

The Complexity of Constraint Satisfaction Problems and Symmetric Datalog

László Egri

*School of Computer Science
McGill University, Montréal
October, 2007*

A thesis submitted to the Faculty of Graduate Studies and
Research in partial fulfillment of the requirements of the degree
of Master of Science.

Copyright ©László Egri 2007.

Abstract

Constraint satisfaction problems (CSPs) provide a unified framework for studying a wide variety of computational problems naturally arising in combinatorics, artificial intelligence and database theory. To any finite domain D and any constraint language Γ (a finite set of relations over D), we associate the constraint satisfaction problem $\text{CSP}(\Gamma)$: an instance of $\text{CSP}(\Gamma)$ consists of a list of variables x_1, x_2, \dots, x_n and a list of constraints of the form “ $(x_1, x_2, \dots, x_n) \in R$ ” for some relation R in Γ . The goal is to determine whether the variables can be assigned values in D such that all constraints are simultaneously satisfied. The computational complexity of $\text{CSP}(\Gamma)$ is entirely determined by the structure of the constraint language Γ and, thus, one wishes to identify classes of Γ such that $\text{CSP}(\Gamma)$ belongs to a particular complexity class.

In recent years, logical and algebraic perspectives have been particularly successful in classifying CSPs. A major weapon in the arsenal of the logical perspective is the database-theory-inspired logic programming language called Datalog. A Datalog program can be used to solve a restricted class of CSPs by either accepting or rejecting a (suitably encoded) set of input constraints. Inspired by Dalmau’s work on linear Datalog and Reingold’s breakthrough that undirected graph connectivity is in logarithmic space, we use a new restriction of Datalog called symmetric Datalog to identify a class of CSPs solvable in logarithmic space. We establish that expressibility in symmetric Datalog is equivalent to expressibility in a specific restriction of

second order logic called Symmetric Restricted Krom Monotone SNP that has already received attention for its close relationship with logarithmic space.

We also give a combinatorial description of a large class of CSPs lying in L by showing that they are definable in symmetric Datalog. The main result of this thesis is that directed *st*-connectivity and a closely related CSP cannot be defined in symmetric Datalog. Because undirected *st*-connectivity can be defined in symmetric Datalog, this result also sheds new light on the computational differences between the undirected and directed *st*-connectivity problems.

Résumé

Les problèmes de satisfaction de contraintes (ou CSP) forment un cadre particulièrement riche permettant de formaliser de façon uniforme un grand nombre de problèmes algorithmiques tirés de l'optimisation combinatoire, de l'intelligence artificielle et de la théorie des bases de données. À chaque domaine D et chaque langage de contraintes Γ (i.e. un ensemble de relations sur D), on associe le problème CSP(Γ) suivant. Une instance du problème est constituée d'une liste de variables x_1, \dots, x_n et d'une liste de contraintes de la forme $(x_1, x_2, \dots, x_n) \in R$, où $R \in \Gamma$. On cherche à déterminer si des valeurs de D peuvent être assignées aux variables de telle sorte que les contraintes soient toutes satisfaites simultanément. La complexité algorithmique de CSP(Γ) est entièrement fonction de la structure du langage de contraintes Γ et on cherche alors à identifier des classes de contraintes pour lesquelles CSP(Γ) appartient à une classe de complexité spécifique.

Au cours des dernières années, cette classification des CSP a grandement progressé grâce à des approches logique et algébrique. Le langage de programmation logique Datalog, né de la théorie des bases de données, est un des outils principaux de l'approche logique. Un programme Datalog peut être utilisé pour résoudre efficacement certains CSP en acceptant ou en rejetant un ensemble de contraintes (encodé adéquatement). En s'inspirant des travaux de Dalmau sur le Datalog linéaire et du résultat fondamental de Reingold sur la complexité de la connexité dans les graphes non-dirigés, nous présentons un nouveau fragment de Datalog, le Datalog symétrique, et identifions ainsi

une classe de CSP résolubles en espace logarithmique. Nous montrons que l'expressivité de Datalog symétrique coïncide avec celle d'un fragment de la logique de second-ordre appelé Krom SNP Symétrique Monotone Restreint qui avait déjà fait l'objet d'étude à cause de son lien aux problèmes résolubles en espace logarithmique.

Nous présentons également une large classe de CSP résolubles en espace logarithmique grâce au Datalog symétrique. Le principal résultat de ce mémoire établit que la *st*-connexité d'un graphe dirigé (et un CSP intimement relié à ce problème) ne sont pas définissables dans Datalog symétrique. Cela met de nouveau en lumière les différences importantes entre les cas dirigé et non-dirigé du problème de connexité des graphes.

Acknowledgments

First and foremost, I wish to express my deepest gratitude to my supervisors Pascal Tesson and Denis Thérien. I am grateful to Denis for getting me interested in complexity theory during his two fantastic courses and giving me the opportunity to work in this field. I have also been fortunate to participate in Denis' Barbados workshops where I learnt a great deal about complexity theory and got to know many excellent researchers. I also thank him for the financial support he provided me during the last year.

I thank Pascal for teaching me countless things and sharing his invaluable insights with me. He introduced me to constraint satisfaction problems and suggested the main topic of this thesis, the wonderful definition of symmetric Datalog. I would like to thank Pascal and also Benoît Larose for many fruitful discussions and suggestions.

I have had the privilege of having a bunch of superb office mates: Anil Ada, Arkadev Chattopadhyay, Navin Goyal and Mark Mercer.

I thank the administrative and academic staff of the School of Computer Science of McGill University. Through their work and dedication, they provide a great environment for study and research.

I also want to thank NSERC and FQRNT for their financial support.

Finally, I thank my parents for their support and encouragement, Bernard Létourneau and Lise Morin for their unconditional support and help over many years, and my girlfriend Joanne Teng Teng Chong for supporting me with patience.

Contents

1	Introduction	1
1.1	Basic Notions	2
1.2	CSP Examples	3
1.3	Overview	6
2	An Algebraic Approach to Constraint Satisfaction Problems	8
2.1	Constructing Relational Clones	9
2.2	From Relational Clones to Polymorphisms	10
2.3	From Polymorphisms to Algebras	14
2.4	Applications	18
3	Datalog and CSPs in NL and L	22
3.1	Basic Notions	22
3.2	Homomorphism Problems	23
3.3	Datalog	24
3.3.1	Syntax	24
3.3.2	Semantics	25
3.3.3	Derivation Trees	27
3.4	Datalog and CSPs	29
3.4.1	The Relation Between Datalog and CSPs	29
3.4.2	Canonical Datalog Programs	31
3.5	The Complexity of Linear and Symmetric Datalog	33

3.6	Symmetric Datalog, Linear Datalog and Second Order Logic	35
3.6.1	Symmetric Restricted Krom SNP	35
3.6.2	Second Order Logics for Linear and Symmetric Datalog	36
3.7	Capturing NL and L Using Datalog	40
4	A New Class of CSPs Expressible in Symmetric Datalog	42
4.1	Symmetric Datalog and Relational Clones	42
4.2	A Sufficient Condition for Expressibility in Symmetric Datalog	49
5	Directed ST-Connectivity Is Not Definable in Symmetric Datalog	58
5.1	The Mirror Operator	59
5.1.1	The Left-Right Case	63
5.1.2	The Containment Case	66
5.2	The Free Derivation Path	68
5.2.1	Basic Definitions	68
5.2.2	Lemmas Related to The Mirror Operator	71
5.3	The Main Theorem	74
5.4	Disconnecting an Isolated UV-Path	76
5.4.1	The UV-Path Following Diagram	76
5.4.2	The Disconnecting Lemma	78
5.5	Directed ST-Connectivity Is Not in Symmetric Datalog	83
6	Conclusion	87
6.1	Overview	87
6.2	Possibility for Future Work	88

Chapter 1

Introduction

The constraint satisfaction problem (CSP) was introduced in 1974 by Montanari [36] and provides a unified framework for studying a wide variety of computational problems [12, 27, 34, 35, 32, 46, 39] naturally arising in machine vision, belief maintenance, scheduling, temporal reasoning, type reconstruction, graph theory and satisfiability. To any finite domain D and any finite set of relations Γ over D , we associate the constraint satisfaction problem $\text{CSP}(\Gamma)$: an instance of $\text{CSP}(\Gamma)$ consists of a list of variables x_1, \dots, x_n and a list of *constraints* of the form $(x_{i_1}, \dots, x_{i_k}) \in R_j$ for some k -ary relation $R_j \in \Gamma$. The goal is to determine whether the variables can be assigned values in D such that all constraints are simultaneously satisfied.

Understanding the computational complexity of problems involving constraints is one of the most fundamental challenges in constraint programming. The class of all CSPs is NP-hard [33] and therefore it is unlikely that efficient general-purpose algorithms exist for solving them. However, in many practical applications the instances that arise have a special form that allow for efficient heuristics. (See for example [5, 8, 30, 40].)

Another approach to CSPs does not make assumptions about the form of the input instances but rather restricts the type of constraints that are allowed. That is, imposing certain restrictions on Γ often allows to efficiently

1.1 Basic Notions

solve $\text{CSP}(\Gamma)$. The ultimate objective is to make precise statements about the complexity of $\text{CSP}(\Gamma)$ *given only* Γ . This thesis is centered around two approaches, one termed the *logical approach* and the other the *algebraic approach*.

In recent years, these two approaches have been particularly successful in identifying “islands of tractability”, i.e. wide classes of Γ s for which $\text{CSP}(\Gamma)$ is tractable, i.e. solvable in polynomial time (P). A central idea of the logical approach is to relate the tractability of $\text{CSP}(\Gamma)$ with the expressibility of the problem in the database-theory-inspired logic programming language called Datalog and its restrictions.

In the algebraic approach one considers for each Γ the set of operations which *preserve* the relations of Γ . It can be shown that the algebraic properties of this set precisely determine the complexity of $\text{CSP}(\Gamma)$. In particular this point of view yields widely applicable sufficient criteria for the tractability of $\text{CSP}(\Gamma)$ (e.g. if Γ is preserved under a *Mal'tsev* operation [4, 3]).

1.1 Basic Notions

We begin with some simple definitions.

Definition 1.1. For any set D (called **domain**) and any non-negative integer n , the set of all n tuples of elements of D is denoted by D^n . The i^{th} component of a tuple t is denoted by $t[i]$. A subset of D^n is called an n -ary **relation** over D . The set of all finitary relations over D is denoted by \mathbf{R}_D . A **constraint language** over D is a subset of \mathbf{R}_D .

Definition 1.2. For any set D and any constraint language Γ over D , $\text{CSP}(\Gamma)$ is the combinatorial decision problem:

Instance: A triple $\langle V, D, C \rangle$, where

- V is a set of **variables**;

1.2 CSP Examples

- C is a set of **constraints**, $\{C_1, \dots, C_q\}$;
- Each constraint $C_i \in C$ is a pair $\langle s_i, R_i \rangle$, where
 - s_i is a tuple of variables of length n_i , called the **constraint scope**;
 - $R_i \in \Gamma$ is an n_i -ary relation over D , called the **constraint relation**.

Question: Does there exist a **solution**, that is, a function $f : V \rightarrow D$ such that for each constraint $\langle s, R \rangle \in C$ with $s = \langle v_1, \dots, v_n \rangle$ the tuple $\langle f(v_1), \dots, f(v_n) \rangle$ belongs to R ? The set of solutions of a CSP instance $\mathcal{P} = \langle V, D, C \rangle$ will be denoted by $\text{Sol}(\mathcal{P})$.

To determine the complexity of a constraint satisfaction problem we need to specify how instances are encoded as finite strings of symbols. The size of a CSP instance can be taken to be the length of a string specifying the variables, the domain, all constraint scopes and corresponding relations. We shall assume in all cases that this representation is chosen so that the complexity of determining whether a constraint allows a given assignment of values to the variables in its scope is bounded by a polynomial function of the length of the representation.

In this thesis we deal only with finite constraint languages. For finite domains, we can assume that the tuples in the constraint relations are listed explicitly. In fact, it is enough to take the size of an instance to be the number of variables occurring in the constraints as it is polynomially related to the number of possible tuples in the relations of the instance.

1.2 CSP Examples

Example 1.3. 3-SAT is the decision problem in which we are given a 3CNF-formula ϕ with variables x_1, \dots, x_n and clauses c_1, \dots, c_m and we have to decide whether the clauses can be satisfied simultaneously. Now we express 3-SAT

1.2 CSP Examples

as a CSP. For each possible type of clause we create a relation in the following way:

Clause Type	Relation
$(x \vee y \vee z)$	$R_0 = \{0, 1\}^3 \setminus \{(0, 0, 0)\}$
$(\neg x \vee y \vee z)$	$R_1 = \{0, 1\}^3 \setminus \{(1, 0, 0)\}$
$(\neg x \vee \neg y \vee z)$	$R_2 = \{0, 1\}^3 \setminus \{(1, 1, 0)\}$
$(\neg x \vee \neg y \vee \neg z)$	$R_3 = \{0, 1\}^3 \setminus \{(1, 1, 1)\}$

Let $\Gamma = \{R_0, R_1, R_2, R_3\}$. Then the CSP instance $\langle V, D, C \rangle$ has a solution if and only if ϕ is satisfiable, where $V = \{x_1, \dots, x_n\}$, $D = \{0, 1\}$, $C = \{c_1, \dots, c_m\}$ and in $c_i = \langle (x, y, z), R_j \rangle$, where R_j is the relation corresponding to (x, y, z) as given in the table above.

In the next example, we need the following definition.

Definition 1.4. A constraint language Γ is called:

- **Tractable** if $\text{CSP}(\Gamma')$ can be solved in polynomial time, for each subset $\Gamma' \subseteq \Gamma$;
- **NP-complete** if $\text{CSP}(\Gamma')$ is NP-complete for some finite subset $\Gamma' \subseteq \Gamma$.

Example 1.5. A **Boolean** constraint language is a constraint language over a two element domain $D = \{d_0, d_1\}$. Similarly to the encoding used in Example 1.3, we can express SATISFIABILITY [37] as a CSP. It was established by Schaefer in 1978 in a seminal paper [44] that a Boolean constraint language Γ is tractable if (at least) one of the following six conditions holds:

1. Every relation in Γ contains the tuple in which all entries are equal to d_0 ;
2. Every relation in Γ contains the tuple in which all entries are equal to d_1 ;

1.2 CSP Examples

3. Every relation in Γ is definable by a conjunction of clauses, where each clause has at most one positive literal (i.e., a conjunction of Horn clauses);
4. Every relation in Γ is definable by a conjunction of clauses, where each clause has at most one negative literal;
5. Every relation in Γ is definable by a conjunction of clauses, where each clause contains at most two literals;
6. Every relation in Γ is the set of solutions of a system of linear equations over the finite field with two elements, $\text{GF}(2)$.

Otherwise Γ is NP-complete. This result is often called **Schaefer's Dichotomy Theorem** [44].

Example 1.6. The binary **less than** relation over an ordered set D is defined as:

$$<_D = \{\langle d_1, d_2 \rangle \in D^2 : d_1 < d_2\}$$

Let $A = \mathbb{N}$ where \mathbb{N} is the set of natural numbers. Then the class of CSP instances $\text{CSP}(\{<_{\mathbb{N}}\})$ corresponds to the **ACYCLIC DIGRAPH** problem [2] (the problem is to decide whether a graph is acyclic). Note that a directed graph is acyclic if and only if its vertices can be numbered in such a way that every arc leads from a vertex with smaller number to a vertex with a greater one. Since the **ACYCLIC DIGRAPH** problem is tractable, $\{<_{\mathbb{N}}\}$ is tractable.

Example 1.7. Let **disequality** be the following relation over D :

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

$\text{CSP}(\{\neq_D\})$ corresponds to the **GRAPH COLORABILITY** problem [16, 37] with $|D|$ colors. This problem is in polynomial time if $|D| \leq 2$ or $|D| = \infty$ and NP-complete if $3 \leq |D| < \infty$.

1.3 Overview

Example 1.8. Let \mathbb{F} be any finite field and Γ_{Lin} be the constraint language containing all those relations over \mathbb{F} which consist of all the solutions to some system of linear equations over \mathbb{F} . Any relation from Γ_{Lin} , and therefore any instance of $\text{CSP}(\Gamma_{Lin})$ can be represented by a system of linear equations over \mathbb{F} and this system of equations can be computed from the relations in polynomial time [5]. A system of linear equations can be solved in polynomial time (e.g., by Gaussian elimination) and therefore Γ_{Lin} is tractable.

Example 1.9. Let D be a domain. A binary relation $R \subseteq D^2$ is said to be *implicational* or a *0/1/all constraint* if it is one of the following forms (see [26] and [9]):

1. $R = B \times C$ for some $B, C \subseteq D$;
2. $R = \{\langle b, f(b) \rangle : b \in B\}$ where $B \subseteq D$ and f is an injective function;
3. $R = \{b\} \times C \cup B \times \{c\}$ for some $B, C \subseteq D$ with $b \in B$ and $c \in C$.

Dalmau showed [10] that CSPs defined by implicational constraints are in NL. We will say more about implicational constraints in Chapter 4.

1.3 Overview

An important driving force of research on CSPs is a major **dichotomy conjecture** postulating that for every Γ , $\text{CSP}(\Gamma)$ is either in polynomial time or is NP-complete ([15]). (Note that this conjecture is a generalization of the result in Example 1.5 to domains of any size.)

The conjecture has been the subject of intense research since its formulation and we summarize these recent efforts in Chapter 2. Because of the dichotomy conjecture, research has focused almost entirely on classifying $\text{CSP}(\Gamma)$ as either solvable in polynomial time or NP-complete. While this is certainly the important question, we should be satisfied only when we have shown that a problem is complete for some well-known complexity class. The

1.3 Overview

main focus of this thesis is to make progress towards a more refined classification of CSPs and in particular, we focus on CSPs in logarithmic space (L).

In Chapter 3 we introduce a new restriction of Datalog called *symmetric Datalog*. We show that symmetric Datalog programs can be evaluated in logarithmic space using Reingold’s algorithm ([41]). We then show how to define the complement of a CSP in (symmetric) Datalog (when possible) and we conclude that CSPs whose complement is definable in symmetric Datalog are in logarithmic space. We introduce a fragment of second order logic whose expressive power is that of symmetric Datalog. Finally, we show how to capture logarithmic space and non-deterministic logarithmic (NL) space using symmetric Datalog and linear Datalog, respectively.

In Chapter 4 we give a combinatorial description of a large class of constraint languages for which $\neg\text{CSP}(\Gamma)$ is in symmetric Datalog and thus solvable in logarithmic space.

The main contribution of this thesis is Chapter 5 where we prove that an important CSP, $\text{CSP}(\langle\{0, 1\}; \leq, \{0\}, \{1\}\rangle)$ is not definable in symmetric Datalog or equivalently, that directed *st*-connectivity is not definable in symmetric Datalog. Because undirected *st*-connectivity is definable in symmetric Datalog, this result sheds new light on the computational differences between directed and undirected *st*-connectivity.

In Chapter 6 we present some conjectures and outline future research possibilities.

Chapter 2

An Algebraic Approach to Constraint Satisfaction Problems

This chapter relies on [43] and [5]. As we noted earlier, an important driving force of research on CSPs is a conjecture postulating that for every Γ , $\text{CSP}(\Gamma)$ is either solvable in polynomial time or NP-complete ([15]). The conjecture has been the subject of intense research since its formulation and many results in this direction was obtained by understanding algebraic properties of the relations in the constraint languages (e.g. [22, 23, 24, 25]). In this chapter we describe this approach and summarize the state of the art in Theorem 2.31 and Theorem 2.32.

The algebraic approach has three main steps:

1. Given a constraint language Γ , further relations can often be added to it with a polynomial increase in the complexity of the resulting CSP. These enlarged constraint languages or sets of relations are known as **relational clones**. So to separate tractable and NP-complete problem classes we can work with the relational clone of a constraint language.

2.1 Constructing Relational Clones

2. Relational clones can be characterized by their **polymorphisms** which are algebraic operations defined on the same underlying domain.
3. The third step is to link constraint languages with finite algebras.

2.1 Constructing Relational Clones

Definition 2.1. A constraint language Γ **expresses** a relation R if there is an instance $\mathcal{P} = \langle V, D, C \rangle \in \text{CSP}(\Gamma)$ and a list $\langle v_1, \dots, v_n \rangle$ of variables in V such that

$$R = \{ \langle \varphi(v_1), \dots, \varphi(v_n) \rangle : \varphi \in \text{Sol}(\mathcal{P}) \}$$

Definition 2.2. The **expressive power** of a constraint language Γ is the set of all relations that can be expressed by Γ . The expressive power of a constraint language Γ can be characterized in many different ways. For example, it is equal to the set of all relations that can be obtained from the relations in Γ using the *relational join* and *project* operations from relational database theory [18]. In algebraic terminology this set of relations is called the **relational clone** generated by Γ [13], and it is denoted by $\langle \Gamma \rangle$. It is also equal to the set of relations definable by **primitive positive formulas** over the relations in Γ together with the equality relation, where a primitive positive formula is a first-order formula containing only conjunction and existential quantification [21, 5].

Theorem 2.3 ([21, 5]). *For any constraint language Γ and any finite set $\Delta \subseteq \langle \Gamma \rangle$, there is a polynomial time reduction from $\text{CSP}(\Delta)$ to $\text{CSP}(\Gamma)$.*

Proof. Let $\Delta = \{R_1, \dots, R_m\}$ be a finite set of relations over the finite set D , where R_i is expressible by a primitive positive formula involving relations from Γ and the equality relation $=_D$.

2.2 From Relational Clones to Polymorphisms

Any instance $\langle V; D; C \rangle \in \text{CSP}(\Delta)$ can be transformed as follows. For every constraint $\langle s, R \rangle \in C$, where $s = \langle v_1, \dots, v_\ell \rangle$ and R is representable by the primitive positive formula

$$R(v_1, \dots, v_\ell) = \exists u_1, \dots, u_m (R_1(w_1^1, \dots, w_{\ell_1}^1) \wedge \dots \wedge R_n(w_1^n, \dots, w_{\ell_n}^n)),$$

where $w_1^1, \dots, w_{\ell_1}^1, \dots, w_1^n, \dots, w_{\ell_n}^n \in \{v_1, \dots, v_\ell, u_1, \dots, u_m\}$ do the following:

1. Add the auxiliary variables u_1, \dots, u_m to V (renaming if necessary so that none of them occurs before);
2. Add the constraint $\langle \langle w_1^1, \dots, w_{\ell_1}^1 \rangle, R_1 \rangle, \dots, \langle \langle w_1^n, \dots, w_{\ell_n}^n \rangle, R_n \rangle$ to C ;
3. Remove $\langle s, R \rangle$ from C .

