation will continue until we guess that we go into the accept state q_a.

```
WHILE $v_q \neq q_a$ DO
  EITHER(Right, Left, Unmoved)
  IF Right THEN DO
    IF $\overline{m_{r/w}} = n^k$ THEN DO LOOP FOREVER FI
    PUSH $R, (1, \overline{m_{r/w}})$
    INCR $\overline{m_{r/w}}$ FI
  IF Left THEN DO
    IF $\overline{m_{r/w}} = 1^k$ THEN DO LOOP FOREVER FI
    PUSH $L, (1, \overline{m_{r/w}})$
    DECR $\overline{m_{r/w}}$ FI
  IF Unmoved THEN DO
    PUSH $U, (1, \overline{m_{r/w}})$
    GUESS $v_q$; IF $v_q \notin Q$ THEN DO LOOP FOREVER FI
  FOR $\overline{m} = (1^k)$ TO $(n^k)$ DO
    EITHER(Zero, One, Blank)
    IF Zero THEN DO PUSH $(zero, v_q), (1, \overline{m})$ FI
    IF One THEN DO PUSH $(one, v_q), (1, \overline{m})$ FI
    IF Blank THEN DO PUSH $(blank, v_q), (1, \overline{m})$ FI NEXT OD
```

3.8.3 Verification: τ_{end}

We now move into the verification, in which we check that we have effected a legitimate computation.

If t is the length of the simulated computation, then at this point each weight $(1, \overline{m})$ of the queue, representing the $\overline{m}$th square of the Turing tape, will have a stack on it looking something like[4]:

$$\begin{array}{c}
(y^{t}_{\overline{m}}, q_a) \\
(y^{t-1}_{\overline{m}}, q^{t-1}) \\
\vdots \\
(y^{g+1}_{\overline{m}}, q^{g+1}) \\
R \\
(y^{g}_{\overline{m}}, q^{g}) \\
\vdots \\
(y^{h+1}_{\overline{m}}, q^{h+1}) \\
U \\
(y^{h}_{\overline{m}}, q^{h}) \\
\vdots \\
(y^{1}_{\overline{m}}, q^{1}) \\
(y^{1}_{\overline{m}}, q_s) \\
M
\end{array}$$

Note that entries R, L, U, or, indeed, M may never be adjacent. We will read these entries off such that we can consider three adjacent at once. At any point the variables v, v', v'' will hold descending successive entries on the stack. τ_{end} will be:

```
FOR $\overline{m} = n^k$ DOWNTO $1^k$ DO
$v :=$ POP; $v' :=$ POP; $v'' := blank$
WHILE $v'' \neq M$ DO
  $v'' :=$ POP
  $\tau_{check}$
  $v := v'$; $v' := v''$ OD NEXT
```

In the case that $v, v' \notin \{L, R, U\}$, we will simply check that the tape-entry in v is the same as in v'. This is not quite the condition $v = v'$, since each such entry on the queue is a pair of tape entry and state, but it is the condition $(v, v') \in \Gamma$, *i.e.* the tape entries contained in v and v' are the same – even if the states are different.

In the case that $v \in \{L, R, U\}$ we do nothing.

Where v' is the symbol R we check that (y^{i+1}, q^{i+1}) and (y^{i}, q^{i}) (in v and v'', respectively) are the same. It is actually consequent on our simulation method that

[4]The qs with superscript should be considered as representative of some state in Q, just as the ys with superscript are representative of one of $\{zero, one, blank\}$. The qs with subscript, *e.g.* q_s, q_a, are actual states.

$q^{i+1} = q^i$. We must also check whether T has a transition rule in state q^i reading y^i to move right.

We do analogously when v' is L.

Where v' is the symbol U we check that T has a transition rule $((y^i, q^i), (y^{i+1}, q^{i+1}))$ (stored in (v, v'')) in Δ.

Thus τ_{check} will be:

```
IF $v, v' \notin \{L, R, U\}$ THEN DO
  IF $(v, v') \notin \Delta$ THEN DO LOOP FOREVER FI FI
IF $v' = R$ THEN DO
  IF $v \neq v''$ THEN DO LOOP FOREVER FI
  IF $(v, R) \notin \Delta$ THEN DO LOOP FOREVER FI
IF $v' = L$ THEN DO
  IF $v \neq v''$ THEN DO LOOP FOREVER FI FI
  IF $(v, L) \notin \Delta$ THEN DO LOOP FOREVER FI
IF $v' = U$ THEN DO
  IF $(v, v'') \notin \Delta$ THEN DO LOOP FOREVER FI FI
```

The result follows. □

Corollary. NPspace $\subseteq$ NPSPQ^u

Proof. Recall NPSPQ^u_s = NPSPQ^u. □

3.9 A polynomial-time restriction of NPSPQ^u.

Definition. A program scheme $\rho \in$ NPSPQ$^u(k)$ is said to be *polynomially step-bounded* if there exists a j such that, for all structures $\mathcal{A}$, ρ accepts $\mathcal{A}$ if, and only if, ρ accepts $\mathcal{A}$ within n^j steps. Let:

- NPSPQ$^u(k)_{poly}$:=

$$\{\rho \ : \ \rho \in \text{NPSPQ}^u(k) \quad \text{and } \rho \text{ is polynomially step-bounded }\}$$

- *NPSPQ$^u_{poly}$* := $\cup_{k \geq 0}$NPSPQ$^u(k)_{poly}$.

Proposition 16. NPSPQ$^u_{poly} \subseteq$ NPSPQ^b.

Proof. We prove this by simulation. The idea is that we can never attempt to use a weight more than once. Given some $\rho \in$ NPSPQ$^u(k)_{poly}$, and the j that is the polynomial power of its step bound, we will construct a $\rho' \in$ NPSPQ$^b(k + j)$, such that, for all structures $\mathcal{A}$, $\mathcal{A} \models \rho$ iff $\mathcal{A} \models \rho'$.

Let the line on $\overline{m}$ indicate a j-tuple. Assume $\overline{m}$ is a numeric variable tuple not involved in ρ.

Given ρ we construct ρ' by

- adding, after every line, except the last, the instructions

  ```
  INCR $\overline{m}$
  IF $\overline{m} = n^j$ THEN DO LOOP FOREVER FI.
  ```

  ($\overline{m}$ will act as a step-counter in ρ'), and

- converting all instances of PUSH $v, (m_1, \ldots, m_k)$ to PUSH $v, (m_1, \ldots, m_k, \overline{m})$.

□

Corollary. $\mathsf{NP} \subseteq \mathrm{NPSPQ}^b$

Proof. The simulation method we used in proving $\mathsf{NPspace} \subseteq \mathrm{NPSPQ}^u$ will also prove $\mathsf{NP} \subseteq \mathrm{NPSPQ}^u_{poly}$. The result follows from the previous lemma. □

Chapter 4

Adding Universal Quantification

4.1 Introducing $\mathrm{APS}(1)$

The schemes of NPS have existential quantification built-in through their guess instruction. $\mathrm{NPS}(1)$ is devoid of any notion of universal quantification. The higher strata, $\mathrm{NPS}(m)$, have some notion of universal quantification, through negation of existential quantification, but have no facility to combine both types of quantification within while-loop recursion. We consider the effect of explicitly adding universal quantification. We are, once more, without the stack.

Definition (Syntax of $\mathrm{APS}(1)$)**.** The syntax of $\mathrm{APS}(1)$ is as that of NPS(1), except the extant GUESS instruction is renamed $\exists$GUESS, and a new instruction $\forall$GUESS is added, with identical syntax.

The schemes of $\mathrm{NPS}(1)$ accepted a structure, expanded with values for the free variables, iff there existed some accepting computation, *i.e.* at each point the program went through an $\exists$GUESS v, there existed an assignment to v such that thenceforth the scheme made it to output. The schemes of $\mathrm{APS}(1)$ accept an expanded structure iff:

- at each point the program goes through a $\exists$GUESS v, there exists an assignment to v such that thenceforth the computation makes it to output, and
- at each point the program goes through a $\forall$GUESS v, we have that for all assignments to v the computation thenceforth makes it to output.

These instructions have an appealing semantic characterisation in terms of the configurations of a scheme $\rho \in \mathrm{APS}(1)$. When computing on a structure $\mathcal{A}$, we can construct an alternating graph $\mathcal{A}\mathcal{G}^{\rho}_{\mathcal{A}}$ just as we constructed $\mathcal{G}^{\rho}_{\mathcal{A}}$, but with the additional information that a configuration $(\bar{\bar{z}}, \widehat{w}, \overline{u})$ is *universal* iff $\widehat{w}$ represents line l, and the instruction on line $(l+1)$ is a $\forall$GUESS. Observe that, for the edge relation of $\mathcal{AG}^{\rho}_{\mathcal{A}}$, there is no difference between $\exists$GUESS and $\forall$GUESS, since in each case the configuration can move to any configuration that is identical excepting the guess for the pertinent variable. Let the the lines on $\overline{\overline{z}}$ denote an i-tuple, the line on $\overline{u},\overline{v}$ denote j-tuples, the hat on $\widehat{w}$ denote a $|\rho|$-tuple, and the line on $\vec{x},\vec{y}$ denote $(i+|\rho|+j)$-tuples.

Proposition 17. *Suppose $\rho \in \mathrm{APS}(1)$ has i free variables and j input-output variables, and that $\psi(\vec{x},\vec{y})$ is a quantifier-free first order formula expressing the edge relation of $\mathcal{AG}^{\rho}_{\mathcal{A}}$, then the following are equivalent:*

- $\mathcal{A} \models \rho(\overline{\overline{z}})$
-
$$\begin{aligned}\mathcal{A} \models \quad & \exists w_1,\ldots,w_{|\rho|}\ w_1 = w_2 \wedge \\ & \exists w'_1,\ldots,w'_{|\rho|}\ w'_1 \neq w'_2 \wedge \ldots \wedge w'_{|\rho|-1} \neq w'_{|\rho|} \wedge \\ & \qquad \exists \overline{u},\overline{v}\ \ \mathrm{AR}[\lambda\vec{x},\vec{y}\psi]((\overline{\overline{z}},w_1,\ldots,w_{|\rho|},\overline{u}),(\overline{\overline{z}},w'_1,\ldots,w'_{|\rho|},\overline{v}))\end{aligned}$$

Proof. Follows immediately from the semantics of $\mathrm{APS}(1)$ and the definition of $\mathcal{AG}^{\rho}_{\mathcal{A}}$. Recall that the bizarre constraints on the ws are our encoding of the first and last lines. $\square$

Just as acceptance in $\mathrm{NPS}(1)$ is a reachability (transitive closure) problem, so acceptance in $\mathrm{APS}(1)$ is an alternating reachability problem.

Corollary. $\mathrm{APS}(1) \subseteq \pm\mathrm{AR}^1[\mathbf{FO}]$.

Not only can the schemes of $\mathrm{APS}(1)$ be recast as formulae of $\mathrm{AR}^1[\mathbf{FO}]$, but a scheme of $\mathrm{APS}(1)$ can express the Alternating Reachability relation. In order to prove this, we will have use for another shorthand that is available to us in the presence of our new instruction $\forall$GUESS.

- Let word1, ..., word$j$ be words representing certain possibilities. Consider:

```
ALL(word1, ..., word$j$)
IF word1 THEN DO $\tau_1$ FI
  ⋮
IF word$j$ THEN DO $\tau_j$ FI
```

  to be shorthand for:

```
$\forall$GUESS $v_1, \ldots, v_j$
IF $v_1 = v_2$ THEN DO $\tau_1$ FI
IF $(v_1 \neq v_2) \wedge (v_2 = v_3)$ THEN DO $\tau_2$ FI
  $\vdots$
IF $(v_1 \neq v_2) \wedge \ldots \wedge (v_{j-2} \neq v_{j-1}) \wedge (v_{j-1} = v_j)$ THEN DO $\tau_{j-1}$ FI
IF $(v_1 \neq v_2) \wedge \ldots \wedge (v_{j-1} \neq v_j)$ THEN DO $\tau_j$ FI
```

ALL is the universal counterpoint to the existential EITHER. When a program scheme meets an EITHER instruction it will accept iff one of those choices leads to acceptance; when a program scheme meets the ALL instruction, it will accept iff all of the choices lead to acceptance.

Proposition 18. *There is a program scheme $\rho_{\mathrm{AR}}(u,v) \in \mathrm{APS}(1)$ with two free variables that expresses the relation* AR. *Formally, for all alternating graphs $\mathcal{A}$, and vertices $a, a' \in \mathcal{A}$:*

$$\mathcal{A} \models \rho_{\mathrm{AR}}(a,a') \text{ iff } \mathcal{A} \models \mathrm{AR}(a,a') \quad \textit{(there is an alt. path in } \mathcal{A} \textit{ from } a \textit{ to } a'\textit{)}$$

Proof. We will construct ρ_{AR}. First we note that the relation $\mathrm{AR}(u,v)$ may be written in LFP[**FO**] as $\mathrm{LFP}[\lambda Pxy\psi](u,v)$, where $\psi(P,x,y) :=$

$$(x=y) \vee (\exists s P(x,s) \wedge \neg U(s) \wedge E(s,y)) \vee (\exists s P(x,s) \wedge U(s) \wedge [\forall r E(s,r) \rightarrow P(r,y)])$$

This can be re-written as $\psi(P,x,y) \equiv$

$$(x=y) \vee \exists s P(x,s) \wedge ([\neg U(s) \wedge E(s,y)] \vee [U(s) \wedge \forall r(\neg E(s,r) \vee P(r,y))])$$

Note that the $\exists s$ quantifies everything to its right. We will denote the two conjuncts after that quantification as Left and Right. Thus:

- Left is $P(x,s)$, and
- Right is $([\neg U(s) \wedge E(s,y)] \vee [U(s) \wedge \forall r(\neg E(s,r) \vee P(r,y))])$.

Let $\tau_{\mathrm{AR}}(u,v,x,y,s,r)$ be the sub-routine involving free variables u,v:

```
$x := u; y := v$
WHILE $x \neq y$ DO
  $\exists$GUESS $s$
  ALL(Left,Right)
    IF Left THEN DO $y := s$ FI
    IF Right THEN DO
      IF $\neg U(s) \wedge E(s,y)$ THEN DO $x := y$ FI
      IF $\neg(\neg U(s) \wedge E(s,y)) \wedge \neg U(s)$ THEN DO LOOP FOREVER FI
```

```
$\forall$GUESS $r$
IF $\neg(\neg U(s) \wedge E(s,y)) \wedge \neg E(s,r)$ THEN DO $x := y$ FI
IF $\neg(\neg U(s) \wedge E(s,y)) \wedge E(s,r)$ THEN DO $x := r$ FI OD
```

We now set $\rho_{\mathrm{AR}}(u,v)$ to be:

- INPUT(x,y,r,s); τ_{AR}; OUTPUT(x,y,r,s)

ρ_{AR} evaluates whether (u,v) is in AR from the outside-in, hence x and y are initially set to u and v, respectively. Each path of the computation succeeds only when the variables x and y become equal. ρ_{AR} mimics exactly $\mathrm{LFP}[\lambda Pxy\psi](u,v)$: indeed if the rank of (u,v) in $\mathrm{LFP}[\lambda Pxy\psi](u,v)$ is j, *i.e.* $(u,v) \in \psi^j$ but $(u,v) \notin \psi^{j-1}$, then τ_{AR} will go through the while loop [a maximum of] j times. □

Proposition 19. $\mathrm{AR}^1[\mathbf{FO}] \subseteq \mathrm{APS}(1)$

Proof. Take any formula $\varphi \in \mathrm{AR}^1[\mathbf{FO}]$. Then φ is of the form $\exists u_1 \ldots u_{j'} \exists v_1 \ldots v_{j''}\ \mathrm{AR}[\lambda\bar{x},\bar{y}\psi](\bar{u},\bar{v})$, where ψ is quantifier-free. We construct $\rho_\varphi \in \mathrm{APS}(1)$ such that, for all structures $\mathcal{A}$, $\mathcal{A} \models \rho_\varphi$ iff $\mathcal{A} \models \varphi$.

Let ρ_φ be:

```
INPUT$(\bar{x},\bar{y},\bar{s},\bar{r},u_1,\ldots,u_{j'},v_1,\ldots,v_{j''})$
$\exists$GUESS $u_1,\ldots,u_{j'},v_1,\ldots,v_{j''}$
$\tau_{\mathrm{AR}}(\bar{u},\bar{v},\bar{x},\bar{y},\bar{s},\bar{r})$
OUTPUT$(\bar{x},\bar{y},\bar{s},\bar{r},u_1,\ldots,u_{j'},v_1,\ldots,v_{j''})$
```

□

Theorem 20. $\mathrm{APS}(1) = \mathrm{AR}^1[\mathbf{FO}]$

Proof. Follows from the previous two propositions. □

Corollary. $\mathrm{APS}(1) = \mathrm{AR}^1[\mathbf{FO}] = \mathrm{LFP}[\mathbf{FO}] = \mathrm{AR}^*[\mathbf{FO}]$.

Proof. $\mathrm{AR}^1[\mathbf{FO}] = \mathrm{LFP}[\mathbf{FO}] = \mathrm{AR}^*[\mathbf{FO}]$ is proved in [16]. □

4.2 Introducing APSS(1)

Here we consider the situation where we augment the schemes of $\mathrm{APS}(1)$ with a stack for memory. We will find that we can quantify over the stack in a way that was not possible with NPSS. Consequently, order will not be a problem, and we quickly establish that we subsume NPspace. We will have no need for free variables to generate stratification: as with NPSPQ, we dispense with them.

Definition (Syntax of $\mathrm{APSS}(1)$)**.** Notwithstanding the forbidding of free variables, the syntax is that of $\mathrm{APS}(1)$, with the PUSH and POP instructions of $\mathrm{NPSS}(1)$.

Definition. Suppose the program scheme $\rho \in \mathrm{APSS}(1)$ involves j variables. Then a configuration of ρ, computing on a structure $\mathcal{A}$, is a sequence $(\overline{v}, l, w)$ giving the values of the variables, the number of the line just executed, and the contents of the stack ($w \in |\mathcal{A}|^*$).

Each such program scheme ρ, computing on a structure $\mathcal{A}$, gives rise to an infinite alternating graph $\mathcal{AG}^{\rho}_{\mathcal{A}}$, defined as in the previous section. We say there is a *finite alternating path* between configurations c, c' in $\mathcal{AG}^{\rho}_{\mathcal{A}}$, if there is an $i \in \omega$ such that $(c, c') \in \psi^i(P, x, y)$, where ψ is as in the proof to Proposition 18.

For some structure $\mathcal{A}$, let $\Gamma_{\mathcal{A}}$ be some subset of $|\mathcal{A}|^*$. Then for some signature σ, let Γ be the global set $\{\Gamma_{\mathcal{A}} : \mathcal{A} \in \mathrm{STRUC}(\sigma)\}$.

Definition (Recognising Stacks)**.** We say that the global set Γ is *recognisable* iff there is a $\rho_\Gamma \in \mathrm{APSS}(1)$ such that for all $\mathcal{A}$ the following are equivalent:

- For all $\overline{v}$, there exists $\overline{v}'$ and $w' \in |\mathcal{A}|^*$ such that there is an alternating path in $\mathcal{AG}^{\rho_\Gamma}_{\mathcal{A}}$ from configuration $(\overline{v}, 1, w)$ to configuration $(\overline{v}', |\rho_\Gamma|, w')$.
- $w \in \Gamma_{\mathcal{A}}$.

Suppose that τ_Γ is the subroutine constructed from ρ_Γ be removing the input and output instructions. We are stating that, for each $\mathcal{A}$, the uniform subroutine τ_Γ, when confronted with a stack w, finishes (*i.e.* does not loop forever) if, and only if, $w \in \Gamma_{\mathcal{A}}$. This is independent of the values of all input-output variables going into τ_Γ. Essentially, τ_Γ recognises w.

If a subroutine, computing on $\mathcal{A}$, recognises a stack with contents $w \in |\mathcal{A}|^*$, without ever popping off more than the top entries w' ($|w'| < |w|$), then it follows that that subroutine will recognise any word in $\{w'\}.|\mathcal{A}|^*$. This suggests that sometimes recognition only relates to the top portion of the contents of a stack. This motivates the cartesian product in the following:

Lemma 21. *Let M_1 be a special marker symbol. The following is recognisable as the stack:*

$$\{\{M_1, x_1, \ldots, x_n, M_1 \ : \ x_1, \ldots, x_n \textit{ is an ordering of } |\mathcal{A}|\ \}.|\mathcal{A}|^* \ : \ \mathcal{A} \in \mathrm{STRUC}(\sigma)\}$$

where the '. $|\mathcal{A}|^*$' indicates cartesian product[1]

[1]There is potential for ambiguity here: By $|\mathcal{A}|^*$ we mean any possible finite string of *real* elements of $\mathcal{A}$, *i.e.* anything that could possibly be pushed to the stack, as opposed to just those symbols (encoded as tuples) that *represent* elements of $\mathcal{A}$.

Proof. Recall that Λ is our set of additional special symbols. Therefore $x \notin \Lambda$ iff x represents a *bona fide* element of the structure on which we are computing. We begin by defining τ_{order}:

```
$v_1 :=$ POP ; IF $v_1 \neq M_1$ THEN DO LOOP FOREVER FI
$\forall$GUESS $v_2$
IF $v_2 \notin \Lambda$ THEN DO
  $v_1 :=$ POP
  WHILE $v_1 \neq v_2$ DO
    $v_1 :=$ POP OD
  $v_1 :=$ POP
  IF $v_1 = v_2$ THEN DO LOOP FOREVER FI
  WHILE $v_1 \neq v_2 \wedge v_1 \neq M_1$ DO
    $v_1 :=$ POP
    IF $v_1 = v_2$ THEN DO LOOP FOREVER FI OD FI
```

The subroutine works by checking that every *bona fide* element appears once (lines 4-6), and only once, *i.e.* not again (lines 7-11), between two markers M_1.

The following scheme ρ accepts the global set of the lemma:

```
INPUT$(v_1, v_2)$
$\tau_{order}$
OUTPUT$(v_1, v_2)$.
```

□

We will also define the following subroutine τ_{push}, which pushes a random (non-deterministic) number of random (non-deterministic) choices (except M_1) to the stack:

```
$\exists$GUESS $v', v''$
WHILE $v' \neq v''$ DO
  $\exists$GUESS $w'$
  IF $w' = M_1$ THEN LOOP FOREVER FI
  PUSH $w'$
  $\exists$GUESS $v', v''$ OD
```

4.3 The ACCEPT instruction.

With the inclusion of a universal side to our semantics, we will have need of an ACCEPT instruction which, as its name suggests, tells the computation to immediately accept. Any program scheme ρ' that involves an ACCEPT instruction should be considered shorthand for a scheme $\rho \in \mathrm{APSS}(1)$ in the following way. Assume, w.l.o.g., that ρ' involves variable set V and that $v_1, v_2 \notin V$. Let τ' be ρ' without the input and output instructions. Construct τ from τ' via the substitutions:

- All tests t in while loops in τ' become tests $v_1 \neq v_2 \wedge t$ in τ.
- All instances of ACCEPT in τ' become $v_1 := v_2$ in τ.

Then ρ should be considered as:

```
INPUT$(V, v_1, v_2)$
$v_1 :\neq v_2$
WHILE $v_1 \neq v_2$ DO $\tau$ OD
OUTPUT$(V, v_1, v_2)$
```

Note that, once $v_1 = v_2$, the program can never get trapped in an infinite loop, and, consequently, must make it to output.

4.4 $\mathbf{APSS}_s(1) = \mathbf{APSS}(1)$

Lemma 22. $\mathrm{APSS}_s(1) \subseteq \mathrm{APSS}(1)$.

Proof. We prove the inclusion by simulating the successor relation. We take any scheme $\rho \in \mathrm{APSS}_s(1)$, and construct a scheme $\rho' \in \mathrm{APSS}(1)$ such that, for all structures $\mathcal{A}$, $\mathcal{A} \models \rho$ iff $\mathcal{A} \models \rho'$.

Assume, without loss of generality, that ρ involves element variables V, with $v_1, v_2, v_3 \notin V$, and does not use the marker symbol M_1. Given ρ, we will construct ρ' by adding a special start-routine, and amending pop instructions as well as successor tests in while loops.

Let $V' = V \cup \{v_1, v_2, v_3\}$. Then ρ' will be:

```
INPUT$(V')$
PUSH $M_1$
$\tau_{push}$
PUSH $M_1$
ALL(CheckOrder, Continue)
IF CheckOrder THEN DO $\tau_{order}$; ACCEPT FI
IF Continue THEN DO FI
$\tau_\rho$
OUTPUT$(V')$
```

where τ_ρ is the, as yet undefined, subroutine that actually mimics ρ. Observe how we are using the ALL instruction to use the stack twice, once for verification of the order, and again for whatever we want to do in the rest of the computation. The stack is no longer readable only once, as it was with NPSS. Note that the ALL choice 'Continue' is a dummy, in that any computation that follows that path will continue through the rest of the program.

We will now meet τ_ρ.

4.4.1 Simulation of ρ; τ_ρ

Any computation that gets to this sub-routine will have M_1 as the top element of the stack. Assuming M_1 is not a symbol of ρ, we can use it to ensure that we never stray into the bottom part of the stack, where the putative order is held, during our simulation of ρ. In constructing τ_ρ, we first remove the input and output lines (of ρ). Next we,

- convert all instances of '$v :=$ POP' to:

```
$v_1 := v$
$v :=$ POP
IF $v = M_1$ THEN DO PUSH $v$ ; $v := v_1$ FI
```

We may also have to evaluate quantifier-free successor queries of the form $v' = succ(v)$, that might appear in a test for a while loop, immediately before the test of that while loop. Let Φ be a propositional formula:

- Convert all instances of 'WHILE $\Phi(v' = succ(v))$ DO τ OD' to:

```
$\tau_{succ}$
WHILE $\Phi(v_1 = v_2)$ DO $\tau; \tau_{succ}$ OD
```

Where τ_{succ} is:

```
ALL(CheckSuccv, Continue)
IF CheckSuccv THEN DO
  WHILE $v_1 \neq M_1$ DO $v_1 :=$ POP OD
  WHILE $v_1 \neq v$ DO $v_1 :=$ POP OD
  $v_2 :=$ POP
  IF $v_2 = v'$ THEN DO ACCEPT FI
  LOOP FOREVER FI
IF Continue THEN DO FI
```

Observe that v' and v are free in the sub-routine.

The subroutine works by splitting the computation, both checking that $v' = succ(v)$ (consuming the stack in the process) and continuing the computation with the stack intact. □

Corollary. $\mathrm{APSS}_s(1) = \mathrm{APSS}(1)$

Proof. The converse $\mathrm{APSS}(1) \subseteq \mathrm{APSS}_s(1)$ is trivial. □

4.5 NPspace $\subseteq$ $\mathbf{APSS}_s(1)$

We will ultimately prove this by simulation of a non-deterministic Turing Machine that uses no more than n^k tape squares (for some k), on input $bin(\mathcal{A})$, where $||\mathcal{A}|| = n$. First, we will need some technical lemmas, which are stated for the case when the signature is σ_2, *i.e.* for graphs. Similar lemmas may be obtained for other signatures. Let m be such that $1 \leq m \leq n^k$, then we identify $\overline{m}$ with the lexicographic mth variable k-tuple $\overline{v}$, with respect to the built-in successor.

Lemma 23 (Recognise $bin(\mathcal{G})$). *Let M_1 and M_2 be special marker symbols. The following is recognisable as the stack (the brackets are synthetic, and appear, as the commas, purely for clarity):*

$$\{\ \{(\overline{1},\alpha_{\overline{1}},M_2),(\overline{2},\alpha_{\overline{2}},M_2),\ldots,\ (\overline{n^k},\alpha_{\overline{n^k}},M_2),M_1\}.|\mathcal{G}|^* \\ :\mathcal{G}\in \mathrm{STRUC}_s(\sigma_2),\ \alpha_{\overline{1}}\ldots\alpha_{\overline{n^2}} = bin(\mathcal{G}),\ \alpha_{\overline{n^2+1}},\ldots,\alpha_{\overline{n^k}} = blank\}$$

Proof. Letting $\overline{v} = (v_1,\ldots,v_k)$, we define τ_{bin}:

```
$(\overline{v},v',v'') :=$ POP
IF $\overline{v} \neq \overline{1} \vee v'' \neq M_2$ THEN DO LOOP FOREVER FI
$\overline{w} := \overline{v}$
WHILE $\overline{v} \neq \overline{n^k}$ DO
  $(\overline{v},v',v'') :=$ POP
  IF $\overline{v} \neq succ(\overline{w})$ THEN DO LOOP FOREVER FI
  IF $v'' \neq M_2$ THEN DO LOOP FOREVER FI
  IF $\overline{v} > \overline{n^2} \wedge v' \neq blank$ THEN DO LOOP FOREVER FI
  IF $E(v_{k-1},v_k) \wedge v' = zero$ THEN DO LOOP FOREVER FI
  IF $\neg E(v_{k-1},v_k) \wedge v' = one$ THEN DO LOOP FOREVER FI
  $\overline{w} := \overline{v}$ OD
$v'' :=$ POP; IF $v'' \neq M_1$ THEN DO LOOP FOREVER FI
```

Then ρ, as INPUT$(\overline{v},v',v'',\overline{w})$;$\tau_{bin}$;OUTPUT$(\overline{v},v',v'',\overline{w})$, recognises the global set of the lemma. □

Lemma 24 (Read/Write Head Right)**.** *Let M_1 and M_2 be special marker symbols. The following is recognisable as the stack (the brackets are synthetic, and appear, as the commas, purely for clarity):*

$$\begin{aligned}&\{\ \{(\overline{m},\alpha_{\overline{m}},M_2),\ldots,(\overline{n^k},\alpha_{\overline{n^k}},M_2),(\overline{1},\alpha_{\overline{1}},M_2),\ldots,(\overline{m-1},\alpha_{\overline{m-1}},M_2),M_1,\\&(\overline{m+1},\alpha_{\overline{m+1}},M_2),\ldots,(\overline{n^k},\alpha_{\overline{n^k}},M_2),(\overline{1},\alpha_{\overline{1}},M_2),\ldots,(\overline{m},\alpha_{\overline{m}},M_2),M_1\}.|\mathcal{G}|^*\\&\qquad\qquad : \mathcal{G}\in \mathrm{STRUC}_s(\sigma_2),\ \alpha_{\overline{1}},\ldots,\alpha_{\overline{n^k}}\in\Sigma\}\end{aligned}$$

Proof. We define $\tau_{r/w-right}$ in two parts. One part will check that the numbers $\overline{m}$ to $\overline{m-1}$, and $\overline{m+1}$ to $\overline{m}$ behave correctly; that the markers are placed properly; and that the αs are in Σ. The other part will check that the αs match in the two tape lists, *i.e.* each $\alpha_{\overline{j}}$ that appears before the first M_1 is equal to the $\alpha_{\overline{j}}$ that appears between the first and second M_1s. In the following, the variables $\overline{u_{top}}$ will hold the number of the first entry of the first tape ($\overline{m}$), and $\overline{u_{bottom}}$ will hold the number of the last entry of that tape ($\overline{m-1}$).

Define $\tau_{checkform}$ to be:

```
$(\overline{v},v',v'') :=$ POP
IF $(v_1\in\Lambda)\vee\ldots\vee(v_k\in\Lambda)\vee(v'\notin\Sigma)\vee(v''\neq M_2)$ THEN DO
  LOOP FOREVER FI
$\overline{u_{top}} := \overline{v}$
$\overline{u_{bottom}} := inv.cyc.succ(\overline{u_{top}})$
$(\overline{w},w',w'') :=$ POP
WHILE $\overline{v}\neq\overline{u_{bottom}}$ DO
  $(\overline{v},v',v'') :=$ POP
  IF $\overline{w}\neq cyc.succ(\overline{v})$ THEN DO LOOP FOREVER FI.
  IF $v'\notin\Sigma\vee v''\neq M_2$ THEN DO LOOP FOREVER FI
  $\overline{w} := \overline{v}$ OD
$v'' :=$ POP; IF $v''\neq M_1$ THEN DO LOOP FOREVER FI

$(\overline{v},v',v'') :=$ POP
IF $(v_1\in\Lambda)\vee\ldots\vee(v_k\in\Lambda)\vee(v'\notin\Sigma)\vee(v''\neq M_2)$ THEN DO
  LOOP FOREVER FI
IF $\overline{v}\neq cyc.succ(\overline{u_{top}})$ THEN DO LOOP FOREVER FI
WHILE $\overline{v}\neq\overline{u_{top}}$ DO
  $(\overline{v},v',v'') :=$ POP
  IF $\overline{w}\neq cyc.succ(\overline{v})$ THEN DO LOOP FOREVER FI.
  IF $v'\notin\Sigma\vee v''\neq M_2$ THEN DO LOOP FOREVER FI
  $\overline{w} := \overline{v}$ OD
$v'' :=$ POP; IF $v''\neq M_1$ THEN DO LOOP FOREVER FI
```

The first half of the sub-routine (12 lines) checks the form of the stack up to, and including, the first M_1. The second half (last 9 lines) does the same up to, and including, the second M_1.

Define $\tau_{checkcontent}$ to be:

```
$\forall$GUESS $\overline{w}$
IF $w_1 \in \Lambda \vee \ldots \vee w_k \in \Lambda$ THEN DO ACCEPT FI
$(\overline{v}, v', v'') :=$ POP
$\overline{u_{top}} := \overline{v}$
$\overline{u_{bottom}} := inv.cyc.succ(\overline{u_{top}})$
IF $\overline{v} = \overline{w}$ THEN DO $v_{first\alpha} := v'$ FI
WHILE $\overline{v} \neq \overline{w}$ DO
  $(\overline{v}, v', v'') :=$ POP
  $v_{first\alpha} := v'$ OD

WHILE $\overline{v} \neq \overline{u_{bottom}}$ DO
  $(\overline{v}, v', v'') :=$ POP OD
$v'' :=$ POP

$(\overline{v}, v', v'') :=$ POP
IF $\overline{v} = \overline{w}$ THEN DO $v_{second\alpha} := v'$ FI
WHILE $\overline{v} \neq \overline{w}$ DO
  $(\overline{v}, v', v'') :=$ POP
  $v_{second\alpha} := v'$ OD

IF $v_{first\alpha} \neq v_{second\alpha}$ THEN DO LOOP FOREVER FI
```

The three lines culminating in the middle '$v'' :=$ POP' remove down to, and including, the first M_1.

We now give $\tau_{r/w-right}$:

```
ALL(CheckForm,CheckContent)
IF CheckForm THEN DO $\tau_{checkform}$ FI
IF CheckContent THEN DO $\tau_{checkcontent}$ FI
```

Now, ρ, as:

```
INPUT$(\overline{v}, v', v'', \overline{w}, w', w'', v_{first\alpha}, v_{second\alpha})$
$\tau_{r/w-right}$
OUTPUT$(\overline{v}, v', v'', \overline{w}, w', w'', v_{first\alpha}, v_{second\alpha})$
```

recognises the global set of the lemma. □

Lemma 25 (Read/Write Head Left)**.** *Let M_1 and M_2 be special marker symbols. The following is recognisable as the stack (the brackets are synthetic, and appear, as the commas, purely for clarity):*

$$\{\ \{(\overline{m},\alpha_{\overline{m}},M_2),\ldots,(\overline{n^k},\alpha_{\overline{n^k}},M_2),(\overline{1},\alpha_{\overline{1}},M_2),\ldots,(\overline{m-1},\alpha_{\overline{m-1}},M_2),M_1,$$
$$(\overline{m-1},\alpha_{\overline{m-1}},M_2),\ldots,(\overline{n^k},\alpha_{\overline{n^k}},M_2),(\overline{1},\alpha_{\overline{1}},M_2),\ldots,(\overline{m-2},\alpha_{\overline{m-2}},M_2),M_1\}.|\mathcal{G}|^*$$
$$:\mathcal{G}\in \mathrm{STRUC}_s(\sigma_2),\ \alpha_{\overline{1}},\ldots,\alpha_{\overline{n^k}}\in\Sigma\}$$

Proof. We construct $\tau_{r/w-left}$ in a similar manner to $\tau_{r/w-right}$. □

Proposition 26. $\mathsf{NPspace} \subseteq \mathrm{APSS}_s(1)$

Proof. We aim to prove this by simulation. As before, and w.l.o.g., we assume that $\Omega \in \mathsf{NPspace}$ is a graph problem. Similar lemmas to those previous may be obtained for other signatures. Suppose $\Omega \in \mathsf{NPspace}$ is accepted by the Turing Machine T, with space bound of n^k on input $bin(\mathcal{A})$, where $||\mathcal{A}|| = n$. We construct ρ_Ω such that, for all graphs $\mathcal{G}$, and for all orderings of $\mathcal{G}$, $bin(\mathcal{G}) \in \Omega$ iff $\mathcal{G} \models \rho_\Omega$.

We will consider our alphabet expanded to include the $|Q|+|\Sigma|$ symbols representing T's states and alphabet; we also assume the additional marker symbols M_1 and M_2. We will store T's state in one variable v_q: let q_s and q_a be the distinguished start and accept states. Once again Δ is how we envisage T's transition rules (*cf.* section 3.8). ρ_Ω will be:

```
INPUT($\overline{v}, v', v'', \overline{w}, w', w'', v_{first\alpha}, v_{second\alpha}, v_q, v'_q$)
$v_q := q_s$
PUSH $M_1$
$\tau_{push}$
PUSH $M_1$
ALL(CheckBin,Continue)
IF CheckBin THEN DO $\tau_{bin}$ ; ACCEPT FI
IF Continue THEN DO FI
WHILE $v_q \neq q_a$ DO
  $(\overline{v}, v', v'')$ := POP; PUSH $(\overline{v}, v', v'')$
  EITHER(Right,Left,Unmoved)
  IF Right THEN DO
    IF $(v_q, v', R) \notin \Delta$ THEN DO LOOP FOREVER FI
    PUSH $M_1$; $\tau_{push}$; PUSH $M_1$
    ALL(Verify,Continue2)
    IF Verify THEN DO $\tau_{r/w-right}$ ; ACCEPT FI
    IF Continue2 THEN DO FI FI
  IF Left THEN DO
```

```
    IF $(v_q, v', L) \notin \Delta$ THEN DO LOOP FOREVER FI
    PUSH $M_1$; $\tau_{push}$; PUSH $M_1$
    ALL(Verify,Continue2)
    IF Verify THEN DO $\tau_{r/w-left}$ ; ACCEPT FI
    IF Continue2 THEN DO FI FI
  IF Unmoved THEN DO
    $(\overline{v}, v', v'') :=$ POP
    $\exists$GUESS $w', v'_q$
    IF$(v_q, v', v'_q, w') \notin \Delta$ THEN DO LOOP FOREVER FI
    $v_q := v'_q$ ; PUSH$(\overline{v}, w', v'')$ FI
  OD
OUTPUT$(\overline{v}, v', v'', \overline{w}, w', w'', v_{first\alpha}, v_{second\alpha}, v_q, v'_q)$
```

□

4.6 Summary

Below, we summarise the results of this chapter, and the previous.

$$\begin{array}{ccccccc}
\mathrm{NPSPQ}^b(0) & \subseteq & \mathrm{NPSPQ}^b_s(0) & = & \mathrm{NPS}_s(1) & = & \mathsf{NL} \\
\mathrm{NPSPQ}^u(0) & \subseteq & \mathrm{NPSPQ}^u_s(0) & = & \mathrm{NPSS}_s(1) & = & \mathsf{P} \\
\\
 & & \mathrm{NPSPQ}^b(k) & \subseteq & \mathrm{NPSPQ}^u(k) & & \\
\\
 & & \mathrm{NPSPQ}^b_s(k) & \subseteq & \mathrm{NPSPQ}^{b+}_s(k) & & \\
 & & \mathrm{NPSPQ}^{b+}_s(k) & \subseteq & \mathrm{NPSPQ}^b_s(2k) & & \\
 & & \mathrm{NPSPQ}^{b+}_s & = & \mathrm{NPSPQ}^b_s & & \\
 & & \mathrm{NPSPQ}^{u+}_s(k) & = & \mathrm{NPSPQ}^u_s(k) & & \\
 & & \mathrm{NPSPQ}^{u+}_s & = & \mathrm{NPSPQ}^u_s & & \\
\\
\mathrm{NPSPQ}^{u+} & = & \mathrm{NPSPQ}^{u+}_s & = & \mathrm{NPSPQ}^u_s & = & \mathrm{NPSPQ}^u \\
\\
\mathsf{NPspace} & \subseteq & \mathrm{NPSPQ}^u & & & & \\
\mathsf{NP} & \subseteq & \mathrm{NPSPQ}^u_{poly} & \subseteq & \mathrm{NPSPQ}^b & \subseteq & \mathsf{NPspace} \\
\\
\mathrm{LFP} & = & \mathrm{APS}(1) & & & & \\
\mathsf{P} & = & \mathrm{APS}_s(1) & & & & \\
\mathsf{NPspace} & \subseteq & \mathrm{APSS}_s(1) & = & \mathrm{APSS}(1) & &
\end{array}$$

Figure 4.1: Summary of Results

Chapter 5

Classes of Structure on which $\mathsf{P} = \pm\mathbf{PS}^k[\mathbf{FO}]$

In [42], various classes of structure $\mathbf{C}$ were studied, on which, for some k, $\mathrm{NPSS}(k) = \mathrm{PS}^k[\mathbf{FO}]$ captures exactly P. The method used in the proofs involved building a canonical order in $\mathrm{NPSS}(k-1)$, whereupon, since $\mathrm{NPSS}(k) = \mathsf{P}$ on ordered structures, the result followed. The following is a consequence of that work:

Proposition 27 ([42]). *Let* $\mathbf{C}$ *be any class of structures, and let* $\overline{x},\overline{y},\overline{z}$ *be variable* j*-tuples. Suppose there are formulae* $R(w_1,\ldots,w_m,\overline{x},\overline{y},\overline{z}) \in \pm\mathrm{PS}^k[\mathbf{FO}]$ *and* $\psi(w_1,\ldots,w_m) \in \mathrm{PS}^l[\mathbf{FO}]$ *such that, for all* $\mathcal{A} \in \mathbf{C}$*:*

- *R is commutative in $\overline{x}$ and $\overline{y}$, and deterministic in $\overline{z}$, i.e.,*

$$\mathcal{A} \models \forall w_1,\ldots,w_m \overline{x}\,\overline{y}\,\overline{z}\, R(w_1,\ldots,w_m,\overline{x},\overline{y},\overline{z}) \leftrightarrow R(w_1,\ldots,w_m,\overline{y},\overline{x},\overline{z})$$
$$\mathcal{A} \models \forall w_1,\ldots,w_m \overline{x}\,\overline{y}\, \exists\overline{z}\, R(w_1,\ldots,w_m,\overline{x},\overline{y},\overline{z}) \rightarrow \exists!\overline{z}\, R(w_1,\ldots,w_m,\overline{x},\overline{y},\overline{z})$$

- $\mathcal{A} \models \psi(w_1,\ldots,w_m)$ *if, and only if*

  in the deterministic, commutative Hypergraph specified by $R(w_1,\ldots,w_m,\overline{x},\overline{y},\overline{z})$ *on* $|(\mathcal{A},w_1,\ldots,w_m)|^j$*, we have, for all* $u \in |\mathcal{A}|$*,* $u^j = (u,\ldots,u)$ *is accessible from* ${w_1}^j$*.*

- $\mathcal{A} \models \exists w_1,\ldots,w_m \psi(w_1,\ldots,w_m)$.

Then $\mathsf{P} = \pm\mathrm{PS}^{max\{k,l\}+1}[\mathbf{FO}]$ *on the class* $\mathbf{C}$.

Any tuple $(w_1,\ldots,w_m)$ s.t. $\mathcal{A} \models \psi(w_1,\ldots,w_m)$ may be considered a *generating* tuple for $\mathcal{A}$. Generating tuples will be denoted $(g_1,\ldots,g_m)$, and their underlying generating set $\{g_1,\ldots,g_m\}$ as G.

The principle results of [42] were:

- On the class of locally-ordered strongly connected digraphs, $\mathsf{P} = \pm\mathrm{PS}^1[\mathbf{FO}]$.
- On the class of planar triangulations, $\mathsf{P} = \pm\mathrm{PS}^2[\mathbf{FO}]$.

In these cases, the construction of ψ, as in the proposition, is fairly straightforward. However, if we are prepared to sacrifice a few levels in $\pm\mathrm{PS}^*[\mathbf{FO}]$, we can disregard ψ altogether.

Given some relation $R \in \pm\mathrm{PS}^k[\mathbf{FO}]$, as in the proposition, there must necessarily be some $\psi \in \mathrm{PS}^{k+3}[\mathbf{FO}]$ that will satisfy the required conditions. We may take $\psi(w_1,\ldots,w_m) :=$

$$\forall u \ \mathrm{PS}[\lambda\overline{x},\overline{y},\overline{z},R(w_1,\ldots,w_m,\overline{x},\overline{y},\overline{z})]({w_1}^j,u^j)$$

Since we may write the $\forall u$ as $\neg\exists u\neg$, $R \in \pm\mathrm{PS}^k[\mathbf{FO}]$ indeed implies that $\psi \in \pm\mathrm{PS}^{k+3}[\mathbf{FO}]$.

The following is now immediate:

Corollary. *Let* $\mathbf{C}$ *be any class of structures, and let* $\overline{x},\overline{y},\overline{z}$ *be variable* j*-tuples. Suppose there is a formula* $R(w_1,\ldots,w_m,\overline{x},\overline{y},\overline{z}) \in \pm\mathrm{PS}^k[\mathbf{FO}]$ *such that, for all* $\mathcal{A} \in \mathbf{C}$*, there exists a generating tuple* $(g_1,\ldots,g_m) \in |\mathcal{A}|^m$ *such that:*

- $R(g_1,\ldots,g_m,\overline{x},\overline{y},\overline{z})$ *is commutative in* $\overline{x}$ *and* $\overline{y}$*, and deterministic in* $\overline{z}$
- *For all* $u \in |\mathcal{A}|$*,* u^j *is R-accessible from* $(g_1)^j$*.*

Then $\mathsf{P} = \pm\mathrm{PS}^{k+4}[\mathbf{FO}]$ *on the class* $\mathbf{C}$*.*

5.1 Finitely generated sets

Intuitively, a set A on which some partial functions are defined is described as m-generated if it has a (generating) subset of m elements such that all elements of A may be obtained by (possibly nested) applications of these partial functions on the elements of this subset.

When F is a finite set of partial functions, each of some finite arity, we will want to define the set F^* of all functions that can be created from those in F by repeated relabelling and substitution. So long as F contains a non-unary function, F^* must be infinite (even under equivalent relabellings), since it will have functions of all arities. Throughout this chapter we will use a bracketed superscript to indicate the arity of variable tuples or partial functions. Thus, whilst for an element x, x^k denotes *the* k-tuple of xs, the notation $\overline{v}^{(k)}$ specifies a variable k-tuple, whose different positions may hold different values.

Definition. For some finite structure $\mathcal{A}$, let $F = \{f_1, \ldots, f_j\}$ be partial functions of respective arities $a_1, \ldots, a_j$ (*i.e.* $f_1^{(a_1)} : |\mathcal{A}|^{a_1} \to |\mathcal{A}|$, ..., $f_j^{(a_j)} : |\mathcal{A}|^{a_j} \to |\mathcal{A}|$). Then F^* is defined inductively via:

- $f_1(v_1, \ldots, v_{a_1}), \ldots, f_j(v_1, \ldots, v_{a_j}) \in F^*$.
- (Projection/Reordering.) If $f \in F^*$ of arity a, and $a' \leq a$, then let $\{n_1, \ldots, n_a\}$ and $\{n'_1, \ldots, n'_{a'}\}$ be subsets of $\mathbb{Z}$ of order a and a', respectively. If we have a function $p : \{n_1, \ldots, n_a\} \to \{n'_1, \ldots, n'_{a'}\}$, then $f' \in F^*$, of arity a', where:

$$f'(v_{n_1}, \ldots, v_{n_{a'}}) := f(v_{p(n_1)}, \ldots, v_{p(n_a)})$$

- (Composition.) If $f, f' \in F^*$, with respective arities a, a', then $f'' \in F^*$, of arity $(a - 1 + a')$, where:

$$f''(v_1, \ldots, v_{a-1}, v_a, \ldots, v_{a+a'-1}) := f(v_1, \ldots, v_{a-1}, f'(v_a, \ldots, v_{a+a'-1}))$$

Projection/Reordering is nothing more than relabelling of the variables. Because of the Reordering rule, we have no need to explicitly mention compositions that occur other than at the right hand end of the outer partial function. The minimum depth of nestings of Composition in a partial function $f \in F^*$ will be known as the *rank* of f in F^*.

Definition. Let $F = \{f_1, \ldots, f_j\}$ be a set of functions of respective arities $a_1, \ldots, a_j$. Let $\sigma_F = \langle f_1, \ldots, f_j \rangle$ be the associated signature. Then:

- $\mathrm{STRUC}(\sigma_F, m)$ is the class of finite structures over σ_F, such that, for all $\mathcal{A} \in \mathrm{STRUC}(\sigma_F, m)$, there exists a generating subset $G = \{g_1, \ldots, g_m\} \subseteq |\mathcal{A}|$, such that, for every $u \in |\mathcal{A}|$, there exists an arity r, a $f^{(r)} \in F^*$, and a $\overline{w}^{(r)} = (w_1, \ldots, w_r) \in G^r$, such that $u = f^{(r)}(\overline{w}^{(r)})$.

$\mathrm{STRUC}(\sigma_F, m)$ is said to be the class of structures that can be *m-generated* by the set of partial functions F.

5.1.1 F contains a single k-ary partial function f_0.

Theorem 28. *For each m, we have that* $\mathsf{P} = \pm\mathrm{PS}^4[\mathbf{FO}]$ *on the class* $\mathrm{STRUC}(\sigma_{\{f_0\}}, m)$.

Proof. Let k be the arity of the partial function f_0, and let $\overline{x}, \overline{y}, \overline{z}$ be variable $(k+1)$-tuples. We will define a deterministic, commutative Hypergraph relation $R(w_1, \ldots, w_m, \overline{x}, \overline{y}, \overline{z})$, in quantifier-free **FO**, such that, for all structures $\mathcal{A} \in$

STRUC$(\sigma_{\{f_0\}}, m)$, there exists $g_1, \ldots, g_m \in |\mathcal{A}|$ such that, for all $u \in |\mathcal{A}|$, u^{k+1} is accessible from ${g_1}^{k+1}$. We may then appeal to the Corollary of Proposition 27.

We will specify $R(w_1, \ldots, w_m, \overline{x}, \overline{y}, \overline{z})$ as $R(\overline{x}, \overline{y}, \overline{z})$, where the entries of $\overline{x}, \overline{y}, \overline{z}$ may be among the variables $w_1, \ldots, w_m$. We will define R over $(k+1)$-tuples from $(\{P, Q, S, \sqcup\} \uplus |\mathcal{A}|)^{k+1}$.

The symbols P, Q, and S are used for switching rules, and $\sqcup$ represents blank. (Note that we can enlarge our alphabet to include these special symbols in precisely the manner we did in Section 3.4. Thus, each 'variable' we discuss here, will, in point of fact, be a quintuple of actual variables.)

We begin with the 'start' rules:

- $R[(w_1^{k+1}), (w_1^{k+1}), (P, \sqcup^{k-1}, w_1)]$.
- $R[(P, \sqcup^{k-1}, w_i), (P, \sqcup^{k-1}, w_i), (P, \sqcup^{k-1}, w_{i+1})]$ for $1 \leq i < m$.

We now progress to the 'active' rules:

Switching:

- $R[(P, \sqcup^{k-i}, \overline{x}^{(i)}), (P, \sqcup^{k-i}, \overline{x}^{(i)}), (Q, \sqcup^{k-i}, \overline{x}^{(i)})]$ for $\overline{x}^{(i)} \in |\mathcal{A}|^i$, and $i \leq k$.
- $R[(Q, \sqcup^{k-i}, \overline{x}^{(i)}), (Q, \sqcup^{k-i}, \overline{x}^{(i)}), (S, \sqcup^{k-i}, \overline{x}^{(i)})]$ for $\overline{x}^{(i)} \in |\mathcal{A}|^i$, and $i \leq k$.

Concatenation:

- $R[(P, \sqcup^{k-i}, \overline{x}^{(i)}), (Q, \sqcup^{k-j}, \overline{y}^{(j)}), (P, \sqcup^{k-j-i}, \overline{x}^{(i)}, \overline{y}^{(j)})]$ for $\overline{x}^{(i)} \in |\mathcal{A}|^i$, $\overline{y}^{(j)} \in |\mathcal{A}|^j$, and $i + j \leq k$.

Production:

- $R[(Q, \sqcup^i, \overline{x}^{(k-i)}), (S, \sqcup^j, \overline{y}^{(k-j)}), (P, \sqcup^{k-1}, z)]$ for $i + j = k$, and $f_0(\overline{x}, \overline{y}) = z$.

and the 'finish' rule:

- $R[(P, \sqcup^{k-1}, x), (S, \sqcup^{k-1}, x), (x^{k+1})]$ for $x \in |\mathcal{A}|$.

Finally, we consider R to be the symmetric closure of the above rules, *i.e.* for all $\overline{x}^{(k+1)}, \overline{y}^{(k+1)}, \overline{z}^{(k+1)}$,

$$R(\overline{x}^{(k+1)}, \overline{y}^{(k+1)}, \overline{z}^{(k+1)}) \Rightarrow R(\overline{y}^{(k+1)}, \overline{x}^{(k+1)}, \overline{z}^{(k+1)}).$$

This ensures the commutativity of R.

R is clearly deterministic in $\overline{z}^{(k+1)}$, and can be written in quantifier-free **FO**. We will now prove that, for all $u \in |\mathcal{A}|$, u^{k+1} is accessible from ${g_1}^{k+1}$. It follows, from the start and finish rules, that this is equivalent to the question of whether $(P, \sqcup^{k-1}, u)$ is accessible from the collection $(P, \sqcup^{k-1}, g_1), \ldots, (P, \sqcup^{k-1}, g_m)$.

We know that, for each such u, there exists a partial function $f^{(r)} \in \{f_0\}^*$, and tuple $(w_1,\ldots,w_r) \in \{g_1,\ldots,g_m\}^r$, such that $u = f^{(r)}(w_1,\ldots,w_r)$. We prove $(P,\sqcup^{k-1},u)$ is accessible by induction on the rank of $f^{(r)}$.

(Base Case.) When the rank of $f^{(r)}$ is 0, then $r \leq k$, and it follows that $u = f_0(w_1,\ldots,w_r)$ for some $w_1,\ldots,w_r \in \{g_1,\ldots,g_m\}$. We may access $(P,\sqcup^{k-1},u)$ from $(P,\sqcup^{k-1},g_1),\ldots,(P,\sqcup^{k-1},g_m)$ by repeated use of Switching, then repeated Concatenation, and finally a single application of Production.

(Inductive Step). Assuming it works for rank δ, we prove it works for rank $\delta+1$. If $f^{(r)}$ is of rank $\delta+1$, then it follows from the definition of rank, and the inductive hypothesis, that $u = f_0(w_1,\ldots,w_r)$, where $(P,\sqcup^{k-1},w_1)$,…,$(P,\sqcup^{k-1},w_r)$ have been accessed (since $w_1,\ldots,w_r$ are generated by partial functions of strictly lower rank). Again, we access $(P,\sqcup^{k-1},u)$ by repeated use of Switching, then repeated Concatenation, then a single application of Production. □

5.1.2 F contains multiple partial functions.

Theorem 29. *Let F be a finite set of partial function symbols. For each m, we have that* $\mathsf{P} = \pm\mathrm{PS}^4[\mathbf{FO}]$ *on the class* $\mathrm{STRUC}(\sigma_F,m)$.

Proof. We reduce this case to the previous. Suppose F contains j partial functions of respective arities $a_1,\ldots,a_j$. Let $a = max\{a_1,\ldots,a_j\}$. We aim to construct a single partial function f_F, of arity $(a+j)$, that simulates all the functions in F. $\overline{w}^{(j)} = (w_1,\ldots,w_j)$ will represent functions f_1 to f_j according to our ubiquitous scheme:

- if $w_1 = w_2$ then $\overline{w}^{(j)}$ represents f_1,
- if $w_1 \neq w_2$ but $w_2 = w_3$ then $\overline{w}^{(j)}$ represents f_2,
- $\vdots$
- if $w_1 \neq w_2$, …, $w_{j-2} \neq w_{j-1}$ but $w_{j-1} = w_j$ then $\overline{w}^{(j)}$ represents f_{j-1}, and
- if $w_1 \neq w_2$, …, $w_{j-1} \neq w_j$ then $\overline{w}^{(j)}$ represents f_j.

Suppose $\widehat{w}$ represents f_i whose arity is a_i. Consider $f_F(\overline{x}^{(a_i)},\overline{y}^{(a-a_i)},\widehat{w})$ to be $f_i(\overline{x}^{(a_i)})$, if f_i is defined at $\overline{x}^{(a_i)}$, and undefined otherwise.

Since j is fixed, this construction of f_F can be specified in quantifier-free **FO**. □

5.1.3 An application: finitely generated groups

We say that a finite group H is m-generated if there exists a set of m generating elements $G = \{g_1, \ldots, g_m\}$, such that for every $x \in H$ we have some $y \in G^*$ such that $x \equiv_H y$ (where $*$ is the usual Kleene star). Clearly, all groups of order $\leq m$ are m-generated.

Corollary. *For each m, on the class of m-generated finite groups,* $\pm\mathrm{PS}^4[\mathbf{FO}] = \mathsf{P}$.

Corollary. *On the class of finite simple groups,* $\pm\mathrm{PS}^4[\mathbf{FO}] = \mathsf{P}$.

Proof. Recall that finite simple groups are 2-generated [3, 39]. $\square$

Despite finitely generated groups being a paradigm for our systems, it should be noted that the results of the previous section go well beyond groups; beyond single functions, and beyond associativity.

5.2 Hamiltonian Outerplanar graphs

A graph is said to be *outerplanar* if it can be drawn in the plane with all its vertices on the outer face. Such a drawing will be called an OP-drawing.

Definition. A Hamilton cycle in a graph $\mathcal{G}$, where $||\mathcal{G}|| = n$, is a sequence s_{cyc} of distinct vertices v_i (for $1 \leq i \leq n$) such that, for $1 \leq i < n$, $E^{\mathcal{G}}(v_i, v_{i+1})$, and also $E^{\mathcal{G}}(v_n, v_1)$.

We consider a *hamiltonian outerplanar graph* (HOP) to be an antireflexive, undirected, outerplanar graph with a Hamilton cycle. We start by noting some basic properties of outerplanar graphs that have a Hamilton cycle.

Lemma 30.

(i) *Consider an* HOP *graph $\mathcal{G}$, with Hamilton cycle s_{cyc}. Then, in any* OP-*drawing of $\mathcal{G}$, s_{cyc} must be on the outer face.*

(ii) *For any* HOP *graph $\mathcal{G}$, the subgraph given by any Hamilton cycle is unique.*

(iii) *$\mathcal{G}$ has a unique* OP-*drawing in the plane, up to combinatorial isomorphism.*

Proof. (i) Note that any OP-drawing of s_{cyc} is combinatorially equivalent to the n-gon ($n = ||\mathcal{G}||$). Thus, s_{cyc} must appear on the outer face of any OP-drawing.

(ii) Consider a graph $\mathcal{G}$ with two Hamilton cycles, s_{cyc} and s'_{cyc}, that give rise to different subgraphs. In any OP-drawing of $\mathcal{G}$, s_{cyc} and s'_{cyc} must be drawn as distinct n-gons over the same vertices. Yet not both can be on the outer face, violating part (i) .

(iii) The unique Hamilton cycle subgraph dictates the unique OP-drawing. $\square$

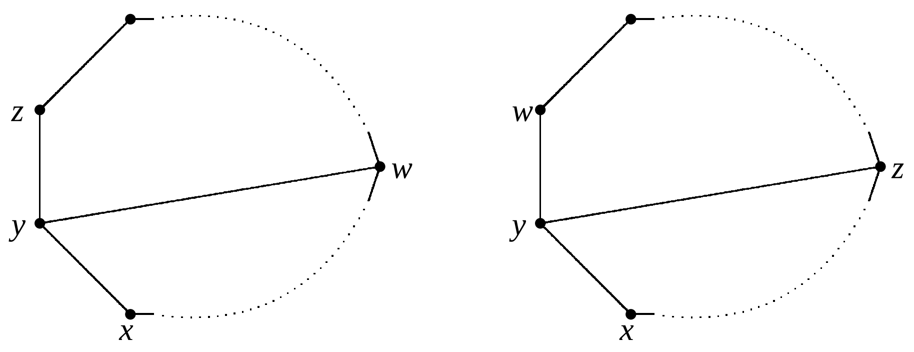


Figure 5.1: Diagrams for Lemma 31.

We make use of this unique OP-drawing by now referring, unambiguously, to *the* outer face.

Lemma 31. *There is a formula $\varphi(x,y,z) \in \pm\mathrm{PS}^3[\mathbf{FO}]$ that holds on an* HOP $\mathcal{G}$ *if, and only if, x and z are the distinct neighbours of y, on the outer face.*

Proof. We first define $P(x,z,y,w) \in \pm\mathrm{PS}^1[\mathbf{FO}]$, intended to mean that there is a path from x to z avoiding both y and w. We define P as the Transitive Closure (though in Path System logic) of the following formula θ:

$$\theta(p,q,y,w) \ := \ p \neq y \wedge p \neq w \wedge q \neq y \wedge q \neq w \wedge E(p,q)$$

Thus,

$$P(x,z,y,w) := \mathrm{PS}[\lambda p, p, q\theta](x,z)$$

Now, $\varphi(x,y,z) \ :=$

$$E(x,y) \wedge E(y,z) \wedge x \neq z \wedge \\ \forall w[(w \neq x \wedge w \neq z \wedge E(y,w) \rightarrow \neg P(x,z,y,w)]$$

We now prove that x and z are the distinct neighbours of y, on the outer face of $\mathcal{G}$, if, and only if, $\mathcal{G} \models \varphi(x,y,z)$.