Clearly, this instance has a solution if and only if the original instance has a solution and it belongs to $\text{CSP}(\Gamma \cup \{=_{\mathcal{D}}\})$. Furthermore, all constraints of the form $\langle \langle v_1, v_2 \rangle, =_{\mathcal{D}} \rangle$ can be eliminated by replacing all occurrences of v_2 by v_1 . This transformation can be done in polynomial time. \square

Remark: In fact, this reduction can be made much weaker as shown in [29]. We will state and partially prove this result in Theorem 4.1 of Chapter 4.

Corollary 2.4. *A set of relations Γ is tractable if and only if $\langle \Gamma \rangle$ is tractable. Similarly, a set of relations Γ is NP-complete if and only if $\langle \Gamma \rangle$ is NP-complete.*

This result reduces the problem of characterizing tractable constraint languages to the problem of characterizing tractable relational clones.

2.2 From Relational Clones to Polymorphisms

In the previous section we have shown that to analyse the complexity of arbitrary constraint languages over finite domains it is sufficient to consider

2.2 From Relational Clones to Polymorphisms

relational clones. Now we describe a convenient way to represent relational clones.

Definition 2.5. Let D be a set and k be a non-negative integer. A mapping $f : D^k \rightarrow D$ is called a k -ary **operation** on D . The set of all finitary operations on D is denoted by \mathbf{O}_D .

There is a fundamental relationship between operations and relations. Observe that any operation on a set D can be extended in a standard way to an operation on tuples of elements from D as follows. A k -ary operation f maps k n -tuples $t_1, \dots, t_k \in D^n$ to the n tuple

$$\langle f(t_1[1], \dots, t_k[1]), \dots, f(t_1[n], \dots, t_k[n]) \rangle.$$

Definition 2.6. A k -ary operation $f \in \mathbf{O}_D$ **preserves** an n -ary relation $R \in \mathbf{R}_D$ if $f(t_1, \dots, t_k) \in R$ for all choices of $t_1, \dots, t_k \in R$. Equivalently, f is a **polymorphism of R** , or **R is invariant under f** .

For any given sets $\Gamma \subseteq \mathbf{R}_D$ and $F \subseteq \mathbf{O}_D$ the mappings Pol and Inv are defined as follows:

- $Pol(\Gamma) = \{f \in \mathbf{O}_D : f \text{ preserves each relation from } \Gamma\}$
- $Inv(F) = \{R \in \mathbf{R}_D : R \text{ is invariant under each operation from } F\}$

To better understand the relation between Pol and Inv we need the concept of Galois-correspondence (see for example [13]).

Definition 2.7. A **Galois-correspondence** between sets A and B is a pair (σ, τ) of mappings between the power sets¹ $\mathcal{P}(A)$ and $\mathcal{P}(B)$:

$$\sigma : \mathcal{P}(A) \rightarrow \mathcal{P}(B), \text{ and } \tau : \mathcal{P}(B) \rightarrow \mathcal{P}(A).$$

σ and τ must satisfy the following conditions. For all $X, X' \subseteq A$ and all $Y, Y' \subseteq B$

¹Given a set S , the power set of S , written $\mathcal{P}(S)$ is the set of all subsets of S .

2.2 From Relational Clones to Polymorphisms

1. $X \subseteq X' \rightarrow \sigma(X) \supseteq \sigma(X')$, and $Y \subseteq Y' \rightarrow \tau(Y) \supseteq \tau(Y')$;
2. $X \subseteq \tau\sigma(X)$, and $Y \subseteq \sigma\tau(Y)$.

It is easy to check that the mappings Pol and Inv form a Galois-correspondence between \mathbf{R}_D and \mathbf{O}_D .

Lemma 2.8. *Let the pair (σ, τ) with*

$$\sigma : \mathcal{P}(A) \rightarrow \mathcal{P}(B), \text{ and } \tau : \mathcal{P}(B) \rightarrow \mathcal{P}(A)$$

be a Galois-correspondence between the sets A and B . Then $\sigma\tau\sigma = \sigma$ and $\tau\sigma\tau = \tau$.

Proof. Let $X \subseteq A$. By the second Galois-correspondence property, $X \subseteq \tau\sigma(X)$. By the first property, applying σ gives $\sigma(X) \supseteq \sigma\tau\sigma(X)$. But by applying the second property to $\sigma(X)$, we also have that $\sigma(X) \subseteq \sigma\tau(\sigma(X))$. Therefore $\sigma\tau\sigma(X) = \sigma(X)$. The second claim is proved similarly. \square

Corollary 2.9. *$Pol(\Gamma) = Pol(Inv(Pol(\Gamma)))$ and $Inv(F) = Inv(Pol(Inv(F)))$ for any given sets $\Gamma \subseteq \mathbf{R}_D$ and $F \subseteq \mathbf{O}_D$*

In universal algebra it is well-known that a relational clone generated by a set of relations over a finite set is determined by the polymorphisms of those relations [38]. Here we give a proof of this result using purely constraint-based reasoning.

Definition 2.10. Let Γ be a finite constraint language over a finite set D . For any positive integer k , the **indicator problem of order k** for Γ is the CSP instance $\langle V, D, C \rangle \in \text{CSP}(\Gamma)$ where

- $V = D^k$, i.e. each variable of \mathcal{C} is a k -tuple of domain elements;
- $C = \{\langle s, R \rangle : R \in \Gamma \text{ and } s \text{ matches (see below) } R\}$.

2.2 From Relational Clones to Polymorphisms

A list of k tuples $s = \langle v_1, \dots, v_n \rangle$ **matches** a relation R if n is equal to the arity of R and for each $i \in \{1, 2, \dots, k\}$ the n -tuple $\langle v_1[i], \dots, v_n[i] \rangle$ is in R .

Fact 2.11. *The solutions to the indicator problem of order k for Γ are mappings from D^k to D that preserve each relation in Γ . These mappings are precisely the k -ary elements of $Pol(\Gamma)$.*

Theorem 2.12 ([38, 21]). *For any constraint language Γ over a finite set, $\langle \Gamma \rangle = Inv(Pol(\Gamma))$.*

Proof. The following are easy to check. If an operation f is a polymorphism of two relations then f is also a polymorphism of the relation obtained by taking the conjunction of those two relations. The equality relation has every operation as a polymorphism. Finally, if f is a polymorphism for a relation then f is also a polymorphism for any relation that is obtained by existential quantification of that relation. Therefore for any $R \in \langle \Gamma \rangle$ we have that

$$\begin{aligned} Pol(\{R\}) \supseteq Pol(\Gamma) &\rightarrow Pol(\langle \Gamma \rangle) \supseteq Pol(\Gamma) \\ &\rightarrow Inv(Pol(\langle \Gamma \rangle)) \subseteq Inv(Pol(\Gamma)) \\ &\rightarrow \langle \Gamma \rangle \subseteq Inv(Pol(\Gamma)). \end{aligned}$$

To establish the converse let $R \in Inv(Pol(\Gamma))$, where Γ is a constraint language over a finite set D . Let r be the arity of R . We show that $R \in \langle \Gamma \rangle$. Let k denote the number of tuples in R . Construct the indicator problem \mathcal{P} of order k for Γ . Choose a list of variables (elements of D^k) $t = \langle v_1, \dots, v_r \rangle$ in \mathcal{P} such that each of the r -tuples $\langle v_1[i], \dots, v_r[i] \rangle$ for $i \in 1, \dots, k$, is a distinct element of R . Let R_t be the relation $\{\langle f(v_1), \dots, f(v_r) \rangle : f \in Sol(\mathcal{P})\}$. By construction, R_t can be expressed using the constraint language Γ . In other words, $R_t \in \langle \Gamma \rangle$. Now we show that $R = R_t$.

It is clear that the k projection operations which return one of their arguments are k -ary polymorphisms of Γ . It follows from this and Fact 2.11 that $R \subseteq R_t$. Conversely, every polymorphism of Γ preserves R so $R_t \subseteq R$. Therefore $R = R_t$. \square

2.3 From Polymorphisms to Algebras

Corollary 2.13. *A relation R over a finite set can be expressed by a constraint language Γ if and only if $Pol(\Gamma) \subseteq Pol(\{R\})$.*

Proof. If $\{R\} \subseteq \langle \Gamma \rangle$ then $\{R\} \subseteq Inv(Pol(\Gamma))$ which implies that $Pol(\{R\}) \supseteq Pol(\Gamma)$. If $Pol(\Gamma) \subseteq Pol(\{R\})$ then $Inv(Pol(\Gamma)) \supseteq Inv(Pol(\{R\}))$. Therefore $\langle \Gamma \rangle \supseteq \langle R \rangle \supseteq \{R\}$. \square

Corollary 2.14. *For any constraint languages Γ and Δ over a finite set, if Δ is finite and $Pol(\Gamma) \subseteq Pol(\Delta)$ then there is a polynomial time reduction from $CSP(\Delta)$ to $CSP(\Gamma)$.*

Proof. We have that for any $R \in \Delta$, $Pol(\Gamma) \subseteq Pol(\Delta) \subseteq Pol(\{R\})$. Therefore $\{R\} \in \langle \Gamma \rangle$ so $\Delta \subseteq \langle \Gamma \rangle$ and using Theorem 2.3 we have the result. \square

By Corollary 2.14, for any finite constraint language Γ over a finite set, the complexity of $CSP(\Gamma)$ is determined by the polymorphisms of Γ up to a polynomial time reduction.

Definition 2.15. A set of operations $F \subseteq \mathbf{O}_D$ is called:

- **Tractable** if $Inv(F)$ is tractable;
- **NP-complete** if $Inv(F)$ is NP-complete.

2.3 From Polymorphisms to Algebras

Definition 2.16. An **algebra** \mathcal{A} is an ordered pair $\langle D, F \rangle$ such that D is a nonempty set and F is a family of finitary operations on D . The set D is called the universe of \mathcal{A} and the operations in F are called **basic**. An algebra with a finite universe is referred to as a finite algebra.

To complete our objective to link constraint languages to finite algebras, we associate with a set of operations F on a fixed set D the algebra $\langle D, F \rangle$. We define what it means for an algebra to be tractable as follows.

2.3 From Polymorphisms to Algebras

Definition 2.17. An algebra $\mathcal{A} = \langle D, F \rangle$ is said to be

- **Tractable** if the set of basic operations F is tractable;
- **NP-complete** if the set of basic operations F is NP-complete.

Now we define an equivalence relation linking algebras such that any two algebras in the same equivalence class correspond to the same constraint language. Recall that by Corollary 2.9 $Inv(Pol(Inv(F))) = Inv(F)$ so we can extend the set of operations F to the set $Pol(Inv(F))$ without changing the family of associated relations. Before we continue we need the following definition.

Definition 2.18. If f is an m -ary operation on a set D , and g_1, g_2, \dots, g_m are k -ary operations on D then the **composition of f and g_1, g_2, \dots, g_m** is the k -ary operation h on D defined as

$$h(a_1, a_2, \dots, a_k) = f(g_1(a_1, \dots, a_k), \dots, g_m(a_1, \dots, a_k)).$$

The set $Pol(Inv(F))$ consists of all operations that can be obtained by taking arbitrary compositions of operations from the set

$$F \cup \{f : f \text{ is a projection operation}\}.$$

Any set of operations that contains the projection operations and is closed under compositions is called a **clone**. The clone of operations obtained from a set $F \subseteq \mathbf{O}_D$ in this way is referred to as the set of **term operations** over F . This motivates the following definition:

Definition 2.19. For any algebra $\mathcal{A} = \langle D, F \rangle$, an operation f on D is called a **term operation** of \mathcal{A} if $f \in Pol(Inv(F))$. The set of all term operations of \mathcal{A} will be denoted by $Term(\mathcal{A})$. Two algebras \mathcal{A} and \mathcal{B} with the same universe are called **term equivalent** $Term(\mathcal{A}) = Term(\mathcal{B})$.

2.3 From Polymorphisms to Algebras

Fact 2.20. *Two algebras $\mathcal{A} = \langle D, F_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle D, F_{\mathcal{B}} \rangle$ are term equivalent if and only if they have the same set of associated invariant relations.*

Proof. If $Term(\mathcal{A}) = Term(\mathcal{B})$ then

$$\begin{aligned} Inv(F_{\mathcal{A}}) &= Inv(Term(\mathcal{A})) \\ &= Inv(Term(\mathcal{B})) \\ &= Inv(F_{\mathcal{B}}). \end{aligned}$$

If $Inv(F_{\mathcal{A}}) = Inv(F_{\mathcal{B}})$ then $Pol(Inv(F_{\mathcal{A}})) = Pol(Inv(F_{\mathcal{B}}))$ so $Term(\mathcal{A}) = Term(\mathcal{B})$. \square

By Fact 2.20, it is enough to characterize tractable algebras only up to term equivalence. Now we show that it is sufficient to consider special classes of algebras. The first simplification we can do is stated in Theorem 2.21.

Theorem 2.21 ([24], [21]). *Let Γ be a constraint language over a set D and let f be a unary operation in $Pol(\Gamma)$. Then $CSP(\Gamma)$ is polynomial-time equivalent to $CSP(f(\Gamma))$, where $f(\Gamma) = \{f(R) : R \in \Gamma\}$ and $f(R) = \{f(t) : t \in R\}$.*

It is easy to see that if we apply Theorem 2.21 with a unary polymorphism f which has the smallest possible range out of all unary polymorphisms in $Pol(\Gamma)$ then $f(\Gamma)$ is a constraint language whose unary polymorphisms are all surjective. Such a constraint language is called a **reduced** constraint language.

Definition 2.22. An algebra is called **surjective** if all of its term operations are surjective.

Fact 2.23. *A finite algebra \mathcal{A} is surjective if and only if all of its unary term operations are surjective.*

2.3 From Polymorphisms to Algebras

Proof. One direction is trivial. Now assume that all unary operations are surjective but $Term(\mathcal{A})$ contains a k -ary operation f that is not surjective where $k > 1$. Define the unary operation g as $g(x) = f(p(x), \dots, p(x))$ where $p(x)$ is the unary projection operation. Observe that g is a non-surjective unary operation in $Term(\mathcal{A})$ which is a contradiction. \square

Now we will see that it is enough to consider only surjective algebras which are idempotent.

Definition 2.24. An operation f on D is called **idempotent** if it satisfies $f(x, \dots, x) = x$ for all $x \in D$. The **full idempotent reduct** of an algebra $\mathcal{A} = \langle D, F \rangle$ is the algebra $\langle D, IdTerm(\mathcal{A}) \rangle$ where $IdTerm(\mathcal{A})$ is the set of all idempotent operations in $Term(\mathcal{A})$.

Clearly, an operation f on a set D is idempotent if and only if it preserves the relation $\Gamma_{const} = \{\{\langle a \rangle\} : a \in D\}$. Therefore $IdTerm(\mathcal{A}) = Pol(Inv(\Gamma) \cup \Gamma_{const})$ which implies that $\langle Inv(\Gamma) \cup \Gamma_{const} \rangle = Inv(IdTerm(\mathcal{A}))$. In other words, considering only the full idempotent reduct of an algebra is equivalent to considering only those constraint languages in which we have access to constants corresponding to each domain element.

Theorem 2.25 ([5]). *A finite surjective algebra is tractable if and only if its full idempotent reduct is tractable. Furthermore, a finite surjective algebra is NP-complete if and only if its full idempotent reduct is NP-complete.*

Another useful tool to analyse the complexity of algebras is to study their subalgebras and homomorphic images. In many cases this makes it possible to consider an algebra similar to the original one but with smaller domain. This in turn corresponds to considering instead of the original constraint language a constraint language over a smaller domain.

Definition 2.26. Let $\mathcal{A}_1 = \langle D_1, F_1 \rangle$ be an algebra and $D_2 \subseteq D_1$ such that for any $f \in F_1$ and any tuple $\langle t_1, \dots, t_k \rangle \in (D_2)^k$ where k is the arity of f ,

2.4 Applications

$f(t_1, \dots, t_k) \in D_2$. Then the algebra $\mathcal{A}_2 = \langle D_2, F|_{D_2} \rangle$ is called a **subalgebra** of \mathcal{A}_1 , where $F|_{D_2}$ is the set of restrictions of all the operations in F_1 to D_2 .

Definition 2.27. Let $\mathcal{A}_1 = \langle D_1, F_1 \rangle$ and $\mathcal{A}_2 = \langle D_2, F_2 \rangle$ be algebras such that $F_1 = \{f_i^1 : i \in I\}$ and $F_2 = \{f_i^2 : i \in I\}$, where both f_i^1 and f_i^2 are k_i -ary for all $i \in I$ and I is an index set. A map $\phi : D_1 \rightarrow D_2$ is called a **homomorphism** from \mathcal{A}_1 to \mathcal{A}_2 if

$$\phi(f_i^1(d_1, \dots, d_{k_i})) = f_i^2(\phi(d_1), \dots, \phi(d_{k_i}))$$

for all $i \in I$ and all $d_1, \dots, d_{k_i} \in D_1$. Furthermore, if ϕ is surjective then \mathcal{A}_2 is called a **homomorphic image** of \mathcal{A}_1 .

Definition 2.28. A homomorphic image of a subalgebra of an algebra \mathcal{A} is called a **factor** of \mathcal{A} .

Theorem 2.29 ([5]). *If \mathcal{A} is a tractable finite algebra then so is every factor of \mathcal{A} . If \mathcal{A} has any NP-complete factor then \mathcal{A} is NP-complete.*

2.4 Applications

In this section we state some important results related to the algebraic approach described above. We need to define the following operations.

Definition 2.30. Let f be a k -ary operation on a set D .

- If $k = 2$ and f is associative, i.e. $f(x, f(y, z)) = f(f(x, y), z)$, commutative, i.e. $f(x, y) = f(y, x)$, and idempotent, i.e. $f(x, x) = x$ then f is called a **semilattice operation**;
- If f satisfies the identity $f(x_1, \dots, x_k) \in \{x_1, \dots, x_k\}$ then f is called a **conservative operation**;
- If $k \geq 3$ and f satisfies the identities $f(y, x, \dots, x) = f(x, y, x, \dots, x) = \dots = f(x, \dots, x, y) = x$ then f is called a **near-unanimity operation**;

2.4 Applications

- If $k = 3$ and f satisfies the identities $f(y, x, x) = f(x, x, y) = y$ then f is called a **Mal'tsev operation**;
- If $k \geq 3$ and if for all $d_1, d_2 \in D$,
 - $f(y, x, \dots, x) = f(x, y, \dots, x) = \dots = f(x, x, \dots, y) = x$ for all $x, y \in \{d_1, d_2\}$ or
 - $f(y, x, \dots, x) = f(x, x, \dots, y)$ for all $x, y \in \{d_1, d_2\}$

then f is called a **generalized majority-minority operation**.

- Assume that there exists a non-constant unary operation g on D and an index $i, 1 \leq i \leq k$ such that f satisfies the identity $f(x_1, \dots, x_k) = g(x_i)$. Then f is called an **essentially unary operation**. If g is the identity operation then f is called a **projection**;
- If $k \geq 3$ and f satisfies the identity $f(x_1, \dots, x_k) = x_i$ for some fixed i whenever $|\{x_1, \dots, x_k\}| < k$ but f is not a projection then f is called a **semiprojection**.

Note that both near-unanimity and Mal'tsev operations are generalized majority-minority operations. Now we are ready to summarize recent progress towards the Feder-Vardi dichotomy conjecture in Theorem 2.31 and Theorem 2.32.

Theorem 2.31. *For any constraint language Γ over a finite set D , $\text{CSP}(\Gamma)$ is tractable if one of the following holds:*

1. $\text{Pol}(\Gamma)$ contains a semilattice operation ([24]);
2. $\text{Pol}(\Gamma)$ contains a conservative commutative binary operation ([6]);
3. $\text{Pol}(\Gamma)$ contains a k -ary near-unanimity operation ([22]);
4. $\text{Pol}(\Gamma)$ contains a Mal'tsev operation ([4, 3]).

2.4 Applications

5. $\text{Pol}(\Gamma)$ contains a generalized majority-minority operation ([11]).

Note that 3 and 4 follows from 5 but 3 and 4 were proved before 5. Also, it has recently been shown that if the algebra associated with a constraint language Γ has few subpowers then $\text{CSP}(\Gamma)$ is tractable ([19]). Furthermore, if $\text{Pol}(\Gamma)$ contains a generalized majority-minority operation then the associated algebra has few subpowers.

Rosenberg's analysis of minimal clones [42, 45] gives the following theorem.

Theorem 2.32. *For any reduced constraint language Γ on a finite set D , at least one of the following conditions holds:*

1. $\text{Pol}(\Gamma)$ contains a constant operation (tractable);
2. $\text{Pol}(\Gamma)$ contains a near-unanimity operation of arity 3 (tractable);
3. $\text{Pol}(\Gamma)$ contains a Mal'tsev operation (tractable);
4. $\text{Pol}(\Gamma)$ contains an idempotent binary operation which is not a projection (unknown);
5. $\text{Pol}(\Gamma)$ contains a semi-projection (unknown);
6. $\text{Pol}(\Gamma)$ contains only essentially unary surjective operations (NP-complete).

In Theorem 2.32, we indicated after each case whether the corresponding CSP is tractable, NP-complete or the complexity is unknown. Case 1 is trivially tractable because each (non-empty) relation in Γ contains a tuple $\langle d, \dots, d \rangle$ where d is the value of the constant operation. Case 2 and 3 are tractable by Theorem 2.31. Cases 4 and 5 are inconclusive although over the Boolean domain it is not hard to show that case 4 is tractable and case 5 cannot occur.

2.4 Applications

Case 6 is NP-complete. To see this when $|D| = 2$ observe that in this case, $Inv(Pol(\Gamma))$ includes the not-all-equal relation

$$N_D = D^3 \setminus \{\langle d_0, d_0, d_0 \rangle, \langle d_1, d_1, d_1 \rangle\}.$$

As in Example 1.3, it is easy to see that $CSP(\{N_D\})$ corresponds to the NOT-ALL-EQUAL SATISFIABILITY problem [44] which is NP-complete. If $|D| > 2$ then observe that $Inv(Pol(\Gamma))$ includes the disequality relation \neq_D and $CSP(\{\neq_D\})$ is NP-complete (see Example 1.7). Notice that Theorem 2.32 implies Schaefer's Dichotomy Theorem.

A similar argument shows the following slightly more general result.

Theorem 2.33 ([21]). *Any set of essentially unary operations over a finite set is NP-complete.*

There is a long-standing conjecture that this condition is sufficient to characterize *all* forms of intractability of constraint languages ([7]).

Chapter 3

Datalog and CSPs in NL and L

In this chapter we rephrase the CSP problem for a fixed constraint language Γ as a restricted HOMOMORPHISM PROBLEM. We introduce the database-inspired logic programming language Datalog and its linear and symmetric restrictions. We describe the relations between linear and symmetric Datalog and logarithmic space and non-deterministic logarithmic, respectively. We also introduce second-order logic fragments that have an expressive power equivalent to that of linear and symmetric Datalog. Finally, we show how to capture L and NL using symmetric Datalog and linear Datalog, respectively. The results of this chapter, obtained with Benoît Larose and Pascal Tesson, appeared in [14].

3.1 Basic Notions

A **vocabulary** is a finite set of relation symbols. In the following, τ denotes a vocabulary. Every relation symbol R in τ has an associated **arity** r . A **relational structure** \mathbf{A} over the vocabulary τ consists of a set A called the **universe** of \mathbf{A} , and a relation $R^{\mathbf{A}} \subseteq A^r$ for every relation symbol $R \in \tau$, where r is the arity of R . We also call such a relational structure a τ -structure. The set of all τ -structures is denoted by $\text{STR}[\tau]$. We use boldface letters

3.2 Homomorphism Problems

to denote relational structures. Note the similarity between the concept of constraint languages and relational structures.

Definition 3.1. Let $\mathbf{A} = \langle A; R_1^{\mathbf{A}}, \dots, R_q^{\mathbf{A}} \rangle$ and $\mathbf{B} = \langle B; R_1^{\mathbf{B}}, \dots, R_q^{\mathbf{B}} \rangle$ be relational structures where $R_i^{\mathbf{A}}$ and $R_i^{\mathbf{B}}$ are both r_i -ary, for all $i = 1, \dots, q$. A function $h : A \rightarrow B$ is called a **homomorphism** from \mathbf{A} to \mathbf{B} if

$$\langle f(a_1), \dots, f(a_{r_i}) \rangle \in R_i^{\mathbf{B}}$$

whenever $\langle a_1, \dots, a_{r_i} \rangle \in R_i^{\mathbf{A}}$, for all $i = 1, \dots, q$. If there is a homomorphism from \mathbf{A} to \mathbf{B} we denote this fact by $\mathbf{A} \rightarrow \mathbf{B}$. If there is a homomorphism h from \mathbf{A} to \mathbf{B} it is denoted by $\mathbf{A} \xrightarrow{h} \mathbf{B}$. The relational clone $\langle \mathbf{A} \rangle$ of a relational structure \mathbf{A} is defined similarly to the relational clone of a constraint language (see Definition 2.2).

3.2 Homomorphism Problems

Consider the following problem. We fix a relational structure \mathbf{B} and a relational structure \mathbf{A} is given as input. The task is to decide if there is a homomorphism from \mathbf{A} to \mathbf{B} . We denote the set of relational structures that admit a homomorphism to \mathbf{B} by $\text{Hom}(\mathbf{B})$. To see that this homomorphism problem is equivalent to a CSP, think of the elements in \mathbf{A} as variables, the elements in \mathbf{B} as values, the tuples in the relations in \mathbf{A} as constraint scopes and the relations of \mathbf{B} as constraint relations. It is easy to see that the solutions to this CSP are precisely the homomorphisms from \mathbf{A} to \mathbf{B} . The other direction is similar.

From now on, we abuse notation and when we write $\text{CSP}(\mathbf{B})$ we actually mean $\text{Hom}(\mathbf{B})$. Note that now \mathbf{B} in $\text{CSP}(\mathbf{B})$ denotes a relational structure, not a constraint language.

3.3 Datalog

Datalog is a query and rule language for deductive databases that syntactically is a subset of Prolog. Its origins date back to the beginning of logic programming but it became prominent as a separate area around 1978. The term Datalog was coined in the mid 1980's by a group of researchers interested in database theory (<http://en.wikipedia.org/wiki/Datalog>).

3.3.1 Syntax

A Datalog program over a vocabulary τ is a finite set of rules of the form

$$t_0 \leftarrow t_1; \dots; t_m$$

where each t_i is an atomic formula $R(v_1, \dots, v_m)$. The relational predicates in the heads (leftmost predicate in the rule) of the rules are called **intensional database predicates (IDBs)** and are not in τ . All other relational predicates are called **extensional database predicates (EDBs)** and are in τ .

A rule of a Datalog program is said to be **linear** if its body contains at most one IDB and is said to be **non-recursive** if its body contains only EDBs. A linear but recursive rule is of the form

$$I_1(\bar{x}) \leftarrow I_2(\bar{y}); E_1(\bar{z}_1); \dots; E_k(\bar{z}_k)^1$$

where I_1, I_2 are IDBs and the E_i are EDBs. Each such rule has a **symmetric rule**

$$I_2(\bar{y}) \leftarrow I_1(\bar{x}); E_1(\bar{z}_1); \dots; E_k(\bar{z}_k).$$

A Datalog program \mathfrak{D} is said to be linear if all its rules are linear. We further say that \mathfrak{D} is a **symmetric Datalog program** if the symmetric of

¹Note that the variables occurring in $\bar{x}, \bar{y}, \bar{z}_i$ are not necessarily distinct.

3.3 Datalog

any recursive rule of \mathfrak{D} is also a rule of \mathfrak{D} .

Let j and k be integers such that $0 \leq j \leq k$. We say that a Datalog program \mathfrak{D} has **width** (j, k) if every rule of \mathfrak{D} has at most k variables and at most j variables in the head. (j, k) -Datalog denotes the set of all Datalog programs of width (j, k) .

3.3.2 Semantics

Before we give the formal definition we give an example.

Example 3.2. Consider the problem of two-coloring. Clearly, an undirected graph is two-colorable if and only if it is homomorphic to an undirected edge. In fact, two-coloring is the problem $\text{CSP}(\mathbf{B})$ where \mathbf{B} has domain $\{0, 1\}$ and contains only the inequality relation $\{\langle 0, 1 \rangle, \langle 1, 0 \rangle\}$. Observe that $\neg\text{CSP}(\mathbf{B})$ is the set of graphs which contain a cycle of odd length. The following Datalog program \mathfrak{D} defines $\neg\text{CSP}(\mathbf{B})$ because the goal predicate becomes non-empty if and only if the input graph contains an odd cycle.

$$\begin{aligned} O(x, y) &\leftarrow E(x, y) \\ O(x, y) &\leftarrow O(x, w); E(w, z); E(z, y) \\ O(x, w) &\leftarrow O(x, y); E(w, z); E(z, y) \\ G &\leftarrow O(x, x) \end{aligned}$$

Here E is the binary EDB representing the adjacency relation in the input graph, O is a binary IDB whose intended meaning is “there exists an odd length path from x to y ” and G is the 0-ary goal predicate. Intuitively, the program begins with finding a path of length one using the only non-recursive rule and after, iteratively increasing the path length, each time by two. Whenever the path begins and ends at the same vertex x , the goal predicate becomes non-empty indicating the presence of a cycle of odd length.

Note that the two middle rules form a symmetric pair. In the above

3.3 Datalog

description, we have not included the symmetric of the last rule. In fact, the fairly counterintuitive rule $O(x, x) \leftarrow G$ can be added to the program without changing the class of structures accepted by the program since the rule only becomes relevant if an odd cycle has already been detected in the graph.

Let us define the semantics of a Datalog program \mathfrak{D} over τ . Let τ_{IDB} be the set of intensional predicates and let $\tau' = \tau \cup \tau_{\text{IDB}}$. \mathfrak{D} defines a function $\Phi_{\mathfrak{D}} : \text{STR}[\tau] \rightarrow \text{STR}[\tau']$. Intuitively, $\Phi_{\mathfrak{D}}(\mathbf{A})$ is the smallest τ' -structure over the universe A of \mathbf{A} such that for each rule $P(\bar{x}) \leftarrow P_1(\bar{y}_1); \dots; P_m(\bar{y}_m)$ of \mathfrak{D} , and any interpretation of the variables the implication defined by the rule is valid.

Formally, for a τ -structure \mathbf{A} , let $\mathbf{A}^{\mathfrak{D}[0]}$ denote the τ' structure over A such that $R^{\mathbf{A}^{\mathfrak{D}[0]}} = R^{\mathbf{A}}$ if $R \in \tau$ and $R^{\mathbf{A}^{\mathfrak{D}[0]}} = \emptyset$ if $R \in \tau_{\text{IDB}}$. $\mathbf{A}^{\mathfrak{D}[n+1]}$ is defined inductively as follows. First, if $R \in \tau$ then $R^{\mathbf{A}^{\mathfrak{D}[n+1]}} = R^{\mathbf{A}^{\mathfrak{D}[n]}} = R^{\mathbf{A}}$

Suppose now that $R \in \tau_{\text{IDB}}$ of arity r . Let $h \leftarrow b_1; \dots; b_m$ be a rule of \mathfrak{D} over the variables x_1, \dots, x_k . An **interpretation** of the variables of the rule over the domain A is a function $f : \{x_1, \dots, x_k\} \rightarrow A$. Then $R^{\mathbf{A}^{\mathfrak{D}[n+1]}}$ is defined as the union of $R^{\mathbf{A}^{\mathfrak{D}[n]}}$ and all r -tuples $\langle a_1, \dots, a_r \rangle \in A^r$ such that for some rule \mathfrak{R} with head $R(x_{i_1}, \dots, x_{i_r})$ and some interpretation f such that $f(x_{i_j}) = a_j$ with $1 \leq j \leq r$ we have for all predicates $T(x_{l_1}, \dots, x_{l_q})$ in the body of \mathfrak{R} that $\langle f(x_{l_1}), \dots, f(x_{l_q}) \rangle \in T^{\mathbf{A}^{\mathfrak{D}[n]}}$.

By definition, $R^{\mathbf{A}^{\mathfrak{D}[n]}} \subseteq R^{\mathbf{A}^{\mathfrak{D}[n+1]}}$ and so the iterative process above is monotone and has a least fixed point which we denote as $R^{\mathbf{A}^{\mathfrak{D}}}$. Accordingly, we define $\Phi_{\mathfrak{D}}(\mathbf{A})$ as the τ' -structure defined by the relations $R^{\mathbf{A}^{\mathfrak{D}}}$.

As defined above, the output of a Datalog program is a τ' -structure but we want to view a Datalog program \mathfrak{D} primarily as a way to define a class of τ -structures. For this purpose, we chose in \mathfrak{D} an IDB G known as the **goal predicate** and say that the τ -structure \mathbf{A} is **accepted** by \mathfrak{D} if $G^{\mathbf{A}^{\mathfrak{D}}}$ is non-empty. Note that changing the arity of the goal predicate to zero does not affect whether \mathfrak{D} accepts or rejects a structure and therefore unless otherwise

3.3 Datalog

stated, in the rest of this thesis we assume that all Datalog programs have a goal predicate of arity zero. Note that in this case \mathfrak{D} accepts if $G^{\mathfrak{A}^{\mathfrak{D}}}$ contains the tuple ϵ of arity zero and rejects otherwise. A class \mathcal{C} of τ -structures is *definable* in Datalog if there exists a program \mathfrak{D} such that $\mathbf{A} \in \mathcal{C}$ if and only if \mathfrak{D} accepts \mathbf{A} . Note that any such class \mathcal{C} is homomorphism closed, i.e. if $\mathbf{A} \xrightarrow{h} \mathbf{B}$ and $\mathbf{A} \in \mathcal{C}$ then $\mathbf{B} \in \mathcal{C}$. (This is easy to see using the concept of derivation trees in the next section.)

Furthermore, as noted in Example 3.2, the presence of the symmetric of a rule whose head is the goal predicate also does not affect whether \mathfrak{D} accepts or rejects a structure. Therefore in our symmetric Datalog programs we always exclude the symmetric of a rule whose head is the goal predicate.

3.3.3 Derivation Trees

Assume that a Datalog program \mathfrak{D} accepts a structure \mathbf{A} . Intuitively, a derivation tree is a tree-representation of the “proof” that \mathfrak{D} accepts \mathbf{A} . Before we give the formal definition we illustrate the concept with an example.

Example 3.3. Let \mathfrak{D} be a (linear) Datalog program whose input vocabulary contains a binary relation symbol E and two unary relation symbols S and T , and accepts if and only if there is a path in E from a vertex in S to a vertex in T . For example, let \mathfrak{D} be

$$\begin{aligned} I(y) &\leftarrow S(y) \\ I(y) &\leftarrow I(x); E(x, y) \\ G &\leftarrow I(y); T(y) \end{aligned}$$

with goal predicate G . Let \mathbf{A} be the input structure in Figure 3.3. Notice that there is a path v_5, v_6, v_3, v_4 from a vertex in S to a vertex in T . Therefore one possible derivation tree for \mathfrak{D} over \mathbf{A} is shown in Figure 3.3. Intuitively, the derivation tree follows the path from v_5 to v_4 .

3.3 Datalog

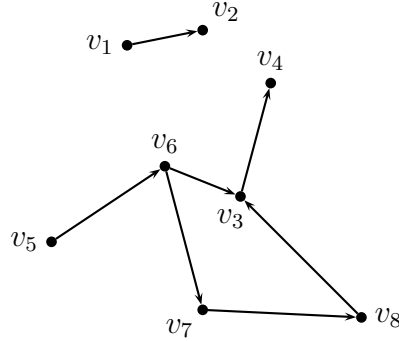


Figure 3.1: The input structure \mathbf{A} with $S = \{\langle v_5 \rangle\}$ and $T = \{\langle v_4 \rangle\}$.

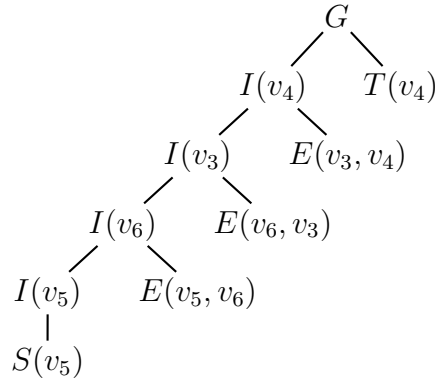


Figure 3.2: A derivation tree for \mathfrak{D} over \mathbf{A} .

Now we define a derivation tree formally. A **derivation tree** \mathcal{T} for a Datalog program \mathfrak{D} over a structure \mathbf{A} is a tree such that:

- The root of \mathcal{T} is the goal predicate;
- An internal node (including the root) together with its children correspond to a rule \mathfrak{R} of \mathfrak{D} in the following sense. The internal node is the head IDB of \mathfrak{R} and the children are the predicates in the body of \mathfrak{R} ;
- The internal nodes of \mathcal{T} are IDB predicates $I(\bar{a})$ where if I has arity r then $\bar{a} \in A^r$;

3.4 Datalog and CSPs

- The predicate children of a parent node inherit an instantiation to elements of A from their parents;
- The leaf nodes of \mathcal{T} are EDB predicates $E(\bar{b})$ such that:
 - If E has arity s then $b \in A^s$;
 - $b \in E^{\mathbf{A}}$.

It is easy to see that a derivation tree for a Datalog program \mathfrak{D} and a structure \mathbf{A} exist if and only if \mathfrak{D} accepts \mathbf{A} .

3.4 Datalog and CSPs

3.4.1 The Relation Between Datalog and CSPs

It is sometimes possible to define the set of structures $\text{-CSP}(\mathbf{B})$ in Datalog. Here we elaborate this relationship between Datalog and CSPs.

A **quasi-ordering** on a set S is a reflexive and transitive relation \leq on S . Let $\langle S, \leq \rangle$ be a quasi-ordered set. Let $S', S'' \subseteq S$. S' is a **filter** if it is closed under \leq upward, i.e. if $x \in S'$ and $x \leq y$ then $y \in S'$. The **filter generated** by S'' is the set $F(S'') = \{y \in S : \exists x \in S'' x \leq y\}$. S' is an **ideal** if it is closed under \leq downward; that is, if $x \in S'$ and $y \leq x$ then $y \in S'$. The **ideal generated** by S'' is the set $I(S'') = \{y \in S : \exists x \in S'' y \leq x\}$. Note that every subset S' of S and its complement $S \setminus S'$ satisfy the following relation: S' is an ideal if and only if $S \setminus S'$ is a filter.

Let \rightarrow be the homomorphism relation between τ -structures and observe that $\langle \text{STR}[\tau], \rightarrow \rangle$ is a quasi-ordered set. Furthermore, a set of τ -structures \mathcal{C} is an ideal if

$$\mathbf{B} \in \mathcal{C}, \mathbf{A} \rightarrow \mathbf{B} \implies \mathbf{A} \in \mathcal{C}.$$

Now let I be an ideal of $\langle S, \leq \rangle$. We say that a set $O \subseteq S$ forms an

3.4 Datalog and CSPs

obstruction set for I if

$$x \in I \text{ iff } \forall y \in O \text{ it is the case that } y \not\leq x.$$

In particular, let $I \subseteq \text{STR}[\tau]$ be an ideal. Then $O \subseteq \text{STR}[\tau]$ is an obstruction set for I if

$$\mathbf{B} \in I \text{ iff } \forall \mathbf{O} \in O \text{ it is the case that } \mathbf{O} \not\rightarrow \mathbf{B}.$$

Observe that for any relational structure \mathbf{B} , $\text{CSP}(\mathbf{B}) = \text{I}(\mathbf{B})$. For $\text{CSP}(\mathbf{B})$, sometimes it is possible to define the filter of τ -structures $\neg\text{CSP}(\mathbf{B})$ in Datalog. Notice also the simple fact that $\neg\text{CSP}(\mathbf{B})$ is an obstruction set for $\text{CSP}(\mathbf{B})$. The following lemma is an example that demonstrates the above definitions and it will be needed in Chapter 5.

Lemma 3.4. *Let $\tau = \langle E, S, T \rangle$ be a vocabulary containing a binary and two unary relation symbols. Let Δ be a τ -structure as follows. Let $\{0, 1\}$ be the domain, E^Δ be the less-than-or-equal-to relation, i.e. $\{\langle 0, 0 \rangle \langle 0, 1 \rangle \langle 1, 1 \rangle\}$, $S^\Delta = \{1\}$ and $T^\Delta = \{0\}$. Let O be the set of all τ -structures \mathbf{O} such that*

- $E^\mathbf{O}$ is a single directed path (including the path of length zero);
- $S^\mathbf{O} = \{s\}$, where s is the first vertex of the directed path;
- $T^\mathbf{O} = \{t\}$, where t is the last vertex of the directed path.

Then O is an obstruction set for $\text{CSP}(\Delta)$.

Proof. Let \mathbf{A} be a τ -structure and assume that for some $\mathbf{O} \in O$, $\mathbf{O} \rightarrow \mathbf{A}$. Then $E^\mathbf{A}$ contains a path from some vertex in $S^\mathbf{A}$ to some vertex in $T^\mathbf{A}$ and clearly, $\mathbf{A} \not\rightarrow \Delta$.

Conversely, assume that for all $\mathbf{O} \in O$, $\mathbf{O} \not\rightarrow \mathbf{A}$ and therefore there are no vertices $u \in S^\mathbf{A}$ and $v \in T^\mathbf{A}$ such that there is path from u to v in $E^\mathbf{A}$. We construct a homomorphism h from \mathbf{A} to Δ . Map each vertex in the connected component of $S^\mathbf{A}$ to 1 and map the rest of the vertices to 0.

3.4 Datalog and CSPs

Observe that each vertex in $S^{\mathbf{A}}$ is mapped to 1 and each vertex in $T^{\mathbf{A}}$ is mapped to 0. Furthermore, if $\langle a, b \rangle$ is an edge in $E^{\mathbf{A}}$ then we did not map a to 1 and b to 0 because if a is mapped to 1 then b must also be in the connected component of $S^{\mathbf{A}}$. This completes the construction of h . \square

3.4.2 Canonical Datalog Programs

In this section we define the **canonical (j, k) -Datalog program for $\neg\text{CSP}(\mathbf{B})$** and prove that $\neg\text{CSP}(\mathbf{B})$ can be solved by a (j, k) -Datalog program if and only if the canonical (j, k) -Datalog program solves it.

The canonical (j, k) -Datalog program for $\neg\text{CSP}(\mathbf{B})$, $\text{Can}(\mathbf{B})$, is constructed in the following way. Let \mathcal{C} be the set of all at most j -ary relations definable over \mathbf{B} by a primitive positive formula. In addition, add to \mathcal{C} the 0-ary empty relation. Construct an IDB I_R for each $R \in \mathcal{C}$. Let R_1, \dots, R_m be elements of \mathcal{C} where we allow repetition. Let \bar{E} be an arbitrary conjunction of atomic formulae. Consider the implication

$$R(\bar{x}) \leftarrow R_1(\bar{x}, \bar{y}_1) \wedge \dots \wedge R_{m-1}(\bar{x}, \bar{y}_{m-1}) \wedge \bar{E}(\bar{x}, \bar{y}_m)^2. \quad (3.1)$$

If implication 3.1 holds and it contains at most k variables then the rule

$$I_R(\bar{x}) \leftarrow I_{R_1}(\bar{x}, \bar{y}_1); \dots; I_{R_{m-1}}(\bar{x}, \bar{y}_{m-1}); \bar{E}(\bar{x}_m, \bar{y}_m) \quad (3.2)$$

is added to $\text{Can}(\mathbf{B})$. The IDB whose subscript is the 0-ary empty relation serves as the goal IDB. Because there are finitely many inequivalent implications, the canonical (j, k) -Datalog program is finite. We say that implication 3.1 is the **associated implication** of rule 3.2.

Theorem 3.5 ([15]). *$\neg\text{CSP}(\mathbf{B})$ can be solved by a (j, k) -Datalog program if and only if the canonical (j, k) -Datalog program solves it.*

²Note again that the variables occurring in \bar{x} and the \bar{y}_i are not necessarily distinct.