(Forwards.) If x and z are the distinct neighbours of y on the outer face, then the first three conjuncts of φ are clearly satisfied. Furthermore, the edge between any distinct w and y must cut across the OP-drawing of $\mathcal{G}$ (see Figure 5.1). It follows that all paths from x to z must go through either y or w. Hence $\mathcal{G} \models \varphi(x,y,z)$.

(Backwards.) Suppose x and z are not the distinct neighbours of y on the outer face. If x and z are not distinct, or y is not adjacent to both, then we fail on one of the first three conjuncts of φ. So, assume x and z are distinct, and y is adjacent to both, but x and z are not the two neighbours of y on the outer face. If we choose some distinct w that is such a neighbour, then there is clearly a path from x to z avoiding both y and w (see Figure 5.1). In any case, $\mathcal{G} \not\models \varphi(x,y,z)$. ☐

Theorem 32. *On the class* HOP, $\mathsf{P} = \pm\mathrm{PS}^7[\mathbf{FO}]$.

Proof. We will define a deterministic, commutative Hypergraph relation $R(w_1,w_2,w_3,x_1,x_2,y_1,y_2,z_1,z_2)$, in $\pm\mathrm{PS}^3[\mathbf{FO}]$ s.t. for all structures $\mathcal{G} \in$ HOP, there exists $g_1,g_2,g_3 \in |\mathcal{G}|$ such that, for all $u \in |\mathcal{G}|$, u^2 is accessible from ${g_1}^2$. We may then appeal to the Corollary of Proposition 27, with $j = 2$.

The rules will be the symmetric closure of:

- $R[(w_1,w_1),(w_1,w_1),(w_1,w_2)]$
- $R[(w_2,w_2),(w_2,w_2),(w_2,w_3)]$
- $R[(w,x),(x,y),(y,z)]$ if $\varphi(w,x,y) \wedge \varphi(x,y,z)$ (where φ is as in Lemma 31).
- $R[(w,x),(w,x),(x,x)]$ if $w \neq x$.

That the rules are deterministic, commutative, and can be written in $\pm\mathrm{PS}^3[\mathbf{FO}]$ is straightforward. It is also clear that, starting with any g_1,g_2,g_3 such that $\varphi(g_1,g_2,g_3)$, all vertices are accessible: from g_1 we access g_2 (*some* next vertex on the outer face – which determines whether we are moving clockwise or anti-clockwise around a certain OP-drawing of $\mathcal{G}$), then g_3 (*the* next vertex on the outer face – now direction is set), then all the way round the outer face until we reach the final vertex on the Hamilton cycle (before we reach g_1 again). ☐

Remark. We can easily extend our result to hamiltonian outerplanar graphs that are not undirected or not antireflexive, by considering their undirected, antireflexive versions. Specifically change all instances of $E(x,y)$, in the prior discourse, to $(E(x,y) \vee E(y,x)) \wedge x \neq y$.

Chapter 6

Dichotomies in Boolean Constraint Satisfaction

6.1 Introduction

Let σ range over all relational signatures. We define the relational class of boolean structures $\textsc{Bool}$.

$$\textsc{Bool} \;:=\; \{\mathcal{A} \,:\, \mathcal{A} \text{ is a } \sigma\text{-structure and } ||\mathcal{A}|| = 2\}$$

We denote tuples of variables (resp. boolean constants) in bold, *e.g.* **x** (resp. **t**). These tuples are not of a uniform arity.

Definition. For $\mathcal{A} \in \textsc{Bool}$, and each with the question as to whether $\mathcal{A} \models \varphi$,

- the problem $\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$ has input $\varphi := \exists \mathbf{x} Q(\mathbf{x})$,
- the problem $\mathrm{Q}\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$ has input, for some $n \geq 1$, of the form,

$$\varphi := \forall \mathbf{x_1} \exists \mathbf{x_2} \forall \mathbf{x_3} \exists \mathbf{x_4} \ldots \forall \mathbf{x_{2n+1}} \exists \mathbf{x_{2n+2}} Q(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{2n+2}})$$

- the problem Π_{2n+1}-$\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$ has input,

$$\varphi := \forall \mathbf{x_1} \exists \mathbf{x_2} \forall \mathbf{x_3} \exists \mathbf{x_4} \ldots \forall \mathbf{x_{2n+1}} Q(\mathbf{x_1}, \mathbf{x_2} \ldots \mathbf{x_{2n+1}})$$

- the problem Π_{2n+2}-$\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$ has input,

$$\varphi := \forall \mathbf{x_1} \exists \mathbf{x_2} \forall \mathbf{x_3} \exists \mathbf{x_4} \ldots \forall \mathbf{x_{2n+1}} \exists \mathbf{x_{2n+2}} Q(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{2n+2}})$$

- the problem Σ_{2n+1}-$\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$ has input,

$$\varphi := \exists \mathbf{x_1} \forall \mathbf{x_2} \exists \mathbf{x_3} \forall \mathbf{x_4} \ldots \exists \mathbf{x_{2n+1}} Q(\mathbf{x_1}, \mathbf{x_2} \ldots \mathbf{x_{2n+1}})$$

- the problem Σ_{2n+2}-$\text{SAT}_{\text{NC}}(\mathcal{A})$ has input,

$$\varphi := \exists \mathbf{x_1} \forall \mathbf{x_2} \exists \mathbf{x_3} \forall \mathbf{x_4} \ldots \forall \mathbf{x_{2n+2}} Q(\mathbf{x_1}, \mathbf{x_2} \ldots \mathbf{x_{2n+1}})$$

where, in each case, Q is a conjunction of positive atoms.

The problems $\text{SAT}_{\text{C}}(\mathcal{A})$ *etc* are defined analogously, but with the two boolean constants 0 and 1 built-in to the signature.

It is clear that Π_{2n+2}-$\text{SAT}_{\text{NC}}(\mathcal{A})$ (respectively, Σ_{2n+1}-$\text{SAT}_{\text{NC}}(\mathcal{A})$) is in the complexity class Π^{P}_{2n+2} (respectively, $\Sigma^{\mathsf{P}}_{2n+1}$). In fact, it follows from [43] that they are complete for those classes, for certain $\mathcal{A}$. It is also clear that Π_1-$\text{SAT}_{\text{NC}}(\mathcal{A})$ is tractable, for all $\mathcal{A}$, since we may check each extensional relation independently, one-by-one, for an invalidating assignment. Indeed, if the maximum arity of a relation in $\mathcal{A}$ is a, then the complexity of Π_1-$\text{SAT}_{\text{NC}}(\mathcal{A})$ is $O(n^a)$, where n is the size of the input. It follows, by similar argument [43], that Π_{2n+1}-$\text{SAT}_{\text{NC}}(\mathcal{A})$ (respectively, Σ_{2n+2}-$\text{SAT}_{\text{NC}}(\mathcal{A})$) is in the complexity class Π^{P}_{2n} (respectively, $\Sigma^{\mathsf{P}}_{2n+1}$).

The comments of the previous paragraph apply equally to the problems Π_i- and Σ_i-SAT_{C}, *i.e.* in the situation where the Boolean constants are available.

Definition. For a relation R, of arity a, define:

- $\exists$-$\text{FORM}(R)$ to be the set of formulae formed from the closure of the atoms $R(\mathbf{x})$ (where $\mathbf{x}$ is an a-tuple of not necessarily distinct variables), under conjunction and existential quantification.

- Π_2-$\text{FORM}(R)$ to be the set of formulae of the form $\forall \mathbf{x} \varphi(\mathbf{x}, \mathbf{y})$, where $\varphi \in$ $\exists$-$\text{FORM}(R)$.

- $\forall/\exists$-$\text{FORM}(R)$ to be the set of formulae formed from the closure of the atoms $R(\mathbf{x})$, under conjunction, existential quantification, and universal quantification.

We define $\exists$-$\text{REL}(R)$ to be the set of relations expressible by formulae in $\exists$-$\text{FORM}(R)$, when reading the variables lexicographically. We do likewise for Π_2-$\text{REL}(R)$ and $\forall/\exists$-$\text{REL}(R)$. These sets are sometimes known as relational clones [14].

We may refer to boolean relations by some propositional formula that expresses them, reading the propositional variables lexicographically, *e.g.* $[A \vee B]$ expresses $\{(0,1),(0,1),(1,1)\}$; $[A \neq B]$ expresses $\{(0,1),(1,0)\}$.

Definition. A relation R, of arity a, is:

(i) 0-*valid* iff it contains the tuple (0^a).

(ii) 1-*valid* iff it contains the tuple (1^a).

(iii) *horn* iff it may be expressed by a propositional formula in CNF where each clause has at most one positive literal.

(iv) *dual horn* iff it may be expressed by a propositional formula in CNF where each clause has at most one negative literal.

(v) *bijunctive* iff it may be expressed by a propositional formula in 2-CNF.

(vi) *affine* iff it may be expressed by a propositional formula that is the conjunction of linear equations over $\mathbb{Z}_2$.

Given a template $\mathcal{A}$, over signature involving relations $R_1, \ldots, R_j$, of respective arities $a_1, \ldots, a_j$, we construct the relation $R^{\mathcal{A}}$ thus:

- If $\mathcal{A}$ has $j' \leq j$ non-empty relations, then let R'_i be the ith non-empty relation of $\mathcal{A}$.
- Let $R^{\mathcal{A}} = R'_1 \times \ldots \times R'_{j'}$.

This construction will enable us to consider signatures with multiple relations, as though they only had one. This is because each of the six attributes from the previous definition hold over all the relations of $\mathcal{A}$ if, and only if, they hold for $R^{\mathcal{A}}$. Note that all the relations of $\mathcal{A}$, except possibly the empty relation ϕ, are present in $\exists$-REL$(R_{\mathcal{A}})$.

Theorem 33 (I-III: Schaefer [38], and IV: Dalmau/Creignou et al [15, 14])**.**

I. SAT$_{\mathrm{C}}(\mathcal{A})$ *is tractable if* $R^{\mathcal{A}}$ *satisfies any of conditions* $(iii)-(vi)$*, and is* NP*-complete otherwise.*

II. SAT$_{\mathrm{NC}}(\mathcal{A})$ *is tractable if* $R^{\mathcal{A}}$ *satisfies any of conditions* $(i)-(vi)$*, and is* NP*-complete otherwise.*

III. QSAT$_{\mathrm{C}}(\mathcal{A})$ *is tractable if* $R^{\mathcal{A}}$ *satisfies any of conditions* $(iii)-(vi)$*, and is* Pspace*-complete otherwise.*

IV. QSAT$_{\mathrm{NC}}(\mathcal{A})$ *is tractable if* $R^{\mathcal{A}}$ *satisfies any of conditions* $(iii)-(vi)$*, and is* Pspace*-complete otherwise.*

We will briefly consider the methods involved in proving these dichotomies. When the boolean constants are present, Schaefer was able to take any $R^{\mathcal{A}}$ not in classes $(i)-(iv)$, and construct the ternary boolean not-all-equal relation, which is known to give rise to an NP-complete SAT, and Pspace-complete QSAT. When constants are not present, there are the degenerate cases of 0- and 1-validity, which become trivial. For the other tractable sub-classes of templates $\mathcal{A}$, we clearly have that $\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$ is polynomially reducible to $\textsc{Sat}_{\mathrm{C}}(\mathcal{A})$, guaranteeing its tractability in the no-constants scenario. It remained for him to prove that $\textsc{Sat}_{\mathrm{C}}(\mathcal{A})$ is polynomially reducible to $\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$ for those $\mathcal{A}$ not in classes $(i)-(vi)$. He did this by simulating the boolean constants. Call a relation $R_{\mathcal{A}}$ *complementative* if, for all tuples $\mathbf{x}$ in $R_{\mathcal{A}}$, the tuple $\mathbf{x}'$, obtained from $\mathbf{x}$ by swapping the 0s and 1s, is also in $R_{\mathcal{A}}$. Schaefer proved the following:

Lemma 34 ([38]). *For some $R_{\mathcal{A}}$, not in classes $(i)-(vi)$, either $[A],[\neg A]\in\exists\text{-}\textsc{Rel}(R_{\mathcal{A}})$, or $[A\neq B]\in\exists\text{-}\textsc{Rel}(R_{\mathcal{A}})$ and $R_{\mathcal{A}}$ is complementative.*

Before going further, we will need the following lemma.

Lemma 35 ([15]). *If $R_{\mathcal{A}}$ is complementative, then all relations in $\forall/\exists\text{-}\textsc{Rel}(R_{\mathcal{A}})$ are complementative.*

Proof. We prove this by induction on the term-complexity of $\varphi\in\forall/\exists\text{-}\textsc{Rel}(R_{\mathcal{A}})$. The base case is trivial. For the inductive step, note that:

- $R(\mathbf{x})$ and $R'(\mathbf{x}')$ complementative, implies $R\wedge R'(\mathbf{x},\mathbf{x}')$ complementative.
- $R(\mathbf{x})$ complementative, implies $\exists x_1 R(\mathbf{x})$ is complementative.
- $R(\mathbf{x})$ complementative, implies $\forall x_1 R(\mathbf{x})$ is complementative.

□

We can now sketch Schaefer's result, and method.

Proposition 36. *If $R^{\mathcal{A}}$ is in none of the classes $(i)-(vi)$ above, then $\textsc{Sat}_{\mathrm{C}}(\mathcal{A})\leq_{\mathrm{P}}\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$.*

Proof. By Lemma 34, we need to consider two cases.

(Case 1.) We have $\exists\mathbf{y_1}Q_1(\mathbf{y_1},b),\exists\mathbf{y_0}Q_0(\mathbf{y_0},a)\in\exists\text{-}\textsc{Form}(R^{\mathcal{A}})$ expressing $[X],[\neg X]$, where Q_1 and Q_0 are positive conjunctive. We are now in a position to simulate the constants 0 and 1, for, given an input $\exists\mathbf{x}\varphi(\mathbf{x},0,1)$ for $\textsc{Sat}_{\mathrm{C}}(\mathcal{A})$, we know,

$$\exists\mathbf{x}\varphi(\mathbf{w},0,1)\ \Leftrightarrow\ \exists\mathbf{x}\exists a\exists b\ \varphi(\mathbf{x},a,b)\wedge\exists\mathbf{y_1}Q_1(\mathbf{y_1},b)\wedge\exists\mathbf{y_0}Q_0(\mathbf{y_0},a)$$

The latter formula is an input for $\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$, when the inner existential quantifiers are drawn out, putting it in prenex form.

(Case 2.) We have $\exists\mathbf{y}Q'(\mathbf{y},a,b)\in\exists\text{-}\textsc{Form}(R^{\mathcal{A}})$ that expresses $[A\neq B]$, and $R^{\mathcal{A}}$ is complementative. It follows that,

$$\exists\mathbf{x}\varphi(\mathbf{x},0,1)\in\textsc{Sat}_{\mathrm{C}}(\mathcal{A})\ \Leftrightarrow\ \exists\mathbf{x}\exists a\exists b\ \varphi(\mathbf{x},a,b)\wedge\exists\mathbf{y}Q'(\mathbf{y},a,b)\in\textsc{Sat}_{\mathrm{NC}}(\mathcal{A})$$

since φ must be complementative (by Lemma 35). □

The problem of removing the constants in $\mathrm{Q}\textsc{Sat}_{\mathrm{C}}$ was not attended to by Schaefer. It was finally settled many years later by Dalmau [15], and, independently, by Creignou et al. [14].

6.2 Technical results

Before progressing, we will need a number of technical lemmas.

Lemma 37 (Quantifier Re-ordering)**.** *Let the variables a, b not appear in $\mathbf{x}$. The following are equivalent on all boolean structures for all conjunctive positive Q:*

$$\exists \mathbf{x} \exists a \forall b \; B(a,b) \wedge Q(a,\mathbf{x})$$

$$\forall b \exists \mathbf{x} \exists a \; B(a,b) \wedge Q(a,\mathbf{x})$$

If, and only if:

- *B is ϕ (empty), singleton, $\{(0,0),(1,0)\}$, $\{(1,1),(0,1)\}$, or*
- *B contains $(0,0)$ and $(0,1)$.*
- *B contains $(1,0)$ and $(1,1)$.*

Proof. If B is ϕ, singleton, $\{(0,0),(1,0)\}$, or $\{(1,1),(0,1)\}$ then both sentences will be false irrespective of Q.

If B contains both $(0,0)$ and $(0,1)$, it may easily be verified that both sentences are equivalent. The case where B contains both $(1,0)$ and $(1,1)$ is symmetric.

The remaining possibilities for B are $\{(0,0),(1,1)\}$ and $\{(1,0),(0,1)\}$, which will be each false in the former sentence, but may be true in the latter (*e.g.* if Q is logically valid). $\square$

Given the boolean k-tuples $\mathbf{t_1} = (t_1^1, \ldots, t_1^k)$ and $\mathbf{t_2} = (t_2^1, \ldots, t_2^k)$ we define $\mathbf{t_1} \oplus \mathbf{t_2}$ to be $(t_1^1 + t_2^1, \ldots, t_1^k + t_2^k)$ where the addition is modulo 2.

Lemma 38 (0-affine case. [13])**.** *Let R be a boolean relation of arity k. The following are equivalent:*

(a) *R is 0-valid and affine.*

(b) *$0^k \in R$, and, for all assignments $\mathbf{t_1}, \mathbf{t_2} \in R$, we have $\mathbf{t_1} \oplus \mathbf{t_2} \in R$.*

Definitions. For a relation R of arity k, and any set $T = \{i_1, \ldots, i_j\}$ of j positions $0 \leq i_1 < \ldots < i_j \leq k$, we define $R|T$ to be the j-ary relation $\exists x_{l_1} \ldots \exists x_{l_{k-j}} R(x_1, \ldots, x_k)$, where $\{l_1, \ldots, l_{k-j}\} = \{1, \ldots, k\} - T$. Observe that $R|T$ is in $\exists$-REL(R).

For any $\mathbf{t} \in \{0,1\}^k$ and $T \subset \{1, \ldots, k\}$, we define $\mathbf{t}|T$ as the assignment $\mathbf{t}' \in \{0,1\}^{|T|}$ that agrees with $\mathbf{t}$ in the positions indexed by T.

Let R be a relation of arity k, and $\mathbf{t} \in \{0,1\}^k$ an assignment. We say that $\mathbf{t}$ is *j-compatible* (w.r.t. R) if, for every subset $T \subset \{1, \ldots, k\}$ (of size $|T| < j$), we can find some assignment $\mathbf{t}' \in R$ such that $\mathbf{t}$ and $\mathbf{t}'$ agree in the positions indexed by T. This is equivalent to the condition that any j-ary sub-tuple of $\mathbf{t}$ can be extended to some k-ary $\mathbf{t}'$ in R. Clearly this is trivially true when $\mathbf{t}$ is itself in R; the interesting cases are when it is not.

The notion of j-compatibility is key in characterising horn logical relations.

Lemma 39 (horn case. [15])**.** *Let R be a boolean relation of arity k. The following are equivalent:*

(a) *R is horn.*

(b) *for all $\mathbf{t_1}, \mathbf{t_2} \in R$, we have $\mathbf{t_1} \wedge \mathbf{t_2} \in R$.*

(c) *for every $T \subset \{1, \dots, k\}$, and for every $|T|$-compatible (w.r.t. $R|T$) assignment $\mathbf{t} \in \{0,1\}^{|T|}$ not in $R|T$, we have that $\mathbf{t}$ contains at most a single 0.*

Lemma 40 (Adapted from [15])**.** *If $R^{\mathcal{A}}$ is 0-valid and non-Horn, then there is a relation S^4_λ definable in $\exists$-*Rel*$(R^{\mathcal{A}})$ such that:*

$\in S^4_\lambda$	$\notin S^4_\lambda$
$(0,0,0,0)$	$(0,0,0,1)$
$(0,1,0,1)$	
$(0,0,1,1)$	

Proof. Since $R^{\mathcal{A}}$ is non-horn, we may guarantee to break part (c) of the previous lemma. This implies that there is a subset of indices $T \subseteq \{1, \dots, k\}$ and a $|T|$-compatible assignment $\mathbf{t}$ not in $R|T$ that contains at least two zeros. We may benefit from dwelling on what exactly this means. It guarantees us some $|T|$-ary relation $R|T$, in $\exists$-Rel$(R^{\mathcal{A}})$, and a $|T|$-tuple $\mathbf{t}$ s.t.

- $\mathbf{t} \notin R|T$,
- for any $\mathbf{t}'$ that agrees with $\mathbf{t}$ in all but one position, $\mathbf{t}' \in R|T$, and
- $\mathbf{t}$ contains at least two zeros.

We therefore consider $R|T(v_1, \dots, v_{|T|})$ and two indices $\alpha, \beta \in \mathbb{Z}_{|T|}$ at which $\mathbf{t}$ has zeros. Note that $\mathbf{t}$ can not be all zeros, since R is 0-valid yet $\mathbf{t} \notin R|T$. Let I' be the set of indices at which $\mathbf{t}$ is one. Finally, let I'' be the set of indices, other than i, j, at which $\mathbf{t}$ is zero[1]. We obtain $S^4_\lambda(x, v_\alpha, v_\beta, y)$ from $R|T(v_1, \dots, v_{|T|})$ by substituting

[1]It is possible that I'' is empty, in which case S^4_λ will actually be a ternary relation. This will cause no problems, and will come out in the wash in Lemma 42.

all variables v_i s.t. $i \in I'$ by the variable y and substituting all variables v_i s.t. $i \in I''$ by the variable x.

We already know that $(0,0,0,1) \notin S^4_\lambda$, and we have assumed that R is 0-valid, hence $(0,0,0,0) \in S^4_\lambda$. Since $(0,0,0,1) \notin S^4_\lambda$, we will have $(0,1,0,1),(0,0,1,1) \in S^4_\lambda$ by the $|T|$-compatibility, since these two assignments each only change the value of a single variable in $R|T$. ☐

Lemma 41 (Adapted from [15])**.** *If $R^{\mathcal{A}}$ is* 0*-valid and non-affine, then there is a relation S^4_μ definable in* $\exists$-REL$(R^{\mathcal{A}})$ *such that:*

$\in S^4_\mu$	$\notin S^4_\mu$
$(0,0,0,0)$	$(0,1,1,0)$
$(0,1,0,1)$	
$(0,0,1,1)$	

Proof. Since R is 0-valid and affine, it follows from Lemma 38 that there are assignments $\mathbf{t_1},\mathbf{t_2} \in R$ such that $\mathbf{t_1} \oplus \mathbf{t_2} \notin R$. For $R(x_1,\ldots,x_k)$, define:

- $V_{00} = \{v : v \in \{x_1,\ldots,x_k\}$ v is 0 in t_1 and 0 in t_2 $\}$
- $V_{01} = \{v : v \in \{x_1,\ldots,x_k\}$ v is 0 in t_1 and 1 in t_2 $\}$
- $V_{10} = \{v : v \in \{x_1,\ldots,x_k\}$ v is 1 in t_1 and 0 in t_2 $\}$
- $V_{11} = \{v : v \in \{x_1,\ldots,x_k\}$ v is 1 in t_1 and 1 in t_2 $\}$

Let $S^4_\mu(y_{00},y_{01},y_{10},y_{11})$ be $R(x_1,\ldots,x_k)$ with the substitutions y_{00} for V_{00}, y_{01} for V_{01}, y_{10} for V_{10}, and y_{11} for V_{11}. The claimed properties follow immediately. ☐

Lemma 42. *If $R^{\mathcal{A}}$ is* 0*-valid and non-Horn and non-affine, then there is a relation $S^4_{\lambda\mu}$ definable in* $\exists$-REL$(R^{\mathcal{A}})$ *such that:*

$\in S^4_{\lambda\mu}$	$\notin S^4_{\lambda\mu}$
$(0,0,0,0)$	$(0,0,0,1)$
$(0,1,0,1)$	$(0,1,1,0)$
$(0,0,1,1)$	

Proof. $S^4_{\lambda\mu} := S^4_\lambda \wedge S^4_\mu$. ☐

6.3 A dichotomy theorem for Π_2-SAT$_{\mathrm{NC}}$

It follows from Schaefer's work and [43] that Π_2-SAT$_{\mathrm{C}}(\mathcal{A})$ is tractable if $R^{\mathcal{A}}$ is in any of the classes $(iii)-(vi)$, and Π_2^{P}-complete otherwise. Borrowing much from Dalmau, we will show that Π_2-SAT$_{\mathrm{NC}}$ exhibits the same dichotomy.

Our proof rests on the following:

Proposition 43. *Let $\mathcal{A} \in \mathcal{B}$. If $R^{\mathcal{A}}$ is neither horn, dual horn, affine, nor bijunctive, then* Π_2-SAT$_{\mathrm{C}}(\mathcal{A})$ *is polynomially reducible to* Π_2-SAT$_{\mathrm{NC}}(\mathcal{A})$.

If $R^{\mathcal{A}}$ is neither 0-valid nor 1-valid, we may appeal to Schaefer's method for simulating the constants. This may only result in more existential quantifiers on the inside of the input instance, which will not jeopardise our being in Π_2^{P}. However, if we need formulae with universal quantifiers to simulate the constants, then we find ourselves potentially outside Π_2^{P}, with more than a single alternation of quantifiers in the input instance.

Recall that we are only concerned with $R^{\mathcal{A}}$ that are non-horn, non-dual-horn, non-bijunctive, non-affine, and either 1-valid or 0-valid. We will consider four cases.

6.3.1 Case 1 : $R^{\mathcal{A}}$ is 0-valid and not 1-valid.

In this case we have the constant 0 for free, since $R^{\mathcal{A}}(a,\dots,a)$ expresses $[\neg A]$.

Let S_3^2 be the boolean relation $\{(0,0),(1,0),(1,1)\}$.

Lemma 44. *If $R^{\mathcal{A}}$ is 0-valid, non-horn, non-affine, and not 1-valid, then S_3^2 is definable in* $\exists$-REL$(R^{\mathcal{A}})$.

Proof. We consider two further possibilities for the relation $S_{\lambda\mu}^4$ above.

- If $S_{\lambda\mu}^4$ also contains $(0,0,1,0)$, then $S_3^2 = \exists a S_{\lambda\mu}^4(a,a,b,c) \wedge R^{\mathcal{A}}(a,\dots,a)$.
- If $S_{\lambda\mu}^4$ does not contain $(0,0,1,0)$, then $S_3^2 = \exists a \exists a' S_{\lambda\mu}^4(a,a',c,b) \wedge R^{\mathcal{A}}(a,\dots,a)$.

$\square$

In both cases S_3^2 is of the form $\exists \mathbf{w} Q_S(\mathbf{w},b,c)$, where Q_S is positive conjunctive. Note that $[A]$ is expressed by $\forall c S_3^2(b,c)$.

Lemma 45. *If $R^{\mathcal{A}}$ is 0-valid, non-horn, non-affine, and not 1-valid, then* Π_2-SAT$_{\mathrm{C}}(\mathcal{A})$ *polynomially reduces to* Π_2-SAT$_{\mathrm{NC}}(\mathcal{A})$.

Proof. Given an input $\forall \mathbf{x_1} \exists \mathbf{x_2} Q(\mathbf{x_1}, \mathbf{x_2}, 0, 1)$ for Π_2-SAT$_{\text{C}}(\mathcal{A})$, observe,

$$\begin{array}{llll}
 & \forall \mathbf{x_1} \exists \mathbf{x_2} & Q(\mathbf{x_1}, \mathbf{x_2}, 0, 1) & \\
\Leftrightarrow & \forall \mathbf{x_1} \exists \mathbf{x_2} \exists a \exists b & Q(\mathbf{x_1}, \mathbf{x_2}, a, b) & \wedge \forall c\, S_3^2(b, c) \\
 & & & \wedge R^{\mathcal{A}}(a, \dots, a) \\
\Leftrightarrow & \forall \mathbf{x_1} \exists \mathbf{x_2} \exists a \exists b & Q(\mathbf{x_1}, \mathbf{x_2}, a, b) & \wedge \forall c \exists \mathbf{w}\, Q_S(\mathbf{w}, b, c) \\
 & & & \wedge R^{\mathcal{A}}(a, \dots, a) \\
\Leftrightarrow & \forall \mathbf{x_1} \forall c \exists \mathbf{x_2} \exists a \exists b & Q(\mathbf{x_1}, \mathbf{x_2}, a, b) & \wedge \exists \mathbf{w}\, Q_S(\mathbf{w}, b, c) \\
 & & & \wedge R^{\mathcal{A}}(a, \dots, a)
\end{array}$$

Note that the final line is a valid input for Π_2-SAT$_{\text{NC}}(\mathcal{A})$. The final equivalence holds by the Quantifier Re-ordering Lemma, with $B := \exists \mathbf{w}\, Q_S(\mathbf{w}, b, c)$. ☐

6.3.2 Case 2 : $R^{\mathcal{A}}$ is 1-valid and not 0-valid.

This is the symmetric case of the previous (where zero is replaced with one, and vice-versa).

6.3.3 Case 3 : $R^{\mathcal{A}}$ is 0-valid and 1-valid, but not complimentative.

Since $R^{\mathcal{A}}$ is 0-valid, 1-valid, and yet not complimentative, there exists a tuple **t** in $R^{\mathcal{A}}$ s.t **t**s complement is not in $R^{\mathcal{A}}$. Let I index the set of positions at which **t** is zero and let J index those positions at which **t** is one. If $R^{\mathcal{A}}(v_1, \dots, v_k)$ is a k-ary relation, consider $Q_{atom}(a, b)$ to be $R^{\mathcal{A}}$ under the substitution a for all variables indexed by I and b for all variables indexed by J. Q_{atom} is atomic, and it expresses S_3^2. We now have that $\forall b S_3^2(a, b)$ expresses $[A]$, and $\forall a S_3^2(a, b)$ expresses $[\neg A]$.