3.4 Datalog and CSPs

Proof. One direction is trivial. Let \mathfrak{D} be a (j, k) -Datalog program that defines $\text{-CSP}(\mathbf{B})$. First assume that \mathfrak{D} rejects a structure \mathbf{A} . We show that $\text{Can}(\mathbf{B})$ also rejects \mathbf{A} . Because \mathfrak{D} rejects \mathbf{A} we can find a homomorphism h from \mathbf{A} to \mathbf{B} . For the sake of contradiction assume that $\text{Can}(\mathbf{B})$ accepts \mathbf{A} witnessed by the derivation tree \mathcal{T} . The associated implications of the rules used in \mathcal{T} correspond via the substitution $x \mapsto h(x)$ to a set of valid implications in \mathbf{B} . This set of implications implies the 0-ary empty relation corresponding to the goal IDB which is a contradiction.

Conversely, we show that if \mathfrak{D} accepts a structure \mathbf{A} then $\text{Can}(\mathbf{B})$ also accepts \mathbf{A} . Let \mathcal{T} be a derivation tree for \mathfrak{D} over \mathbf{A} . Let the IDBs of \mathcal{T} be $J_1(\bar{a}_1), J_2(\bar{a}_2), \dots, J_q(\bar{a}_q)$ from the bottom level of \mathcal{T} to the root (then J_q must be the root). We construct a sequence of primitive positive formulae $\varphi_1, \dots, \varphi_q$ with at most j free variables and corresponding IDBs $I_{\varphi_1}, \dots, I_{\varphi_q}$ which are in $\text{Can}(\mathbf{B})$.

Let $J_i(\bar{y}) \leftarrow E_1(\bar{y}_1); \dots; E_n(\bar{y}_n)$ be the rule of corresponding to an IDB $J_i(\bar{a}_i)$ of \mathcal{T} at the bottommost level. Define φ_i to be the formula $\exists \bar{z}' E_1(\bar{y}_1) \wedge \dots \wedge E_n(\bar{y}_n)$ where \bar{z}' is a tuple of variables containing all variables not in the head of this rule. Define the IDB I_{φ_1} from J_i by keeping only one copy of those variables which are free in φ_i . Define \bar{a}'_i from \bar{a}_i accordingly. It is clear that $I_{\varphi_i}(\bar{a}'_i)$ is derived by $\text{Can}(\mathbf{B})$. Repeat this construction for each IDB of \mathcal{T} at the bottommost layer.

Now inductively, assume that we completed this construction for each IDB of \mathcal{T} up to the ℓ -th level counting from the bottom. Let $J_i(\bar{y}) \leftarrow R_1(\bar{y}_1); \dots; R_n(\bar{y}_n)$ be a rule of \mathcal{T} at the $(\ell + 1)$ -th level. Notice that now R_p could be an IDB. Define φ_i to be the formula $\exists \bar{z}' R_1(\bar{y}_1) \wedge \dots \wedge R_n(\bar{y}_n)$ where \bar{z}' is a tuple of variables containing all variables not in the head of this rule and any R_p which is an IDB stands for φ_p in φ_i . Define the IDB I_{φ_i} from J_i by keeping only one copy of those variables which are free in φ_i . Define \bar{a}'_i from \bar{a}_i accordingly. It is easy to see that $I_{\varphi_i}(\bar{a}'_i)$ is derived by $\text{Can}(\mathbf{B})$. Repeat this construction for the remaining IDBs in \mathcal{T} .

3.5 The Complexity of Linear and Symmetric Datalog

It remains to show that φ_m is false over \mathbf{B} . Assume that φ_m is true over \mathbf{B} . Rewrite φ_m in prenex normal form. Assume that the vocabulary of \mathbf{B} is τ . Construct a τ -structure \mathbf{F}_{φ_m} from φ_m by considering each variable of φ_m as a distinct element of the universe F . For each relation symbol $R(\bar{x})$ in φ_m add the tuple \bar{x} to $R^{\mathbf{F}_{\varphi_m}}$. A simple modification of \mathcal{T} shows that \mathfrak{D} accepts \mathbf{F}_{φ_m} .

For the sake of contradiction assume that φ_m is true over \mathbf{B} . Then it is easy to see that there is a homomorphism from \mathbf{F}_{φ_m} to \mathbf{B} which implies that \mathfrak{D} rejects \mathbf{F}_{φ_m} . This leads to a contradiction. \square

3.5 The Complexity of Linear and Symmetric Datalog

In this section we show that symmetric Datalog programs can be evaluated in logarithmic space and linear Datalog programs can be evaluated in non-deterministic logarithmic space.

Theorem 3.6. *A symmetric Datalog program \mathfrak{D} can be evaluated in logarithmic space. In fact, there exists a logspace transducer which on input \mathbf{A} produces some representation of $\Phi_{\mathfrak{D}}(\mathbf{A})$.*

Proof. Suppose that I_1, \dots, I_s are the IDBs in \mathfrak{D} . Let \mathbf{A} be the input τ -structure and let n denote the size of its universe A . We define the **execution graph** $G_{\mathbf{A}}$ as follows: for each I_j of arity k , we introduce for each of the n^k k -tuples of A^k a vertex labeled $I_j(a_1, \dots, a_k)$. Furthermore, we add an extra vertex labeled S . Edges are now determined by the EDBs in \mathbf{A} and the rules of \mathfrak{D} . We add an edge from $I_j(a_1, \dots, a_k)$ to $I_{j'}(b_1, \dots, b_\ell)$ if \mathfrak{D} contains a rule of the form³

$$I_{j'}(\bar{x}) \leftarrow I_j(\bar{y}); E_{s_1}(\bar{z}_1); \dots; E_{s_r}(\bar{z}_r)$$

³Note again that the variables occurring in \bar{x}, \bar{y} , and the \bar{z}_i are not necessarily distinct.

3.5 The Complexity of Linear and Symmetric Datalog

such that there exists an interpretation f of the variables of this rule over A such that $f(\bar{x}) = (b_1, \dots, b_\ell)$, $f(\bar{y}) = (a_1, \dots, a_k)$ and for each $f(\bar{z}_i) \in E_{s_t}^{\mathbf{A}}$. Informally, such an edge represents the fact that if \mathfrak{D} places the tuple (a_1, \dots, a_k) in I_j then it will also add the tuple (b_1, \dots, b_ℓ) to $I_{j'}$. We further add a bi-directional edge between our special vertex S and $I_j(a_1, \dots, a_k)$ if there exists a non-recursive rule

$$I_j(\bar{x}) \leftarrow E_{s_1}(\bar{z}_1); \dots; E_{s_r}(\bar{z}_r)$$

and an interpretation f such that $f(\bar{x}) = (a_1, \dots, a_k)$ and $f(\bar{z}_i) \in E_{s_t}^{\mathbf{A}}$ for each EDB occurring in the body. Since \mathfrak{D} is symmetric, the graph $G_{\mathbf{A}}$ is symmetric and can thus be regarded as an undirected graph. Moreover, the graph can clearly be constructed in logarithmic space and we have $(a_1, \dots, a_k) \in I_j^{\mathbf{A}^Q}$ if and only if the vertex $I_j(a_1, \dots, a_k)$ is reachable from S in $G_{\mathbf{A}}$. Since undirected connectivity can be computed in logarithmic space [41], a representation of $\Phi_{\mathfrak{D}}(\mathbf{A})$ can be produced in logarithmic space. \square

Corollary 3.7. *If $\neg\text{CSP}(\mathbf{B})$ is definable in symmetric Datalog then $\text{CSP}(\mathbf{B}) \in \text{L}$.*

An argument similar to the one in the proof of Theorem 3.6 can be used to show Theorem 3.8.

Theorem 3.8. *A linear Datalog program \mathfrak{D} can be evaluated in non-deterministic logarithmic space.*

Corollary 3.9. *If $\neg\text{CSP}(\mathbf{B})$ is definable in linear Datalog then $\text{CSP}(\mathbf{B}) \in \text{NL}$.*

3.6 Symmetric Datalog, Linear Datalog and Second Order Logic

In this section we define two second order logic fragments which have the same expressive power as linear and symmetric Datalog.

3.6.1 Symmetric Restricted Krom SNP

Let τ be a vocabulary. SNP is the class of sentences of the form

$$\exists S_1, \dots, S_l \forall v_1, \dots, v_m \varphi(v_1, \dots, v_m)$$

where S_1, \dots, S_l are second-order variables and φ is a quantifier-free first-order formula over the vocabulary $\tau \cup \{S_1, \dots, S_l\}$ with variables among v_1, \dots, v_m . We assume that φ is in CNF. In **monotone SNP**, every occurrence of a relation symbol from τ is negated. In **j -adic SNP**, every second order variable S_i , $1 \leq i \leq l$ has arity at most j . In **k -ary SNP**, the number of universally quantified first order variables is at most k , i.e. $m \leq k$. In **Krom SNP**, every clause of the quantifier-free first-order part φ has at most two occurrences of a second order variable. In **restricted Krom SNP**, every clause of the quantifier-free first-order part φ has at most one positive occurrence of a second-order variable and at most one negative occurrence of a second-order variable. **Symmetric restricted Krom SNP** is the subset of restricted Krom SNP formulae that contain with every clause of the form $\psi \vee S_i \vee \neg S_j$ also the clause $\psi \vee \neg S_i \vee S_j$ (where ψ contains no second-order variables).

Example 3.10. Consider again the problem of two-coloring from Example 3.2. It is easy to check that a graph with adjacency relation E is two-colorable if and only if the following symmetric restricted Krom SNP sentence

3.6 Symmetric Datalog, Linear Datalog and Second Order Logic

is true.

$$\begin{aligned}
& \exists O, G \forall x, y, z, w (O(x, y) \vee \neg E(x, y)) \wedge \\
& \quad (O(x, y) \vee \neg O(x, w) \vee \neg E(w, z) \vee \neg E(z, y)) \wedge \\
& \quad (\neg O(x, y) \vee O(x, w) \vee \neg E(w, z) \vee \neg E(z, y)) \wedge \\
& \quad (G \vee \neg O(x, x)) \wedge \\
& \quad \neg G,
\end{aligned}$$

where O has arity two and G has arity zero. Intuitively, O is defined by the first three clauses such that if $\langle x, y \rangle \in O$ then there is a path of odd length from x to y . The third clause is the symmetric of the second clause. The fourth clause checks if a path of odd length is a cycle and if yes then G becomes non-empty. But if G is non-empty then the last clause is false so the formula must evaluate to false. On the other hand, if there is no odd cycle in the input graph then G remains empty and the formula evaluates to true.

3.6.2 Second Order Logics for Linear and Symmetric Datalog

Let \mathbf{A} be a τ -structure and $\varphi(x_1, \dots, x_n)$ be a logical formula with free variables among x_1, \dots, x_n . We write $\mathbf{A}, a_1, \dots, a_n \models \varphi(x_1, \dots, x_n)$ to denote that $\phi(a_1, \dots, a_n)$ is true when the relations of \mathbf{A} are used in φ . The symbol \models stands for “models”. Note that in second order logic the free variables could be free relation variables.

Theorem 3.11. *A Datalog(\neg, \neq) is a Datalog program in which we allow the negation of the EDBs and inequalities. Let \mathcal{C} be a collection of τ -structures. Then 1 is equivalent to 2, and 3 is equivalent to 4.*

1. \mathcal{C} is definable in symmetric Datalog;

3.6 Symmetric Datalog, Linear Datalog and Second Order Logic

2. $\neg\mathcal{C}$ is definable in symmetric restricted Krom monotone SNP;
3. \mathcal{C} is definable in symmetric Datalog(\neg, \neq);
4. $\neg\mathcal{C}$ is definable in symmetric restricted Krom SNP.

A similar theorem can be stated with linear Datalog and restricted Krom SNP with an analogous proof [10]. We use the following lemma.

Lemma 3.12. *Let \mathfrak{D} be a Datalog(\neg, \neq) program over τ with IDBs I_1, \dots, I_m . Let*

$$\psi(I_1, \dots, I_m, x_1, \dots, x_n) = \bigwedge_{h \leftarrow b_1; \dots; b_q \in \mathfrak{D}} h \leftarrow (b_1 \wedge \dots \wedge b_q).$$

Let \mathbf{A} be a τ -structure such that there exist relations R_1, \dots, R_m such that

$$\mathbf{A}, R_1, \dots, R_m \models \forall x_1, \dots, x_n \psi(I_1, \dots, I_m, x_1, \dots, x_n).$$

Then $I_i^{\mathbf{A}^{\mathfrak{D}}}(t) \rightarrow R_i(t)$ for each $i, 1 \leq i \leq m$ and each $t \in A^r$ where r is the arity of I_i .

Notation: For convenience, we write $I_i^{\mathbf{A}^{\mathfrak{D}[j]}} \rightarrow R_i$ to denote that for each tuple $t \in A^r$ where r is the arity of I_i , $I_i^{\mathbf{A}^{\mathfrak{D}[j]}}(t) \rightarrow R_i(t)$. We write $I^{\mathbf{A}^{\mathfrak{D}[j]}} \rightarrow R$ to denote that for each $i, 1 \leq i \leq m$ and for each tuple $t \in A^r$ where r is the arity of I_i , $I_i^{\mathbf{A}^{\mathfrak{D}[j]}}(t) \rightarrow R_i(t)$.

Proof. Consider the sequence $\mathbf{A}^{\mathfrak{D}[0]}, \mathbf{A}^{\mathfrak{D}[1]}, \dots, \mathbf{A}^{\mathfrak{D}}$ and for the sake of contradiction assume that for some i and for some tuple $t' \in A^{r'}$, $I_i^{\mathbf{A}^{\mathfrak{D}}}(t')$ is true but $R_i(t')$ is false. Then there exists a smallest j , an index k , and a tuple $t \in A^r$ such that $I_k^{\mathbf{A}^{\mathfrak{D}[j]}}(t)$ is true but $R_k(t)$ is false.

Let $I_k(x_{k_1}, \dots, x_{k_r}) \leftarrow b_1; \dots; b_q$ be the rule which added t to $I_k^{\mathbf{A}^{\mathfrak{D}[j]}}$ using the interpretation $f : x_1, \dots, x_n \rightarrow A$. Notice that $t = \langle f(x_{k_1}), \dots, f(x_{k_r}) \rangle$. By the choice of j , $I^{\mathbf{A}^{\mathfrak{D}[j-1]}} \rightarrow R$. Therefore $\mathbf{A}, R_1, \dots, R_m, f(x_1), \dots, f(x_n) \models$

3.6 Symmetric Datalog, Linear Datalog and Second Order Logic

$b_1 \wedge \dots \wedge b_q$ and $b_1 \wedge \dots \wedge b_q \rightarrow R_k(f(x_{k_1}), \dots, f(x_{k_r}))$ in ψ . This implies that $R_k(t)$ is true which is a contradiction. \square

Proof of Theorem 3.11. We prove the equivalence of **1** and **2**. The equivalence of **3** and **4** is an obvious modification of the proof.

1 \rightarrow **2**: Let \mathfrak{D} be a symmetric Datalog program with IDB predicates I_1, \dots, I_m one of which is the goal predicate I_g . Let ψ be a CNF formula defined as

$$\begin{aligned} \psi(I_1, \dots, I_m, x_1, \dots, x_n) &= \neg I_g \wedge \bigwedge_{h \leftarrow b_1; \dots; b_q \in \mathfrak{D}} h \leftarrow (b_1 \wedge \dots \wedge b_q) \\ &\equiv \neg I_g \wedge \bigwedge_{h \leftarrow b_1; \dots; b_q \in \mathfrak{D}} h \vee \neg b_1 \vee \dots \vee \neg b_q. \end{aligned}$$

Let ϕ be the following symmetric restricted Krom monotone SNP sentence

$$\phi = \exists R_1, \dots, R_m \forall x_1, \dots, x_n \psi(I_1, \dots, I_m, x_1, \dots, x_n).$$

We show that ϕ is satisfied exactly by those structures that are rejected by \mathfrak{D} . Assume that \mathfrak{D} rejects \mathbf{A} . Then clearly,

$$\mathbf{A}, I_1^{\mathbf{A}^\mathfrak{D}}, \dots, I_m^{\mathbf{A}^\mathfrak{D}} \models \forall x_1, \dots, x_n \psi(I_1, \dots, I_m)$$

so $\mathbf{A} \models \phi$.

Conversely, assume that \mathbf{A} is a structure such that $\mathbf{A} \models \phi$. Then there exist relations R_1, \dots, R_m such that

$$\mathbf{A}, R_1, \dots, R_m \models \forall x_1, \dots, x_n \psi(I_1, \dots, I_m, x_1, \dots, x_n).$$

By Lemma 3.12, $I^{\mathbf{A}^\mathfrak{D}} \rightarrow R$ and in particular $I_g^{\mathbf{A}^\mathfrak{D}} \rightarrow R_g$. Therefore because R_g is false $I_g^{\mathbf{A}^\mathfrak{D}}$ is also false, i.e. \mathfrak{D} rejects \mathbf{A} .

2 \rightarrow **1**: Let $\phi = \exists I_1, \dots, I_m \forall x_1, \dots, x_n \psi(I_1, \dots, I_m, x_1, \dots, x_n)$ be an arbitrary sentence in symmetric restricted Krom monotone SNP. We rewrite ϕ

3.6 Symmetric Datalog, Linear Datalog and Second Order Logic

in an equivalent “implicational” form which we call ϕ' and the modified ψ becomes ψ' . Parallel to this, we construct a symmetric Datalog program \mathfrak{D} as follows.

1. Add a new second order variable I_{m+1} to the existential quantifier block of ϕ and a new clause to ψ , ($I_{m+1} = False$). The IDBs of \mathfrak{D} are I_1, \dots, I_{m+1} and I_{m+1} is the goal predicate. Let C be a clause of ψ (but not the newly added clause);
2. If $C = h \vee b_1 \vee \dots \vee b_q$ where h, b_1, \dots, b_q are literals and C contains a non-negated second order variable which we denoted by h then rewrite C as $h \leftarrow (\neg b_1 \wedge \dots \wedge \neg b_q)$. Add the rule $h \leftarrow \neg b_1; \dots; \neg b_q$ to \mathfrak{D} in which the IDBs are the second order variables of C ;
3. If $C = b_1 \vee \dots \vee b_q$ where C does not contain a non-negated second order variable then rewrite C as $I_{m+1} \leftarrow (\neg b_1 \wedge \dots \wedge \neg b_q)$. Add $I_{m+1} \leftarrow \neg b_1; \dots; \neg b_q$ to \mathfrak{D} .

Observe that \mathfrak{D} is a symmetric Datalog program. We show that \mathfrak{D} accepts exactly the same set of structures which falsify ϕ' . Assume that \mathfrak{D} rejects \mathbf{A} . Then clearly,

$$\mathbf{A}, I_1^{\mathbf{A}^{\mathfrak{D}}}, \dots, I_{m+1}^{\mathbf{A}^{\mathfrak{D}}} \models \forall x_1, \dots, x_n \psi'(I_1, \dots, I_m, x_1, \dots, x_n),$$

so $\mathbf{A} \models \phi'$.

Conversely, assume that \mathbf{A} is a structure such that $\mathbf{A} \models \phi'$. Then there exist relations R_1, \dots, R_{m+1} such that

$$\mathbf{A}, R_1, \dots, R_{m+1} \models \forall x_1, \dots, x_n \psi'(I_1, \dots, I_{m+1}, x_1, \dots, x_n).$$

By Lemma 3.12, $I_i^{\mathbf{A}^{\mathfrak{D}}} \rightarrow R_i, 1 \leq i \leq m+1$. In particular $I_{m+1}^{\mathbf{A}^{\mathfrak{D}}} \rightarrow R_{m+1}$ where R_{m+1} is forced to be false. Therefore $I_{m+1}^{\mathbf{A}^{\mathfrak{D}}}$ is false, i.e. \mathfrak{D} rejects \mathbf{A} . \square

3.7 Capturing NL and L Using Datalog

A **finite successor structure** is a structure whose domain is $\{0, 1, \dots, n-1\}$ (for some $n \in \mathbb{N}$) and whose vocabulary contains the two constant symbols min and max and the binary predicate S whose interpretations are the constants $0, n-1$ and the successor relation $S = \{\langle x, x+1 \rangle : x < n-1\}$. \mathcal{O} denotes the set of all finite successor structures.

A logic **captures** the complexity class C if for every problem $P \subseteq \mathcal{O}$, P is in C if and only if there exists a formula ψ in the logic such that $P = \{\mathcal{R} \in \mathcal{O} : \mathcal{R} \models \psi\}$.

Theorem 3.13. *Over the set of finite successor structures, symmetric Datalog(\neg, \neq) captures L.*

Proof. It follows from [20] and [41] that over the set of finite successor structures, $[STC_{\bar{x}, \bar{y}}\psi(\bar{x}, \bar{y})](\overline{min}, \overline{max})$ captures L. Here ψ is a quantifier-free first-order formula and $[STC_{\bar{x}, \bar{y}}\psi(\bar{x}, \bar{y})]$ denotes the reflexive, symmetric and transitive closure of the binary relation defined by ψ . Now we use an idea from Theorem 6.4 in [17].

If P is a problem in L then the complement of P can be defined by a formula $\phi = \neg[STC_{\bar{x}, \bar{y}}\psi(\bar{x}, \bar{y})](\overline{min}, \overline{max})$ where ψ is quantifier-free. Let $\bigvee_i \psi_i$ be the disjunctive normal form of ψ and build the formula

$$\exists R \forall \bar{x}, \bar{y}, \bar{z} R(\bar{x}, \bar{x}) \wedge \bigwedge_i (\psi_i(\bar{y}, \bar{z}) \rightarrow (R(\bar{x}, \bar{y}) \leftrightarrow R(\bar{x}, \bar{z}))) \wedge \neg R(\overline{min}, \overline{max}).$$

This formula is equivalent to ϕ and it can be rewritten as a symmetric restricted Krom SNP formula:

$$\begin{aligned} \exists R \forall \bar{x}, \bar{y}, \bar{z} R(\bar{x}, \bar{x}) \wedge \bigwedge_i (\neg \psi_i(\bar{y}, \bar{z}) \vee \neg R(\bar{x}, \bar{y}) \vee R(\bar{x}, \bar{z})) \wedge \\ \bigwedge_i (\neg \psi_i(\bar{y}, \bar{z}) \vee R(\bar{x}, \bar{y}) \vee \neg R(\bar{x}, \bar{z})) \wedge \neg R(\overline{min}, \overline{max}). \end{aligned}$$

3.7 Capturing NL and L Using Datalog

Now we use Theorem 3.11 to define P in symmetric Datalog(\neg, \neq).

Conversely, a symmetric Datalog(\neg, \neq) can be evaluated in L by a simple extension of Theorem 3.6. \square

A similar argument can be used using transitive closure instead of symmetric transitive closure to show that over the set of finite successor structures, linear Datalog(\neg, \neq) captures NL. Similarly, over the set of finite successor structures, Datalog(\neg, \neq) captures P (see for example [31]).

Chapter 4

A New Class of CSPs Expressible in Symmetric Datalog

In this chapter we show expressibility results related to symmetric Datalog. First we show that if $\neg\text{CSP}(\mathbf{B})$ is definable in symmetric Datalog and $\mathbf{C} \subseteq \langle \mathbf{B} \rangle$ then $\neg\text{CSP}(\mathbf{C})$ is also definable in symmetric Datalog. We then identify a large set of constraints closely related to implicational constraints such that the corresponding CSP is expressible in symmetric Datalog. The results of this chapter, obtained with Benoît Larose and Pascal Tesson, appeared in [14].

4.1 Symmetric Datalog and Relational Clones

In this section we prove the following result.

Theorem 4.1. *If \mathbf{B} is a relational structure such that $\neg\text{CSP}(\mathbf{B})$ is definable in symmetric Datalog and \mathbf{C} is a finite relational structure whose relations are defined by primitive positive formulas over \mathbf{B} then $\neg\text{CSP}(\mathbf{C})$ is definable in symmetric Datalog.*

4.1 Symmetric Datalog and Relational Clones

Proof. The theorem relies on results in [29]. Since \mathbf{C} is a finite subset of $\langle \mathbf{B} \rangle$ it can be obtained from the set \mathbf{B} by a finite sequence of applications of six basic constructions, five of which are shown in Lemma 6.5 of [29] to preserve expressibility in symmetric Datalog. It remains to show that if $\neg\text{CSP}(\mathbf{B})$ is expressible in symmetric Datalog then so is $\neg\text{CSP}(\mathbf{B} \cup \{=\})$. This is proved in Lemma 4.2. \square

It is interesting to compare this result with Theorem 2.3 in Chapter 2.

Lemma 4.2. *Let \mathbf{B} be a relational structure such that $\neg\text{CSP}(\mathbf{B})$ is expressible in symmetric Datalog. Then $\neg\text{CSP}(\mathbf{B} \cup \{=\})$ is also expressible in symmetric Datalog.*

Proof. First we describe a reduction from $\text{CSP}(\mathbf{B} \cup \{=\})$ to $\neg\text{CSP}(\mathbf{B})$. Let \mathbf{T} and \mathbf{T}' denote the “target” structures for $\text{CSP}(\mathbf{B})$ and $\text{CSP}(\mathbf{B} \cup \{=\})$, respectively. In other words, \mathbf{T} and \mathbf{T}' are the structures whose base set is B and whose basic relations are those of \mathbf{B} and $\mathbf{B} \cup \{=\}$, respectively. Given an input \mathbf{S}' for $\text{CSP}(\mathbf{T}')$, construct an input \mathbf{S} for $\text{CSP}(\mathbf{T})$ as follows.

Let E be the relation in \mathbf{S}' that corresponds to $\{=\}$ in \mathbf{T}' . We say that two k -tuples \bar{x} and \bar{x}' are E -related if $\text{STC}[E](x_i, x'_i)$ ¹ for each $1 \leq i \leq k$. We denote the fact that \bar{x} and \bar{x}' are E -related by $\bar{x} \equiv_E \bar{x}'$. Notice that \equiv_E is an equivalence relation.

Now we define the structure \mathbf{S} by specifying its basic relations. For each relation symbol R , let \bar{x} be in $R^{\mathbf{S}}$ whenever there exists \bar{x}' in $R^{\mathbf{S}'}$ such that $\bar{x} \equiv_E \bar{x}'$. We claim that \mathbf{S} admits a homomorphism to \mathbf{T} if and only if \mathbf{S}' admits a homomorphism to \mathbf{T}' . One direction is trivial.

Now suppose that $\mathbf{S} \xrightarrow{h} \mathbf{T}$ and let x and y be such that $\text{STC}[E](x, y)$. Let h' be the function obtained from h by setting $h'(x) = h(y)$ and $h'(z) = h(z)$ for all $z \neq x$. By construction, if a tuple of the relation $R^{\mathbf{S}}$ has an occurrence of x then the tuple obtained by replacing all occurrences of x by y is also in $R^{\mathbf{S}}$, and hence h' is also a homomorphism. Therefore if

¹For simplicity, we will denote $[\text{STC}_{u,v} E(u, v)](x_i, x'_i)$ by $\text{STC}[E](x_i, x'_i)$.

4.1 Symmetric Datalog and Relational Clones

there is a homomorphism from \mathbf{S} to \mathbf{T} then there is a homomorphism h'' such that $h''(x) = h''(y)$ for each y such that $\text{STC}[E](x, y)$. Clearly, h'' is a homomorphism from \mathbf{S}' to \mathbf{T}' .

Let \mathfrak{D} be a symmetric Datalog program for $\neg\text{CSP}(\mathbf{B})$. We have proved the equivalence of [3](#) and [4](#) below and by the definition of \mathfrak{D} , [2](#) and [3](#) are equivalent. It remains to show the equivalence of [1](#) and [2](#) and we will define \mathfrak{E} later.

1. \mathfrak{E} accepts \mathbf{S}' ;
2. \mathfrak{D} accepts \mathbf{S} ;
3. $\mathbf{S} \in \neg\text{CSP}(\mathbf{T})$, i.e. there exists no homomorphism from \mathbf{S} to \mathbf{T} ;
4. $\mathbf{S}' \in \neg\text{CSP}(\mathbf{T}')$, i.e. there exists no homomorphism from \mathbf{S}' to \mathbf{T}' .

Before we construct \mathfrak{E} , we “normalize” \mathfrak{D} such that each rule has at most one EDB. We construct \mathfrak{D}' from \mathfrak{D} as follows. For each *non-recursive* rule

$$I(\bar{x}) \quad \leftarrow \quad R_1(\bar{z}_1); R_2(\bar{z}_2); \dots; R_n(\bar{z}_n)$$

in \mathfrak{D} add the rules

$$\begin{aligned} I_1(\bar{z}_1) & \quad \leftarrow \quad R_1(\bar{z}_1) \\ I_2(\bar{z}_1, \bar{z}_2) & \quad \leftarrow \quad I_1(\bar{z}_1); R_2(\bar{z}_2) \\ & \quad \quad \quad \vdots \\ I_{n-1}(\bar{z}_1, \dots, \bar{z}_{n-1}) & \quad \leftarrow \quad I_{n-2}(\bar{z}_1, \dots, \bar{z}_{n-2}); R_{n-1}(\bar{z}_{n-1}) \\ I(\bar{x}) & \quad \leftarrow \quad I_{n-1}(\bar{z}_1, \dots, \bar{z}_{n-1}); R_n(\bar{z}_n) \end{aligned}$$

to \mathfrak{D}' . Similarly, for each *recursive* rule

$$I(\bar{x}) \quad \leftarrow \quad J(\bar{y}); R_1(\bar{z}_1); R_2(\bar{z}_2); \dots; R_n(\bar{z}_n)$$

4.1 Symmetric Datalog and Relational Clones

in \mathfrak{D} add the rules

$$\begin{array}{ll}
 I_1(\bar{y}, \bar{z}_1) & \leftarrow J(\bar{y}); R_1(\bar{z}_1) \\
 I_2(\bar{y}, \bar{z}_1, \bar{z}_2) & \leftarrow I_1(\bar{y}, \bar{z}_1); R_2(\bar{z}_2) \\
 & \vdots \\
 I_{n-1}(\bar{y}, \bar{z}_1, \dots, \bar{z}_{n-1}) & \leftarrow I_{n-2}(\bar{y}, \bar{z}_1, \dots, \bar{z}_{n-2}); R_{n-1}(\bar{z}_{n-1}) \\
 I(\bar{x}) & \leftarrow I_{n-1}(\bar{y}, \bar{z}_1, \dots, \bar{z}_{n-1}); R_n(\bar{z}_n)
 \end{array}$$

to \mathfrak{D}' . Obviously, \mathfrak{D}' is equivalent to \mathfrak{D} . Clearly, adding in the symmetric rules in \mathfrak{D}' has no effect. Now we construct \mathfrak{D}'' in which we allow the use of the reflexive symmetric transitive closure operator in the body of the rules. Let E be the relational symbol (i.e. EDB) in \mathbf{S}' that corresponds to $\{=\}$ in \mathbf{T}' . To obtain \mathfrak{D}'' , replace each EDB

$$R(z_1, \dots, z_r)$$

in each rule of \mathfrak{D}' with

$$R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r).$$

Notice that \mathfrak{D}' accepts \mathbf{S} if and only if \mathfrak{D}'' accepts \mathbf{S}' . Now we construct a symmetric Datalog program \mathfrak{E} that is equivalent to \mathfrak{D}'' but does not contain the symmetric transitive closure operator. To obtain \mathfrak{E} , replace back each

$$R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r).$$

with

$$R(z_1, \dots, z_r)$$

4.1 Symmetric Datalog and Relational Clones

and for each IDB I of arity k in \mathfrak{D}'' and for each $1 \leq i \leq k$, add the rules

$$\begin{aligned} I(x_1, \dots, x_k) &\leftarrow I(x_1, \dots, x_{i-1}, y_i, x_{i+1}, \dots, x_k); E(x_i, y_i) \\ I(x_1, \dots, x_k) &\leftarrow I(x_1, \dots, x_{i-1}, y_i, x_{i+1}, \dots, x_k); E(y_i, x_i) \end{aligned} \quad (4.1)$$

to \mathfrak{E} . Notice that \mathfrak{E} is symmetric.

First we show that **if \mathfrak{D}'' accepts \mathbf{S}' then \mathfrak{E} also accepts \mathbf{S}'** . Assume that \mathfrak{D}'' accepts \mathbf{S}' and let $\mathcal{T}^{\mathfrak{D}''}$ be the corresponding derivation tree. We claim that if in any IDB $I^{\mathfrak{D}''}(\bar{x})$ of $\mathcal{T}^{\mathfrak{D}''}$ \bar{x} is instantiated to $\bar{t}_{\bar{x}}$ then $\bar{t}_{\bar{x}}$ is also in the corresponding IDB $I^{\mathfrak{E}}$ of \mathfrak{E} and in particular, the 0-ary tuple ϵ is in the goal predicate. We induct from bottom to top on $\mathcal{T}^{\mathfrak{D}''}$. Assume that the bottommost rule in $\mathcal{T}^{\mathfrak{D}''}$ is

$$I^{\mathfrak{D}''}(\bar{x}) \leftarrow R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r)$$

and that \bar{x} is instantiated to $\bar{t}_{\bar{x}}$ in $\mathcal{T}^{\mathfrak{D}''}$. We can put $\bar{t}_{\bar{x}}$ in $I^{\mathfrak{E}}$ using the rule $I^{\mathfrak{E}}(\bar{x}) \leftarrow R(\bar{z})$ and the rules in (4.1) with IDB $I^{\mathfrak{E}}$.

Now let

$$I^{\mathfrak{D}''}(\bar{x}) \leftarrow J^{\mathfrak{D}''}(\bar{y}); R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r) \quad (4.2)$$

be a rule in $\mathcal{T}^{\mathfrak{D}''}$ other than the bottommost rule and assume that \bar{x} is instantiated to $\bar{t}_{\bar{x}}$, \bar{y} is instantiated to $\bar{t}_{\bar{y}}$, and \bar{z}' is instantiated to $\bar{t}_{\bar{z}'}$ in $\mathcal{T}^{\mathfrak{D}''}$. Construct the tuple $\bar{t}'_{\bar{x}}$ from $\bar{t}_{\bar{x}}$ as follows. Assume that the j -th element of \bar{x} is z_i for some $1 \leq i \leq r$. Then replace the j -th element of $\bar{t}_{\bar{x}}$ with the value of z'_i . Repeat this for each occurrence of a z_i in \bar{x} . Similarly, obtain $\bar{t}'_{\bar{y}}$ from $\bar{t}_{\bar{y}}$. Observe that $\bar{t}'_{\bar{x}} \equiv_E \bar{t}_{\bar{x}}$ and $\bar{t}'_{\bar{y}} \equiv_E \bar{t}_{\bar{y}}$.

Now consider the rule in \mathfrak{E} that corresponds to rule (4.2) in \mathfrak{D}'' ,

$$I^{\mathfrak{E}}(\bar{x}) \leftarrow J^{\mathfrak{E}}(\bar{y}); R(z_1, \dots, z_r).$$

R contains $\bar{t}_{\bar{z}'}$ and $J^{\mathfrak{E}}$ contains $\bar{t}_{\bar{y}}$ by the inductive hypothesis. $J^{\mathfrak{E}}$ also con-

4.1 Symmetric Datalog and Relational Clones

tains \bar{t}'_y because of the rules in (4.1) with IDB $J^{\mathfrak{E}}$. So $J^{\mathfrak{E}}(\bar{t}'_y) \wedge R(\bar{t}'_z)$ is true and therefore $I^{\mathfrak{E}}$ contains \bar{t}'_x . Because $\bar{t}_x \equiv_E \bar{t}'_x$, using the rules in (4.1) with IDB $I^{\mathfrak{E}}$ gives that $I^{\mathfrak{E}}$ also contains \bar{t}_x .

Conversely, we have to show that **if \mathfrak{E} accepts \mathbf{S}' then \mathfrak{D}'' also accepts \mathbf{S}'** . First, it is clear that taking out the rules of the form (4.1) from \mathfrak{E} and replacing each occurrence of an IDB

$$I^{\mathfrak{E}}(y_1, \dots, y_r)$$

in the body of a rule with

$$I^{\mathfrak{E}}(y'_1, \dots, y'_r); \text{STC}[I^{\mathfrak{E}}](y'_1, y_1); \dots; \text{STC}[I^{\mathfrak{E}}](y'_r, y_r)$$

results in a program equivalent to \mathfrak{E} . Now construct from this program the program \mathfrak{D}''' by replacing each EDB

$$R(z_1, \dots, z_r)$$

in the body of a rule with

$$R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r).$$

It is clear that if \mathfrak{E} accepts then \mathfrak{D}''' also accepts. Now we show that if \mathfrak{D}''' accepts then \mathfrak{D}'' also accepts. Assume that \mathfrak{D}''' accepts \mathbf{S}' and let $\mathcal{T}^{\mathfrak{D}'''}$ be the corresponding derivation tree. We show that if we replace each recursive rule of $\mathcal{T}^{\mathfrak{D}'''}$

$$I^{\mathfrak{D}''}(\bar{x}) \leftarrow J^{\mathfrak{D}'''}(y'_1, \dots, y'_p); \text{STC}[J^{\mathfrak{D}'''}](y'_1, y_1); \dots; \text{STC}[I^{\mathfrak{D}'''}](y'_p, y_p); \\ R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r)$$

4.1 Symmetric Datalog and Relational Clones

with the corresponding rule

$$I^{\mathfrak{D}''}(\bar{x}) \leftarrow J^{\mathfrak{D}''}(y_1, \dots, y_p); R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r)$$

and the bottommost non-recursive rule

$$I^{\mathfrak{D}'''}(\bar{x}) \leftarrow R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r)$$

with the corresponding rule

$$I^{\mathfrak{D}''}(\bar{x}) \leftarrow R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r)$$

in \mathfrak{D}'' , then the resulting tree $\mathcal{T}^{\mathfrak{D}''}$ with a variable instantiation defined later is a valid derivation tree for \mathfrak{D}'' over \mathbf{S}' . It remains to show how to instantiate the variables of $\mathcal{T}^{\mathfrak{D}''}$. Let $P(i)$ be the following statement.

Let $I_i^{\mathfrak{D}'''}(\bar{x})$ be the head of $\text{Sub}(\mathcal{T}^{\mathfrak{D}'''}, i)$ ². Assume that \bar{x} is instantiated to $\bar{t}_{\bar{x}}$ and $\bar{t}'_{\bar{x}}$ is any tuple such that $\bar{t}'_{\bar{x}} \equiv_E \bar{t}_{\bar{x}}$. Let $I_i^{\mathfrak{D}''}(\bar{x})$ be the root of $\text{Sub}(\mathcal{T}^{\mathfrak{D}''}, i)$ and instantiate \bar{x} in $I_i^{\mathfrak{D}''}(\bar{x})$ to $\bar{t}'_{\bar{x}}$. Then we can assign values to the remaining variables of $\text{Sub}(\mathcal{T}^{\mathfrak{D}''}, i)$ such that $\text{Sub}(\mathcal{T}^{\mathfrak{D}''}, i)$ is valid.

Notice that $P(h)$ implies that we can construct a valid derivation tree for \mathfrak{D}'' with input \mathbf{S}' , where h is the number of IDBs in $\mathcal{T}^{\mathfrak{D}'''}$.

$P(1)$ is obviously true. Assume that $P(i)$ is true. Let the i -th IDB from the bottom be $J^{\mathfrak{D}'''}$ and the $(i+1)$ -th be $I^{\mathfrak{D}'''}$. Furthermore, let

$$\begin{aligned} I^{\mathfrak{D}'''}(\bar{x}) \leftarrow & J^{\mathfrak{D}'''}(y'_1, \dots, y'_p); \text{STC}[J^{\mathfrak{D}'''}](y'_1, y_1); \dots; \text{STC}[I^{\mathfrak{D}'''}](y'_p, y_p); \\ & R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r) \end{aligned} \quad (4.3)$$

²Let \mathcal{T} be a derivation tree. Then $\text{Sub}(\mathcal{T}, i)$ denotes the subtree of \mathcal{T} rooted at the i -th IDB counting from the bottom.

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

be the rule used at this point in $\mathcal{T}^{\mathfrak{D}''}$ and let the corresponding rule in $\mathcal{T}^{\mathfrak{D}''}$ be

$$I^{\mathfrak{D}''}(\bar{x}) \leftarrow J^{\mathfrak{D}''}(y_1, \dots, y_p); R(z'_1, \dots, z'_r); \text{STC}[E](z'_1, z_1); \dots; \text{STC}[E](z'_r, z_r). \quad (4.4)$$

Assume that in (4.3), \bar{x} is instantiated to $\bar{t}_{\bar{x}}$, \bar{y} is instantiated to $\bar{t}_{\bar{y}}$, \bar{y}' is instantiated to $\bar{t}_{\bar{y}'}$, \bar{z} is instantiated to $\bar{t}_{\bar{z}}$ and \bar{z}' is instantiated to $\bar{t}'_{\bar{z}}$. Let $\bar{t}'_{\bar{x}}$ be any tuple such that $\bar{t}'_{\bar{x}} \equiv_E \bar{t}_{\bar{x}}$. In (4.4), instantiate \bar{x} to $\bar{t}'_{\bar{x}}$ and \bar{z}' to $\bar{t}'_{\bar{z}}$.

Let \bar{v} be the same tuple as $\bar{t}_{\bar{y}}$ except that we make the following changes. For each x_i and y_j , if x_i and y_j are the same variables then change the value v_j to the i -th element of $\bar{t}'_{\bar{x}}$. Notice that because $\bar{t}_{\bar{y}'} \equiv_E \bar{t}_{\bar{y}}$ and $\bar{t}_{\bar{y}} \equiv_E \bar{v}$, we have that $\bar{t}_{\bar{y}'} \equiv_E \bar{v}$. Therefore by the inductive hypothesis, we can instantiate \bar{y} in (4.4) to \bar{v} .

Finally, let \bar{w} be the same tuple as $\bar{t}_{\bar{z}}$ except that we make the following changes. For each x_i and z_j , if x_i and z_j are the same variables then change the value w_j to the i -th element of $\bar{t}'_{\bar{x}}$. Then $\bar{w} \equiv_E \bar{t}_{\bar{z}}$ so we can instantiate \bar{z} in (4.4) to \bar{w} . \square

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

In this section we define a large class of CSPs definable in symmetric Datalog (see the statement of Theorem 4.3).

Theorem 4.3. *Let \mathbf{B} be a relational structure such that for some $d \in B$, each relation R in \mathbf{B} is either*

1. *A unary relation $S \subseteq B$;*
2. *A binary relation R_π such that $R_\pi = \{(b, \pi(b)) : b \in B\}$ for some permutation π with $\pi(d) = d$;*

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

3. A k -ary relation R_k with $k \geq 2$ and $R_k = \{\langle b_1, \dots, b_k \rangle : \exists i b_i = d\}$.

Then $\neg\text{CSP}(\mathbf{B})$ is definable in symmetric Datalog.

Similarly, if \mathbf{C} is a structure such that each relation $R^{\mathbf{C}}$ is a unary relation $R^{\mathbf{C}} \subseteq C$ or a binary relation $R_\pi^{\mathbf{C}}$ for some permutation π of C (with no fixed point condition) then $\neg\text{CSP}(\mathbf{C})$ is definable in symmetric Datalog.

Proof of Theorem 4.3. Let us assume that the permutations have a fixed point d (see comment following the proof) and that \mathbf{B} contains relations of the form $R_k^{\mathbf{B}} = \{\langle b_1, \dots, b_k \rangle : \exists i b_i = d\}$. Before describing the symmetric Datalog program, we make a few basic observations and recast the problem in a more graph-theoretic fashion. First note that if $j \leq k$, then the relation $R_j^{\mathbf{B}} = \{\langle b_1, \dots, b_j \rangle : \exists i b_i = d\}$ is simply the set of tuples such that $\langle b_1, \dots, b_j, \dots, b_j \rangle \in R_k^{\mathbf{B}}$. By Theorem 4.1, we can thus assume without loss of generality that \mathbf{B} contains a single relation $R_k^{\mathbf{B}}$.

If \mathbf{A} is an input structure and $\mathbf{A} \xrightarrow{h} \mathbf{B}$ then for each $a \in A$ we must have $h(a) \in \bigcap_{a \in U^{\mathbf{A}}} U_a^{\mathbf{B}}$. For convenience, we denote by $U_a^{\mathbf{B}}$ this intersection which corresponds to the subset of values in B which respect the unary constraints imposed on a .