Lemma 46. *If $R^{\mathcal{A}}$ is 0-valid, non-horn, non-affine, 1-valid, but not complementative, then Π_2-SAT$_{\text{C}}(\mathcal{A})$ polynomially reduces to Π_2-SAT$_{\text{NC}}(\mathcal{A})$.*

Proof. Given an input $\forall \mathbf{x_1} \exists \mathbf{x_2} Q(\mathbf{x_1}, \mathbf{x_2}, 0, 1)$ for Π_2-SAT$_{\text{C}}(\mathcal{A})$, observe,

$$\begin{array}{llll}
 & \forall \mathbf{x_1} \exists \mathbf{x_2} & Q(\mathbf{x_1}, \mathbf{x_2}, 0, 1) & \\
\Leftrightarrow & \forall \mathbf{x_1} \exists \mathbf{x_2} \exists a \exists b & Q(\mathbf{x_1}, \mathbf{x_2}, a, b) & \wedge \forall b'\, S_3^2(a, b') \\
 & & & \wedge \forall a'\, S_3^2(a', b) \\
\Leftrightarrow & \forall \mathbf{x_1} \exists \mathbf{x_2} \exists a \exists b & Q(\mathbf{x_1}, \mathbf{x_2}, a, b) & \wedge \forall b'\, Q_{atom}(a, b') \\
 & & & \wedge \forall a'\, Q_{atom}(a', b) \\
\Leftrightarrow & \forall \mathbf{x_1} \forall b' \forall a' \exists \mathbf{x_2} \exists a \exists b & Q(\mathbf{x_1}, \mathbf{x_2}, a, b) & \wedge\, Q_{atom}(a, b') \\
 & & & \wedge\, Q_{atom}(a', b)
\end{array}$$

The final line is a valid input for Π_2-SAT$_{\text{NC}}(\mathcal{A})$. The final equivalence holds via two applications of the Quantifier Re-ordering Lemma. ☐

6.3.4 Case 4 : $R^{\mathcal{A}}$ is 0-valid and 1-valid, and complimentative.

In this case $S^4_{\lambda\mu}$ will look like:

$\in S^4_{\lambda\mu}$	$\notin S^4_{\lambda\mu}$
$(0,0,0,0)$	$(0,0,0,1)$
$(0,1,0,1)$	$(0,1,1,0)$
$(0,0,1,1)$	
$(1,1,1,1)$	$(1,1,1,0)$
$(1,0,1,0)$	$(1,0,0,1)$
$(1,1,0,0)$	

Note that $\forall d \exists a S^4_{\lambda\mu}(a,b,c,d)$ expresses $[B \neq C]$. Since $S^4_{\lambda\mu} \in \exists\text{-}\mathrm{REL}(R^{\mathcal{A}})$, it follows that $[A \neq B]$ is expressed by $\forall d \exists a \exists \mathbf{w} Q_{\lambda\mu}(\mathbf{w},a,b,c,d)$, where $Q_{\lambda\mu}$ is positive conjunctive.

Lemma 47. *If $R^{\mathcal{A}}$ is 0-valid, non-horn, non-affine, 1-valid, and complementative, then $\Pi_2\text{-}\mathrm{SAT_C}(\mathcal{A})$ polynomially reduces to $\Pi_2\text{-}\mathrm{SAT_{NC}}(\mathcal{A})$.*

Proof. Given an input $\forall \mathbf{x_1} \exists \mathbf{x_2} Q(\mathbf{x_1},\mathbf{x_2},0,1)$ for $\Pi_2\text{-}\mathrm{SAT_C}(\mathcal{A})$, observe,

$$\begin{array}{llll} & \forall \mathbf{x_1} \exists \mathbf{x_2} & Q(\mathbf{x_1},\mathbf{x_2},0,1) & \\ \Leftrightarrow & \forall \mathbf{x_1} \exists \mathbf{x_2} \exists b \exists c & Q(\mathbf{x_1},\mathbf{x_2},b,c) & \wedge \forall d \exists a S^4_{\lambda\mu}(a,b,c,d) \\ \Leftrightarrow & \forall \mathbf{x_1} \exists \mathbf{x_2} \exists b \exists c & Q(\mathbf{x_1},\mathbf{x_2},b,c) & \wedge \forall d \exists a \exists \mathbf{w} Q_{\lambda\mu}(\mathbf{w},a,b,c,d) \end{array}$$

Now this is not, in general, equivalent to:

$$\forall \mathbf{x_1} \forall d \exists \mathbf{x_2} \exists b \exists c\, Q(\mathbf{x_1},\mathbf{x_2},b,c) \wedge \exists a \exists \mathbf{w} Q_{\lambda\mu}(\mathbf{w},a,b,c,d)$$

because, when $b = c$, that formula may be true, but the previous ones are always false. However, we claim that:

$$\forall \mathbf{x_1} \exists \mathbf{x_2} \exists b \exists c\, Q(\mathbf{x_1},\mathbf{x_2},b,c) \wedge \forall d \exists a \exists \mathbf{w} Q_{\lambda\mu}(\mathbf{w},a,b,c,d)$$

is equivalent to

$$\forall \mathbf{x_1} \forall d \forall d' \exists \mathbf{x_2} \exists b \exists c\, Q(\mathbf{x_1},\mathbf{x_2},b,c) \wedge \exists a \exists \mathbf{w} Q_{\lambda\mu}(\mathbf{w},a,b,c,d) \wedge \exists a' \exists \mathbf{w} Q_{\lambda\mu}(\mathbf{w},a',b,c,d')$$

which is an input for $\Pi_2\text{-}\mathrm{SAT_{NC}}(\mathcal{A})$. It remains for us to prove this equivalence.

(forwards.) This direction is trivial. For each given $\mathbf{x_1}$ in both formulae: any $b,c,\mathbf{x_2}$ that witness the first formula will also witness the second .

(backwards.) For each given $\mathbf{x_1}$ in both formulae: if $d \neq d'$, it follows that any true valuation of the second formula has $b \neq c$. This ensures that, if the second formula is true, that the first formula will also be, witnessed by some $b \neq c$. □

Theorem 48. Π_2-$\textsc{Sat}_{\text{NC}}(\mathcal{A})$ *is tractable if* $R^{\mathcal{A}}$ *is horn, dual horn, bijunctive, or affine, and is* Π_2^{P}*-complete otherwise.*

Proof. We know Π_2-$\textsc{Sat}_{\text{C}}$ has the proposed dichotomy. Trivially, the tractability of Π_2-$\textsc{Sat}_{\text{C}}(\mathcal{A})$ implies the tractability of Π_2-$\textsc{Sat}_{\text{NC}}(\mathcal{A})$. Furthermore, we have proved, for $R^{\mathcal{A}}$ outside the listed classes, that Π_2-$\textsc{Sat}_{\text{C}}(\mathcal{A})$ polynomially reduces to the Pspace-complete Π_2-$\textsc{Sat}_{\text{NC}}(\mathcal{A})$. The result follows. □

Corollary. *For* $i > 2$, Π_i-$\textsc{Sat}_{\text{NC}}$ *and* Σ_i-$\textsc{Sat}_{\text{NC}}$ *exhibit the same dichotomy as* Π_2-$\textsc{Sat}_{\text{NC}}$ *and* $\text{Q}\textsc{Sat}_{\text{NC}}$.

Proof. Let $j > 1$. Our manipulation of the innermost universal quantifiers in the pertinent Π_3 or Π_4 formulae, such that we build equivalent ones in Π_2, will clearly also work on Π_{2j+1} or Π_{2j+2} (resp. Σ_{2j+1} or Σ_{2j}) formulae to obtain equivalent ones in Π_{2j} (resp. Σ_{2j-1}). Consequently, our proof is equally valid for these problems. □

Remark. We are left with the class of problems Σ_2-$\textsc{Sat}_{\text{NC}}$. As noted before, these are in NP, and they exhibit the same dichotomy as $\textsc{Sat}_{\text{NC}}$.

Remark. A similar proof to this dichotomy theorem appears in [27]. The result is also inferred in [18].

Some recent work has been undertaken in alternation-bounded QCSP, by Chen [12]. He studies certain templates for which the complexity of the problem collapses to co-NP-completeness for all levels of the polynomial hierarchy above or equal to Π_2.

Chapter 7

Quantified Constraints on Graphs

7.1 Introduction

The *uniform constraint satisfaction problem*, as used in Artificial Intelligence, is usually defined as follows (see *e.g.* [32]).

- Input: a finite set of variables A, a finite domain of values T, and a set of constraints $\{C(S_1),\ldots,C(S_c)\}$ where each S_i is an a_i-tuple of (not necessarily distinct) variables from A and each $C(S_i)$ is an a_i-ary relation over T.

- Question: is there an assignment to the variables over the domain that mutually satisfies all of the constraints?

It is clear that T, together with the relations $S_1,\ldots,S_c$, is a first-order structure $\mathcal{T}$ (over some signature σ of the form $\langle S_1^{a_1},\ldots,S_c^{a_c}\rangle$). It is also clear that the question we are posing of this structure concerns the existence of a simultaneous solution to a conjunction of atomic relational constraints. Therefore, we will prefer to use the following formulation of the uniform CSP (see *e.g.* [5]).

- Input: a structure $\mathcal{T}$ and a sentence $\varphi = \exists \mathbf{x}\ Q(\mathbf{x})$, where Q is a conjunction of positive atoms.

- Question: does $\mathcal{T} \models \varphi$?

In this thesis we will be concerned only with the *non-uniform* variant of CSP, which is a family of problems parameterised by the template $\mathcal{T}$. Thus, for each template $\mathcal{T}$, $\mathrm{CSP}(\mathcal{T})$ is the decision problem with:

- Input: a sentence $\varphi = \exists \mathbf{x}\ Q(\mathbf{x})$, where Q is a conjunction of positive atoms.

- Question: does $\mathcal{T} \models \varphi$?

Note that the well-known NP-complete problems 3-SAT (satisfiability of a formula in conjunctive normal form with exactly three literals per clause) and 3-COL (graph 3-colourability) correspond to constraint satisfaction problems. The problem 3-QSAT is a popular generalisation of 3-SAT to quantified formulae, which is Pspace-complete. In this context, it makes sense to generalise constraint satisfaction problems to *quantified constraint satisfaction problems*.

Definition ([5])**.** The *non-uniform quantified constraint satisfaction problem* with template $\mathcal{T}$, denoted by $\mathrm{QCSP}(\mathcal{T})$, is the decision problem with:

- Input: a sentence ψ of the form

$$\forall \mathbf{x_1} \exists \mathbf{x_2} \forall \mathbf{x_3} \exists \mathbf{x_4} \ldots \forall \mathbf{x_{2n+1}} \exists \mathbf{x_{2n+2}} Q(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{2n+2}})$$

  (for some $n \geq 1$), where Q is a conjunction of positive atoms.

- Question: does $\mathcal{T} \models \psi$?

3-QSAT is easily recast as a QCSP, but we will be more interested in a variant problem. Let $\mathcal{B}^3_{\mathrm{NAE}}$ be the boolean structure with a single ternary not-all-equal relation

$$\mathrm{NAE}^3 = \{(0,0,1),(0,1,0),(1,0,0),(1,1,0),(1,0,1),(0,1,1)\}$$

The problem $\mathrm{QCSP}(\mathcal{B}^3_{\mathrm{NAE}})$ is known to be Pspace-complete [38].

Much effort has gone into identifying the $\mathcal{T}$ for which $\mathrm{CSP}(\mathcal{T})$ is tractable (*e.g.* [32, 34]) and NP-complete (*e.g.* [33]) . It has been conjectured in [19] that $\mathrm{CSP}(\mathcal{T})$ is always either tractable, or NP-complete. (Indeed, it has even been conjectured in [8] where this separation lies.) However, the grand classification into dichotomy remains incomplete. Some partial results are known: many years ago Schaefer proved the dichotomy for $\mathcal{T}$ ranging over boolean domains [38]. That was recently extended to domains of size 3, through methods of universal algebra, by Bulatov in [7]. Of greater interest to us is the dichotomy theorem for undirected, antireflexive graphs of Hell and Nešetřil. They prove in [25] that an undirected template $\mathcal{T}$ gives rise to a $\mathrm{CSP}(\mathcal{T})$ that is tractable, if $\mathcal{T}$ is bipartite, and a $\mathrm{CSP}(\mathcal{T})$ that is NP-complete otherwise. This dichotomy extends trivially to all undirected graphs, since templates with self-loops will give rise to a trivial CSP. Bang-Jenson, Hell and MacGillavray prove a similar dichotomy theorem for tournament templates in [4]. Specifically, they prove that $\mathrm{CSP}(\mathcal{T})$ is tractable, if $\mathcal{T}$ is a tournament with at most one (directed) cycle, and that $\mathrm{CSP}(\mathcal{T})$ is NP-complete, if $\mathcal{T}$ is any other tournament. Both of these graph dichotomy results are proved by non-constructive means.

Following on from Schaefer's work [38], Dalmau [15] and Creignou et al. [14] eventually proved a dichotomy (tractable or Pspace-complete) for QCSP on boolean domains. A trichotomy (tractable, NP-complete or Pspace-complete) has been proved for QCSP on templates where all graphs of permutations appear as relations [5]. A significant body of tractability results has been established for QCSP, largely along the same lines as for CSP, in [5, 11]. However, so far, no overarching polychotomy for QCSP has been conjectured.

It is well known from work by Chandra and Merlin [9], on the problem of *Conjunctive Query Containment* from database theory, that (existential positive) conjunctive queries are directly related to the existence of homomorphism between structures. Defining constraint satisfaction problems in terms of structure homomorphism became popular after the seminal paper by Feder and Vardi [19]. The *non-uniform homomorphism problem* with template $\mathcal{T}$, denoted $\textsc{Hom}(\mathcal{T})$, is the decision problem with:

- Input: a structure $\mathcal{A}$.
- Question: does $\mathcal{A} \xrightarrow{h} \mathcal{T}$?

If $\mathcal{K}_3$ is the 3-clique, then it is clear that $\textsc{Hom}(\mathcal{K}_3)$ is the problem of graph 3-colourability.

We define the *canonical query* $\varphi_{\mathcal{A}}$ associated with $\mathcal{A}$ to be the existential quantification of the conjunction of the facts of $\mathcal{A}$. For example, $\mathcal{K}_3$ has the canonical query

$$\varphi_{K_3} := \exists x \exists y \exists z\, E(x,y) \wedge E(y,x) \wedge E(y,z) \wedge E(z,y) \wedge E(z,x) \wedge E(x,z).$$

$\textsc{Hom}(\mathcal{T})$ and $\mathrm{CSP}(\mathcal{T})$ are essentially two views of the same problem. Specifically, they are equivalent under the bijective (up to structural isomorphism and labelling of variables) reduction $r(\mathcal{A}) = \varphi_{\mathcal{A}}$, *i.e.* $\mathcal{A} \in \textsc{Hom}(\mathcal{T})$ iff $\varphi_{\mathcal{A}} \in \mathrm{CSP}(\mathcal{T})$.

In this chapter, we introduce a new problem $\textsc{Alt-Hom}(\mathcal{T})$, which is to $\mathrm{QCSP}(\mathcal{T})$ what $\textsc{Hom}(\mathcal{T})$ is to $\mathrm{CSP}(\mathcal{T})$. It is defined in terms of *alternating-homomorphism* from a partitioned structure to a non-partitioned template. We also give a characterisation of this problem through the existence of winning strategies in a certain game. Such a method has been used independently by Chen (*e.g.* in [11, 10, 12]).

7.2 Preliminaries

7.2.1 Structures and Logic.

We consider only finite, non-empty structures. Let $\mathcal{A}$ and $\mathcal{T}$ be such structures over σ. We denote the universe, or domain, of $\mathcal{A}$ by $|\mathcal{A}|$, and the cardinality of $|\mathcal{A}|$ by $||\mathcal{A}||$. For each relation R_i of σ, with arity a_i, $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$ is the interpretation of R_i over $\mathcal{A}$. When it does not lead to confusion we may be sloppy in identifying

R_i and $R_i^{\mathcal{A}}$. A structure $\mathcal{A}$ is *connected* if, and only if, it is *not* the disjoint union of some structures $\mathcal{A}'$ and $\mathcal{A}''$. An *isolated element* of a structure $\mathcal{A}$ is one that does not appear in any tuple of any relation of $\mathcal{A}$.

A homomorphism from $\mathcal{A}$ to $\mathcal{T}$ is a function $h : |\mathcal{A}| \to |\mathcal{T}|$ such that, for all relations R_i of σ, with arity a_i, and for all $(x_1, \ldots, x_{a_i}) \in |\mathcal{A}|^{a_i}$, we have that $R_i^{\mathcal{A}}(x_1, \ldots, x_{a_i})$ implies $R_i^{\mathcal{T}}(h(x_1), \ldots, h(x_{a_i}))$. If there exists a homomorphism from $\mathcal{A}$ to $\mathcal{T}$, then we write $\mathcal{A} \xrightarrow{h} \mathcal{T}$. If we have both $\mathcal{A} \xrightarrow{h} \mathcal{T}$ and $\mathcal{T} \xrightarrow{h} \mathcal{A}$ then we describe $\mathcal{A}$ and $\mathcal{T}$ as homomorphically equivalent.

A quantifier-free first-order formula Q is *positive conjunctive* if it is a conjunction of positive atoms, *i.e.* of the form,

$$Q(\mathbf{x}) = R_{i_1}(\mathbf{x_1}) \wedge R_{i_2}(\mathbf{x_2}) \wedge \ldots \wedge R_{i_n}(\mathbf{x_n}),$$

where, for every $1 \leq j \leq n$, R_{i_j} is a relational symbol from σ, and $\mathbf{x_j}$ is a tuple of variables of suitable length (*i.e.*, of the same length as the arity of R_{i_j}). Note that a variable may occur more than once in a given tuple.

7.2.2 Alternating-homomorphism problems.

For $n \in \mathbb{N}$, let $\kappa_n = \{U_1, E_2, U_3, E_4, \ldots, U_{2n+1}, E_{2n+2}\}$ be a set of unary symbols that do not occur in σ. Define an *n-partitioned structure* $\mathfrak{P}$ over σ to be a finite structure over the signature $\sigma \cup \kappa_n$, such that the interpretation of the symbols from κ_n is a partition of the structure: *i.e.,*

- $|\mathfrak{P}| = \bigcup_{i=0}^{n}(|U_{2i+1}| \cup |E_{2i+2}|)$; and,
- for any $0 \leq i < j \leq n$, the sets U_{2i+1}, E_{2i+2}, U_{2j+1} and E_{2j+2} are pairwise disjoint.

We write $\mathcal{S}_{\mathfrak{P}}$ to denote the σ-structure underlying $\mathfrak{P}$. We write $\mathfrak{P}\restriction_{U_i}$ (respectively, $\mathfrak{P}\restriction_{E_i}$) to denote the substructure of $\mathcal{S}_{\mathfrak{P}}$ induced by U_i (respectively, E_i). When this does not cause confusion, we write U_i (respectively, E_i) for the sake of brevity.

We say there is an *alternating-homomorphism* from the n-partitioned structure $\mathfrak{P}$ over σ to the (non-partitioned) σ-structure $\mathcal{T}$, and we write $\mathfrak{P} \xrightarrow{alt} \mathcal{T}$ if, and only if,

- for all functions $f_{U_1} : U_1 \to |\mathcal{T}|$,
- there exists a function $f_{E_2} : E_2 \to |\mathcal{T}|$, such that,

  $\vdots$

- for all functions $f_{U_{2n+1}} : U_{2n+1} \to |\mathcal{T}|$,

- there exists a function $f_{E_{2n+2}} : E_{2n+2} \to |\mathcal{T}|$, such that,
- $f_{U_1} \cup f_{E_2} \cup \dots f_{U_{2n+1}} \cup f_{E_{2n+2}}$ is a homomorphism from $\mathcal{S}_{\mathfrak{P}}$ to $\mathcal{T}$.

A *partitioned structure* is one that is n-partitioned, for some n.

Definition (Alternating-homomorphism problem)**.** The *non-uniform alternating-homomorphism problem* with template $\mathcal{T}$, denoted by ALT-HOM$(\mathcal{T})$, is the decision problem with:

- Input: a partitioned structure $\mathfrak{P}$.
- Question: does $\mathfrak{P} \xrightarrow{alt} \mathcal{T}$?

Examples. Consider the graph $\mathcal{G}$ with vertices $\{a,b,c,d\}$ and edge set $\{(a,b),(b,a),(c,d),(d,c)\}$. We define three partitioned structures which have $\mathcal{G}$ as their underlying graph:

- $\mathfrak{P}_1$ such that $U_1 = \{a\}, E_2 = \{b\}, U_3 = \{c\}, E_4 = \{d\}$.
- $\mathfrak{P}_2$ such that $U_1 = \{a\}, E_2 = \{b\}, U_1 = \{c\}, E_2 = \{d\}$.
- $\mathfrak{P}_3$ such that $U_1 = \{a\}, E_8 = \{b\}, U_1 = \{c\}, E_{10} = \{d\}$.

These partitioned structures are depicted in Figure 7.1.

The above partitioned structures are equivalent in the sense that, for any structure $\mathcal{T}$, if for any one of them there exists an alternating-homomorphism to $\mathcal{T}$, then there exists also an alternating-homomorphism to $\mathcal{T}$ from the others. This leads us to define the following *rewrite* scheme to transform an n-partitioned structure $\mathfrak{P}$ to a *rewrite-reduced* partitioned structure, denoted $\overline{\mathfrak{P}}$.

1. If all relations of $\mathcal{S}_{\mathfrak{P}}$ are empty, *i.e.* all elements of $\mathcal{S}_{\mathfrak{P}}$ are isolated, then set $\overline{\mathfrak{P}}$ to be the singleton with $|\mathcal{S}_{\overline{\mathfrak{P}}}| = \{0\}$, all relations of $\mathcal{S}_{\overline{\mathfrak{P}}}$ empty, and $E2 = \{0\}$. *Otherwise*:
2. Remove all isolated elements of $\mathfrak{P}$.
3. Suppose $\mathfrak{P}$ is the disjoint union of m connected substructures $\mathfrak{P}_1, \dots, \mathfrak{P}_m$. For each $1 \le l \le m$, construct $\overline{\mathfrak{P}}_l$ from $\mathfrak{P}_l$ thus:
   (a) while there is a minimal $i \le n$ such that U_{2i+1} is empty, move every element of E_{2j+2} into E_{2j}, for all $i \le j \le n$ and every element of U_{2j+3} into U_{2j+1}, for all $i \le j \le n-1$.

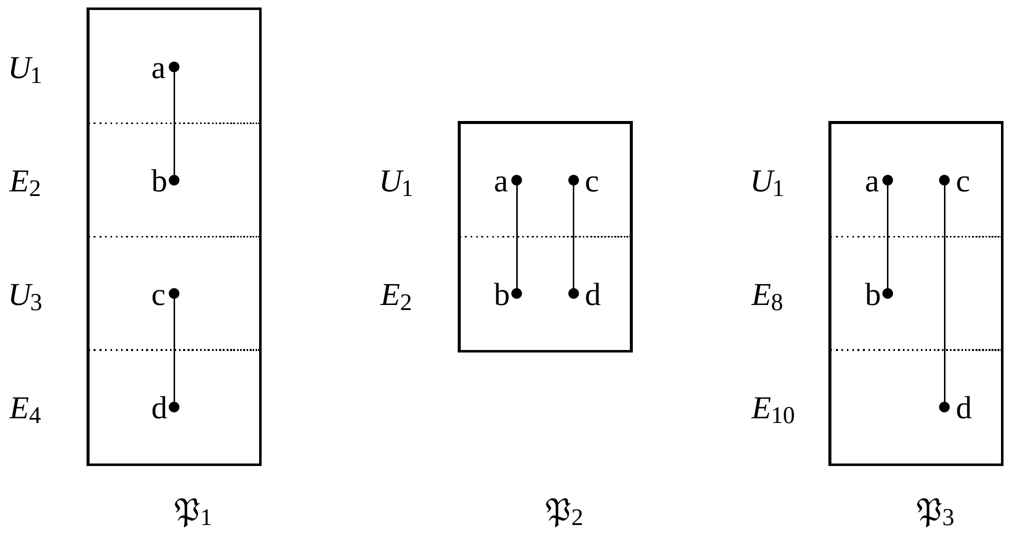


Figure 7.1: Three Partitioned Graphs.

  (b) while there is a minimal $i \leq n$ such that E_{2i} is empty, move every element of E_{2j+2} into E_{2j}, for all $i \leq j \leq n$ and every element of U_{2j+1} into U_{2j-1}, for all $i \leq j \leq n$.

4. Set $\overline{\mathfrak{P}}$ to be $\overline{\mathfrak{P}}_1 \uplus \ldots \uplus \overline{\mathfrak{P}}_m$.

5. Remove as many empty partitions as possible, so obtaining an n'-partitioned structure (for some $n' \leq n$).

The rewrite scheme is deterministic, up to the order in which the connected substructures are considered, and so $\overline{\mathfrak{P}}$ is well-defined. We say that two partitioned structures $\mathfrak{P}$ and $\mathfrak{P}'$ are *rewrite-equivalent* if $\overline{\mathfrak{P}} = \overline{\mathfrak{P}}'$. In figure 7.1, the structures $\mathfrak{P}_1$, $\mathfrak{P}_2$, and $\mathfrak{P}_3$ are all rewrite equivalent, and $\mathfrak{P}_2$ is rewrite-reduced. Note that, for any $\mathfrak{P}$, we can compute its rewrite-reduced $\overline{\mathfrak{P}}$ in polynomial time.

Two partitioned structures $\mathfrak{P}$ and $\mathfrak{P}'$ are said to be *problem-equivalent* if, for all templates $\mathcal{T}$, we have $\mathfrak{P} \in \text{ALT-HOM}(\mathcal{T})$ iff $\mathfrak{P}' \in \text{ALT-HOM}(\mathcal{T})$.

Proposition 49. *Let $\mathfrak{P}$ and $\mathfrak{P}'$ be two partitioned structures. If $\mathfrak{P}$ and $\mathfrak{P}'$ are rewrite-equivalent then they are problem-equivalent.*

Proof. It is easy to see that the rewrite rules preserve the existence of alternating-homomorphism. □

Note the converse does not hold as the following example shows.

Example. Consider the 1-partitioned digraphs:

- $\mathfrak{P}_4$: with domain $\{x,y,z\}$, edge set $\{(x,y),(z,y)\}$, and partitions $U_1 = \{x\}$, $E_2 = \{y,z\}$.
- $\mathfrak{P}_5$: with domain $\{x,y\}$, edge set $\{(x,y)\}$, and partitions $U_1 = \{x\}$, $E_2 = \{y\}$.

Whilst they are not rewrite equivalent, they are problem equivalent. $\mathfrak{P}_4$ is equivalent to the sentence $\forall x \exists y \exists z E(x,y) \wedge E(y,z)$ and $\mathfrak{P}_5$ is equivalent to the sentence $\forall x \exists y E(x,y)$. Both sentences have the same class of finite models.

We note, for any $\mathfrak{P}$ and its rewrite-reduced $\overline{\mathfrak{P}}$, that their underlying structures $\mathcal{S}_{\mathfrak{P}}$ and $\mathcal{S}_{\overline{\mathfrak{P}}}$ differ by possibly only some isolated elements. $\mathcal{S}_{\mathfrak{P}}$ and $\mathcal{S}_{\overline{\mathfrak{P}}}$ are homomorphically equivalent, and, for all templates $\mathcal{T}$, $\mathcal{S}_{\mathfrak{P}} \xrightarrow{h} \mathcal{T}$ iff $\mathcal{S}_{\overline{\mathfrak{P}}} \xrightarrow{h} \mathcal{T}$.

7.2.3 QCSP versus ALT-HOM.

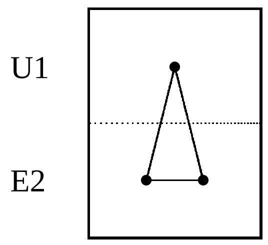
In this section, we show that QCSP and ALT-HOM are essentially the same problem.

Theorem 50. *Let $\mathcal{T}$ be a finite σ-structure. The problems* QCSP$(\mathcal{T})$ *and* ALT-HOM$(\mathcal{T})$ *are equivalent under logspace reduction.*

Proof. We will modify the bijective reduction $r(\mathcal{A}) = \varphi_{\mathcal{A}}$, mapping a structure to its canonical query, that proved the equivalence of HOM and CSP in the introduction to this chapter. From r we build the function s from partitioned structures to prenex quantified formulae whose quantifier-free part is positive conjunctive. Given a partitioned structure $\mathfrak{P}$, consider the canonical query $\varphi_{\mathcal{S}_{\mathfrak{P}}}$ of its underlying structure $\mathcal{S}_{\mathfrak{P}}$. Given this existential query $\varphi_{\mathcal{S}_{\mathfrak{P}}}$, we produce the query $\varphi_{\mathfrak{P}}$ by replacing all instances of $\exists x$, for variables x that correspond to elements of $\mathfrak{P}$ in a universal partition, by $\forall x$. The map $s(\mathfrak{P}) = \varphi_{\mathfrak{P}}$ is bijective (up to isomorphism of the rewrite-reduced $\overline{\mathfrak{P}}$ and labelling of variables) and, along with its inverse, may be computed in logarithmic space. It follows directly from the definitions that s and s^{-1} are reductions between QCSP$(\mathcal{T})$ and ALT-HOM$(\mathcal{T})$. $\square$

Just as we refer to the canonical query of a non-partitioned structure, so we will refer to the canonical query of a partitioned structure as being the sentence it reduces to, as in the previous theorem.

Example. Let $\mathcal{T}$ be any graph. The partitioned structure and sentence of Figure 7.2 give rise to equivalent instances of, respectively, ALT-HOM$(\mathcal{T})$ and QCSP$(\mathcal{T})$, that reduce to one another in logarithmic space. The sentence is the canonical query of the partitioned structure, as just defined.



$$\forall x \exists y \exists z \; E(x,y) \wedge E(y,x) \wedge E(y,z) \wedge E(z,y) \wedge E(z,x) \wedge E(x,z)$$

Figure 7.2: Canonical sentence of a Partitioned Structure.

7.2.4 Alternating-homomorphisms as winning strategies.

We give a game characterisation of QCSP. The game we are about to define corresponds exactly to a standard model-checking game, also known as a Hintikka game [20]. We define this game in order to use the game parlance in subsequent proofs.

Definition (Game for QCSP)**.** Let $\mathfrak{P}$ be an n-partitioned structure and $\mathcal{T}$ a (non-partitioned) template. The $(\mathfrak{P},\mathcal{T})$-*game* goes as follows. Opponent plays on the universal partitions and Proponent plays on the existential partitions. They play alternate partitions, in ascending order, until all the partitions have been played. For $0 \leq i \leq n$:

- $\underline{\text{Opponent}}$ (U-move): for every element in partition U_{2i+1}, Opponent chooses an element in $\mathcal{T}$, that is, Opponent gives a function $opp_{2i+1} : U_{2i+1} \to |\mathcal{T}|$.
- $\underline{\text{Proponent}}$ (E-move): for every element in partition E_{2i+2}, Proponent chooses an element in $\mathcal{T}$, that is, Proponent gives a function $pro_{2i+2} : E_{2i+2} \to |\mathcal{T}|$.

If, at any stage of the game, the function defined by the union of the moves of both players, $opp_1 \cup pro_2 \cup opp_3 \cup \ldots$, is not a partial homomorphism from $\mathcal{S}_{\mathfrak{P}}$ to $\mathcal{T}$, then Opponent wins. Otherwise this finite game will finish with some homomorphism from $\mathcal{S}_{\mathfrak{P}}$ to $\mathcal{T}$ having been constructed, and Proponent wins. It is Proponent's aim to construct such a homomorphism, and it is Opponent's aim to stop her. (In deference to the conventions of Ehrenfeucht-Fraisse games, Proponent is considered female, and Opponent male.) Note that, if at some point the partial function defined by the play can not be extended to a homomorphism no matter how either side plays, then Opponent must necessarily win the game.