For a structure \mathbf{A} , we construct an edge-labeled and vertex-labeled directed graph $G^{\mathbf{A}} = \langle A, E \rangle$ as follows (the vertex set is the universe of \mathbf{A}). For any $a_1, a_2 \in A$ such that $\langle a_1, a_2 \rangle \in R_\pi^{\mathbf{A}}$ we add an edge $\langle a_1, a_2 \rangle$ labeled with π and an edge $\langle a_2, a_1 \rangle$ labeled with π^{-1} . Note that this edge construction makes the graph essentially undirected. Finally, we color each vertex $a \in A$ with $U_a^{\mathbf{B}}$. Obviously, if for any $a \in A$, $U_a^{\mathbf{B}} = \emptyset$ then there can be no homomorphism from \mathbf{A} to \mathbf{B} . Any path p between two vertices of A can be thought of as labeled by the permutation π_p which is the product of the labels of individual edges on that path.

For any $\langle a_1, \dots, a_k \rangle \in R_k^{\mathbf{A}}$ we say that the k -tuple $\langle a'_1, \dots, a'_k \rangle$ is a **k -blocking pattern** in $G^{\mathbf{A}}$ if for each $1 \leq i \leq k$ there is a path from a_i to a'_i and $d \notin U_{a'_i}$. If a k -blocking pattern exists for some $\langle a_1, \dots, a_k \rangle \in R_k^{\mathbf{A}}$ then there is no homomorphism from \mathbf{A} to \mathbf{B} . To see this, assume that $\mathbf{A} \xrightarrow{h} \mathbf{B}$

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

and observe that h does not map any of the a'_i to d and d is a fixed point of all permutations π . It follows that h does not map any of the a_i to d and so $\langle h(a_1), \dots, h(a_k) \rangle \notin R_k^{\mathbf{A}}$. This is a contradiction.

Similarly, let a be a vertex of the graph with $U_a^{\mathbf{B}} = \{b_1, \dots, b_t\}$. For each b_i , $1 \leq i \leq t$, consider a path p_i of length at least 1 from a to some vertex a_i in $G^{\mathbf{A}}$. We say that the set of paths p_1, \dots, p_t is **permutation blocking** for the vertex a if for each $1 \leq i \leq t$ we have either $\pi_{p_i}(b_i) \notin U_{a_i}^{\mathbf{B}}$ or $a = a_i$ and $\pi_{p_i}(b_i) \neq b_i$. By design, if $G^{\mathbf{A}}$ contains a permutation blocking pattern then there can be no homomorphism from \mathbf{A} to \mathbf{B} . To see this, assume that $\mathbf{A} \xrightarrow{h} \mathbf{B}$. It follows that $h(a) \in U_a^{\mathbf{B}}$ and also that if $h(a) = b_i$ then $h(a_i) = \pi_{p_i}(b_i) \in U_{a_i}^{\mathbf{B}}$. This is a contradiction. Furthermore, notice that we need to consider only paths of length at least 1 because if a path has zero length then it is always the case that $\pi_{\text{Id}}(b_i) \in U_a^{\mathbf{B}}$ and $\pi_{\text{Id}}(b_i) = b_i$, where π_{Id} is the identity permutation.

We claim that $\mathbf{A} \in \text{CSP}(\mathbf{B})$ if and only if for each $a \in A$, $U_a^{\mathbf{B}} \neq \emptyset$ and $G^{\mathbf{A}}$ contains no permutation blocking or k -blocking patterns. Note that we have already established one direction. Let us suppose that for each $U_a^{\mathbf{B}} \neq \emptyset$ and $G^{\mathbf{A}}$ contains no blocking patterns and explicitly construct a homomorphism from \mathbf{A} to \mathbf{B} . Consider an arbitrary element a of \mathbf{A} . If $d \in U_{a'}^{\mathbf{B}}$ for all a' in the connected component of a , then we set $h(a) = d$. Otherwise, since $G^{\mathbf{A}}$ contains no permutation blocking pattern, there exists some $b \in U_a^{\mathbf{B}}$ such that for each path p from a to any vertex a' either $a = a'$ and $\pi_p(b) = b$ or $a \neq a'$ and $\pi_p(b) \in U_{a'}^{\mathbf{B}}$. We set $h(a) = b$ and $h(a') = \pi_p(b)$ for each vertex a' reachable from a by a path p . Note that this assignment is well defined. Indeed, assume that there are two (or more) distinct paths p and p' from a to a given a' and assume that $\pi_p(b) = b'$. Notice that pp'^{-1} is a path from a to a and so $\pi_{pp'^{-1}}(b) = b$. This implies that $\pi_{p'^{-1}}(b') = b$ so $\pi_{p'}(b) = b'$, i.e. $\pi_p(b) = \pi_{p'}(b)$.

This stage defines $h(a')$ for each a' in the connected component of a . We can repeat the argument to similarly fix the values of h in each other

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

connected component of $G^{\mathbf{A}}$. We need to prove that h is indeed a homomorphism. By construction, we have $h(a) \in U_a^{\mathbf{B}}$ for each a and $\langle h(a_1), h(a_2) \rangle \in R_\pi$ for any $\langle a_1, a_2 \rangle \in R_\pi^{\mathbf{A}}$. It remains to show that if $\langle a_1, \dots, a_k \rangle \in R_k^{\mathbf{A}}$ then $\langle h(a_1), \dots, h(a_k) \rangle \in R_k^{\mathbf{B}}$ which, by definition of $R_k^{\mathbf{B}}$ is equivalent to the requirement that $h(a_j) = d$ for some j . Since $G^{\mathbf{A}}$ contains no k -blocking pattern, there exists some a_j such that all a' in the connected component of a_j satisfy $d \in U_{a'}^{\mathbf{B}}$ and so $h(a_j) = d$.

Our symmetric Datalog program contains rules of one of nine types summarized in Figure 4.1. The program is a reflection of the above graph-theoretic construction. The goal predicate in these rules is denoted by G . We start by understanding the rules of type (U) . Let $C = \{U_1, \dots, U_q\}$ be a set of relation symbols such that $\bigcap_{U \in C} U^{\mathbf{B}} = \emptyset$. For each such C there is a rule of type (U) in which $\tilde{U} = \bigwedge_{U \in C} U(x)$. Clearly, there exists an $a \in A$ such that $U_a^{\mathbf{B}} = \emptyset$ if and only if the body of a rule of type (U) is satisfied.

Next, we have to detect k -blocking patterns. We use an IDB

$$J(x_1, \dots, x_k, y_1, \dots, y_k)$$

of arity $2k$ to represent the fact that for each i there is a path from x_i to y_i . Clearly, we can initialize this IDB with the non-recursive rule of type $(k1)$. This rule says that for each x_i , $1 \leq i \leq k$, there is a path of length zero to itself. We also have symmetric rules of the form $(k2)$ and $(k3)$ for each $1 \leq j \leq k$ and each R_π to construct the k paths in $G^{\mathbf{A}}$. Notice that the rules of type $(k2)$ and $(k3)$ form symmetric pairs. The presence of these symmetric rules is justified because if there is an edge from y_j to z in $G^{\mathbf{A}}$ then by construction there is also an edge from z to y_j . Furthermore, the subscripts of the IDBs are empty because we do not care about the labels of the paths.

To detect k -blocking patterns, consider the rules of type $(k4)$. First, the variables $x_1, \dots, x_k, y_1, \dots, y_k$ are instantiated to some elements $a_1, \dots, a_k, a'_1, \dots, a'_k$ of A such that for each $1 \leq i \leq k$ there is a path from a_i to a'_i . This

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

Rules to detect if for any $a \in A$, $U_a^{\mathbf{B}}$ is empty:

$$G \leftarrow \tilde{U}(x) \quad (U)$$

Rules to detect k -blocking patterns:

$$J(x_1, \dots, x_k, x_1, \dots, x_k) \leftarrow \quad (k1)$$

$$J(x_1, \dots, x_k, y_1, \dots, z, \dots, y_k) \leftarrow J(x_1, \dots, x_k, y_1, \dots, y_j, \dots, y_k); \quad (k2)$$

$$R_\pi(y_j, z)$$

$$J(x_1, \dots, x_k, y_1, \dots, y_j, \dots, y_k) \leftarrow J(x_1, \dots, x_k, y_1, \dots, z, \dots, y_k); \quad (k3)$$

$$R_\pi(y_j, z)$$

$$G \leftarrow J(x_1, \dots, x_k, y_1, \dots, y_k); \quad (k4)$$

$$R_k(x_1, \dots, x_k); \tilde{U}_{y_1}(y_1); \dots; \tilde{U}_{y_k}(y_k)$$

Rules to detect permutation blocking patterns:

$$I_{\pi_1, \dots, \pi_{|B|}}(x, y_1, \dots, y_{|B|}) \leftarrow R_{\pi_1}(x, y_1); \dots; R_{\pi_{|B|}}(x, y_{|B|}) \quad (P1)$$

$$I_{\pi_1, \dots, \rho\pi_j, \dots, \pi_{|B|}}(x, y_1, \dots, z, \dots, y_{|B|}) \leftarrow R_\rho(y_j, z); \quad (P2)$$

$$I_{\pi_1, \dots, \pi_j, \dots, \pi_{|B|}}(x, y_1, \dots, y_j, \dots, y_{|B|})$$

$$I_{\pi_1, \dots, \pi_j, \dots, \pi_{|B|}}(x, y_1, \dots, y_j, \dots, y_{|B|}) \leftarrow R_\rho(y_j, z) \quad (P3)$$

$$I_{\pi_1, \dots, \rho\pi_j, \dots, \pi_{|B|}}(x, y_1, \dots, z, \dots, y_{|B|})$$

$$G \leftarrow I_{\pi_{y_1}, \dots, \pi_{y_{|B|}}}(x, y_1, \dots, y_{|B|}); \quad (P4)$$

$$\tilde{U}_x; \tilde{U}_{y_1}; \dots; \tilde{U}_{y_{|B|}}$$

Figure 4.1: Types of rules for the program of Theorem 4.3

is the role of the IDB $J(x_1, \dots, x_k, y_1, \dots, y_k)$ in the body. Now we have to define \tilde{U}_{y_i} for $1 \leq i \leq k$ such that $\tilde{U}_{y_i}(y_i)$ is true if and only if $d \notin U_{y_i}^{\mathbf{B}}$. Let

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

$C = \{U_1, \dots, U_q\}$ be a set of unary relation symbols such that $d \notin \bigcap_{U \in C} U^{\mathbf{B}}$. Let \mathfrak{C} be the set of all such C -s. For any k -tuple $\langle C_{y_1}, \dots, C_{y_k} \rangle \in \mathfrak{C}^k$ we have a rule of type (k4) in which we let $\tilde{U}_{y_i} = \bigwedge_{U \in C_{y_i}} U(y_i)$, $1 \leq i \leq k$. Clearly, the body of a rule of type (k4) is satisfied if and only if there is a k -blocking pattern.

Finally, we have to detect permutation blocking patterns. For each $|B|$ -tuple $\langle \pi_1, \dots, \pi_{|B|} \rangle$ of permutations of B , we create a $|B| + 1$ -ary IDB $I_{\pi_1, \dots, \pi_{|B|}}(x, y_1, \dots, y_{|B|})$ which is intended to represent the fact for each $1 \leq j \leq |B|$ there is a path p in $G^{\mathbf{A}}$ labeled with π_j from x to y_j . We include non-recursive initialization rules of type (P1) stating that if \mathbf{A} includes for each j the constraint $\langle x, y_j \rangle \in R_{\pi_j}$ then the tuple $\langle x, y_1, \dots, y_{|B|} \rangle$ lies in the IDB $I_{\pi_1, \dots, \pi_{|B|}}$.

We next include recursive rules of type (P2) and (P3) to build the paths for the permutation blocking patterns. Notice that rules of type (P2) and (P3) form symmetric pairs. Consider two IDBs $I_{\pi_1, \dots, \rho\pi_j, \dots, \pi_{|B|}}$ and $I_{\pi_1, \dots, \pi_j, \dots, \pi_{|B|}}$ whose index differs only in the j th permutation. If $x, y_1, \dots, y_{|B|}$ are such that there exist for each i a path from x to y_i labeled by π_i and if further \mathbf{A} contains a constraint $\langle y_j, z \rangle \in R_{\rho}$ then there is a path from x to z labeled $\rho\pi_j$. But symmetrically, if there is a path from x to z labeled by $\rho\pi_j$, we have a path from x to y_j labeled by $\rho^{-1}\rho\pi_j = \pi_j$. This observation justifies the rules of the form (P2) and (P3) in our program and it is clear that these IDBs behave as intended.

Permutation blocking patterns are identified using rules of type (P4). In these types of rules each of the y_i could be a different variable than x or it could be the same. Therefore there are $2^{|B|}$ possible variable patterns in the IDB in the body. For each possible variable pattern we will construct a set of rules. To detect a permutation blocking pattern, first the variables $x, y_1, \dots, y_{|B|}$ are instantiated to some elements $a, a_1, \dots, a_{|B|}$ such that for each $1 \leq i \leq |B|$ there is a path from a to a'_i . This is the role of the IDB $I_{\pi_{y_1}, \dots, \pi_{y_{|B|}}}(x, y_1, \dots, y_{|B|})$ in the body.

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

For each variable pattern in the IDB in the body construct the following rules. Let the permutations in the subscripts of the IDB in the body be $\pi_{p_1}, \dots, \pi_{p_t}$. Let $C = \{U_1, \dots, U_q\}$ be any set of unary relation symbols and let \mathfrak{C}_x be the class of all C s. For each set $C \in \mathfrak{C}_x$ construct the following set of rules. Let $\tilde{U}_x(x) = \bigwedge_{U \in C} U(x)$. Assume that $\bigcap_{U \in C} U^{\mathbf{B}} = \{b_1, \dots, b_t\}$. Note that $t \leq |B|$. If $i > t$ then we set \tilde{U}_{y_i} to be the empty conjunction. Now assume that $i \leq t$. If y_i is a different variable than x then \tilde{U}_{y_i} is responsible for detecting whether $\pi_{y_i}(b_i) \in U_{a_i}^{\mathbf{B}}$. If \tilde{U}_{y_i} is the same variable as x then we have to detect if $\pi_{y_i}(b_i) = b_i$. For each y_i that is different from x , construct a class of sets of relation symbols \mathfrak{C}_{y_i} as follows. Let $C = \{U_1, \dots, U_q\}$ be a set of unary relation symbols such that $\pi_{y_i}(b_i) \notin \bigcap_{U \in C} U^{\mathbf{B}}$. Let \mathfrak{C}_{y_i} be the set of all such C s. Now take a tuple in $\langle C_{y_1}, \dots, C_{y_i} \rangle \in \mathfrak{C}_{y_1} \times \dots \times \mathfrak{C}_{y_p}$ and set $\tilde{U}_{y_i} = \bigwedge_{U \in C_{y_i}} U(y_i)$. Furthermore, add this rule to the program only if for each y_i that is the same variable as x it is the case that $\pi_{y_i}(b_i) \neq b_i$.

To see that the rules of type (P4) set the goal predicate to true if and only if there is permutation blocking pattern first we assume that there is a permutation blocking pattern defined by the input structure \mathbf{A} and construct a rule \mathfrak{R} which is of type (P4) and whose body is satisfied for the input structure \mathbf{A} . If there is a permutation blocking pattern then there is a vertex $a \in A$ with $U_a^{\mathbf{B}} = \{b_1, \dots, b_t\}$ such that there is a set of t paths p_1, \dots, p_t from a to some $a_1, \dots, a_t \in A$ such that for each $1 \leq i \leq t$ we have either $\pi_{p_i}(b_i) \notin U_{a_i}^{\mathbf{B}}$ or $a = a_i$ and $\pi_{p_i}(b_i) \neq b_i$.

For each i such that $a_i = a$, let y_i be the same variable as x . Let the unary conditions on x be as follows. Let C be a set of relation symbols such that $U_a^{\mathbf{B}} = \bigcap_{U \in C} U^{\mathbf{B}}$ and let the unary conditions on x be $\tilde{U}_x(x) = \bigwedge_{U \in C} U(x)$. Let the permutations in the subscripts of the IDB in the body be $\pi_{p_1}, \dots, \pi_{p_t}$ and we arbitrarily set the permutations $\pi_{p_{t+1}}, \dots, \pi_{p_{|B|}}$ to be π_{p_1} . For each $1 \leq i \leq t$ such that $a_i \neq a$ assume that D is a set such that $U_{a_i}^{\mathbf{B}} = \bigcap_{U \in D} U^{\mathbf{B}}$. Because we have a permutation blocking pattern we know that $\pi(b_i) \notin U_{a_i}^{\mathbf{B}}$ so in \mathfrak{R} we can set the unary constraints on y_i to be $\tilde{U}_{y_i}(y_i) = \bigwedge_{U \in D} U(y_i)$.

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

Furthermore, for each $1 \leq i \leq k$ such that $a_i = a$ it is not the case that $\pi_{p_1}(b_i) = b_i$ so this rule is actually in the program. Clearly, the body of this rule is satisfied for input \mathbf{A} .

Conversely, assume that on some input structure \mathbf{A} a rule \mathfrak{R} of type (P4) sets the goal predicate to true and construct a permutation blocking pattern. Assume that the variables of \mathfrak{R} are instantiated to $a, a_1, \dots, a_{|B|}$. Let the permutations in the subscript of the IDB in the body be $\pi_{y_1}, \dots, \pi_{y_{|B|}}$. Let C be the set of unary relation symbols such that x is constrained by $\bigwedge_{U \in C} U(x)$. Consider $\bigcap_{U \in C} U^{\mathbf{B}} = \{b_1, \dots, b_t\}$. We show for each $1 \leq i \leq t$, if $a \neq a_i$ then $\pi_{y_i}(b_i) \notin U_{a_i}^{\mathbf{B}}$ or if $a = a_i$ then $\pi_{y_i}(b_i) \neq b_i$.

It is clear that for each $1 \leq i \leq t$, the symmetric Datalog program detected a path p_i from a to a_i and that π_{y_i} is the corresponding permutation. If $a \neq a_i$ then x is a different variable than y_i . Let D be the set of unary relation symbols such that $\tilde{U}_{y_i}(y_i) = \bigwedge_{U \in D} U(y_i)$. Then by construction we have that $\pi_{y_i}(b_i) \notin \bigcap_{U \in D} U^{\mathbf{B}}$ and because the body of \mathfrak{R} is satisfied we have that $a_i \in \bigcap_{U \in D} U^{\mathbf{A}}$. Therefore $U_{a_i}^{\mathbf{B}} \subseteq \bigcap_{U \in D} U^{\mathbf{B}}$ so $\pi_{y_i}(b_i) \notin U_{a_i}^{\mathbf{B}}$. Now assume that $a = a_i$. If x is a different variable than y_i then by the above analysis $\pi_{y_i}(b_i) \notin U_{a_i}^{\mathbf{B}}$, i.e. $\pi_{y_i}(b_i) \notin U_a^{\mathbf{B}}$. Because $b_i \in U_a^{\mathbf{A}}$ we have that $\pi_{y_i}(b_i) \neq b_i$. If y_i is the same variable as x then because \mathfrak{R} is present in the program we have by construction that $\pi_{y_i}(b_i) \neq b_i$. \square

Remark: We assumed in our proof that d was a fixed point of the permutations. However, it is clear from the argument that this requirement is only needed in the presence of the R_k relations. If \mathbf{B} consists solely of permutations and unary relations, $\text{-CSP}(\mathbf{B})$ can be defined in symmetric Datalog.

However, if \mathbf{B} contains a relation R_k but contains a permutation of which d is *not* a fixed point expressibility in symmetric Datalog cannot be guaranteed. Indeed, over the two-element domain, the relation R_2 is the binary OR relation and the non-trivial permutation π is disequality. The problem $\text{CSP}(\text{OR}_2, \neq)$ is NL-hard and it easily follows from Theorem 5.19 in the next

4.2 A Sufficient Condition for Expressibility in Symmetric Datalog

chapter that $\neg \text{CSP}(\text{OR}_2, \neq)$ does not lie in symmetric Datalog.

We also note that the conditions in Theorem 4.3 are motivated by the algebraic approach. (See last section of [14].)

Finally, it is interesting to compare this class of CSPs with the one defined in Example 1.9 in Chapter 1:

Corollary 4.4. *Let \mathbf{D} be a finite relational structure such that each relation R in \mathbf{D} is either of the form 1 or 2 in Example 1.9. Then $\text{CSP}(\mathbf{D})$ is expressible in symmetric Datalog.*

Proof. A binary relation of the form $B \times C$ can be expressed as the conjunction of the unary relations B and C . Similarly, if $B \subseteq D$ and $f : B \rightarrow D$ is injective then f can be extended to a permutation π of D such that $\pi|_B = f$. The implicational relation $R = \{\langle b, f(b) \rangle : b \in B\}$ can then be expressed as a conjunction of the relation R_π and the unary relation B . Thus, \mathbf{D} is a finite relational structure whose relations are defined by primitive positive formulas over \mathbf{E} for some \mathbf{E} of the form given in Theorem 4.3 and the result follows by Theorem 4.1. \square

To the best of our knowledge, all currently known CSPs in L are definable in symmetric Datalog. It is an interesting open problem whether there exists a CSP that lies in L but which is not definable in symmetric Datalog.

Chapter 5

Directed ST-Connectivity Is Not Definable in Symmetric Datalog

I am grateful to Benoît Larose and Pascal Tesson for many fruitful discussions and suggestions about this chapter.

Ajtai and Fagin gave the first proof that directed st -connectivity is “harder” for directed graphs than for undirected graphs [1] in a precise technical sense. They showed that unlike undirected st -connectivity, directed st -connectivity is not definable in monadic Σ_1^1 . In this chapter we demonstrate a different approach to show that directed st -connectivity is harder than the undirected version. We prove that unlike undirected st -connectivity, directed st -connectivity is not definable in symmetric Datalog.

First we show that the reflexive transitive closure of a binary relation is not definable in symmetric Datalog. In fact, let \mathfrak{D} be a symmetric Datalog program with goal predicate $G(u, v)$. Let s and t be two nodes of a graph defined by the edge relation E . Set u to s and v to t and if \mathfrak{D} works correctly then it accepts if and only if there is a directed path from s to t in E . We show that no such symmetric Datalog program exists and here we give an

5.1 The Mirror Operator

intuitive overview the proof.

Assume that there is a symmetric Datalog program \mathfrak{D} that outputs the reflexive transitive closure of a binary relation. Construct an input graph that is a long enough path p (“long enough” will be specified in the proofs) and query whether the first vertex s and the last vertex t of this path are connected. If \mathfrak{D} works correctly then it must detect that s and t are connected and therefore it must produce a derivation path \mathcal{P} that witnesses this fact. The main strategy is to obtain a derivation path \mathcal{P}' from \mathcal{P} such that

1. \mathcal{P}' contains no path from s and t ;
2. On some input structure \mathbf{P} , \mathcal{P}' is a valid derivation path for \mathfrak{D} over \mathbf{P} .

The key idea is to blow up \mathcal{P} , *using the fact that \mathfrak{D} is symmetric*, such that we “disconnect” each path from s to t . Intuitively, we will “zig-zag” on \mathcal{P} going back and forth until we disconnect two internal vertices of an st -path and this leads to a contradiction.

After we show that the following statements are equivalent.

1. The problem of finding the reflexive transitive closure of a binary relation is definable in symmetric Datalog;
2. The complement of a specific CSP (defined later) is definable in symmetric Datalog;
3. Directed st -connectivity is definable in symmetric Datalog.

Before getting to the main proof in the second part of this chapter, we need to introduce some concepts and prove Lemma 5.1. The proof of this lemma is not difficult but laborious and it can be skipped on first reading.

5.1 The Mirror Operator

A **zig-zag** is a sequence of integers $Z = t_1, \dots, t_p$ such that $|t_i - t_{i+1}| = 1$. Given two integers $[a, b]$ such that $a < b$, $\text{MaxP}_{[a,b]}(Z)$ denotes the set of all

5.1 The Mirror Operator

index pairs $\langle k, \ell \rangle$ such that

- $k < \ell$;
- If $t \in \{t_k, t_{k+1}, \dots, t_\ell\}$, then $a \leq t \leq b$ and it is not the case that $a \leq t_{k-1} \leq b$ or $a \leq t_{\ell+1} \leq b$;
- $a, b \in \{t_k, t_{k+1}, \dots, t_\ell\}$.

Let \mathcal{Z} denote the set of all zig-zags. A mirror operator $\mu_{[a,b],r} : \mathcal{Z} \rightarrow \mathcal{Z}$ is a function with integer parameters $a < b$ and $r \in \mathbb{Z}^+$. Let Z be a zig-zag. Then $\mu_{[a,b],r}(Z)$ is the zig-zag such that for each index pair $\langle i, j \rangle \in \text{MaxP}_{[a,b],r}(Z)$ we insert r consecutive copies of the sequence $t_{j-1}, t_{j-2}, \dots, t_i, t_{i+1}, \dots, t_j$ after t_j in Z . Pictorially, we can think of this process as “folding out” the subsequence t_i, t_{i+1}, \dots, t_j $2r$ times. The objective of this section is to prove Lemma 5.1.

Lemma 5.1. *If Z is a zig-zag and μ and ν are mirror operators then*

$$\nu(\mu(Z)) = \mu(\nu(Z)).$$

We call the pairs in $\text{MaxP}_{[a,b]}(Z)$ **maximal pairs associated with** $\mu_{[a,b],r}$. For a maximal pair $\langle i, j \rangle \in \text{MaxP}_{[a,b]}(Z)$, we call the sequence t_i, t_{i+1}, \dots, t_j a **maximal interval associated with** $\mu_{[a,b],r}$. Let $\text{MaxI}_{[a,b]}(Z)$ denote the set of all maximal intervals associated with $\mu_{[a,b],r}$. Let $I \in \text{MaxI}_{[a,b]}(Z)$ be an interval. Then I_l denotes the index of the leftmost element of I and I_r denotes the index of the rightmost element of I . We say that two pairs of indices $\langle i, j \rangle$ such that $i < j$ and $\langle k, \ell \rangle$ such that $k < \ell$ are **disjoint** if either $j + 1 < k$ or $\ell + 1 < i$. Notice that any two different pairs in $\text{MaxP}_{[a,b]}(Z)$ for some zig-zag Z and integers a and b are disjoint. Let $\langle i, j \rangle, \langle k, \ell \rangle$ and $[a, b], [c, d]$ be two pairs of indices and two pairs of integers, respectively. Then

1. If $i < k$ ($a < c$) and $j < \ell$ ($b < d$) then $\langle i, j \rangle$ ($[a, b]$) is said to be to the **left** of $\langle k, \ell \rangle$ ($[c, d]$);

5.1 The Mirror Operator

2. If $i > k$ ($a > c$) and $j > \ell$ ($b > d$) then $\langle i, j \rangle$ ($[a, b]$) is said to be to the **right** of $\langle k, \ell \rangle$ ($[c, d]$);
3. If $i \leq k$ ($a \leq c$) and $\ell \leq j$ ($d \leq b$) then $\langle i, j \rangle$ ($[a, b]$) is said to **contain** $\langle k, \ell \rangle$ ($[c, d]$);
4. If $k \leq i$ ($c \leq a$) and $j \leq \ell$ ($b \leq d$) then $\langle i, j \rangle$ ($[a, b]$) is said to be **contained in** $\langle k, \ell \rangle$ ($[c, d]$).

Notice that these four cases cover all possibilities. (See Figure 5.1 for examples.)



- (a) The index pair $\langle 2, 4 \rangle$ is to the left of the pair $\langle 3, 6 \rangle$. (b) The index pair $\langle 3, 6 \rangle$ contains the pair $\langle 4, 5 \rangle$.

Figure 5.1: Possible relations between two pairs of integers.

Fact 5.2. Let $[a, b]$ and $[c, d]$ be pairs of integers such that $a < b, c < d$ and $[a, b]$ is either to the left or to the right of $[c, d]$. Let Z be a zig-zag. Let $\langle i, j \rangle \in \text{MaxP}_{[a,b]}(Z)$ and $\langle k, \ell \rangle \in \text{MaxP}_{[c,d]}(Z)$. Then $\langle i, j \rangle$ is either to the left or to the right of $\langle k, \ell \rangle$.

Proof. Assume that $\langle i, j \rangle$ contains $\langle k, \ell \rangle$ (the reverse case is similar). Then $i \leq k < \ell \leq j$ so it must be the case that $a \leq c < d \leq b$, i.e. $[a, b]$ contains $[c, d]$ which is a contradiction. \square

Fact 5.3. Let $[a, b]$ and $[c, d]$ be pairs of integers such that $a < b, c < d$ and $[a, b]$ contains $[c, d]$. Let Z be a zig-zag. Let $\langle i, j \rangle \in \text{MaxP}_{[a,b]}(Z)$ and $\langle k, \ell \rangle \in \text{MaxP}_{[c,d]}(Z)$. Then either $\langle i, j \rangle$ contains $\langle k, \ell \rangle$ or $\langle i, j \rangle$ and $\langle k, \ell \rangle$ are disjoint.

5.1 The Mirror Operator

Proof. Similar to the proof of Fact 5.2. □

By Facts 5.2 and 5.3 we have to analyse two cases. Let $\mu_{[a,b],r}$ and $\mu_{[c,d],s}$ be mirror operators. We call the case when $[a, b]$ is to the left or to the right of $[c, d]$ the **left-right case**. We call the case when $[a, b]$ contains $[c, d]$ the **containment case**.

Definition 5.4. Let $A = a_1, \dots, a_m, c_1, \dots, c_k$ and $B = c_1, \dots, c_k, b_1, \dots, b_n$ be zig-zags. If the last k elements of A and the first k elements of B coincide then

$$A \oplus_k B \stackrel{\text{def}}{=} a_1, \dots, a_m, c_1, \dots, c_k, b_1, \dots, b_n.$$

The operator \oplus stands for \oplus_0 , i.e. simple concatenation.

Definition 5.5. Let $Z = t_1, \dots, t_p$ be a zig-zag and μ and ν be mirror operators. We construct a graph G whose vertex set is $\{t_1, \dots, t_p\}$. We place an edge between two consecutive elements t_i and t_{i+1} if there is a maximal interval associated with μ or ν that contains both t_i and t_{i+1} . Notice that a connected component of G corresponds to a consecutive sequence of elements in Z . We call such a consecutive sequence in Z a **block**.

Definition 5.6. Let Z be a zig-zag and $[a, b]$ be a pair of integers such that $a < b$. We say that an element t of Z is **good** for $[a, b]$ if there exist two sequences p_1 and p_2 as follows. Both p_1 and p_2 are consecutive elements of Z starting at t and ending at a and b , respectively. Furthermore, the elements of p_1 and p_2 belong to $\{a, a + 1, \dots, b\}$. We call the pair of sequences $\langle p_1, p_2 \rangle$ a **pair of good paths**.

Fact 5.7. *An element t of a zig-zag Z belongs to a maximal interval associated with a mirror operator $\mu_{[a,b],r}$ if and only if t is good for $[a, b]$.*

Lemma 5.8. *Let $\mu_{[a,b],r}$ and $\nu_{[c,d],s}$ be mirror operators. Let Z be a zig-zag and find all the blocks Z_1, \dots, Z_n in Z from left to right. Let $G_0, G_1, G_2, \dots, G_n$ be such that*

$$Z = G_0 \oplus Z_1 \oplus G_1 \oplus Z_2 \oplus \dots \oplus Z_n \oplus G_n.$$

5.1 The Mirror Operator

Note that G_i could be empty for any $0 \leq i \leq n$. Then

$$\nu(\mu(Z)) = G_0 \oplus \nu(\mu(Z_1)) \oplus G_1 \oplus \nu(\mu(Z_2)) \oplus \cdots \oplus \nu(\mu(Z_n)) \oplus G_n.$$

Proof. Clearly, $\mu(Z) = G_0 \oplus \mu(Z_1) \oplus G_1 \oplus \mu(Z_2) \oplus \cdots \oplus \mu(Z_n) \oplus G_n$. Now it is enough to show that each maximal interval associated with ν in $\mu(Z)$ has each of its elements within $\mu(Z_i)$ for some i . Assume that there is a maximal interval I in $\mu(Z)$ associated with ν that has elements in $\mu(Z_i)$ for some i and it also has elements outside of $\mu(Z_i)$. There are two cases.

Case 1: Assume that I also has an element in G_i (the case for G_{i-1} is similar). Let that element be g . Let p_1 and p_2 be a corresponding pair of good paths from g to an element of Z having the value c and from g to an element of Z having the value d , respectively. Now we “fold back” $\mu(Z)$ according to μ to get back Z . Clearly, after folding back there must still exist a pair of good paths p'_1 from g to an element in Z having the value c and p'_2 from g to an element in Z having the value d . So g in G_i in Z is still good for $\nu(Z)$ which is a contradiction.

Case 2: Assume that G_i is empty and I also has an element in $\mu(Z_{i+1})$ (the case for $\mu(Z_{i-1})$ is similar). Then the elements of $\mu(Z_i)$ and $\mu(Z_{i+1})$ belong to a single block in $\mu(Z)$. As in the previous case, fold back according to μ and observe that there will be a maximal interval associated in Z that has elements both in Z_i and Z_{i+1} . This contradicts the fact that Z_i and Z_{i+1} are separate blocks in Z . \square

Using Lemma 5.8, to prove Lemma 5.1 we can restrict our attention to zig-zags such that the zig-zag itself is a single block.

5.1.1 The Left-Right Case

Now we are ready to prove the left-right case of Lemma 5.1.

Proof of Lemma 5.1 in the left-right case. Assume that the parameters of μ

5.1 The Mirror Operator

and ν are $[a, b], r$ and $[c, d], s$, respectively. Let $M = \text{MaxI}_{[a,b]} \cup \text{MaxI}_{[c,d]}$. Order the maximal intervals in M according to the right endpoints. Denote the h -th maximal interval in M by $M[h]$. Because we can assume that Z is a single block and we are in the left-right case we have that

- As we traverse the maximal intervals in M , they alternate between being associated with μ and ν ;
- Let I and J be consecutive intervals in M . Then $I_l < J_l \leq I_r < J_r$.

Without loss of generality assume that the first maximal interval in M is associated with μ . Let $Z[h]$ be the sequence t_1, t_2, \dots, t_j where $t_j = I_r$ and I is the h -th element in M . Furthermore, let $Z_{\mu\nu}[h]$ denote

$$\mu(M[1]) \oplus_{q_1} \nu(M[2]) \oplus_{q_2} \mu(M[3]) \oplus_{q_3} \dots \oplus_{q_{h-1}} \nu(M[h])$$

where q_u denotes the number of elements that coincide at the end of $M[u]$ and the beginning of $M[u+1]$ in Z . Now let $P(h)$ be the statement

$$Z_{\mu\nu}[h] = \nu(\mu(Z[h])) = \mu(\nu(Z[h])).$$

Observe that $P(|M|)$ proves the left-right case of the lemma because $Z[|M|] = Z$. Clearly, $P(1)$ is true. Assume that $P(h)$ holds and assume without loss of generality that $M[h]$ is a maximal interval associated with μ (and therefore $M[h+1]$ is associated with ν). It follows that

$$\nu(\mu(Z[h+1])) = \nu(\mu(Z[h] \oplus_{q_h} M[h+1])) \tag{5.1}$$

$$= \nu(\mu(Z[h]) \oplus_{q_h} \mu(M[h+1])) \tag{5.2}$$

$$= \nu(\mu(Z[h]) \oplus_{q_h} M[h+1]) \tag{5.3}$$

$$= \nu(\mu(Z[h])) \oplus_{q_h} \nu(M[h+1]) \tag{5.4}$$

$$= Z_{\mu\nu}[h] \oplus_{q_h} \nu(M[h+1]) \tag{5.5}$$

$$= Z_{\mu\nu}[h+1] \tag{5.6}$$

5.1 The Mirror Operator

Step (5.1): By definition.

Step (5.2): Observe that $M[h + 1]$ does not contain a maximal interval associated with μ because we are in the left-right case. The correctness of Step (5.2) follows.

Step (5.3): Also follows from the fact that $M[h + 1]$ does not contain a maximal interval associated with μ .

Step (5.4): Consider applying μ to $Z[h]$ and in particular observe that $\mu(Z[h])$ ends with the maximal interval $\mu(M[h])$. Let R denote the rightmost copy of $M[h]$ in $\mu(Z[h])$. Observe that the $(q_h + 1)$ -th element of R counting from the right is not in $[c, d]$ because that would contradict the maximality of $M[h + 1]$ in Z . It follows that the rightmost maximal interval associated with ν in $\mu(Z[h]) \oplus_{q_h} M[h + 1]$ is $M[h + 1]$.

Furthermore, R does not contain a maximal interval associated with ν because we are in the left-right case. In particular, the first q_h elements of R counting from the right do not contain a maximal interval associated with ν . The correctness of Step (5.4) follows.

Step (5.5): Follows from the inductive hypothesis.

Step (5.6): By definition of $Z_{\mu\nu}[h + 1]$.

Similarly, we have that

$$\mu(\nu(Z[h + 1])) = \mu(\nu(Z[h] \oplus_{q_h} M[h + 1])) \quad (5.7)$$

$$= \mu(\nu(Z[h]) \oplus_{q_h} \nu(M[h + 1])) \quad (5.8)$$

$$= \mu(\nu(Z[h])) \oplus_{q_h} \mu(\nu(M[h + 1])) \quad (5.9)$$

$$= \mu(\nu(Z[h])) \oplus_{q_h} \nu(M[h + 1]) \quad (5.10)$$

$$= Z_{\mu\nu}[h] \oplus_{q_h} \nu(M[h + 1]) \quad (5.11)$$

$$= Z_{\mu\nu}[h + 1] \quad (5.12)$$

Step (5.7): By definition.

Step (5.8): Observe that the rightmost elements of $Z[h]$ starting immediately after the rightmost element of $M[h - 1]$ do not contain a maximal

5.1 The Mirror Operator

interval associated with ν . To see this notice that these elements belong to $M[h]$ which is a maximal interval associated with μ and that we are in the left-right case. The correctness of Step (5.8) follows.

Step (5.9): Observe that $\nu(Z[h])$ ends in $M[h]$. Furthermore, the element in $\nu(Z[h]) \oplus_{q_h} \nu(M[h+1])$ after the rightmost element of $\nu(Z[h])$ is not in $[a, b]$ because that would contradict the maximality of $M[h]$ in Z . Notice also that $\nu(M[h+1])$ does not contain a maximal interval associated with μ . The correctness of Step (5.9) follows.

Step (5.10): Follows because $\nu(M[h+1])$ does not contain a maximal interval associated with μ .

Step (5.11): Follows from the inductive hypothesis.

Step (5.12): By definition of $Z_{\mu\nu}[h+1]$.

□

5.1.2 The Containment Case

First we describe a form of concatenation that will be important later. Let A and B be zig-zags. If the last integer in A is different from the first integer in B then AB denotes $A \oplus B$. If the last integer in A is the same as the first integer in B then AB denotes $A \oplus_1 B$. For example, if $X = 1, 2$ and $Y = 2, 3$ then $XY = 1, 2, 3$ but if $Y = 3, 4$ then $XY = 1, 2, 3, 4$. Now we are ready to prove Lemma 5.1 in the containment case.

Proof of Lemma 5.1 in the containment case. Let $Z = t_1, \dots, t_p$. Assume that the parameters of μ and ν are $[a, b], r$ and $[c, d], s$, respectively. Without loss of generality, assume that $[a, b]$ contains $[c, d]$. By Lemma 5.8, we can assume that Z is a single block. Since we are in the containment case, we can easily see that Z must be a single maximal interval associated with μ . We denote by N_i the i -th maximal interval associated with ν , counting from the left. Let G_0, G_1, \dots, G_n be such that $Z = G_0 N_1 G_1 \dots N_n G_n$. Note that G_0 and G_n could be empty (but G_k cannot be empty for $1 \leq k \leq n-1$).

5.1 The Mirror Operator

Furthermore, if Y is a sequence of integers s_1, s_2, \dots, s_m then \bar{Y} denotes these integers in the reverse order, i.e. s_m, s_{m-1}, \dots, s_1 . Also, let X^k for some $k \in \mathbb{Z}^+$ denote

$$\underbrace{XX \dots X}_{k \text{ times}}.$$

As an example, let $Y = 1, 2, 1$ and observe that $Y^3 = 1, 2, 1, 2, 1, 2, 1$.

First assume that both G_0 and G_n are non-empty. Then we have that

$$\begin{aligned} \nu(\mu(Z)) &= \nu(\mu(G_0 N_1 \dots G_n)) \\ &= \nu(G_0 N_1 \dots G_n (\overline{G_0 N_1 \dots G_n} G_0 N_1 \dots G_n)^r) \\ &= \nu(G_0 N_1 \dots G_n (\bar{G}_n \dots \bar{N}_1 \bar{G}_0 G_0 N_1 \dots G_n)^r) \\ &= G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n (\bar{G}_n \dots \bar{N}_1 (N_1 \bar{N}_1)^s \bar{G}_0 G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n)^r \end{aligned}$$

and

$$\begin{aligned} \mu(\nu(Z)) &= \mu(\nu(G_0 N_1 \dots G_n)) \\ &= \mu(G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n) \\ &= G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n (\overline{G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n} G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n)^r \\ &= G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n (\bar{G}_n \dots \bar{N}_1 (N_1 \bar{N}_1)^s \bar{G}_0 G_0 N_1 (\bar{N}_1 N_1)^s \dots G_n)^r. \end{aligned}$$

We show the case when G_0 is empty. The cases when G_n is empty or both G_0 and G_n are empty are similar. As above, let us segment Z as $Z = N_1 G_1 \dots G_n$. Then

$$\begin{aligned} \nu(\mu(Z)) &= \nu(\mu(N_1 G_1 \dots G_n)) \\ &= \nu(N_1 G_1 \dots G_n (\bar{G}_n \dots \bar{G}_1 \bar{N}_1 N_1 G_1 \dots G_n)^r) \\ &= N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n (\bar{G}_n \dots \bar{G}_1 \bar{N}_1 N_1 (\overline{\bar{N}_1 N_1 \bar{N}_1 N_1})^s G_1 \dots G_n)^r \\ &= N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n (\bar{G}_n \dots \bar{G}_1 \bar{N}_1 N_1 (\bar{N}_1 N_1 \bar{N}_1 N_1)^s G_1 \dots G_n)^r \\ &= N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n (\bar{G}_n \dots \bar{G}_1 (\bar{N}_1 N_1)^{2s+1} G_1 \dots G_n)^r \end{aligned}$$

5.2 The Free Derivation Path

and

$$\begin{aligned}
\mu(\nu(Z)) &= \mu(\nu(N_1 G_1 \dots G_n)) \\
&= \mu(N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n) \\
&= N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n \overline{(N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n)^r} \\
&= N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n (\bar{G}_n \dots \bar{G}_1 (\bar{N}_1 N_1)^s \bar{N}_1 N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n)^r \\
&= N_1 (\bar{N}_1 N_1)^s G_1 \dots G_n (\bar{G}_n \dots \bar{G}_1 (\bar{N}_1 N_1)^{2s+1} G_1 \dots G_n)^r.
\end{aligned}$$

□

5.2 The Free Derivation Path

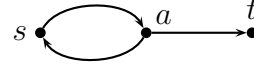
In the previous section we introduced zig-zags and the mirror operator and proved results that we need later. Now we connect these concepts with symmetric Datalog and derivation paths.

5.2.1 Basic Definitions

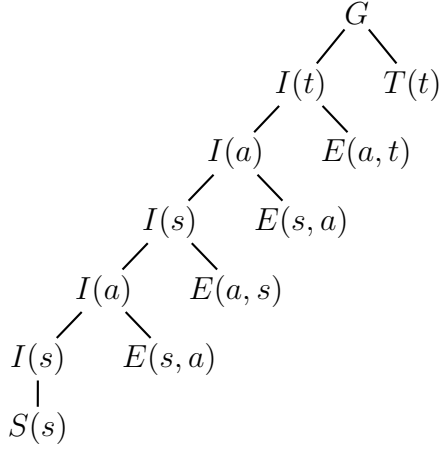
A **free derivation tree** is obtained from a derivation tree by replacing the domain elements with the underlying variables and renaming all quantified variables to different names. (See Example 5.9.) Note that in symmetric Datalog each rule contains at most one IDB in its body so the free derivation trees are in fact **free derivation paths**.