A strategy specifies how Opponent or Proponent are to play, given what has been played before:

- (Proponent) A strategy for partition E_{2i+2} in the $(\mathfrak{P},\mathcal{T})$-game is a function $\sigma_{E_{2i+2}} : E_{2i+2} \times \Pi_{\lambda<i}(E_{2\lambda+2} \times \mathcal{T}) \times (U_{2\lambda+1} \times \mathcal{T}) \rightarrow \mathcal{T}$.
- A strategy σ for Proponent in the $(\mathfrak{P},\mathcal{T})$-game is the union of her strategies for all the existential partitions, viz $\bigcup_{\lambda\leq n} \sigma_{E_{2\lambda+2}}$.
- (Opponent) A strategy for partition U_{2i+1} in the $(\mathfrak{P},\mathcal{T})$-game is a function $\tau_{U_{2i+1}} : U_{2i+1} \times \Pi_{\lambda<i}(E_{2\lambda} \times \mathcal{T}) \times (U_{2\lambda+1} \times \mathcal{T}) \rightarrow \mathcal{T}$.
- A strategy τ for Opponent in the $(\mathfrak{P},\mathcal{T})$-game is the union of his strategies for all the universal partitions, viz $\bigcup_{\lambda\leq n} \tau_{U_{2\lambda+1}}$.

A *winning strategy* for Proponent is a strategy σ that beats all Opponent strategies τ.

Theorem 51. *Let $\mathfrak{P}$ be a partitioned structure, and $\varphi_{\mathfrak{P}}$ the corresponding canonical query. The following are equivalent.*

(i) $\mathfrak{P} \xrightarrow{alt} \mathcal{T}$.

(ii) Proponent has a winning strategy in the $(\mathfrak{P},\mathcal{T})$-game.

(iii) $\mathcal{T} \models \varphi_{\mathfrak{P}}$.

Proof. The equivalence of (i) and (iii) follows from Theorem 50. It is well known that Proponent has a winning strategy in the **FO**-model-checking game on $(\varphi_{\mathfrak{P}},\mathcal{T})$ if, and only if, $\mathcal{T} \models \varphi_{\mathfrak{P}}$ [20]. The game we define is the model-checking game restricted to sentences in prenex form whose quantifier-free part is positive conjunctive. The equivalence of (ii) and (iii) follows. ☐

7.2.5 Graphs

A digraph is a structure over the signature containing a single binary relation E. An undirected graph is one whose edge relation is symmetric.

Definitions.

(Cliques.) Let $n \geq 1$. Let $\mathcal{K}_n$ be the *(antireflexive) n-clique*, that is the graph with vertices $\{0,1,\ldots,n-1\}$ such that *all distinct* vertices are adjacent. Let $\mathcal{K}_n^{ref}$

be the *reflexive n-clique*, that is the graph with vertices $\{0,1,\ldots,n-1\}$ such that *all* vertices are adjacent (when $i = j$, we call the corresponding edge a *self-loop*).

(Paths.) Let $\mathcal{P}_n$ be the undirected antireflexive n-path, *i.e.* with vertices $\{0,1,\ldots,n-1\}$ such that $E(i,j)$ iff $j = i+1$ or $j = i-1$. It follows that the 1-path $\mathcal{P}_1$ is $\mathcal{K}_1$ and the 2-path $\mathcal{P}_2$ is $\mathcal{K}_2$.

(Cycles.) Let $\mathcal{DC}_n$ be the directed antireflexive n-cycle, *i.e.* with vertices $\{0,1,\ldots,n-1\}$ such that $E(i,j)$ iff $j = i+1 \bmod n$. Let $\mathcal{C}_n$ be the undirected antireflexive n-cycle, *i.e.* the symmetric closure of $\mathcal{DC}_n$. It follows that $\mathcal{C}_3$ is the 3-clique $\mathcal{K}_3$.

It is proved in [5] that, for $n \geq 3$, $\mathrm{QCSP}(\mathcal{K}_n)$ is Pspace-complete. It follows immediately that $\textsc{Alt-Hom}(\mathcal{K}_n)$ is also Pspace-complete, for $n \geq 3$.

An induced sub-digraph $\mathcal{G}' \subseteq \mathcal{G}$ is a *retract* of $\mathcal{G}$ iff there is a homomorphism $h : \mathcal{G} \to \mathcal{G}$ s.t. $\mathcal{G}'$ is the image of h. A graph $\mathcal{G}'$ is a *core* if it contains no proper retracts. For an arbitrary digraph $\mathcal{G}$, we define a *core of* $\mathcal{G}$ to be any minimal (w.r.t. size) retract that is itself a core. It is well-documented (*e.g.* [26]) that the core of a digraph is unique, up to isomorphism.

A bipartite graph is an undirected graph that is 2-colourable. A graph is bipartite iff it has either $\mathcal{K}_1$ or $\mathcal{K}_2$ as its core. Note that bipartite graphs are antireflexive.

7.3 Basic graph results

Most of these results are given for digraphs; we will specifically consider undirected graphs in the next section. When we discuss edges *between* vertices x and y, these may be oriented either way, or not at all (double edges).

7.3.1 Restricting partitions

We define restrictions on the input partitioned structure which will ultimately lead to tractability.

Definitions (restricting partitions)**.** Let $\mathfrak{P}$ be a partitioned digraph. We say that $\mathfrak{P}$ is in Σ_1-*form* (respectively, Π_2-*form*), if the only non-empty partition is E_2 (respectively, if the only non-empty partitions are among $\{U_1, E_2\}$). If $\mathfrak{P}$ is in in Π_2-form and there is at most one vertex in U_1, then we say that $\mathfrak{P}$ is in Π_2-*fan form*. If, moreover, the vertex $x \in U_1$ exists, and is adjacent to some vertex $y \in E_2$, then we say that $\mathfrak{P}$ is in *strict* Π_2-*fan form*. Finally, we say that $\mathfrak{P}$ is in Π_2-*multifan form*, if $\mathfrak{P}$ is the finite disjoint union of structures in Π_2-fan form.

Note that, if $\mathfrak{P}$ is in Σ_1-form, then $\mathfrak{P}$ is *a fortiori* in Π_2-fan form. Any $\mathfrak{P}$ in Π_2-fan form, but not in strict Π_2-fan form, has a rewrite-reduced $\overline{\mathfrak{P}}$ in Σ_1-form.

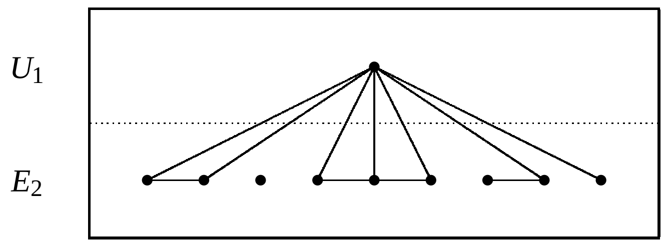


Figure 7.3: A Partitioned Graph in strict Π_2-fan form.

Proposition 52 (Π_2-multifan form)**.** *Let $\mathcal{T}$ be a digraph. The restriction of* $\text{ALT-HOM}(\mathcal{T})$ *to inputs in Π_2-multifan form is* NP*-complete, whenever* $\text{HOM}(\mathcal{T})$ *is* NP*-complete.*

Proof. Let $\mathfrak{P}$ be the disjoint union of $\mathfrak{P}_1, \ldots, \mathfrak{P}_m$ all in Π_2-fan form. Note that $\mathfrak{P} \in \text{ALT-HOM}(\mathcal{T})$ iff $\mathfrak{P}_i \in \text{ALT-HOM}(\mathcal{T})$, for $1 \leq i \leq m$.

(Membership of NP) For each $1 \leq i \leq m$, if $\mathfrak{P}_i$ is not in strict Π_2-fan form, then it is equivalent to $\overline{\mathfrak{P}_i}$ in Σ_1-form, and we may simply guess a homomorphism and verify in polynomial time. If $\mathfrak{P}_i$ is in strict Π_2-fan form, we test all possible maps for the single element in U_1, guessing the rest of the homomorphism and verifying in polynomial time.

(NP-hardness) If $\mathfrak{P}$ is in Σ_1-form, then $\mathfrak{P} \in \text{ALT-HOM}(\mathcal{T})$ if, and only if, $\mathcal{S}_{\mathfrak{P}} \in \text{HOM}(\mathcal{T})$. Hence, $\text{ALT-HOM}(\mathcal{T})$ is NP-hard provided that $\text{HOM}(\mathcal{T})$ is NP-complete.

$\square$

We will find, for a wide range of templates $\mathcal{T}$, that every input which is not in Π_2-multifan form can be discarded. This will be because inputs $\mathfrak{P}$ not in Π_2-multifan form are either easily seen to be no-instances of $\text{ALT-HOM}(\mathcal{T})$, or to be equivalent to the rewrite-reduced $\overline{\mathfrak{P}}$ which *is* in Π_2-multifan form. Further, we will find that we can split up inputs $\overline{\mathfrak{P}}$ in Π_2-multifan form into their constituent Π_2-fan components (as in the previous proof). Thus, structures in Π_2-fan form are central to our discourse. Such a structure appears in Figure 7.3.

Remark. The 'converse' of Theorem 52, that the restriction of $\text{ALT-HOM}(\mathcal{T})$ to Π_2-multifan form being NP-complete implies $\text{HOM}(\mathcal{T})$ is NP-complete, does not in general hold. For example, take $\mathcal{T}$ to be (the disjoint union) $\mathcal{K}_3 \uplus \mathcal{K}_1^{ref}$. The self-loop $\mathcal{K}_1^{ref}$ makes $\text{HOM}(\mathcal{K}_3 \uplus \mathcal{K}_1^{ref})$ trivial (every instance is a yes-instance). However:

Proposition 53. *The restriction of* $\text{ALT-HOM}(\mathcal{K}_3 \uplus \mathcal{K}_1^{ref})$ *to Π_2-multifan form is* NP*-complete.*

Proof. Membership of NP follows as in the first part of the proof to the previous proposition. For completeness, we give a reduction from the NP-complete 3-colourability problem $\textsc{Hom}(\mathcal{K}_3)$.

Let $\mathcal{G}$ be an input for the problem $\textsc{Hom}(\mathcal{K}_3)$. Let $\mathcal{G}_1, \ldots, \mathcal{G}_m$ be the connected components of $\mathcal{G}$ and let $x_1, \ldots, x_m$ be some sequence of vertices in these respective components.

We construct an input for $\textsc{Alt-Hom}(\mathcal{K}_3 \uplus \mathcal{K}_1^{ref})$ thus:

- For each component $\mathcal{G}_i$, construct a partitioned graph $\mathfrak{P}_i$ (in Π_2-fan form) whose underlying graph has vertices $|\mathcal{G}_i| \uplus \{y_i\}$ (where y_i is a new vertex) and edge set $E^{\mathcal{G}_i} \uplus \{(x_i,y_i),(y_i,x_i)\}$, and whose partitions are $U_1 := \{y_i\}$ and $E_2 := |\mathcal{G}_i|$.
- Set $\mathfrak{P}$ to be the disjoint union $\mathfrak{P}_1 \cup \ldots \cup \mathfrak{P}_m$.

Clearly $\mathfrak{P}$ is in Π_2-multifan form: we claim that $\mathcal{G} \in \textsc{Hom}(\mathcal{K}_3)$ if, and only if, $\mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{K}_3 \uplus \mathcal{K}_1^{ref})$.

(forwards.) Suppose $\mathcal{G} \in \textsc{Hom}(\mathcal{K}_3)$. For each of the connected components $\mathcal{G}_i$ there must be a homomorphism h_i to $\mathcal{K}_3$. It suffices to show that there is a winning strategy for Proponent for each of the games $(\mathfrak{P}_i, \mathcal{K}_3 \uplus \mathcal{K}_1^{ref})$. If Opponent plays y_i to the self-loop of $\mathcal{K}_1^{ref}$, then Proponent may play the remainder of $\mathfrak{P}_i$ to the same self-loop to win. If Opponent plays y_i to one of the vertices of the triangle $\mathcal{K}_3$, then Proponent plays x_i to an adjacent vertex on the triangle, and may play the remainder of $\mathfrak{P}_i$ according to [a cyclic permutation of] the homomorphism h_i to win.

(backwards.) Suppose $\mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{K}_3 \uplus \mathcal{K}_1^{ref})$: which implies that for each $\mathfrak{P}_i$ there is a winning strategy for Proponent in the game on $(\mathfrak{P}_i, \mathcal{K}_3 \uplus \mathcal{K}_1^{ref})$. It suffices to show that this must imply the existence of a homomorphism from each $\mathcal{G}_i$ to $\mathcal{K}_3$. This is immediate, for suppose Opponent plays the y_i to some vertex in the triangle $\mathcal{K}_3$, then the remainder of $\mathfrak{P}_i$ must be played to the triangle, since $\mathfrak{P}_i$ is connected, and so the winning strategy provides the necessary homomorphism. $\square$

7.3.2 Basic results.

Proposition 54 (reflexive clique)**.** *If $\mathcal{T}$ is a reflexive clique, then* $\textsc{Alt-Hom}(\mathcal{T})$ *is trivial. Specifically:* $\mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{T})$*, for every* $\mathfrak{P}$.

Proof. If $\mathcal{T}$ is a reflexive clique, then for any $\mathfrak{P}$, all functions from $\mathcal{S}_{\mathfrak{P}}$ to $\mathcal{T}$ will be homomorphisms. $\square$

Definition (*e.g.*[6])**.** A *dominating* vertex z in a digraph $\mathcal{T}$ is one s.t. for all $w \in |\mathcal{T}|$, both $E^{\mathcal{T}}(w,z)$ and $E^{\mathcal{T}}(z,w)$ hold. (It follows that $(z,z) \in E^{\mathcal{T}}$.)

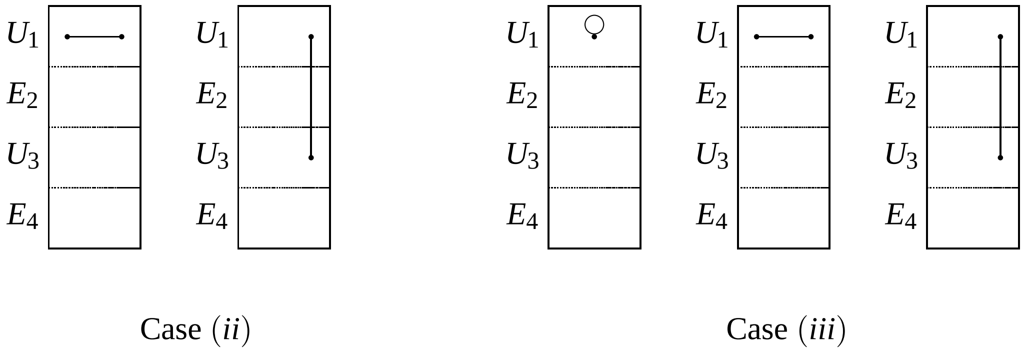


Figure 7.4: Types of forbidden edges in the last two cases of Proposition 55.

Proposition 55 (dominating vertex)**.** *Let $\mathcal{T}$ be a digraph in which there exists a dominating vertex z. Then* ALT-HOM$(\mathcal{T})$ *may be decided in logarithmic space.*

Proof. We consider three cases.

(i) If $\mathcal{T}$ is a reflexive clique, then ALT-HOM$(\mathcal{T})$ is trivial (Proposition 54).

(ii) $\mathcal{T}$ is not a reflexive clique, but is reflexive. In this case $\mathfrak{P} \in$ ALT-HOM$(\mathcal{T})$ iff $\mathfrak{P}$ has no edges between distinct vertices $x \in U_i$ and $y \in U_j$ (for any i, j, see Figure 7.4). This property can clearly be checked in logarithmic space; we prove its correctness. Let a and b be distinct vertices of $\mathcal{T}$ s.t. $\neg E^{\mathcal{T}}(a,b)$.

($\Rightarrow$) By contraposition: if $\mathfrak{P}$ has an edge between distinct vertices $x \in U_i$ and $y \in U_j$, then Opponent may play x on a and y on b to win, proving $\mathfrak{P} \notin$ ALT-HOM$(\mathcal{T})$.

($\Leftarrow$) If $\mathfrak{P}$ has no edge between vertices $x \in U_i$ and $y \in U_j$, then Proponent may follow the strategy of playing all existential vertices to the dominating vertex z. This will overcome all Opponent strategies.

(iii) $\mathcal{T}$ is not a reflexive clique, and is not reflexive. In this case $\mathfrak{P} \in$ ALT-HOM$(\mathcal{T})$ iff $\mathfrak{P}$ has no edges between (not necessarily distinct) vertices $x \in U_i$ and $y \in U_j$ (for any i, j, see Figure 7.4). The proof proceeds as in part (ii), with all instances of the word 'distinct' dropped. □

Lemma 56 (not a reflexive clique)**.** *Let $\mathcal{T}$ be a digraph that is not a reflexive clique, and let $\mathfrak{P}$ be a partitioned digraph. If there is an edge in $\mathcal{T}$ between distinct vertices $x \in U_i$ and $y \in U_j$ (for any i, j) then $\mathfrak{P}$ is a no-instance of* ALT-HOM$(\mathcal{T})$.

Proof. By assumption, there are vertices a and b in $\mathcal{T}$ that are not adjacent (a may be equal to b). Opponent plays a for x and b for y and wins. □

Lemma 57 (antireflexivity)**.** *Let $\mathcal{T}$ be an antireflexive digraph. If there is an edge in $\mathfrak{P}$ between nodes $x \in E_i$ and $y \in U_j$ (for $i < j$) then $\mathfrak{P}$ is a no-instance of* ALT-HOM$(\mathcal{T})$. *If there is an edge in $\mathfrak{P}$ between (not necessarily distinct) nodes $x \in U_i$ and $y \in U_j$ (for any i, j) then $\mathfrak{P}$ is a no-instance of* ALT-HOM$(\mathcal{T})$.

Proof. In the first case, Proponent has chosen some vertex s in $\mathcal{T}$ for x. Opponent also chooses s for y and wins. Similarly, in the second case, Opponent chooses the same vertex s for both x and y. □

Lemma 58 (isolated vertex)**.** *Let $\mathcal{T}$ be a digraph with an isolated vertex s, and let $\mathfrak{P}$ be a partitioned digraph. If there is an edge in $\mathfrak{P}$ between $x \in U_i$ and $y \in E_j$ (for any i, j), or between $x \in U_i$ and $y \in U_j$ (any i, j), then $\mathfrak{P}$ is a no-instance of* ALT-HOM$(\mathcal{T})$.

Proof. We prove the first case; the second may be done similarly. Regardless of what is played before, when Opponent plays x on s, there is no way that partial function can extend to homomorphism, regardless of where Proponent will map, or has mapped, y. □

Proposition 59 (isolated vertex)**.** *If $\mathcal{T}$ is antireflexive and has an isolated vertex, then* ALT-HOM$(\mathcal{T})$ *and* HOM$(\mathcal{T})$ *are logspace equivalent.*

Proof. The reduction of HOM$(\mathcal{T})$ to ALT-HOM$(\mathcal{T})$ is trivial.

We reduce ALT-HOM$(\mathcal{T})$ to HOM$(\mathcal{T})$ as follows. Let $\mathcal{N}$ be a fixed no-instance of HOM$(\mathcal{T})$ (say, $\mathcal{T}$ augmented with one vertex adjacent to every vertex of $\mathcal{T}$). If $\mathfrak{P}$ has an edge as in the previous lemma then we know that it is a no-instance and we reduce $\mathfrak{P}$ to $\mathcal{N}$. If $\mathfrak{P}$ has no such edge then every element in a universal partition is isolated. Thus, Opponent's moves have no bearing on Proponent's moves, and we may disregard every element occurring in a universal partition. Indeed, the rewritten-reduced graph $\overline{\mathfrak{P}}$ will be in Σ_1-form. We reduce $\mathfrak{P}$ to its underlying graph $\mathcal{S}_{\mathfrak{P}}$. □

It is important that $\mathcal{T}$ be antireflexive, to guarantee the existence of an $\mathcal{N}$ in the previous proof. The following proposition is a cousin of the previous.

Proposition 60. *If $\mathcal{T}$ has an isolated vertex, then* ALT-HOM$(\mathcal{T})$ *and* HOM$(\mathcal{T})$ *are equivalent under logspace Turing reductions.*

Proof. Again, the reduction from HOM$(\mathcal{T})$ to ALT-HOM$(\mathcal{T})$ is trivial.

We give the reduction ALT-HOM$(\mathcal{T})$ to HOM$(\mathcal{T})$. If $\mathfrak{P}$ has an edge as in the previous lemma then we reject the input. If $\mathfrak{P}$ has no such edge then we reduce it to the underlying graph $\mathcal{S}_{\mathfrak{P}}$. □

Example. The problem ALT-HOM$(\mathcal{K}_3 \uplus \mathcal{K}_1)$ is NP-complete. It is equivalent to HOM$(\mathcal{K}_3)$, which in turn is the well-known NP-complete problem 3-COL.

7.3.3 Non-connected templates.

Lemma 61 (forbidden paths and non-connected templates)**.** *Let $\mathcal{T}$ be a non- connected graph. If there is a path in $\mathfrak{P}$ between any $x \in E_i$ and $y \in U_j$ (for $i < j$), then $\mathfrak{P}$ is a no-instance of* Alt-Hom$(\mathcal{T})$. *If there is a path in $\mathfrak{P}$ between any $x \in U_i$ and $y \in U_j$ (for any i, j), then $\mathfrak{P}$ is a no-instance of* Alt-Hom$(\mathcal{T})$.

Proof. We prove the first case; the second may be done similarly. Wherever Proponent plays x, Opponent need only play y in another connected component to win. □

Lemma 62 (forbidden paths yield Π_2-multifan form)**.** *If there is no path in a partitioned digraph $\mathfrak{P}$ between any $x \in E_i$ and $y \in U_j$ (for $i < j$), or between any $x \in U_i$ and $y \in U_j$ (for any i, j), then the rewrite-reduced $\overline{\mathfrak{P}}$ will be in Π_2-multifan form.*

Proof. Consider $\mathfrak{P}$ with all isolated vertices removed, and split into disjoint connected components. It suffices to prove that each of these is in Π_2-fan form.

For any such component $\mathfrak{P}'$, let $0 < i \leq n$ be the largest integer such that U_{2i+1} is non-empty. It follows that there is an $x \in U_{2i+1}$, and that all other elements of $\mathfrak{P}'$ are in existential partitions of index at least $2i+2$, for otherwise there would be a path that violates our assumptions. The rewrite rules may be applied to move x to U_1 and all other vertices in the component to E_2. The result follows. □

Note that we can determine in polynomial time whether or not a digraph $\mathcal{T}$ has any of the paths of the previous lemma.

Lemma 63 (non-connected)**.** *Let $\mathcal{T}$ be a digraph that is not connected, then* Alt-Hom$(\mathcal{T})$ *is in* NP.

Proof. Let $\mathfrak{P}$ be a partitioned input digraph. If there are any of the paths in $\mathfrak{P}$ as in Lemma 61 then we may reject the instance. Otherwise, it follows from the previous lemma that the rewrite-reduced $\overline{\mathfrak{P}}$ is in Π_2-multifan form, and we can use the algorithm of Theorem 52. □

Proposition 64 (non-connected)**.** *Let $\mathcal{T}$ be a digraph that is not connected. If* Hom$(\mathcal{T})$ *is* NP*-complete then* Alt-Hom$(\mathcal{T})$ *is* NP*-complete.*

Proof. By Lemma 63, Alt-Hom$(\mathcal{T})$ is in NP. Hom$(\mathcal{T})$ reduces trivially to Alt-Hom$(\mathcal{T})$, and completeness follows. □

Remark. As in the remark after Theorem 52, the 'converse' of the previous proposition is not in general true: *i.e.* there are $\mathcal{T}$ such that Hom$(\mathcal{T})$ is tractable but Alt-Hom$(\mathcal{T})$ is not. For example, when $\mathcal{T} = \mathcal{K}_3 \uplus \mathcal{K}_1^{ref}$, Alt-Hom$(\mathcal{T})$ is readily seen to be NP-complete. (Membership follows from Lemma 63 and completeness follows from Proposition 53.)

7.4 Quantified $\mathcal{H}$-colouring

7.4.1 Bipartite templates.

Lemma 65 (forbidden paths and bipartite template)**.** *Let $\mathcal{H}$ be a bipartite graph. If there is a path in $\mathfrak{P}$ between any $x \in E_i$ and $y \in U_j$ (for $i < j$), then $\mathfrak{P}$ is a no-instance of* ALT-HOM$(\mathcal{H})$*. If there is a path in $\mathfrak{P}$ between any $x \in U_i$ and $y \in U_j$ (for any i, j), then $\mathfrak{P}$ is a no-instance of* ALT-HOM$(\mathcal{H})$*.*

Proof. We prove the first case; the second may be done similarly. Let a be any vertex in $\mathcal{H}$ on which Proponent plays x. If a is an isolated vertex, then Opponent wins (*cf.* proof of Lemma 58). Assume that a is not isolated. If the path in $\mathfrak{P}$ between x and y is of even length, then Opponent plays y on b, where b is adjacent to a. A winning strategy for Proponent would imply the existence of an odd cycle in $\mathcal{H}$. This contradicts the fact that $\mathcal{H}$ is bipartite, thus it follows that Opponent wins. If the path in $\mathfrak{P}$ is of odd length, then Opponent plays y on a and wins by the same argument. □

Lemma 66 (Π_2-multifan form and bipartite)**.** *Let $\mathcal{H}$ be a bipartite graph. If $\mathcal{H}$ has no isolated vertices then, for any $\mathfrak{P}$ in Π_2-multifan form, the following are equivalent:*

(i) $\mathfrak{P} \xrightarrow{alt} \mathcal{H}$

(ii) $\mathfrak{P} \xrightarrow{alt} \mathcal{K}_2$

(iii) $\mathcal{S}_{\mathfrak{P}} \xrightarrow{h} \mathcal{K}_2$

Proof. If $\mathfrak{P}$ is the disjoint union of $\mathfrak{P}_1, \ldots, \mathfrak{P}_m$ all in Π_2-fan form, recall that $\mathfrak{P} \in$ ALT-HOM$(\mathcal{T})$ iff $\mathfrak{P}_i \in$ ALT-HOM$(\mathcal{T})$, for $1 \leq i \leq m$.

For each $\mathfrak{P}_i$ in Π_2-fan form. When $\mathfrak{P}_i$ has no vertex in U_1, the result holds trivially. Otherwise, let x be the unique vertex in U_1. Again, if x is isolated then $\mathfrak{P}$ can be rewrite-reduced to $\overline{\mathfrak{P}}_i$ in Σ_1-form. So, assume that $\mathfrak{P}_i$ is in strict Π_2-fan form, and that x is adjacent to some y in E_2.

- $(i) \Rightarrow (ii)$: Given a winning strategy σ for Proponent in the $(\mathfrak{P}_i, \mathcal{H})$-game, we construct a winning strategy for Proponent in the $(\mathfrak{P}_i, \mathcal{K}_2)$-game. Suppose, w.l.o.g., that Opponent plays the x on the 1 in $\mathcal{K}_2$. All remaining moves are Proponent's. So, Proponent chooses any homomorphism $h : \mathcal{H} \to \mathcal{K}_2$, and a vertex a in $\mathcal{H}$ such that $h(a) = 1$. She then plays the rest of the vertices (all in E_2) according to the strategy $h \circ \sigma$ (where she assumes Opponent played the x to a in the oracle-game on $(\mathfrak{P}_i, \mathcal{H})$). Since h is a homomorphism, any outcome of the game on $(\mathfrak{P}_i, \mathcal{H})$ under strategy σ

that is a homomorphism will lead to an outcome of the game on $(\mathfrak{P}_i, \mathcal{K}_2)$ under strategy $h \circ \sigma$ that is a homomorphism. We know that, under strategy σ, all outcomes of $(\mathfrak{P}_i, \mathcal{H})$ are homomorphisms, so the result follows.

- $(ii) \Rightarrow (i)$: Given a winning strategy σ for Proponent in the $(\mathfrak{P}_i, \mathcal{K}_2)$-game, we construct a winning strategy for Proponent in the $(\mathfrak{P}_i, \mathcal{H})$-game. Suppose Opponent plays the x on a vertex a in $\mathcal{H}$. We know that a is not isolated, and has a distinct neighbour b. Let $h' : \mathcal{K}_2 \to \mathcal{H}$ be the homomorphism $\{(1,a),(2,b)\}$. All remaining moves are Proponent's. Proponent now plays the rest of the vertices (all in E_2) according to the strategy $h' \circ \sigma$ (where she assumes Opponent played the x to 1 in the oracle-game on $(\mathfrak{P}_i, \mathcal{K}_2)$). The argument concludes as before.

- $(ii) \Leftrightarrow (iii)$: Since each $\mathfrak{P}_i$ is in strict Π_2-fan form, the result follows immediately from the symmetry of $\mathcal{K}_2$.

□

Theorem 67 (bipartite)**.** *Let $\mathcal{H}$ be a bipartite graph. The problem* ALT-HOM$(\mathcal{H})$ *is tractable.*

Proof. We propose the following algorithm to solve ALT-HOM$(\mathcal{H})$.

The input $\mathfrak{P}$ is first scanned to check whether it has any of the forbidden paths of Lemma 65. If there are any, then the input is rejected.

If there are none of the forbidden paths and $\mathcal{H}$ has an isolated vertex, then we evaluate HOM$(\mathcal{H})$ on input $\mathcal{S}_{\mathfrak{P}}$. That this is correct follows from Proposition 59; that it is tractable follows from Hell and Nešetřil's dichotomy theorem [25].

Otherwise, if there are none of the forbidden paths and $\mathcal{H}$ has no isolated vertex, then we check whether $\mathcal{S}_{\mathfrak{P}}$ is 2-colorable, and answer accordingly. This is clearly polynomial: we prove its correctness. We know that if $\mathfrak{P}$ has none of the forbidden paths, then it is rewrite-equivalent to the reduced $\overline{\mathfrak{P}}$ in Π_2-multifan form, by Lemma 62. In particular, $\overline{\mathfrak{P}} \xrightarrow{alt} \mathcal{H}$ if, and only if, $\mathfrak{P} \xrightarrow{alt} \mathcal{H}$. By Lemma 66, we know that $\overline{\mathfrak{P}} \xrightarrow{alt} \mathcal{H}$ if, and only if, $\mathcal{S}_{\overline{\mathfrak{P}}} \xrightarrow{h} \mathcal{K}_2$. Moreover, by definition of $\overline{\mathfrak{P}}$, $\mathcal{S}_{\overline{\mathfrak{P}}}$ is the same as $\mathcal{S}_{\mathfrak{P}}$ up to possibly some isolated vertices. Hence, $S_{\overline{\mathfrak{P}}} \xrightarrow{h} K_2$ if, and only if, $\mathcal{S}_{\mathfrak{P}} \xrightarrow{h} \mathcal{K}_2$. It follows that $\mathfrak{P} \xrightarrow{alt} \mathcal{H}$ if, and only if, $\mathcal{S}_{\mathfrak{P}} \xrightarrow{h} \mathcal{K}_2$. □

7.4.2 Odd Catherine Wheels

We have already met the problem QCSP$(\mathcal{B}^3_{\text{NAE}})$, known to be Pspace-complete. Let $\mathcal{B}^n_{\text{NAE}}$ be the boolean structure with single n-ary not-all-equal relation

$$\text{NAE}^n := \{0,1\}^n - \{(0^n),(1^n)\}$$

For all $n \geq 3$, $\mathrm{QCSP}(\mathcal{B}^n_{\mathrm{NAE}})$ is Pspace-complete, by a trivial reduction from $\mathrm{QCSP}(\mathcal{B}^3_{\mathrm{NAE}})$.

Definition. We consider an undirected graph $\mathcal{W}$ to be an *odd catherine wheel* (OCW) if it is isomorphic to some graph $\mathcal{G}$ constructed as follows. For some k, take the $(2k+1)$-cycle $\mathcal{C}_{2k+1}$, together with $(2k+1)$ undirected paths $\mathcal{P}^0, \ldots, \mathcal{P}^{2k}$ (each of any finite length, where $\mathcal{K}_1$ is considered the 0-path). Construct $\mathcal{G}$ by identifying an end of each path $\mathcal{P}^i$ with vertex i of $\mathcal{C}_{2k+1}$.

As in that construction, an OCW may be given an ordering over its $(2k+1)$-cycle, which we will call a *listing*. An OCW may have up to $2.(2k+1)$ distinct listings, corresponding to orientation of the cycle, together with position of the zero (first) vertex. We will usually refer to a listing by a corresponding sequence of paths.

Definition. For an undirected graph $\mathcal{G}$, and a subset $A \subseteq |\mathcal{G}|$ define:

- $d(x,y)$ to be the length of the shortest path in $\mathcal{G}$ from x to y,
- $d(A,y) = \min\{d(x,y) : x \in A\}$, and,
- $D(\{p,q\}) = \max\{d(\{p,q\},y) : y \in |\mathcal{G}|\}$.

$D(\{p,q\})$ is minimal such that there is an m'-walk (for some $m' \leq D(\{p,q\})$) from $\{p,q\}$ to every vertex of $\mathcal{G}$. We will only be concerned with $D(\{p,q\})$ when p and q are adjacent vertices on the cycle of an OCW.

Definition. For any OCW $\mathcal{W}$, define:

$$m_{\mathcal{W}} = \min\{D(\{p,q\}) : p,q \text{ adjacent on the cycle of } \mathcal{W}\ \}$$

A *D-minimal listing* $\mathcal{P}^0, \ldots, \mathcal{P}^{2k}$ of $\mathcal{W}$ is one in which $D(\{k,k+1\}) = m_{\mathcal{W}}$.

A D-minimal listing is one in which the maximal distance from the vertices $\{k,k+1\}$ is minimised. These middle vertices k and $k+1$ will eventually play the role of TRUE and FALSE in a reduction from $\mathrm{QCSP}(\mathcal{B}^{2k+1}_{\mathrm{NAE}})$. It is the following property of D-minimal listings that is important.

Lemma 68. *Given a D-minimal listing $\mathcal{P}^0, \ldots, \mathcal{P}^{2k}$ of an* OCW *$\mathcal{W}$, i.e. one in which $D(\{k,k+1\}) = m_{\mathcal{W}} = m$, there exists:*

- *a $t \in |\mathcal{W}|$ s.t. there is an m-walk from t to k, but no m-walk from t to $k+1$, and*
- *an $s \in |\mathcal{W}|$ s.t. there is an m-walk from s to $k+1$, but no m-walk from s to k.*

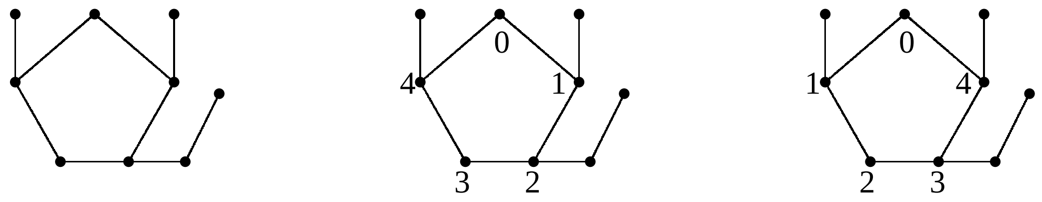


Figure 7.5: An OCW and its two D-minimal listings.

Proof. Let the respective lengths of the paths $\mathcal{P}^i$ be λ_i. It follows that:

$$\begin{aligned} m &= D(\{k,k+1\}) \\ &= \max\{ \begin{array}{ll} & \lambda_0 + k\ , \\ \lambda_1 + [k-1]\ , & \lambda_{2k} + [k-1]\ , \\ \vdots & \vdots \\ \lambda_{k-1} + 1\ , & \lambda_{k+2} + 1\ , \\ \lambda_k\ , & \lambda_{k+1} \end{array} \} \end{aligned}$$

We consider three cases.

(exists i, $1 \le i \le k$, s.t. $m = \lambda_i + [k-1]$.) Take such a branch i that has a vertex at maximal distance from $\{k, k+1\}$. This vertex will be at the end of the path $\mathcal{P}^i$. Label this vertex t, and its neighbour, the penultimate vertex along $\mathcal{P}_i$, s. (If the path $\mathcal{P}^i$ is $\mathcal{K}_1$, *i.e.* there is no path leaving the cycle, then consider $i+1$ on the cycle to be the 'penultimate' vertex). It follows that there will be an m-walk from t to k but not to $k+1$ (by maximality of m, together with the fact that t *must* be closer to k than $k+1$). It also follows that there will be an m-walk from s to $k+1$ but not k (we can not go the long way round the cycle, and any backstepping increases the walk by an even amount).

(exists i, $k+1 \le i \le 2k$, s.t. $m = \lambda_i + [i-k-1]$.) This case is symmetric to the previous.

(previous cases fail, and $m = \lambda_0 + k$.) This is the case in which the ultimate vertex of $\mathcal{P}^0$ is the *unique* vertex at maximal distance m from $\{k, k+1\}$. In this case, we make two claims:

(i) there exists $1 \le i \le k$ s.t. $m - 1 = \lambda_i + [k-1]$, and

(ii) there exists $k+1 \le i \le 2k$ s.t. $m - 1 = \lambda_i + [i-k-1]$.

For the first claim, if no such i exists, then we do not have a D-minimal listing, since $D(\{k+1, k+2\}) < D(\{k, k+1\})$. (If the vertex 0 is always considered at the top of a drawing of $\mathcal{W}$, then this represents rotating the wheel one place

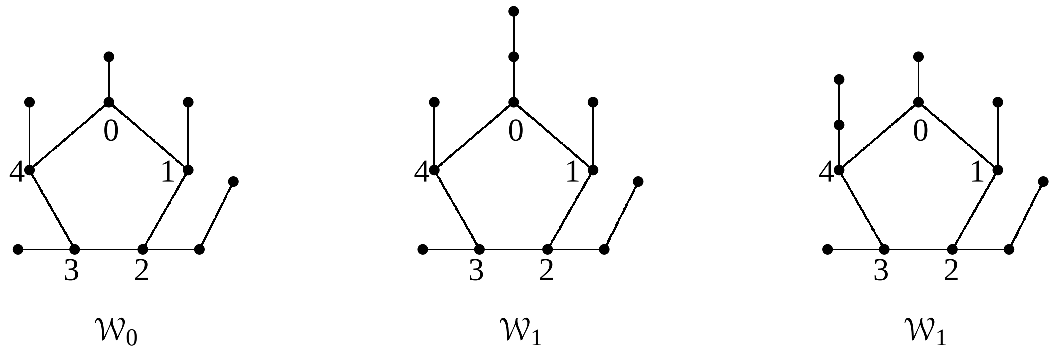


Figure 7.6: The listing on $\mathcal{W}_0$ is D-minimal, but only the rightmost listing of $\mathcal{W}_1$ is D-minimal.

anticlockwise.) The proof of the second claim is symmetric. We may now take t and s to be the ultimate vertices on some paths that fulfill the the second and first criteria, respectively. The proof concludes as with the previous cases. ☐

Before we go on, we will need the following.

Lemma 69 (*e.g.* [25])**.** *For all vertices x of the* $(2k+1)$*-cycle* $\mathcal{C}_{2k+1}$*, there is no* $(2k-1)$*-walk from x to x, but there is a* $(2k-1)$*-walk from x to all distinct vertices y.*

Proof. That there is no walk from x to itself follows by a parity argument, together with the fact that the walk can not go round the entire cycle. We now construct a $(2k-1)$ walk from vertex 0 to any vertex $1 \leq i \leq 2k$, whereafter we may appeal to symmetry. If i even, then walk backwards (anticlockwise) until i is attained, in an odd number of moves, and waste the even number of moves remaining walking between i and some neighbour. If i odd, then walk forwards (clockwise) until i is attained, in an odd number of moves, and waste the remaining moves as before. ☐

Pspace-**completeness.**

Theorem 70. *If* $\mathcal{W}$ *is an* OCW*, then* ALT-HOM$(\mathcal{W})$ *is* Pspace*-complete.*

Proof. Suppose $\mathcal{W}$ has a $(2k+1)$-cycle, and let $m_{\mathcal{W}} = m$ be given. The proof is by direct reduction from $\mathrm{QCSP}(\mathcal{B}^{2k+1}_{\mathrm{NAE}})$. It is based on that given in [5]. Let φ be an instance of $\mathrm{QCSP}(\mathcal{B}^{2k+1}_{\mathrm{NAE}})$. Without loss of generality, we assume that φ has at least one universal quantifier: if there is none we can introduce a dummy[1]. Suppose φ has ν variables and κ clauses: we will construct a partitioned graph $\mathfrak{P}$ such that $\varphi \in \mathrm{QCSP}(\mathcal{B}^{2k+1}_{\mathrm{NAE}})$ iff $\mathfrak{P} \in$ ALT-HOM$(\mathcal{W})$.

[1]This restriction is actually unnecessary in the reduction we use, but it saves us considering as a special case the situation where there are no universally quantified variables.

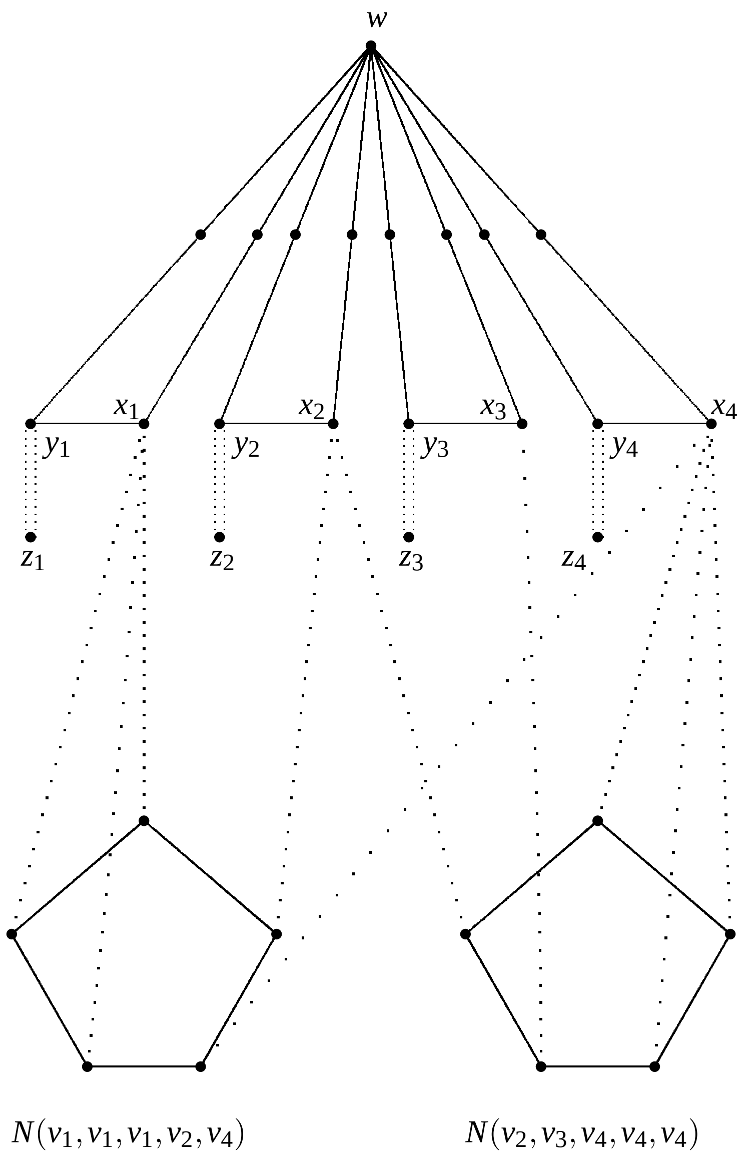


Figure 7.7: Underlying Graph in reduction from QCSP$(\mathcal{B}_{NAE}^{2k+1})$. The dotted lines are $(2k-1)$-paths; the double dotted lines are m-paths.

To build the underlying graph $\mathcal{S}_{\mathfrak{P}}$, we first take ν copies of the $(2k+1)$-gon $\mathcal{C}_{2k+1}$, one for each variable. Consider each of these $(2k+1)$-gons $\mathcal{C}^i_{2k+1}$ to have identified vertex w_i, and labelled twin vertices x_i,y_i farthest away in $\mathcal{C}^i_{2k+1}$ from w_i (it is irrelevant which way round x_i and y_i are). Attach to each $\mathcal{C}^i_{2k+1}$ an m-path from y_i, and label the end-vertex on this path z_i. Now identify the w_is as a single vertex w. (A case involving pentagons, with $\nu = 4$, is shown in the top half of Figure 7.7.) We now take κ further copies of $\mathcal{C}_{2k+1}$, one for each clause. Each vertex of each of these $(2k+1)$-gons represents a variable in the clause. For each variable v_i in such a clause, add a $(2k-1)$-path from the vertex representing v_i to the x_i previously introduced. The case in Figure 7.7 has $\kappa = 2$, with clauses $(v_1, v_1, v_1, v_2, v_3)$ and $(v_2, v_3, v_4.v_4, v_4)$. It remains for us to partition the vertices of $\mathcal{S}_{\mathfrak{P}}$. There is nothing in partition U_1, and w is on its own in partition E_2. Now we read the quantifiers in φ, from the outside. For each existentially quantified variable v_i, we add z_i, its path, and all the rest of $\mathcal{C}^i_{2k+1}$, to the next strictly higher existential partition. For each universally quantified variable v_i, we add just z_i to the next available universal partition. We then add the rest of z_i's path, and all the rest of $\mathcal{C}^i_{2k+1}$, to the next existential partition. When we have gone through all the quantifiers of φ, we add all of the remaining vertices, *i.e.* those in the clause $(2k+1)$-gons, and in the paths that reach them, to the next available existential partition.

This construction is clearly polynomial. It remains for us to prove its correctness. Note that Proponent can not successfully play all the x_i associated with some clause $(2k+1)$-gon to a single vertex of $\mathcal{W}$ (if she plays off the cycle she clearly loses; if she plays on the cycle she loses by Lemma 69).

($\varphi \in \mathrm{QCSP}(\mathcal{B}^{2k+1}_{\mathrm{NAE}}) \rightarrow \mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{W})$). We give Proponent's strategy in the game $(\mathfrak{P}, \mathcal{W})$. She should play w on some vertex 0 on the $(2k+1)$-cycle of $\mathcal{W}$ such that this gives rise to a D-minimal listing of $\mathcal{W}$. Whenever Opponent plays a z_i in $\mathcal{W}$, there will be an m-walk such that y_i may be played on one of vertices k or $k+1$ of the cycle of $\mathcal{W}$. These vertices represent $\textsc{True}$ and $\textsc{False}$ respectively. Since, no matter whether the universal variables are true or false, there is a valuation of the existential variables that gives the clauses a not-all-equal valuation, Proponent may ensure that not all x_i associated with each clause are mapped to $\textsc{True}$ (respectively, $\textsc{False}$). She should play this valuation, finally playing each clause $(2k+1)$-gon and the path to it according to Lemma 69, ensuring homomorphism.

($\mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{W}) \rightarrow \varphi \in \mathrm{QCSP}(\mathcal{B}^{2k+1}_{\mathrm{NAE}})$). If w is not played to a vertex 0 in the $(2k+1)$-cycle of $\mathcal{W}$ such that this gives a D-minimal listing, then Opponent may play any universal z_i (by assumption there is at least one) to some vertex in $\mathcal{W}$ that does not have an m-walk to either k or $k+1$, and Proponent loses. (Such a vertex exists by minimality of m.) Thus, in a winning strategy, w must be played to some 0 on the $(2k+1)$-cycle that gives rise to a D-minimal listing. Note that now Proponent must play each $\{x_i, y_i\}$ to $\{k, k+1\}$, with which being played to which specifying truth or falsity of variable v_i. Thereafter, for any play of z_i, there is an m-walk to either k or $k+1$ (TRUE or FALSE respectively), by the definition of m over a D-minimal listing. For certain z there is an m-walk to k but not to $k+1$, and vice-versa, as guaranteed by Lemma 68. This ensures that Proponent must answer to all valuations of the universal variables. Finally, when the clause $(2k+1)$-gons are reached, if Proponent can extend to homomorphism, then not all the x_is of each clause were played to k (respectively $k+1$), and we have a not-all-equal assignment $\square$

Corollary. *Let $\mathcal{G}$ be a (undirected antireflexive) connected graph that has a unique cycle, which is of odd length. Then* ALT-HOM$(\mathcal{G})$ *is* Pspace-*complete.*

Proof. Such a graph $\mathcal{G}$ is of form similar to an OCW, but with trees affixed to the vertices of the odd cycle, instead of paths. The completeness result holds for such graphs under exactly the same reduction. (The length of such a tree should be considered the maximal depth from its root on the odd cycle.) $\square$

7.4.3 A trichotomy theorem

Theorem 71. *The class of antireflexive undirected graphs with at most one cycle exhibits* ALT-HOM*-trichotomy. Specifically:*

- *If $\mathcal{H}$ is bipartite, then* ALT-HOM$(\mathcal{H})$ *is tractable.*
- *If $\mathcal{H}$ has an odd cycle and is not connected, then* ALT-HOM$(\mathcal{H})$ *is* NP-*complete.*
- *If $\mathcal{H}$ has an odd cycle and is connected, then* ALT-HOM$(\mathcal{H})$ *is* Pspace-*complete.*

Proof. We have just proved the final part. The first part is proved in Theorem 67. The second part is a consequence of Proposition 64, and Hell and Nešetřil's Dichotomy Theorem [25]. $\square$

Remark. The same trichotomy holds on the class of antireflexive undirected graphs with exactly one cycle. This is because bipartite graphs may contain even cycles.

7.5 Closure properties

We examine some closure properties on templates that may be used for proving Pspace-hardness. Later we look at the question of problem equivalence in QCSP.

7.5.1 Indicator construction

Hell and Nešetřil defined three graph constructions to prove their dichotomy theorem for undirected graphs in [25]. One of them is known as the *Indicator construction*. An Indicator is a digraph $\mathcal{I}$ with two identified vertices i and j.

Definition (Hell and Nešetřil [25]). The indicator construction $\mathcal{T}^*$ of a digraph $\mathcal{T}$ with respect to Indicator $(\mathcal{I}, i, j)$ is the graph with vertex set $|\mathcal{T}|$; and, edge set:

$$\{(a,b): \text{ exists hom. } h : \mathcal{I} \to \mathcal{T} \text{ s.t. } h(i) = a \text{ and } h(j) = b\}.$$

Remark. In the case of undirected graphs it makes sense to consider only Indicators $(\mathcal{I}, i, j)$ that have an automorphism swapping i and j. This ensure that $\mathcal{T}^*$ remains undirected when $\mathcal{T}$ is undirected. For an example of this construction, see Figure 7.8.

Lemma 72 (Hell and Nešetřil [25]). *Let $\mathcal{T}$ be an undirected graph and $(\mathcal{I}, i, j)$ an indicator that has an automorphism swapping i and j. If* $\textsc{Hom}(\mathcal{T}^*)$ *is* NP-*complete then* $\textsc{Hom}(\mathcal{T})$ *is* NP-*complete.*

Their result readily extends to digraphs; we extend it to QCSP.

Theorem 73 (Indicator Construction). *Let $\mathcal{G}$ be a digraph and $(\mathcal{I}, i, j)$ an indicator.*

- *If* $\mathrm{CSP}(\mathcal{G}^*)$ *is* NP-*complete then* $\mathrm{CSP}(\mathcal{G})$ *is* NP-*complete.*
- *If* $\mathrm{QCSP}(\mathcal{G}^*)$ *is* Pspace-*complete then* $\mathrm{QCSP}(\mathcal{G})$ *is* Pspace-*complete.*

Proof. We prove the first claim by reducing $\mathrm{CSP}(\mathcal{G}^*)$ to $\mathrm{CSP}(\mathcal{G})$. The proof broadly follows that of Hell and Nešetřil. Let $m := ||\mathcal{I}||$. Take the canonical query θ of $\mathcal{I}$ and remove the two existential quantifiers for i and j. It is now of the form:

$$\theta(z,z') := \exists y_1 \exists y_2 \ldots \exists y_{m-2} Q(z,z',y_1,y_2,\ldots,y_{m-2}),$$

where Q is positive conjunctive. It follows directly from the definitions that $E^{\mathcal{G}^*}(a,b)$ holds if, and only if,

$$\mathcal{G} \models \theta(z/a, z'/b) = \exists y_1 \exists y_2 \ldots \exists y_{m-2} Q(a,b,y_1,y_2,\ldots,y_{m-2}).$$

Hence, given an instance of $\mathrm{CSP}(\mathcal{G}^*)$, we can replace each occurrence of $E(z,z')$ by $\theta(z,z')$, ensuring that variables introduced are new variables. More precisely, let $z_{k_1}, \ldots, z_{k_{2r}}$ be (not necessarily distinct) variables among $\mathbf{z}$, and let $\varphi := \exists \mathbf{z} \bigwedge_{i=1}^{r} E(z_{k_{2i-1}}, z_{k_{2i}})$ be an instance of $\mathrm{CSP}(\mathcal{G}^*)$. We have $\mathcal{G}^* \models \varphi$ iff $\mathcal{G} \models \psi$, where

$$\psi := \exists \mathbf{z} \bigwedge_{i=1}^{r} (\exists y_1^i \exists y_2^i \ldots \exists y_{m-2}^i) Q(z_{k_{2i-1}}, z_{k_{2i}}, y_1^i, y_2^i, \ldots, y_{m-2}^i).$$

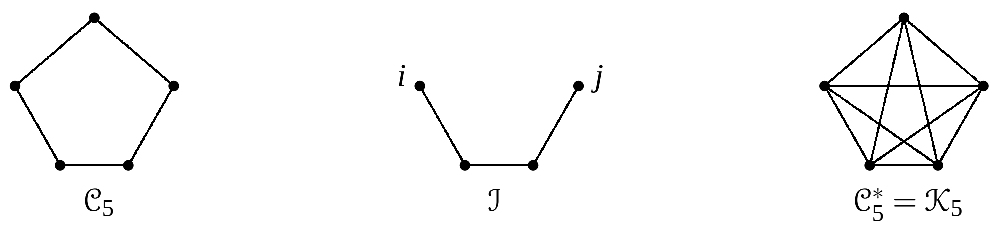


Figure 7.8: Example of the indicator construction.

Note that ψ can be built from φ in polynomial time (remember that the indicator is a fixed graph). Both $\mathrm{CSP}(\mathcal{G})$ and $\mathrm{CSP}(\mathcal{G}^*)$ are in NP, so it follows immediately that NP-completeness of the latter implies NP-completeness of the former.

We use the same method to prove the second claim, by reducing $\mathrm{QCSP}(\mathcal{G}^*)$ to $\mathrm{QCSP}(\mathcal{G})$. We have seen that the edge relation of $\mathcal{G}^*$ can be defined in existential positive conjunctive first-order logic on $\mathcal{G}$. Thus, by replacing in the same way each occurrence of $E(z,z')$ by $\theta(z,z')$, where every variable apart from z and z' is a new one, we get a quantified formula ψ. The proof concludes as before, but for Pspace instead of NP. $\square$

We now have an alternative proof, based on that in [25], of the following.

Corollary. *For every undirected cycle $\mathcal{C}_{2k+1}$ ($k \geq 1$), the problem* ALT-HOM$(\mathcal{C}_{2k+1})$ *is* Pspace-*complete.*

Proof. Recall that ALT-HOM$(\mathcal{K}_{2k+1})$ is Pspace-complete. For $\mathcal{G} := \mathcal{C}_{2k+1}$, choose the indicator $(\mathcal{I}, i, j)$ to be the undirected $(2k+1)$-path from the vertex i to the vertex j. It follows from Lemma 69 that $\mathcal{G}^* = \mathcal{K}_{2k+1}$. The result follows from Theorem 73. The case $k = 2$ is depicted in Figure 7.8. $\square$

Remark. In [25], Hell and Nešetřil introduced two other graph constructions. When $\mathcal{H}$ is a core, they defined the graphs $\mathcal{H}^\sim$ [respectively, $\mathcal{H}^\wedge$] with respect to sub-indicator $\mathcal{J}$ [respectively, edge-sub-indicator $\mathcal{J}'$]. We do not go into the details of these constructions here. They proved that either of HOM$(\mathcal{H}^\sim)$ or HOM$(\mathcal{H}^\wedge)$ being NP-complete implies HOM$(\mathcal{H})$ is NP-complete. We note that this result is unlikely to extend to ALT-HOM (QCSP) in the case of the sub-indicator construction. That is, ALT-HOM$(\mathcal{H}^\sim)$ being Pspace-complete does not imply ALT-HOM$(\mathcal{H})$ is Pspace-complete, under the assumption that NP $\neq$ Pspace. We do not prove this here.

7.5.2 Adding a vertex to a core.

Definition. Let $\mathcal{H}$ be an antireflexive core (*i.e.* any core other than the self-loop $\mathcal{K}_1^{ref}$). Let $\mathcal{H}^+$ be $\mathcal{H}$ with a new vertex γ added, adjacent to all vertices of $\mathcal{H}$, but not adjacent to itself (so it does not introduce a self-loop). Formally:

- $|\mathcal{H}^+| = |\mathcal{H}| \uplus \{\gamma\}$.
- $E^{\mathcal{H}^+} = E^{\mathcal{H}} \cup \{(x,\gamma),(\gamma,x) : x \in \mathcal{H}\}$.

We aim to establish that $\mathcal{H}^+$ is a core.

Lemma 74. *Any homomorphism $h : \mathcal{H}^+ \rightarrow \mathcal{H}^+$ is such that there is an automorphism i of $\mathcal{H}^+$ that swaps γ and $h(\gamma)$.*

Proof. We may assume $h(\gamma) \neq \gamma$. We prove that $h(\gamma)$ is [forward- and backward-]adjacent to all vertices of $\mathcal{H}^+$ except itself. If that is the case then the function that leaves all vertices unchanged, but swaps γ and $h(\gamma)$, will be an automorphism.

Suppose $h(\gamma)$ were not adjacent to everything in $\mathcal{H}^+$ (except itself), and that its neighbours constitute the proper subgraph $\mathcal{H}' \subset \mathcal{H}^+ - \{h(\gamma)\}$. Since h is a homomorphism, it follows that $h(\mathcal{H}) \subseteq \mathcal{H}'$, so we have:

$$h(\mathcal{H}) \subset \mathcal{H}^+ - \{h(\gamma)\}.$$

Now, γ itself may or may not be in the image $h(\mathcal{H})$. We consider both cases separately.

[$\gamma \notin h(\mathcal{H})$.] We have $h(\mathcal{H})$ is a proper subgraph of $\mathcal{H}$, and we are done since this contradicts $\mathcal{H}$ being a core.

[$\gamma \in h(\mathcal{H})$.] See Figure 7.9. It follows that there is a homomorphism

$$h' : \mathcal{H} \longrightarrow h(\mathcal{H}) - \{\gamma\} \cup \{h(\gamma)\},$$

defined by $h'(x) := h(x)$ except when $h(x) = \gamma$, in which case $h'(x) := h(\gamma)$. Whilst $h(\mathcal{H})$ is not actually a subgraph of $\mathcal{H}$, $h'(\mathcal{H})$ is, and it is proper since it has the same cardinality as $h(\mathcal{H})$. This contradicts $\mathcal{H}$ being a core, and we are done. $\square$

Lemma 75. *$\mathcal{H}^+$ is a core.*

Proof. Suppose there were a homomorphism h from $\mathcal{H}^+$ to a proper subgraph $\mathcal{H}' \subset \mathcal{H}^+$. Since we know there is an automorphism of $\mathcal{G}^+$ that swaps γ and $h(\gamma)$, we may assume w.l.o.g. that $h(\gamma) = \gamma$. But that implies that h maps $\mathcal{H}$ to a proper subgraph of itself, which contradicts $\mathcal{H}$ being a core. $\square$

Theorem 76. *Let $\mathcal{H}$ be a core. Then* ALT-HOM$(\mathcal{H})$ *is logspace reducible to* ALT-HOM$(\mathcal{H}^+)$.

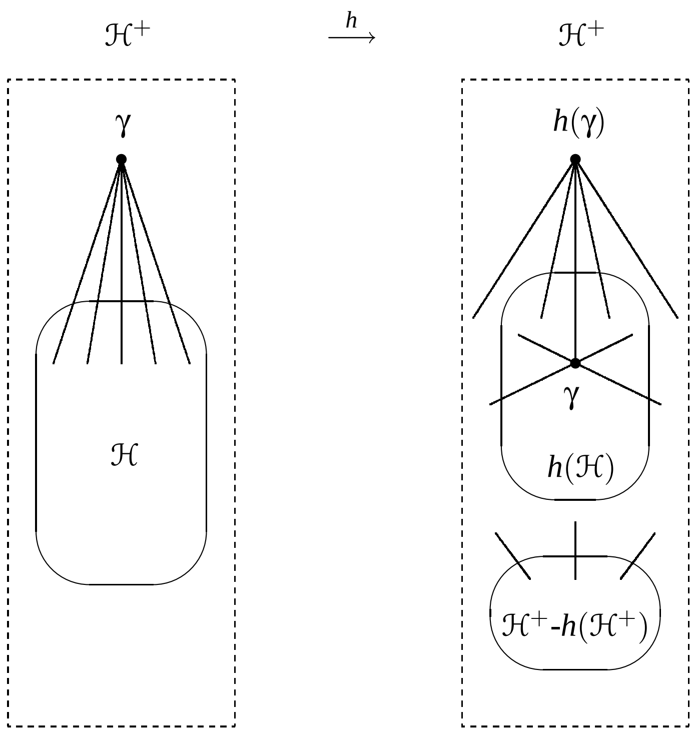


Figure 7.9: The case $\gamma \in h(\mathcal{H})$ in Lemma 74. Note that $\mathcal{H}^+$-$h(\mathcal{H}^+)$ must be non-empty.

Proof. Given an input $\mathfrak{P}$ for ALT-HOM$(\mathcal{H})$, we construct $\mathfrak{P}'$ as an input for ALT-HOM$(\mathcal{H}^+)$ such that $\mathfrak{P} \in$ ALT-HOM$(\mathcal{H})$ iff $\mathfrak{P}' \in$ ALT-HOM$(\mathcal{H}^+)$. We construct $\mathfrak{P}'$ from $\mathfrak{P}$ by introducing a new existential partition E_0 before U_1 (we may renumber later). Into E_0 we place a copy of $\mathcal{H}^+$, adding an edge from the γ of that $\mathcal{H}^+$ to all the existential vertices of $\mathfrak{P}$. Our proof rests on the equivalence of the following:

(i) Proponent has a winning strategy in the game on $(\mathfrak{P}', \mathcal{H}^+)$.

(ii) Proponent has a winning strategy in the game on $(\mathfrak{P}', \mathcal{H}^+)$ where Opponent is forbidden to play γ[2].

(iii) Proponent has a winning strategy in the game on $(\mathfrak{P}, \mathcal{H})$.

In the game $(\mathfrak{P}', \mathcal{H}^+)$, Proponent must play the copy of $\mathcal{H}^+$ in E_0 to itself in the template. Thereafter, Proponent may never play this γ on the template, since $\mathcal{H}^+$ is a core. The equivalence of (ii) and (iii) follows.

The equivalence of (i) and (ii) follows from the fact that γ is adjacent to everything in $\mathcal{H}^+$, so Opponent gains no advantage in playing it. $\square$

[2]Note that γ is not necessarily well-defined in $\mathcal{H}^+$ until Proponent plays the copy of $\mathcal{H}^+$ in $\mathfrak{P}'$ on to the template $\mathcal{H}^+$.

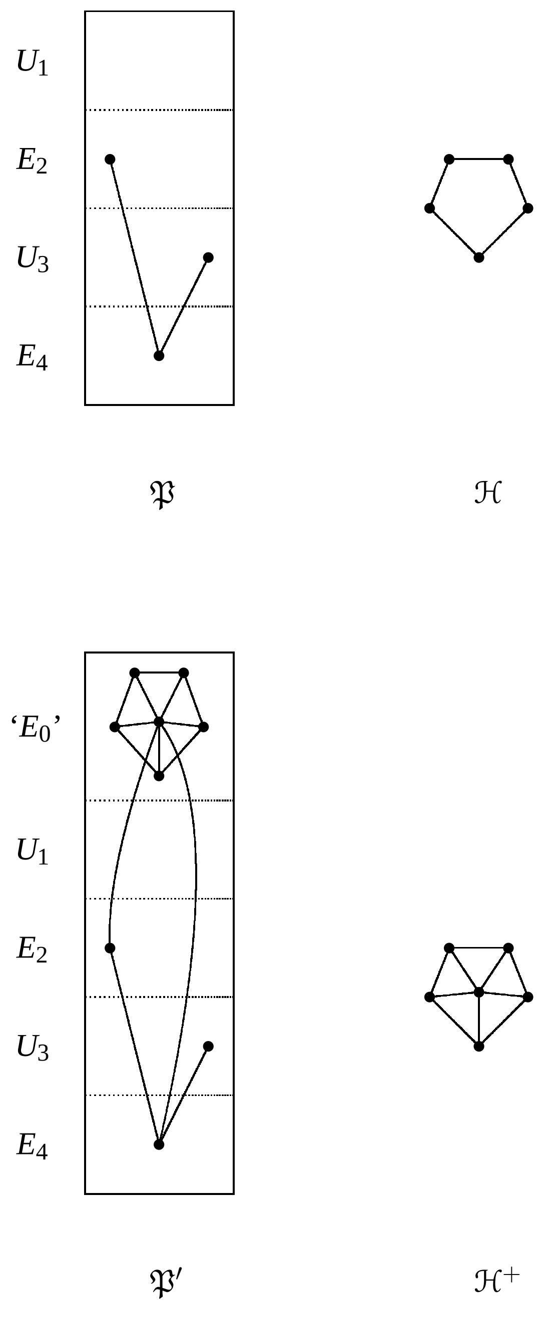


Figure 7.10: An example of the reduction used in Theorem 76, when $\mathcal{H} = \mathcal{C}_5$.

Corollary. *The graphs associated with the* $(2n+1)$*-gonal pyramids give rise to* Pspace*-complete* ALT-HOM *problems.*

Proof. The graph associated with the $(2n+1)$-gonal pyramid is $\mathcal{C}^{+}_{2n+1}$. The case of the pentagonal pyramid appears in Figure 7.10. $\square$

7.5.3 A sufficient condition for ALT-HOM problem equivalence.

It is well known that, for any digraph $\mathcal{G}$ whose core is $\mathcal{H}$, we have $\textsc{Hom}(\mathcal{G}) = \textsc{Hom}(\mathcal{H})$, *i.e.*, for all digraphs $\mathcal{D}$ we have $\mathcal{D} \in \textsc{Hom}(\mathcal{G})$ iff $\mathcal{D} \in \textsc{Hom}(\mathcal{H})$. This result does not extend to ALT-HOM.

Example. Let $\mathcal{G}$ be $\mathcal{K}_3 \uplus \mathcal{K}_3$, therefore its core $\mathcal{H}$ will be $\mathcal{K}_3$. Consider a partitioned graph $\mathfrak{P}$ whose underlying graph is the directed 3-path $\mathcal{DP}_3$. Placing the two end-nodes in partition U_1 and the middle node in partition E_2, we will have $\mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{H})$ whilst $\mathfrak{P} \notin \textsc{Alt-Hom}(\mathcal{G})$. This is because, for $\varphi :=$

$$\forall x \forall z \exists y\, E(x,y) \wedge E(y,z),$$

we have $\mathcal{K}_3 \models \varphi$, but $\mathcal{K}_3 \uplus \mathcal{K}_3 \not\models \varphi$.

However, we propose a digraph construction that preserves the alternating-homomorphism problem.

Definition. Given a digraph $\mathcal{G}$ and specified vertex g, we construct $\mathcal{G}^{+g}$ by *duplicating* the vertex g. Specifically:

- $|\mathcal{G}^{+g}| = |\mathcal{G}| \cup \{g'\}$, and
- $$E^{\mathcal{G}^{+g}} = E^{\mathcal{G}} \cup \{(g',x) : (g,x) \in E^{\mathcal{G}}\} \quad \cup \{(x,g') : (x,g) \in E^{\mathcal{G}}\}$$
  $$\cup \{(g,g'),(g',g),(g',g') : \text{iff } (g,g) \in E^{\mathcal{G}}\}.$$

Theorem 77. *For all digraphs* $\mathcal{G}$*, and any* $g \in \mathcal{G}$*, the problems* $\textsc{Alt-Hom}(\mathcal{G})$ *and* $\textsc{Alt-Hom}(\mathcal{G}^{+g})$ *are equal, i.e. for all partitioned digraphs* $\mathfrak{P}$ *we have* $\mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{G})$ *iff* $\mathfrak{P} \in \textsc{Alt-Hom}(\mathcal{G}^{+g})$.

Proof.

- (Forwards) We prove that a winning strategy σ for Proponent in the $(\mathfrak{P},\mathcal{G})$-game can be translated to a winning strategy σ' for her in the $(\mathfrak{P},\mathcal{G}^{+g})$-game.

  The strategy σ' will tell Proponent that, if Opponent ever plays g' in the game on $(\mathfrak{P},\mathcal{G}^{+g})$, she should behave in the game on $(\mathfrak{P},\mathcal{G})$ exactly as if he had played on g. Since g and g' are adjacent to *exactly* the same vertices in $\mathcal{G}^{+g}$, any play of the $(\mathfrak{P},\mathcal{G})$-game that results in homomorphism must yield a play of the $(\mathfrak{P},\mathcal{G}^{+g})$-game that results in homomorphism.

- (Backwards) We prove that a winning strategy σ for Proponent in the $(\mathfrak{P}, \mathcal{G}^{+g})$-game can be translated to a winning strategy σ' for her in the $(\mathfrak{P}, \mathcal{G})$-game. Indeed, if we take σ and substitiute all instances of g' for g, then we will have such a strategy, for the same reason as before.

$\square$

Definition. For the n-clique $\mathcal{K}_n$, define $\mathcal{K}_n^-$ to be $\mathcal{K}_n$ with any single edge removed.

Corollary. *For all $n \geq 4$, the problem* ALT-HOM$(\mathcal{K}_n^-)$ *is* Pspace-*complete.*

Proof. Observe that $\mathcal{K}_n^- = \mathcal{K}_{n-1}^{+g}$, for any $g \in \mathcal{K}_{n-1}$. $\square$

Corollary. *The graphs associated with the $(2n+1)$-gonal bipyramids give rise to* Pspace-*complete* ALT-HOM *problems.*

Proof. The graph associated with the $(2n+1)$-gonal bipyramid is $(\mathcal{C}_{2n+1}^+)^{+\gamma}$, where γ is the vertex of $\mathcal{C}_{2n+1}^+$ adjacent to everything but itself. $\square$

7.5.4 Why that condition is not necessary: equivalence in fragments of FO.

Definition.

- Let **FO**$\setminus\{=\}$ be first-order logic **FO** deprived of the binary equality relation.
- Let *pos-conj*-**FO** be the fragment of **FO** involving formulae in prenex form, whose quantifier-free portion is positive conjunctive.
- Let $\exists$-*pos-conj*-**FO** be the existential fragment of *pos-conj*-**FO**.

Note the trivial containments $\exists$-*pos-conj*-**FO** $\subseteq$ *pos-conj*-**FO** $\subseteq$ **FO**$\setminus\{=\}$ $\subseteq$ **FO**. These containments are readily seen to be proper; we will return to this later.

$\exists$-*pos-conj*-**FO** is closely related to the problems CSP and HOM. Indeed, it follows from the definition of CSP that, for any two templates $\mathcal{T}$ and $\mathcal{T}'$, CSP$(\mathcal{T})$ = CSP$(\mathcal{T}')$ iff $\mathcal{T}$ and $\mathcal{T}'$ agree on all sentences of $\exists$-*pos-conj*-**FO**. Similarly, it follows from the definition of QCSP that QCSP$(\mathcal{T})$ = QCSP$(\mathcal{T}')$ iff $\mathcal{T}$ and $\mathcal{T}'$ agree on all sentences of *pos-conj*-**FO**. We also have, from the definition of the HOM problems, that HOM$(\mathcal{T})$ = HOM$(\mathcal{T}')$ iff $\mathcal{T}$ is homomorphically equivalent to $\mathcal{T}'$, *i.e.* we have both $\mathcal{T} \xrightarrow{h} \mathcal{T}'$ and $\mathcal{T}' \xrightarrow{h} \mathcal{T}$. This is equivalent to the condition that $\mathcal{T}$ and $\mathcal{T}'$ have isomorphic cores. The concept of core gives us a combinatorial characterisation of what it is to be $\exists$-*pos-conj*-**FO**-equivalent. Such a characterisation seems harder for the logic *pos-conj*-**FO**: in the world of QCSP and ALT-HOM.

However, such a characterisation is at hand with the logic $\mathbf{FO}\backslash\{=\}$, but first we must define the pertinent Ehrenfeucht-Fraisse game. Our exposition is based on that for the standard **FO**-game given in [31].

Definitions. Let $\mathcal{A}$ and $\mathcal{B}$ be digraphs. The $\mathbf{FO}\backslash\{=\}^m_k$-game on a pair $(\mathcal{A}, \alpha_0, \mathcal{B}, \beta_0)$ is played by two players, Spoiler and Duplicator, with k pairs of pebbles over m rounds. A position in such a game, a k-configuration, is a pair of partial functions (α, β), where

- $\alpha : \{v_1, \ldots, v_k\} \rightarrow |\mathcal{A}|$, and
- $\beta : \{v_1, \ldots, v_k\} \rightarrow |\mathcal{B}|$,

and we further have that $dom(\alpha) = dom(\beta)$. The domains of α and β are the pebbles that have already been played. From some position (α_j, β_j), for the next move, Spoiler picks some pebble i (where $1 \leq i \leq k$) and chooses to play it in either $|\mathcal{A}|$ or $|\mathcal{B}|$. If the former [resp. latter], then he adds the pair (v_i, a) to α_j [resp. (v_i, b) to β_j] and Duplicator adds the pair (v_i, b) to β_j [resp. (v_i, a) to α_j], so obtaining the new position $(\alpha_{j+1}, \beta_{j+1})$. Note that v_i may already have been in $dom(\alpha_j) = dom(\beta_j)$, *i.e.* the pebble i may already have been played, in which case some former pairs $(v_i, a') \in \alpha_j$ and $(v_i, b') \in \beta_j$ will have been removed. The initial position of the game is (α_0, β_0). Spoiler wins if at any point the relation $\alpha^{-1} \circ \beta \subseteq |\mathcal{A}| \times |\mathcal{B}|$ does not satisfy:

$$(*) \quad E^{\mathcal{A}}(a, a') \Leftrightarrow E^{\mathcal{B}}(b, b') \quad \text{for all } (a, b), (a', b') \in \alpha^{-1} \circ \beta$$

If Spoiler does not win, then Duplicator wins.

The $\mathbf{FO}\backslash\{=\}$-game on $(\mathcal{A}, \alpha_0, \mathcal{B}, \beta_0)$ is played similarly, but with an unbounded number of pebble pairs and an unbounded number of moves. This means that (α_0, β_0) is an ω-configuration with potentially infinite domain: though only if the initial position were infinite. Since we are concerned with finite digraphs, this will never happen.

The quantifier-rank qr of a formula $\mathbf{FO}\backslash\{=\}$[3] is defined inductively thus: if φ is quantifier-free then $qr(\varphi) = 0$; if $\varphi = \neg\varphi'$ then $qr(\varphi) = qr(\varphi')$; if $\varphi = \varphi' \wedge \varphi''$ then $qr(\varphi) = \max\{qr(\varphi'), qr(\varphi'')\}$; and if $\varphi = \exists v \varphi'$ then $qr(\varphi) = qr(\varphi') + 1$.

Let $\mathbf{FO}\backslash\{=\}^m_k$ be that fragment of $\mathbf{FO}\backslash\{=\}$ whose formulae involve only the variables $v_1, \ldots, v_k$ and whose quantifier-rank is at most m. For (α, β) a k-configuration and $\varphi \in \mathbf{FO}\backslash\{=\}^m_k$ a formula whose free variables are among $dom(\alpha) = dom(\beta)$ we consider φ to be true on $(\mathcal{A}, \alpha)$ [resp. $(\mathcal{B}, \beta)$] if φ is true on $\mathcal{A}$ [resp. $\mathcal{B}$] under free variable assignment α [resp. β]. Letting (α, β) be a k-configuration, we write:

[3] Quantifier-rank is defined identically in **FO**.

- $(\mathcal{A},\alpha) \cong^m_k (\mathcal{B},\beta)$ iff $\mathcal{A}$ and $\mathcal{B}$ agree on all formulae of $\mathbf{FO}\backslash\{=\}^m_k$ (whose free variables are among $dom(\alpha) = dom(\beta)$), and
- $(\mathcal{A},\alpha) \sim^m_k (\mathcal{B},\beta)$ iff Duplicator has a winning strategy for the $\mathbf{FO}\backslash\{=\}^m_k$-game on $(\mathcal{A},\alpha,\mathcal{B},\beta)$.

Letting (α,β) be an ω-configuration, we write:

- $(\mathcal{A},\alpha) \cong (\mathcal{B},\beta)$ iff $\mathcal{A}$ and $\mathcal{B}$ agree on all formulae of $\mathbf{FO}\backslash\{=\}$ (whose free variables must be among $dom(\alpha) = dom(\beta)$), and
- $(\mathcal{A},\alpha) \sim (\mathcal{B},\beta)$ iff Duplicator has a winning strategy for the $\mathbf{FO}\backslash\{=\}$-game on $(\mathcal{A},\alpha,\mathcal{B},\beta)$.

Lemma 78. *There are only finitely many inequivalent formulae of* $\mathbf{FO}\backslash\{=\}^m_k$.

Proof. We prove the result for digraphs: it is easily extended to arbitrary (finite, relational) signatures. We proceed by induction on m.

(Base Case.) For $m = 0$, the formulae we can write are boolean combinations of $E(v_i,v_j)$ (for $i,j \in \{1,\ldots,k\}$). We may consider any $\varphi \in \mathbf{FO}\backslash\{=\}^0_k$ to be a propositional formula in these k^2 propositional variables. We may rewrite this in CNF to obtain a formula $\varphi \in \mathbf{FO}\backslash\{=\}^0_k$ s.t. for all digraphs $\mathcal{G}$, $\mathcal{G} \models \varphi \leftrightarrow \varphi'$. The number distinct clauses for a formula of $\mathbf{FO}\backslash\{=\}^0_k$ in CNF is bounded by 2^{2k^2}, so it follows that the number of inequivalent formulae in CNF is bounded by $2^{2^{2k^2}}$. The result for base case follows.

(Inductive step.) Assume it is true for m. Any formula $\varphi \in \mathbf{FO}\backslash\{=\}^{m+1}_k$ is a boolean combination of formulae of the form $\exists v_i \varphi'$ with $\varphi' \in \mathbf{FO}\backslash\{=\}^m_k$. It follows from the inductive hypothesis that the number of inequivalent such formulae φ is finite, say c, and that therefore the number of inequivalent formulae in $\mathbf{FO}\backslash\{=\}^{m+1}$ is bounded above by 2^{2^c}. ☐

Proposition 79 (Methodology)**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be digraphs and let (α_0,β_0) be an intial k-configuration. Then the following are equivalent:*

(i) $(\mathcal{A},\alpha_0) \sim^m_k (\mathcal{B},\beta_0)$

(ii) $(\mathcal{A},\alpha_0) \cong^m_k (\mathcal{B},\beta_0)$

Proof. (Based on that for **FO** in [31].) We proceed by induction on m. For the base case, $m = 0$, Duplicator wins the zero-round game on $(\mathcal{A},\alpha_0,\mathcal{B},\beta_0)$ iff $\alpha_0^{-1} \circ \beta_0$ satisfies $(*)$ iff $(\mathcal{A},\alpha_0)$ and $(\mathcal{B},\beta_0)$ agree on all quantifier-free formulae φ (whose free variables, indeed, only variables, are in $dom(\alpha_0) = dom(\beta_0)$).

Inductive step: $(i) \Rightarrow (ii)$ (by contraposition). Suppose the proposition is true for m, but that $(\mathcal{A},\alpha_0)$ and $(\mathcal{B},\beta_0)$ disagree on some formula $\varphi \in \mathbf{FO}\backslash\{=\}^{m+1}_k$. If

φ were of the form $\neg\varphi'$ [resp. $\varphi' \wedge \varphi''$] then they would disagree on φ' [resp. one of φ', φ''], so we may assume w.l.o.g. that $\varphi = \exists v_i \varphi'$ ($1 \leq i \leq k$) and that $(\mathcal{A}, \alpha_0) \models \varphi$ and $(\mathcal{B}, \beta_0) \models \neg\varphi$. In playing the $\mathbf{FO}\backslash\{=\}_k^{m+1}$-game on $(\mathcal{A}, \alpha_0, \mathcal{B}, \beta_0)$, Spoiler begins by playing a witness for φ in $\mathcal{A}$, but, no matter where Duplicator replies, we will end up with a position (α_1, β_1) s.t. $(\mathcal{A}, \alpha_1) \models \varphi'$ and $(\mathcal{B}, \beta_1) \models \neg\varphi'$. Since the quantifier-rank of φ' is m, it follows from the inductive hypothesis that Spoiler wins the $\mathbf{FO}\backslash\{=\}_k^{m}$-game on $(\mathcal{A}, \alpha_1, \mathcal{B}, \beta_1)$, and we are done.

Inductive step: $(ii) \Rightarrow (i)$. Suppose that $(\mathcal{A}, \alpha_0) \cong_k^m (\mathcal{B}, \beta_0)$, and let Spoiler take his first move in the $\mathbf{FO}\backslash\{=\}_k^{m+1}$-game on $(\mathcal{A}, \alpha_0, \mathcal{B}, \beta_0)$. Let him place a pebble i on an element of $\mathcal{A}$, so defining α_1. Remembering that there are only finitely inequivalent formulae of $\mathbf{FO}\backslash\{=\}_k^{m}$, let Φ be the conjunction of all of these formulae that $(\mathcal{A}, \alpha_1)$ satisfies. We know that $(\mathcal{A}, \alpha_0) \models \exists v_i \Phi$: so by assumption $(\mathcal{B}, \beta_0) \models \exists v_i \Phi$. Let Duplicator play her pebble i on a witness for $\exists v_i \Phi$ in $\mathcal{B}$. Thus $(\mathcal{A}, \alpha_1)$ and $(\mathcal{B}, \beta_1)$ both satisfy Φ. Since Φ is a complete description of everything satisfied by $(\mathcal{A}, \alpha_1)$ in $\mathbf{FO}\backslash\{=\}_k^{m}$, it follows that $(\mathcal{A}, \alpha_1) \cong_k^m (\mathcal{B}, \beta_1)$, and the result follows from the inductive hypothesis. $\square$

Corollary. *Let $\mathcal{A}$ and $\mathcal{B}$ be digraphs and let (α_0, β_0) be an initial ω-configuration. Then:*

- $(\mathcal{A}, \alpha_0) \sim (\mathcal{B}, \beta_0)$ *iff* $(\mathcal{A}, \alpha_0) \cong (\mathcal{B}, \beta_0)$.
- $(\mathcal{A}, \alpha_0) \sim^{||\mathcal{A}||+||\mathcal{B}||} (\mathcal{B}, \beta_0)$ *iff* $(\mathcal{A}, \alpha_0) \sim (\mathcal{B}, \beta_0)$.

Proof. The first part follows immediately from the previous proposition and the definitions. The second part follows from the fact that Duplicator need only find an answer for the $||\mathcal{A}|| + ||\mathcal{B}||$ positions Spoiler can play: thereafter she may copy previous replies. $\square$

We now introduce the converse of the vertex duplication that we have already seen.

Definition. If a digraph $\mathcal{A}$ possesses two vertices a, a' such that $\{x : E(a,x) \in E^{\mathcal{A}}\} = \{x : E(a',x) \in E^{\mathcal{A}}\}$ and $\{x : E(x,a) \in E^{\mathcal{A}}\} = \{x : E(x,a') \in E^{\mathcal{A}}\}$, then $\mathcal{A}$ may be *folded* to the graph $\mathcal{A}^{-x}$ by collapsing the vertices x and x' to a single vertex x' (alternatively, removing vertex v).

A digraph that has no potential folds is said to be *stiff*. (Similar definitions for fold, and stiff graph, appear, *e.g.*, in [26].)

Theorem 80. *The following are equivalent, for all digraphs $\mathcal{A}, \mathcal{B}$:*

(i) *$\mathcal{A}$ and $\mathcal{B}$ agree on all sentences of* $\mathbf{FO}\backslash\{=\}$.

(ii) *Duplicator wins the* $\mathbf{FO}\setminus\{=\}$*-game on* $(\mathcal{A},\mathcal{B})$.

(iii) *There exists a stiff* $\mathcal{C}$ *such that both* $\mathcal{A}$ *and* $\mathcal{B}$ *may be put through a sequence of folds to derive [isomorphic copies of]* $\mathcal{C}$.

Proof. We already have the equivalence of (i) and (ii).

($(iii) \to (ii)$.) Suppose $f_{\mathcal{A}} : \mathcal{A} \to \mathcal{C}$ and $f_{\mathcal{B}} : \mathcal{B} \to \mathcal{C}$ are the surjective collapsing functions for the respective sequences of folds. For the $\mathbf{FO}\setminus\{=\}$-game on $(\mathcal{A},\mathcal{B})$, if Spoiler plays in $|\mathcal{A}|$ [resp. $|\mathcal{B}|$] then Duplicator should answer with any vertex $b \in |\mathcal{B}|$ s.t. $f_{\mathcal{B}}(b) = f_{\mathcal{A}}(a)$ [resp. any vertex $a \in |\mathcal{A}|$ s.t. $f_{\mathcal{A}}(a) = f_{\mathcal{B}}(b)$] (the existence of such vertices is guaranteed by surjectivity of $f_{\mathcal{A}}$ and $f_{\mathcal{B}}$). This is a winning strategy by the definition of folding and, its inverse, duplication.

($(ii) \to (iii)$.) Suppose Spoiler plays all $||A|| + ||B||$ distinct vertices. Duplicator must be able to answer. Suppose $a_1, \ldots, a_{||A||}$ was answered with $a'_1, \ldots, a'_{||A||}$ and $b_1, \ldots, b_{||B||}$ was answered with $b'_1, \ldots, b'_{||B||}$. In $\mathcal{A}$ repeatedly collapse vertices a_i, a_j to a single vertex iff $a'_i = a'_j$. Continuing until there are no more vertices to collapse, we ultimately build a stiff $\mathcal{C}_{\mathcal{A}}$. In B repeatedly collapse vertices b_i, b_j to a single vertex iff $b'_i = b'_j$, so obtaining a stiff $\mathcal{C}_{\mathcal{B}}$. It follows by transitivity that Duplicator has a winning strategy in the $\mathbf{FO}\setminus\{=\}$-game on $(\mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{B}})$. Let Spoiler play all the positions in $\mathcal{C}_{\mathcal{A}}$ and let Duplicator make her reply. The so obtained $\alpha^{-1} \circ \beta$ that satisifes $(*)$ must also satisfy:

$$\text{injectivity:} \quad \forall a, a' \in \mathcal{C}_{\mathcal{A}}\ \forall b \in \mathcal{C}_{\mathcal{B}} \quad (a,b),(a',b) \in \alpha^{-1} \circ \beta \ \Rightarrow\ a = a'$$

(for otherwise $\mathcal{A}_{\mathcal{C}}$ is not stiff since a may be folded to a'), and

$$\text{surjectivity:} \quad \forall b \in \mathcal{C}_{\mathcal{B}}\ \exists a \in \mathcal{C}_{\mathcal{A}} \quad (a,b) \in \alpha^{-1} \circ \beta$$

(for otherwise $\mathcal{C}_{\mathcal{B}}$ is not stiff). It follows that $\alpha^{-1} \circ \beta$ is an isomorphism, and the result follows. □

Corollary ([26]). *Every digraph* $\mathcal{G}$ *has a unique [up to isomorphism] stiff induced subgraph that it can be put through a sequence of folds to obtain.*

Proof. Take $\mathcal{A} = \mathcal{B} = \mathcal{G}$ in the previous proof. □

We will now unambiguously refer to the *stiff-graph-within* $\mathcal{G}$ as the one which $\mathcal{G}$ reaches through a maximal sequence of folds. We have seen that stiff-graphs-within characterise $\mathbf{FO}\setminus\{=\}$-equivalence in exactly the way that cores characterise $\exists$-*pos-conj*-$\mathbf{FO}$-equivalence. This gives us a new proof of Theorem 77, in which we proved that, for all digraphs $\mathcal{G}$ and $g \in \mathcal{G}$, $\textsc{Alt-Hom}(\mathcal{G}) = \textsc{Alt-Hom}(\mathcal{G}^{+g})$. Since $\mathcal{G}^{+g}$ may be folded to $\mathcal{G}$, it follows that they share the same stiff-graph-within. It is now clear that $\mathcal{G}$ and $\mathcal{G}^{+g}$ agree on all sentences $\mathbf{FO}\setminus\{=\}$, which certainly includes all sentences of *pos-conj*-$\mathbf{FO}$.

It remains for us to ask whether or not stiff-graphs-within capture equivalence in QCSP, *i.e.* whether *pos-conj*-**FO**actually coincides with **FO**$\setminus\{=\}$. It turns out not to be so; we demonstrate the proper containments $\exists$-*pos-conj*-**FO** $\subset$ *pos-conj*-**FO** $\subset$ **FO**$\setminus\{=\}$ $\subset$ **FO**.

Examples.

$\mathcal{K}_3$ and $\mathcal{K}_3 \uplus \mathcal{K}_3$ give rise to the same CSP problem, *i.e.* are equivalent in $\exists$-*pos-conj*-**FO**, but do not give rise to the same QCSP problem, since $\forall x \forall z \exists y E(x,y) \wedge E(y,z)$ is true in the former, yet false in the latter. It follows that $\exists$-*pos-conj*-**FO** can not express that property.

$\mathcal{K}_2 = \mathcal{P}_2$ and $\mathcal{P}_4$ give rise to the same QCSP problems. (In fact, it follows from Lemmas 58 and 66 that there are precisely three classes of ALT-HOM = QCSP problem for bipartite template $\mathcal{T}$: specifically, $\mathcal{T}$ having no edges; or $\mathcal{T}$ has an edge and an isolated vertex; or $\mathcal{T}$ has an edge and no isolated vertices. $\mathcal{P}_2$ and $\mathcal{P}_4$ are both in the last class.) Consider the sentence $\exists w,x,y,z E(w,x) \wedge E(x,y) \wedge E(y,z) \wedge \neg E(z,x)$. This is false in the former, but true in the latter. If follows that the property can not be expressed in *pos-conj*-**FO**.

Finally, consider the query $\forall x \forall y\ E(x,y) \vee x = y$. This can not be expressed in **FO**$\setminus\{=\}$ since $\mathcal{K}_3$ and $\mathcal{K}_4^-$ disagree on it, yet agree on all sentences of **FO**$\setminus\{=\}$.

7.6 Results concerning tournament templates

7.6.1 Template is a directed cycle.

We consider the case where the template $\mathcal{T}$ is a directed n-cycle $\mathcal{DC}_n$. Such a graph is a tournament only when $n = 3$, but the method easily generalises.

Definition. An *oriented path* is a list of vertices $v_1, \ldots, v_m$ and, for $1 \leq i < m$, exactly one of the edges $E(v_i, v_{i+1})$ or $E(v_{i+1}, v_i)$. The *net length* of this oriented path is the number of instances of edges $E(v_i, v_{i+1})$ (forward-edges) minus the number of instances of edges $E(v_{i+1}, v_i)$ (backward-edges). An oriented path in a digraph $\mathcal{G}$ is a (not necessarily induced) subgraph of $\mathcal{G}$ that is an oriented path.

In a directed n-cycle any oriented path between a vertex and itself must have net length 0 *mod n*. Furthermore, any path between a vertex and its forward-neighbour must have net length 1 *mod n*, and every vertex *has* a forward-neighbour. These facts will allow us once again to consider only partitioned inputs in Π_2-multifan form since:

Lemma 81. *For $n \geq 3$,*

- *if there is a path in $\mathfrak{P}$ between any $x \in E_i$ and $y \in U_j$ (for $i < j$), then $\mathfrak{P} \notin$ ALT-HOM$(\mathcal{DC}_n)$, and*

- *if there is a path in $\mathfrak{P}$ between any $x \in U_i$ and $y \in U_j$ (any i, j), then $\mathfrak{P} \notin$ $\text{ALT-HOM}(\mathcal{DC}_n)$.*

Proof. We prove the first claim, the proof of the second is similar. If the path has net length 0 *mod n*, then, if Proponent plays a for x, then Opponent plays the forward-neighbour b of a for y, and wins. If the path has net length other than 0 *mod n*, then, if Proponent plays a for x, then Opponent also plays a for x, and again wins. □

Theorem 82. *If $\mathcal{T}$ is a directed n-cycle then* $\text{ALT-HOM}(\mathcal{T})$ *is tractable.*

Proof. If the input $\mathfrak{P}$ has any of the paths of the previous lemma, then we have a no-instance. We may therefore assume that $\mathfrak{P}$ has no such path, and is equivalent to the rewrite-reduced $\overline{\mathfrak{P}}$ in Π_2-multifan form. We may further split $\overline{\mathfrak{P}}$ into its Π_2-fan form components $\overline{\mathfrak{P}_i}$ $(1 \leq i \leq m)$, for some m, and solve separately for each.

Since $\mathcal{T}$ is rotationally symmetric, each Π_2-fan structure $\overline{\mathfrak{P}_i}$ admits an alternating-homomorphism to $\mathcal{T}$ iff the structure $\mathcal{S}_{\overline{\mathfrak{P}_i}}$ admits a homomorphism to $\mathcal{T}$. It is known that the problem $\text{HOM}(\mathcal{T})$ is tractable [4], and the result follows. □

7.6.2 Template is a digraph with source and sink.

In a digraph, a source (respectively, sink) is a vertex with in-degree (respectively, out-degree) 0.

We consider the case where the template $\mathcal{T}$ is a digraph with both a source s and a sink t. In such cases we need only consider inputs in Σ_1-form since:

Lemma 83. *For any i, j,*

- *if there is a forward-edge in $\mathfrak{P}$ between any $x \in E_i$ and $y \in U_j$, then $\mathfrak{P} \notin$ $\text{ALT-HOM}(\mathcal{T})$,*

- *if there is a backward-edge in $\mathfrak{P}$ between any $x \in E_i$ and $y \in U_j$, then $\mathfrak{P} \notin$ $\text{ALT-HOM}(\mathcal{T})$,*

- *if there is any forward-edge in $\mathfrak{P}$ between any $x \in U_i$ and $y \in U_j$, then $\mathfrak{P} \notin \text{ALT-HOM}(\mathcal{T})$, and*

- *if there is any backward-edge in $\mathfrak{P}$ between any $x \in U_i$ and $y \in U_j$, then $\mathfrak{P} \notin \text{ALT-HOM}(\mathcal{T})$.*

Proof. For the first and third claims, Opponent plays s for y and wins. For the second and fourth claims, Opponent plays t for y and wins. □

The following should be seen as a generalisation of Proposition 59.

Proposition 84. *If $\mathcal{T}$ is an antireflexive digraph with both a source and a sink, then* $\textsc{Alt-Hom}(\mathcal{T})$ *is logspace equivalent to* $\textsc{Hom}(\mathcal{T})$.

Proof. Use the reduction of Proposition 59 (we have the same forbidden edges here as we did there). □

Definition. An n-tournament is a digraph $\mathcal{G}$ with vertex set $\{0,\dots n-1\}$, such that, for all $i,j \in \mathbb{Z}_n$, exactly one of $E(i,j)$ or $E(j,i)$ is an edge of $\mathcal{G}$. The unique n-tournament which contains no directed cycle as a subgraph is known as the transitive n-tournament, and will be denoted $\mathcal{T}^t_n$.

Corollary. *If $\mathcal{T}^t_n$ is the transitive n-tournament then* $\textsc{Alt-Hom}(\mathcal{T}^t_n)$ *is tractable.*

Proof. $\mathcal{T}^t_n$ has both a source and a sink. So, the result follows from the previous theorem, and the fact that $\textsc{Hom}(\mathcal{T}^t_u)$ is tractable [4]. □

7.6.3 Tractable tournament Alt-Hom problems.

The tournaments $\mathcal{T}^u_{m+3}$

We examine the tournaments $\mathcal{T}^u_{m+3}$ which are constructed from the directed 3-cycle by repeatedly adding a source m times. (The superscript u suggests this unique cycle.)

Definition. We define $\mathcal{T}^u_{m+3}$ inductively:

- Let $\mathcal{T}^{(0)} := \mathcal{T}^u_3 = \mathcal{DC}_3$, the directed 3-cycle..
- From $\mathcal{T}^{(r)}$ build $\mathcal{T}^{(r+1)}$ by adding a new source, *i.e.*, $|\mathcal{T}^{(r+1)}| := |\mathcal{T}^{(r)}| \uplus \{r+3\}$[4] and $E^{\mathcal{T}^{(r+1)}} := E^{\mathcal{T}^{(r)}} \uplus \{(r+3,i) : i \in |\mathcal{T}^{(r)}|\}$.
- Let $\mathcal{T}^u_{m+3} := \mathcal{T}^{(m)}$.

Since we have dealt with the case of the directed 3-cycle, we consider $m > 0$, *i.e.* when $\mathcal{T}^u_{m+3}$ has a source.

Lemma 85. *For $m > 0$:*

- *If there is a directed edge in $\mathfrak{P}$ between $x \in E_i$ and $y \in U_j$ ($i < j$), then $\mathfrak{P} \notin$* $\textsc{Alt-Hom}(\mathcal{T}^u_{m+3})$.

[4] $\mathcal{T}^{(r)}$, being a tournament with $r+3$ vertices, will already have vertex numbers 0 to $r+2$.

- *If there is a directed edge in $\mathfrak{P}$ between $x \in U_i$ and $y \in U_j$ (any i, j), then $\mathfrak{P} \notin$ ALT-HOM$(\mathcal{T}^u_{m+3})$.*

- *If there is a directed edge in $\mathfrak{P}$ from $x \in E_j$ to $y \in U_i$ ($i < j$), then $\mathfrak{P} \notin$ ALT-HOM$(\mathcal{T}^u_{m+3})$.*

Proof. The first two parts follow from the antireflexivity of $\mathcal{T}^u_{m+3}$, by Lemma 57. For the final part, if Opponent plays y to the source of $\mathcal{T}^u_{m+3}$, Proponent can have no reply for x. $\square$

Let $\mathfrak{P}$ be a partitioned digraph, we define its cousin $\widetilde{\mathfrak{P}}$ inductively:

- $\mathfrak{P}^{(0)} := \mathfrak{P}$.
- From $\mathfrak{P}^{(r)}$ we build $\mathfrak{P}^{(r+1)}$ by removing *all* sources that are in existential partitions.
- $\widetilde{\mathfrak{P}} := \mathfrak{P}^{(m)}$.

Let $Ex(\mathfrak{P} - \widetilde{\mathfrak{P}})$ be those existential vertices in $\mathfrak{P}$ that are not in $\widetilde{\mathfrak{P}}$, and let $Ex(\widetilde{\mathfrak{P}})$ be those existential vertices in $\mathfrak{P}$ that are also in $\widetilde{\mathfrak{P}}$ (since $\widetilde{\mathfrak{P}} \subseteq \mathfrak{P}$ these are the existential vertices of $\widetilde{\mathfrak{P}}$). Let $PrEx(\mathfrak{P})$ be those vertices of $\mathfrak{P}$ in existential partitions *to* which there is a directed path *from* some vertex in a universal partition. Let $Un(\mathfrak{P})$ be the set of universal vertices of $\mathfrak{P}$. We refer to the vertices of $\mathcal{T}^u_{m+3}$ that are not in the 3-cycle as the *tail* of $\mathcal{T}^u_{m+3}$.

We will benefit from examining which vertices of the underlying graph $\mathcal{S}_{\mathfrak{P}}$ have been removed in the graph $\mathcal{S}_{\widetilde{\mathfrak{P}}}$. It should be clear that vertices in $Un(\mathfrak{P})$ and $PrEx(\mathfrak{P})$ can never be removed, and are, therefore, *protected*. Let us consider the sub-partitioned-graph $\mathfrak{P}_1$ of $\mathfrak{P}$ induced by the existential vertices that are not protected. $\mathfrak{P}_1$ may be put through our given inductive scheme, iteratively removing sources m times, so obtaining $\widetilde{\mathfrak{P}_1}$. It should be clear that $\widetilde{\mathfrak{P}}$ is that subgraph of $\mathfrak{P}$ induced by the set $Un(\mathfrak{P}) \cup PrEx(\mathfrak{P}) \cup |\mathcal{S}_{\widetilde{\mathfrak{P}_1}}|$. Apart from the universal vertices and those existential vertices that are protected, our construction is that given for proving the tractability of HOM$(\mathcal{T}^u_{m+3})$ in [4]. All of the sets we have defined should now be considered as subsets of $\mathfrak{P}$ (though some may be subsets of $\widetilde{\mathfrak{P}}$ too). Before going on we will benefit from the following lemmas.

Lemma 86. *In a winning strategy for Proponent on $(\mathfrak{P}, \mathcal{T}^u_{m+3})$, if Opponent plays all his vertices to the 3-cycle, then Proponent must play all of the vertices of $Ex(\widetilde{\mathfrak{P}})$ [in $\mathfrak{P}$] to the 3-cycle.*

Proof. Again, let $\mathfrak{P}_1$ be the sub-partitioned-graph of $\mathfrak{P}$ induced by those existential vertices that are not protected. Recall that $\widetilde{\mathfrak{P}}$ is the subgraph of $\mathfrak{P}$ induced by $Un(\mathfrak{P}) \cup PrEx(\mathfrak{P}) \cup |\mathcal{S}_{\widetilde{\mathfrak{P}_1}}|$, as in the previous paragraph. So $Ex(\widetilde{\mathfrak{P}})$ is $PrEx(\mathfrak{P}) \cup |\mathcal{S}_{\widetilde{\mathfrak{P}_1}}|$. Since universal vertices are played to the 3-cycle, it follows that all vertices of $PrEx(\mathfrak{P})$ must be played to the 3-cycle. Furthermore, if any vertex of $|\mathcal{S}_{\widetilde{\mathfrak{P}_1}}|$ [in $\mathfrak{P}$] could be played to the tail of $\mathcal{T}^u_{m+3}$, then this could not be extended to a homomorphism from $\mathcal{S}_{\mathfrak{P}_1}$ to $\mathcal{T}^u_{m+3}$ – by definition of $\widetilde{\mathfrak{P}_1}$ – so this could not be a winning strategy for Proponent on $(\mathfrak{P}, \mathcal{T}^u_{m+3})$. The result follows. ☐

Lemma 87. *Assume that* $\mathfrak{P}$ *has none of the edges of Lemma 85. Then Opponent can win the game on* $(\mathfrak{P}, \mathcal{T}^u_{m+3})$ *iff he can win it whilst never playing in the tail of* $\mathcal{T}^u_{m+3}$.

Proof. Since edges of $\mathfrak{P}$ from universal partitions only point toward vertices x in higher existential partitions, if Opponent plays in the tail then he allows Proponent to answer x with anything on the 3-cycle, whereas, if he plays on the 3-cycle he limits Proponent to a single adjacent vertex of the 3-cycle. It is clear that Opponent gains nothing by playing in the tail. ☐

Theorem 88. *The problems* $\text{ALT-HOM}(\mathcal{T}^u_{m+3})$ *are tractable.*

Proof. We already have the result for $m = 0$. For $m > 0$ we will solve $\text{ALT-HOM}(\mathcal{T}^u_{m+3})$ by taking any input $\mathfrak{P}$ for that problem, and constructing a given $\mathfrak{P}'$. We will prove $\mathfrak{P} \in \text{ALT-HOM}(\mathcal{T}^u_{m+3})$ iff $\mathfrak{P}' \in \text{ALT-HOM}(\mathcal{T}^u_3)$, whereupon we may appeal to the known tractability of $\text{ALT-HOM}(\mathcal{T}^u_3)$, and our result will follow.

If $\mathfrak{P}$ has any of the edges of Lemma 85 then we define $\mathfrak{P}'$ to be any set no-instance of $\text{ALT-HOM}(\mathcal{T}^u_3)$ (*e.g.* the transitive 4-tournament $\mathcal{T}^t_4$ with all vertices in E_2). If $\mathfrak{P}$ has none of those edges then we set $\mathfrak{P}'$ to be $\widetilde{\mathfrak{P}}$, via the construction already described. It remains for us to prove that this is correct. It is trivially correct if $\mathfrak{P}$ has any of the edges of Lemma 85: we assume it does not.

($\mathfrak{P} \in \text{ALT-HOM}(\mathcal{T}^u_{m+3}) \Rightarrow \widetilde{\mathfrak{P}} \in \text{ALT-HOM}(\mathcal{T}^u_3)$.) For a winning strategy σ for Proponent in the game on $(\mathfrak{P}, \mathcal{T}^u_{m+3})$, we claim σ is also a winning strategy for her in the game on $(\widetilde{\mathfrak{P}}, \mathcal{T}^u_3)$. This follows immediately from Lemma 86.

($\widetilde{\mathfrak{P}} \in \text{ALT-HOM}(\mathcal{T}^u_3) \Rightarrow \mathfrak{P} \in \text{ALT-HOM}(\mathcal{T}^u_{m+3})$.) From a winning strategy σ for Proponent in the game on $(\widetilde{\mathfrak{P}}, \mathcal{T}^u_3)$, we construct a winning strategy σ' in the game on $(\mathfrak{P}, \mathcal{T}^u_{m+3})$ where Opponent only plays in the 3-cycle. In that game on $(\mathfrak{P}, \mathcal{T}^u_{m+3})$, when Opponent plays on the 3-cycle, then Proponent answers the vertices in $Ex(\widetilde{\mathfrak{P}})$ according to σ, and then maps $Ex(\mathfrak{P} - \widetilde{\mathfrak{P}})$ to the tail of $\mathcal{T}^u_{m+3}$. The result follows from Lemma 87. ☐

The tournaments $\mathcal{T}^u_{3+m}$

These tournaments are analogous to the tournaments $\mathcal{T}^u_{m+3}$, but are constructed by the repeated addition of a sink, rather than a source. It follows by a similar argument that, for all m, ALT-HOM$(\mathcal{T}^u_{m+3})$ is tractable.

The result

Theorem 89. *If $\mathcal{T}$ is a tournament with at most one cycle then* ALT-HOM$(\mathcal{T})$ *is tractable.*

Proof. It follows from standard results about tournaments (see [4]) that $\mathcal{T}$ is either transitive or is $\mathcal{T}^u_3$ with a succession of sources and/or sinks added. If $\mathcal{T}$ has both a source and a sink then we can reduce the problem to HOM$(\mathcal{T})$, which is known to be tractable [4]. If it has no sink, then it is one of the tournaments $\mathcal{T}^u_{m+3}$ above. If it has no source, then it is one of the tournaments $\mathcal{T}^u_{3+m}$ above. $\square$

7.6.4 NP-complete tournament ALT-HOM problems.

Bang-Jenson, Hell, and MacGillavray proved that, for any n-tournament $\mathcal{T}_n$ that has at least two distinct cycles, HOM$(\mathcal{T}_n)$ is NP-complete [4].

Theorem 90. *Let $\mathcal{T}$ be a tournament with more than one cycle and a source and sink, then* ALT-HOM$(\mathcal{T})$ *is* NP-*complete.*

Proof. Follows from Proposition 84 , together with [4]. $\square$

7.6.5 Pspace-complete tournament ALT-HOM problems.

A 2-*walk Tournament* (2WT) is a tournament $\mathcal{T}$ in which, for all distinct $i, j \in \mathcal{T}$, there is a directed 2-walk from x to y.

Theorem 91. *For every* 2WT $\mathcal{T}$, ALT-HOM$(\mathcal{T})$ *is* Pspace-*complete.*

Proof. Note that $||\mathcal{T}|| > 2$. Also, since $\mathcal{T}$ is a tournament, there can be no 2-walk from any vertex $x \in \mathcal{T}$ to itself. Using the directed 2-path from vertex i to vertex j as indicator, we find that $\mathcal{T}^*$ is $\mathcal{K}_{||\mathcal{T}||}$. Pspace-completeness follows from Theorem 73. $\square$

We conclude by proving that the class of 2WTs is infinite.

Definition. For $m \geq 5$, define the Tournament $\mathcal{T}^p_{2m+1}$ thus:

- $|\mathcal{T}^p_{2m+1}| = \{0, \ldots, 2m\}$

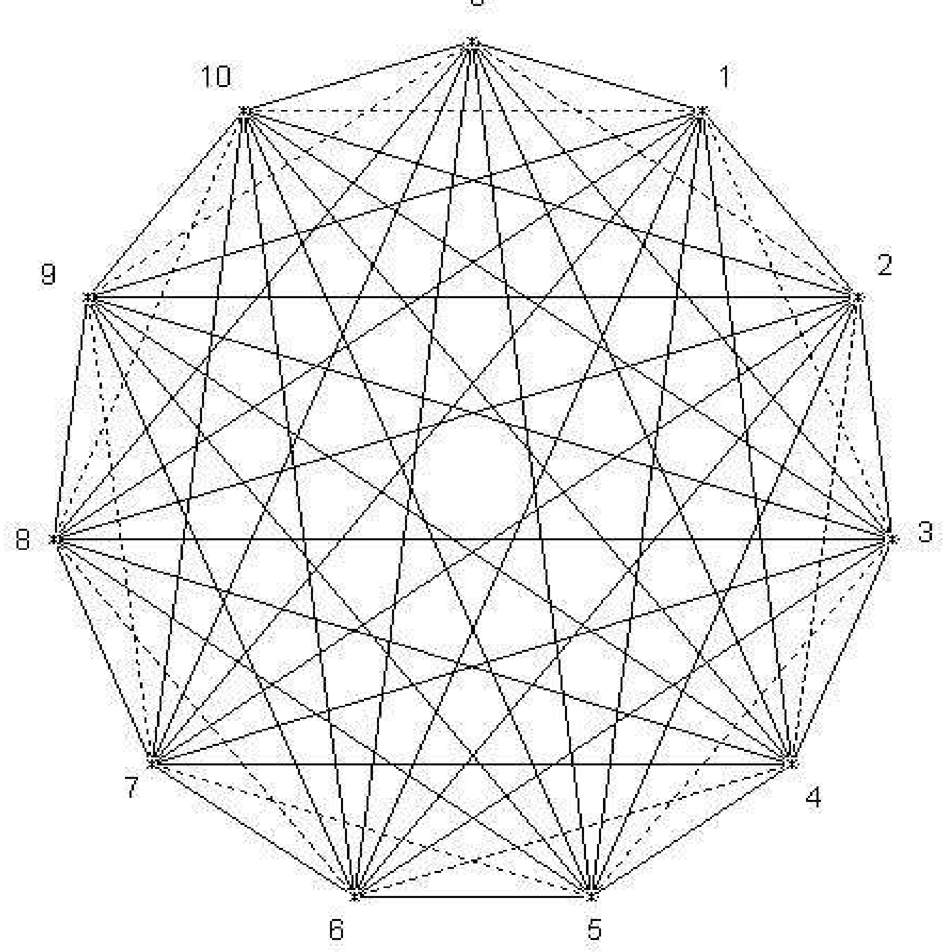


Figure 7.11: The tournament $\mathcal{T}^p_{11}$. The 2-jump dotted edges point anticlockwise; all other edges point clockwise.

- $E^{\mathcal{T}^p_{2m+1}} =$
  - $\{(i,j) : i,j \in \mathbb{Z}_{2m+1} \quad \text{s.t.} \quad i+1 = j \bmod 2m+1\} \cup$
  - $\{(i,j) : i,j \in \mathbb{Z}_{2m+1} \quad \text{s.t.} \quad i-2 = j \bmod 2m+1\} \cup$
  - $\{(i,j) : i,j \in \mathbb{Z}_{2m+1} \quad \text{s.t.} \quad i+3 = j \bmod 2m+1\} \cup$
  - $\{(i,j) : i,j \in \mathbb{Z}_{2m+1} \quad \text{s.t.} \quad i+4 = j \bmod 2m+1\} \cup$
  - $\{(i,j) : i,j \in \mathbb{Z}_{2m+1} \quad \text{s.t.} \quad i+5 = j \bmod 2m+1\} \cup$
  - $\vdots$
  - $\{(i,j) : i,j \in \mathbb{Z}_{2m+1} \quad \text{s.t.} \quad i+m = j \bmod 2m+1\}$

It may easily be verified that $\mathcal{T}^p_{2m+1}$ is a tournament. Note that this is partly a consequence of the odd number of vertices: if we had an even number of vertices under a similar construction we would either have vertices not joined by an edge or vertices joined by a double edge. Observe the aberration of the 2-jumps: if we draw edges on a regular $(2m+1)$-gon with the vertices enumerated clockwise, then all edges point in a clockwise direction, except the 2-jumps which point anticlockwise (see Figure 7.11.)

Lemma 92. *For all distinct vertices* $i, j \in \mathcal{T}^p_{2m+1}$*, there is a directed* 2*-walk from* i *to* j*, i.e.* $\mathcal{T}^p_{2m+1}$ *is a* 2wT.

Proof. We will prove that there is a directed 2-walk from vertex 0 to each of the vertices $1, \dots, 2m$. We may then appeal to symmetry.

It will suffice to show that every number $1, \dots, 2m$ is the sum $(\bmod 2m+1)$ of exactly two elements of the set $\{1, -2, 3, 4, 5, \dots, m\}$. So: $1 = 3-2$; $2 = 4-2$; $3 = 5-2$ (this is why $m \geq 5$), henceforth we may use the positive numbers $\{3, \dots, m\}$ for $6 = 3+3$ through to $2m = m+m$. $\square$

Chapter 8

Conclusions and Further Work

8.1 Program Schemes

Most of our results for program schemes augmented with priority queue are far from tight, and those that are tight are unsurprising. It is hard to see how a Turing machine simulation might prove that NPSPQ^u is contained in a space-bounded (or time-bounded) complexity class. This is because the size of the potential memory of the priority queue in NPSPQ^u seems to be unbounded. We give no better upper bound to NPSPQ^u than the class of recursively enumerable languages. For better lower bounds for NPSPQ^u, our simulation method can go no further than $\mathsf{NPspace}$, since we rely on the fact that we can enumerate the tape squares through a constant number of variables (which may encode only a polynomial quantity of numbers). A similar problem arises in the case of a better lower bound for $\mathrm{APSS}(1)$.

We suggest that an indirect method may be more likely to succeed, perhaps like that used to prove $\mathsf{P} \subseteq \mathrm{NPSS}_s(1)$ – via the path system problem – in [2].

8.2 Classes of Structure on which $\mathsf{P} = \pm\mathbf{PS}^k[\mathbf{FO}]$

Grohe had proved in [23] that $\mathsf{P} = \mathrm{LFP}[\mathbf{FO}]$ on the class of 3-connected planar graphs. Since triangulations are 3-connected planar graphs and it is known that $\mathsf{P} = \pm\mathrm{PS}[\mathbf{FO}]$ on the class of triangulations [42], it is natural to question whether $\mathsf{P} = \mathrm{LFP}[\mathbf{FO}]$ on the class of 3-connected planar graphs. Thus far, we have failed to adapt Grohe's method to settle this question.

8.3 Quantified Constraints on Graphs

The method used to prove the Pspace-completeness of ALT-HOM$(\mathcal{H})$, where $\mathcal{H}$ is an odd catherine wheel, may be applied to prove Pspace-completeness for similar templates. An obvious extension is for graphs that are constructed like odd catherine wheels, but where any bipartite graph (not just a tree) may be appended to each position on the odd cycle.

For quantified $\mathcal{H}$-colouring, we conjecture the following extension to Theorem 71.

Conjecture 93. The class of antireflexive undirected graphs exhibits ALT-HOM-trichotomy. Specifically:

- If $\mathcal{H}$ is bipartite, then ALT-HOM$(\mathcal{H})$ is tractable.
- If $\mathcal{H}$ is not bipartite, and is not connected, then ALT-HOM$(\mathcal{H})$ is NP-complete.
- If $\mathcal{H}$ is not bipartite, and is connected, then ALT-HOM$(\mathcal{H})$ is Pspace-complete.

In order to prove this, it would remain for us to prove that ALT-HOM$(\mathcal{H})$ is Pspace-complete, when $\mathcal{H}$ is antireflexive, undirected and connected, and has more than one odd cycle.

It is well-known that common cores characterise equivalent $\exists$-*pos*-*conj*-**FO**-theories (CSP/ HOM), *i.e.* two templates $\mathcal{T}, \mathcal{T}'$ have the same core iff they agree on all sentences of $\exists$-*pos*-*conj*-**FO**. Similarly, common stiff-graphs-within characterise equivalent **FO**$\setminus\{=\}$-theories. We know of no such characterisation for equivalence of *pos*-*conj*-**FO**-theories (QCSP/ ALT-HOM). It would be interesting to isolate some characteristic on two templates that exactly specifies whether they give rise to the same QCSP problem.

Bibliography

[1] ABITEBOUL, S., AND VIANU, V. Generic computation and its complexity. In *32nd IEEE Symposium on FOCS* (1991).

[2] ARRATIA-QUESEDA, A., CHAUHAN, S., AND STEWART, I. Hierarchies in classes of program schemes. *Journal of Logic and Computation 9:6* (1999), 915–957.

[3] ASCHBACHER, M., AND GURALNICK, R. Some applications of the first cohomology group. *Journal of Algebra 90* (1984), 446–460.

[4] BANG-JENSON, J., HELL, P., AND MAGILLIVRAY, G. The complexity of colourings by semi-complete digraphs. *SIAM Journal on Discrete Mathematics 1:3* (1988).

[5] BORNER, F., KROKHIN, A., BULATOV, A., AND JEAVONS, P. Quantified constraints and surjective polymorphisms. Tech. Rep. PRG-RR-02-11, Oxford University, 2002.

[6] BREWSTER, R., FEDER, T., HELL, P., HUANG, J., AND MACGILLIVRAY, G. Near-unanimity functions and varieties of graphs. 2003.

[7] BULATOV, A. A dichotomy theorem for constraints on a three-element set. In *FOCS'02* (2002).

[8] BULATOV, A., KROKHIN, A., AND JEAVONS, P. Constraint satisfaction problems and finite algebras. In *Proceedings 27th International Colloquium on Automata, Languages and Programming, ICALP'00* (2000), vol. 1853 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 272–282.

[9] CHANDRA, A., AND MERLIN, P. Optimal implementation of conjunctive queries in relational databases. In 9*th ACM Symposium on Theory of Computing* (1979), pp. 77–90.

[10] CHEN, H. Collapsibility and consistency in quantified constraint satisfaction. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (2004).

[11] CHEN, H. *The Computational Complexity of Quantified Constraint Satisfaction*. PhD thesis, Cornell University, August 2004.

[12] CHEN, H. Existentially restricted quantified constraint satisfaction. Tech. Rep. cs.CC/0506059, ACM Computing Research Repository, 2005.

[13] CREIGNOU, N., AND HERMANN, M. Complexity of generalized satisfiability counting problems. *Information and Computation 125* (1996).

[14] CREIGNOU, N., KHANNA, S., AND SUDAN, M. *Complexity classifications of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications 7. 2001.

[15] DALMAU, V. Some dichotomy theorems on constant-free quantified boolean formulas. *Machine Learning 35:3* (1999).

[16] EBBINGHAUS, H.-D., AND FLUM, J. *Finite Model Theory*. Perspective in Mathematical Logic. Springer-Verlag, 1995.

[17] FAGIN, R. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity and Computation: SIAM-AMS 7* (1974).

[18] FEDER, T., AND KOLAITIS, P. Closures and dichotomies for quantified constraints. To appear in SIAM J. Discrete Math.

[19] FEDER, T., AND VARDI, M. Y. The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. *SIAM J. Comput. 28* (1999).

[20] GRADEL, E. Model checking games. *Electronic Notes in Theoretical Computer Science 67* (2002).

[21] GRADEL, E., AND MCCOLM, G. Hierarchies in transitive closure logic, stratified datalog and infinitary logic. *Annals of pure and applied logic 77* (1996), 166–199.

[22] GROHE, M. Existential least fixed-point logic and its relatives. *Journal of Logic and Computation 7* (1997).

[23] GROHE, M. Fixed-point logics on planar graphs. In *Logic and Computer Science: IEEE 13* (1998), pp. 6–15.

[24] GUREVICH, Y. Logic and the challenge of computer science. In *Current trends in Theoretical Computer Science* (1988).

[25] HELL, P., AND NEŠETŘIL, J. On the complexity of H-coloring. *J. Combin. Theory Ser. B 48* (1990).

[26] HELL, P., AND NEŠETŘIL, J. *Graphs and Homomorphisms*. OUP, 2004.

[27] HESMAPANDRA, E. Dichotomy theorems for alternation-bounded quantified boolean formulas. Tech. Rep. cs.CC/0306134, ACM Computing Research Repository, 2003.

[28] IMMERMAN, N. Relational queries computable in polynomial time. *Information and Control 68* (1986).

[29] IMMERMAN, N. Languages that capture complexity classes. *SIAM Journal of Computing 16:4* (1987).

[30] IMMERMAN, N. Nondeterministic space is closed under complementation. *SIAM Journal of Computing 17:5* (1988).

[31] IMMERMAN, N. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, 1998.

[32] JEAVONS, P., COHEN, D., AND GYSSENS, M. A unifying framework for tractable constraints. In *Proceedings 1st International Conference on Constraint Programming, CP'95* (1995), vol. 976 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 276–291.

[33] JEAVONS, P., COHEN, D., AND GYSSENS, M. A test for tractability. In *Proceedings 2nd International Conference on Constraint Programming—CP'96 (Boston, August 1996)* (1996), vol. 1118 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 267–281.

[34] KOLAITIS, P. G., AND VARDI, M. Y. A game-theoretic approach to constraint satisfaction. In *AAAI-2000/IAAI-2000 Proceedings* (2000).

[35] LADNER, R. E. On the structure of polynomial time reducibility. *J. ACM 22* (1975), 155–171.

[36] OTTO, M. *Bounded Variable Logics and Counting*. Lecture Notes in Logic 9. Springer-Verlag, 1997.

[37] PAPADIMITRIOU, C. *Computational Complexity*. Addison-Wesley, 1994.

[38] SCHAEFER, T. The complexity of satisfiability problems. In *STOC* (1978).

[39] STEINBERG, R. Generators for simple groups. *Canad. J. Math. 14* (1962), 277–283.

[40] STEWART, I. Logical description of monotone NP problems. *Journal of Logic and Computation 4* (1994), 337–357.

[41] STEWART, I. Program schemes, arrays, lindstrom quantifiers and zero-one laws. *Theoretical Computer Science 275* (2002), 283–310.

[42] STEWART, I. Using program schemes to logically capture polynomial-time on certain classes of structures. *LMS Journal of Computation and Mathematics 6* (2003), 40–67.

[43] STOCKMEYER, L. The polynomial-time hierarchy. *Theoretical Computer Science 3* (1977).

[44] VARDI, M. Complexity of relational query languages. In *14th Symposium on Theory of Computation* (1982).