Example 5.9. Let \mathfrak{D} be the symmetric Datalog program as in Figure 5.2a. Let the input structure \mathbf{A} be the graph in Figure 5.2b together with the unary relations $S = \{s\}$ and $T = \{t\}$. Then a derivation path \mathcal{P} obtained from \mathfrak{D} over the input \mathbf{A} is shown in Figure 5.2c. Now we obtain the corresponding free derivation path by renaming each variable of \mathcal{P} such that the variables of an IDB and EDBs in the body of a rule inherit the variables of the head IDB and all other variables are renamed to new elements. See Figure 5.2d.

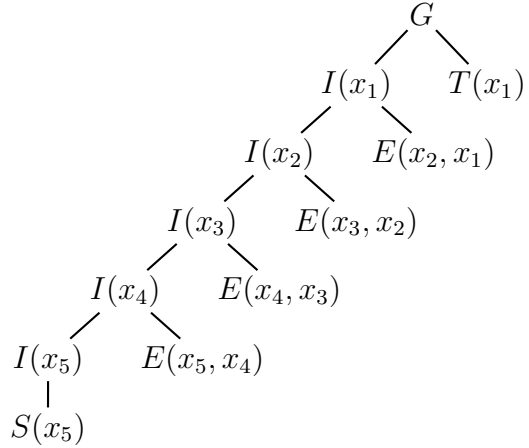
5.2 The Free Derivation Path

$$\begin{aligned}
 I(y) &\leftarrow S(y) \\
 I(y) &\leftarrow I(x); E(x, y) \\
 I(x) &\leftarrow I(y); E(x, y) \\
 G &\leftarrow I(y); T(y)
 \end{aligned}$$


(a) The symmetric Datalog program \mathfrak{D} . (b) The edge relation of the input structure.



(c) A derivation path \mathcal{P} for \mathbf{A} .



(d) The corresponding free derivation path \mathcal{F} for \mathbf{A} . Note that the second a becomes x_4 and the second s becomes x_5 .

Figure 5.2: The construction of a free derivation path.

If we are given a zig-zag $Z = t_1, \dots, t_q$ and a free derivation path \mathcal{F} having q IDBs then we can construct a corresponding **labeled free derivation path** \mathcal{F}^Z as follows. Label the i -th IDB of \mathcal{F}^Z counting from the goal predicate with t_i , $1 \leq i \leq q$. If an IDB I is labeled with t_i we denote this fact by I_{t_i} . Let \mathcal{F}^Z be a labeled free derivation path and $\mu_{[a,b],r}$ be a mirror operator. First we define $\mu(\mathcal{F}^Z)$ and then we give an example.

1. For each pair $\langle i, j \rangle \in \text{MaxP}_{[a,b]}(Z)$ we insert r consecutive copies of the

5.2 The Free Derivation Path

sequence $I_{t_{j-1}}, I_{t_{j-2}}, \dots, I_{t_i}, I_{t_{i+1}}, \dots, I_{t_j}$ after I_{t_j} in \mathcal{F}^Z .

2. Let $I_{t_k} - I_{t_{k+1}}$ be a segment of \mathcal{F}^Z and let \mathfrak{R} be the corresponding rule. Whenever $I_{t_k} - I_{t_{k+1}}$ is inserted in the new derivation path the corresponding rule is \mathfrak{R} and the parent of the EDBs of \mathfrak{R} is I_{t_k} . On the other hand, whenever we insert $I_{t_{k+1}} - I_{t_k}$ the rule corresponding to $I_{t_{k+1}} - I_{t_k}$ is the symmetric pair \mathfrak{R}' of \mathfrak{R} and accordingly, the parent of the EDBs of \mathfrak{R}' is $I_{t_{k+1}}$;
3. $\mu(\mathcal{F}^Z)$ is labeled with $\mu(Z)$;
4. Starting at the goal predicate, traverse the variables of $\mu(\mathcal{F}^Z)$ and rename them to a new name whenever it is possible. This ensures that $\mu(\mathcal{F}^Z)$ is free.

Clearly, if \mathcal{F}^Z is a labeled free derivation path that can be constructed from the rules of a symmetric Datalog program \mathfrak{D} then $\mu(\mathcal{F}^Z)$ can also be constructed from the rules of \mathfrak{D} .

Example 5.10. Let \mathfrak{D} be the same program as in Figure 5.2a. Consider a free derivation path \mathcal{F} for this program shown in Figure 5.3a. Let $Z = 1, 2, 3$ be a zig-zag. Then \mathcal{F}^Z is shown in Figure 5.3b. Figure 5.3c shows the intermediate step when the variables are not yet renamed to new elements and Figure 5.3d shows $\mu_{[2,3],1}(\mathcal{F})$.

Let τ be the input vocabulary of a symmetric Datalog program \mathfrak{D} and let \mathcal{F}^Z be a labeled free derivation path constructed from the rules of \mathfrak{D} . We can associate a τ -structure \mathbf{F} with a \mathcal{F}^Z as follows.

- The domain F of \mathbf{F} consists of all the variables appearing in \mathcal{F}^Z .
- Let $R \in \tau$ have arity r . Put a tuple $\langle x_1, \dots, x_r \rangle \in F^r$ into the relation $R^{\mathbf{F}}$ if $R(x_1, \dots, x_r)$ is present in \mathcal{F}^Z .

5.2 The Free Derivation Path

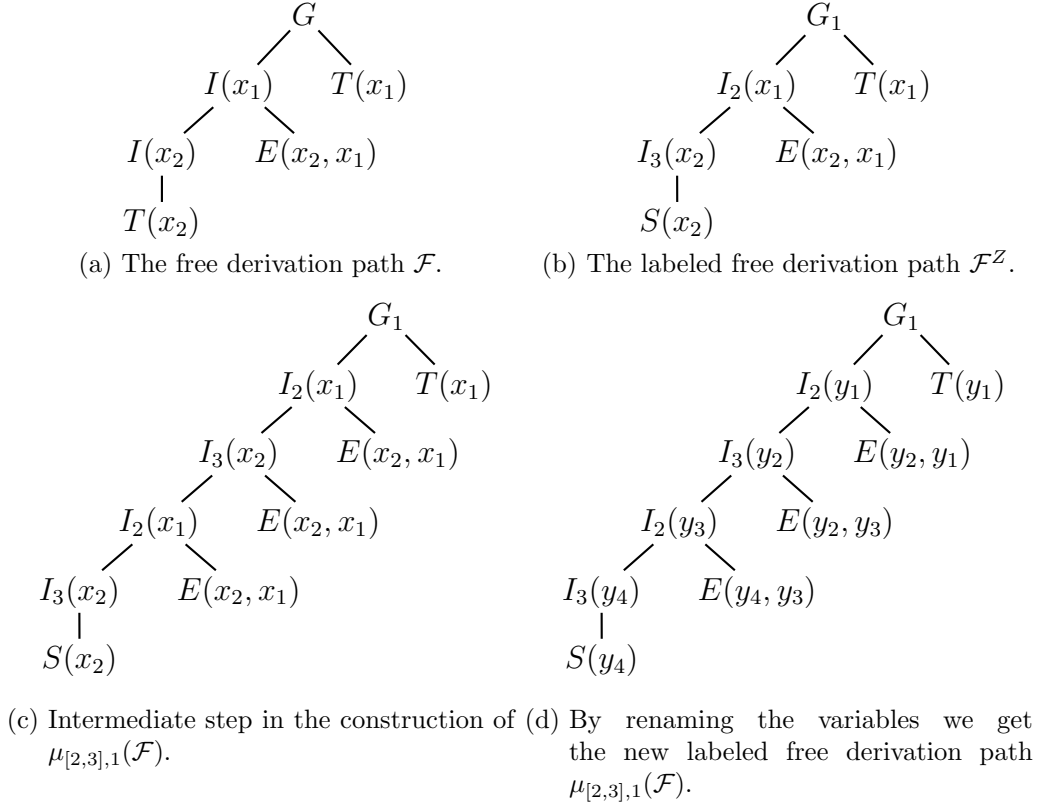


Figure 5.3: Constructing $\mu_{[2,3],1}(\mathcal{F})$.

Let \mathcal{F}^Z and $\mathcal{F}^{Z'}$ be free labeled derivation paths and \mathbf{F} and \mathbf{F}' be the corresponding relational structures, respectively. We say that there is a homomorphism from \mathcal{F}^Z to $\mathcal{F}^{Z'}$ if there is a homomorphism from \mathbf{F} to \mathbf{F}' . We denote the existence of a homomorphism h from \mathcal{F}^Z to $\mathcal{F}^{Z'}$ by $\mathcal{F}^Z \xrightarrow{h} \mathcal{F}^{Z'}$.

5.2.2 Lemmas Related to The Mirror Operator

One of our main tools is Corollary 5.11.

Corollary 5.11. *Let Z be a zig-zag, μ and ν be mirror operators, and \mathcal{F} be a free derivation path. Then*

$$\nu(\mu(\mathcal{F}^Z)) = \mu(\nu(\mathcal{F}^Z))^1.$$

5.2 The Free Derivation Path

Proof. Follows from Lemma 5.1. □

Let \mathcal{F}^Z be a labeled free derivation path and $M = \mu_1, \mu_2, \dots, \mu_n$ be a sequence of mirror operators. Let $M(\mathcal{F}^Z) = \mathcal{F}_0^Z, \mathcal{F}_1^Z, \dots, \mathcal{F}_n^Z$ be a sequence of labeled free derivation defined paths by

$$\begin{aligned}\mathcal{F}_0^Z &= \mathcal{F}^Z \\ \mathcal{F}_1^Z &= \mu_1(\mathcal{F}_0^Z) \\ \mathcal{F}_2^Z &= \mu_2(\mathcal{F}_1^Z) \\ &\vdots \\ \mathcal{F}_n^Z &= \mu_n(\mathcal{F}_{n-1}^Z).\end{aligned}$$

Fact 5.12. *Let \mathcal{F}^Z be a free labeled derivation path and $M = \mu_1, \mu_2, \dots, \mu_n$ be a sequence of mirror operators. Then $\mathcal{F}_n^Z \xrightarrow{h} \mathcal{F}^Z$.*

Proof. Notice that when we apply μ_i to \mathcal{F}_i^Z to obtain \mathcal{F}_{i+1}^Z we insert additional sequences into \mathcal{F}_i^Z and we create new variables by renaming the original variables to new names whenever possible (see Step 4 above). Let h_{i+1} be the function that maps each new variable in \mathcal{F}_{i+1}^Z to the original in \mathcal{F}_i^Z . Clearly, we have that h_{i+1} is a homomorphism from \mathcal{F}_{i+1}^Z to \mathcal{F}_i^Z and therefore $\mathcal{F}_n^Z \xrightarrow{h_1 \circ \dots \circ h_n} \mathcal{F}^Z$.

For example, we obtained the free labeled derivation path in Figure 5.3d by applying $\mu_{[2,3],1}$ to the free labeled derivation path in Figure 5.3b. Define

¹Holds after finding a 1-1 map between the variables of $\nu(\mu(\mathcal{F}^Z))$ and $\mu(\nu(\mathcal{F}^Z))$ and renaming the variables of $\mu(\nu(\mathcal{F}^Z))$ to the variables of $\nu(\mu(\mathcal{F}^Z))$, i.e. up to “isomorphism”.

5.2 The Free Derivation Path

h to be

$$y_1 \mapsto x_1$$

$$y_2 \mapsto x_2$$

$$y_3 \mapsto x_1$$

$$y_4 \mapsto x_2.$$

Observe that h is a homomorphism from $\mu_{[2,3],1}(\mathcal{F})$ to \mathcal{F}^Z . \square

Assume that τ is a vocabulary that contains a single binary relation symbol E . Let \mathbf{A} be a τ -structure. Recall that we want to show that there is no symmetric Datalog program with a binary goal predicate that defines the reflexive transitive closure of $E^{\mathbf{A}}$. In our proof, we assume that such a symmetric Datalog program \mathfrak{D} exists and derive a contradiction.

Given a labeled free derivation path \mathcal{F}^Z over τ we define $\mathfrak{E}(\mathcal{F}^Z)$ as follows. Index the EDBs in \mathcal{F}^Z by their distance from the root $G(u, v)$ and let $\mathfrak{E}(\mathcal{F}^Z)$ be the set of all indexed EDBs that appear in \mathcal{F}^Z . Furthermore, we say that a set $S \subseteq \mathfrak{E}(\mathcal{F}^Z)$ contains a path from x_1 to x_n if there are indexed EDBs in S such that $E_{i_1}(x_1, x_2), E_{i_2}(x_2, x_3), \dots, E_{i_k}(x_{n-1}, x_n)$ for some variables x_2, x_3, \dots, x_{n-1} . The indices of the EDBs are used to differentiate between paths which would be the same if we removed the indices. For example, a labeled free derivation path \mathcal{F}^Z could contain two paths p_1 and p_2 whose EDBs are exactly the same but an EDB $E(x, y)$ of p_1 appears at a different level in \mathcal{F}^Z than the same EDB of p_2 . If we had no indices, this difference would be lost in $\mathfrak{E}(\mathcal{F}^Z)$.

Let $M = \langle \mu_1, \mu_2, \dots, \mu_n \rangle$ be a sequence of mirror operators and $S \subseteq \mathfrak{E}(\mathcal{F}^Z)$. We define the M -**image** $S' \subseteq \mathfrak{E}(\mu_n(\dots \mu_2(\mu_1(\mathcal{F}^Z)) \dots))$ of S inductively as follows. Let $M_i = \mu_1, \dots, \mu_i$. The M_0 -image of S is S . Assume that the M_i -image of S is S_i . Now consider the construction of

$$\mu_{i+1}(\mu_i(\dots \mu_1(\mathcal{F}^Z) \dots))$$

5.3 The Main Theorem

from

$$\mu_i(\dots \mu_1(\mathcal{F}^Z) \dots).$$

For each indexed EDB E_ℓ in S_i add the corresponding indexed EDB in $\mu_{i+1}(\mu_i(\dots \mu_1(\mathcal{F}^Z) \dots))$ to S_{i+1} and any new copies of E_ℓ . (Note that the indices of the EDBs in $\mu_{i+1}(\mu_i(\dots \mu_1(\mathcal{F}^Z) \dots))$ are recomputed. The M -image of S is S_n . We need the following corollary of Lemma 5.1.

Corollary 5.13. *Let Z be a zig-zag, μ and ν be mirror operators, and \mathcal{F} be a free derivation path. Let $\mathcal{F}' = \nu(\mu(\mathcal{F}^Z)) = \mu(\nu(\mathcal{F}^Z))$. Let $S \subseteq \mathfrak{E}(\mathcal{F}^Z)$. Then in \mathcal{F}' , the $\langle \nu, \mu \rangle$ -image of S is the same as the $\langle \mu, \nu \rangle$ -image of S .*

Proof Sketch. This is easy to see by going over the proof of Lemma 5.1 and tracking the EDBs in the construction of $\nu(\mu(\mathcal{F}^Z))$ and $\mu(\nu(\mathcal{F}^Z))$. \square

Finally, we also need the following lemma.

Lemma 5.14. *Let \mathcal{F}^Z be a free labeled derivation path and $M = \mu_1, \mu_2, \dots, \mu_n$ be a sequence of mirror operators. Let $E_{u \not\rightarrow v} \subseteq \mathfrak{E}(\mathcal{F}^Z)$ be a set that contains no path from u to v for some u and v . Let $E'_{u \not\rightarrow v} \subseteq \mathfrak{E}(\mu(\mathcal{F}_n^Z))$ be the $\langle \mu \rangle$ -image of $E_{u \not\rightarrow v}$. Then $E'_{u \not\rightarrow v}$ contains no path from u to v .*

Proof. Assume that $E'_{u \not\rightarrow v}$ contains a path from u to v . Notice that “folding back” $\mu(\mathcal{F}^Z)$ according to $\mu_1, \mu_2, \dots, \mu_n$ yields a path in $E_{u \not\rightarrow v}$ and this leads to a contradiction. \square

5.3 The Main Theorem

In this section we put together the different lemmas we have proved and Lemma 5.15 to prove the main theorem. We state Lemma 5.15 here and prove it later.

Lemma 5.15. *Let \mathcal{F} be a free derivation path originating from a symmetric (j, k) -Datalog program with root $G(u, v)$. Let q be the number of IDBs in \mathcal{F}*

5.3 The Main Theorem

and let $Z = 1, 2, \dots, q$ be a zig-zag. Assume that $\mathfrak{E}(\mathcal{F}^Z)$ contains a path p from u to v of length at least ℓ where ℓ is specified in the proof. Then there exists a mirror operator μ such that the $\langle \mu \rangle$ -image of p in $\mathfrak{E}(\mu(\mathcal{F}^Z))$ does not contain any path from u to v .

Theorem 5.16. *Assume that τ is a vocabulary that contains a single binary relation symbol E . Given any symmetric Datalog program \mathfrak{D} with a binary goal predicate G , there exists a structure \mathbf{A} such that $G^{\mathbf{A}^\mathfrak{D}}$ is not the reflexive transitive closure of $E^{\mathbf{A}}$.*

Proof. Assume that there is a symmetric Datalog program \mathfrak{D} such that for each τ -structure \mathbf{A} , $G^{\mathbf{A}^\mathfrak{D}}$ is the reflexive transitive closure of $E^{\mathbf{A}}$. Let \mathbf{A} be such that $E^{\mathbf{A}}$ is a path p of length ℓ where ℓ satisfies the length condition of Lemma 5.15. Let \mathcal{F} be a free derivation path produced by \mathfrak{D} to derive p . Let q be the number of IDBs in \mathcal{F} and let $Z = 1, 2, \dots, q$ be a zig-zag. Observe that any path from u to v (the root of \mathcal{F}^Z is $G(u, v)$) in $\mathfrak{E}(\mathcal{F}^Z)$ must have length exactly ℓ . Let $P = \{p_1, \dots, p_n\}$ be the set of all paths from u to v in \mathcal{F}^Z . For each $1 \leq i \leq n$, use Lemma 5.15 to find a mirror operator μ_i such that the $\langle \mu_i \rangle$ -image of p_i does not contain a path from u to v . Let $\mathcal{F}^\emptyset = \mu_n(\mu_{n-1}(\dots \mu_1(\mathcal{F}^Z) \dots))$. Observe that by Corollary 5.11

$$\mathcal{F}^\emptyset = \mu_n(\dots \mu_{i-1}(\mu_{i+1}(\dots \mu_1(\mu_i(\mathcal{F})) \dots)) \dots).$$

for each $1 \leq i \leq n$. We claim that $\mathfrak{E}(\mathcal{F}^\emptyset)$ does not contain any path from u to v .

For the sake of contradiction assume that $\mathfrak{E}(\mathcal{F}^\emptyset)$ contains a path p from u to v . Let h be the homomorphism defined in Fact 5.12 from \mathcal{F}^\emptyset to \mathcal{F}^Z . Then $h(p)$ is a path in $\mathfrak{E}(\mathcal{F}^Z)$ and therefore $h(p) = p_i$ for some i . By construction, the $\langle \mu_i \rangle$ -image of p_i in $\mathfrak{E}(\mu_i(\mathcal{F}^Z))$ does not contain a path from u to v . Using Lemma 5.14 we have that the $\langle \mu_i, \mu_1, \dots, \mu_{i-1}, \mu_{i+1}, \dots, \mu_n \rangle$ -image of p_i in $\mathfrak{E}(\mathcal{F}^\emptyset)$ does not contain a path from u to v .

Because by Corollary 5.13 the $\langle \mu_i, \mu_1, \dots, \mu_{i-1}, \mu_{i+1}, \dots, \mu_n \rangle$ -image and

5.4 Disconnecting an Isolated UV-Path

the $\langle \mu_1, \mu_2, \dots, \mu_n \rangle$ -image of p_i are the same, the $\langle \mu_1, \mu_2, \dots, \mu_n \rangle$ -image of p_i in \mathcal{F}^\emptyset does not contain a path from u to v . This leads to a contradiction because p is such a path. \square

5.4 Disconnecting an Isolated UV-Path

5.4.1 The UV-Path Following Diagram

As in the previous section, we assume that the input vocabulary τ contains a single binary relation symbol E . Let \mathcal{F}^Z be a labeled derivation path and assume that \mathcal{F}^Z contains a path $p = E_{i_1}(u, x_1), (x_1, x_2), \dots, E_{i_n}(x_n - 1, v)$ from u to v . For simplicity, we will not write the indices of the EDBs from now on. A ***uv*-path following diagram** is a tool to investigate this path. Figure 5.4 shows a *uv*-path following diagram for p . The x -axis on the top is labeled from left to right by the variables $u, x_1, \dots, x_{n-2}, v$ (we can think of these as vertices) of p in the order as they appear on the path. The y -axis on the left is labeled from top to bottom by the IDBs of \mathcal{F}^Z starting at the goal predicate. Each IDB is associated with a horizontal non-dashed grey line which we call **IDB-lines**. Furthermore, there is a dashed grey line between each pair of IDB-lines, above the IDB line corresponding to the goal predicate and below the bottommost IDB line. We call these lines **edge layers**. Edge layers are intended to represent those EDBs of a rule that belong to p .

To represent p , start with the first EDB $E(u, x_1)$ of p . Identify the rule that produced $E(u, x_1)$ in \mathcal{F}^Z and let I be the IDB in the head and J be the IDB in the body (if any) of this rule. Represent $E(u, x_1)$ in the diagram by putting an arrow on the edge layer between I and J (or the edge layer below the last IDB line) such that the tail of the arrow is under u and its head is under x_1 . Repeat this for each EDB-edge of p . To continue the construction of the diagram we need the following observation.

Let x_i be a vertex (variable) of p . Assume that x_i appears in two different EDBs E_1 and E_2 of \mathcal{F}^Z where E_1 and E_2 have different indices. Assume that

5.4 Disconnecting an Isolated UV-Path

E_1 is closer to the goal predicate than E_2 in \mathcal{F}^Z and that the rule of E_1 has the IDB I in its body and the rule of E_2 has head J . Then it is easy to see that each IDB in \mathcal{F}^Z between (inclusive) I and J must contain x_i among its variables. A similar claim holds when E_2 is closer to the goal predicate.

To finish the construction of the uv -path following diagram, draw a vertical line under each vertex x_i in the following way. Consider the first occurrence of x_i as we traverse \mathcal{F}^Z starting at the goal predicate. Note that the first occurrence of x_i could happen either in an IDB or in an EDB of \mathcal{F}^Z and accordingly, let the vertical line start at the IDB line or the edge layer of the EDB, respectively. Similarly, the vertical line ends at the last occurrence of x_i . These vertical lines are called *bridges*.

We make the following observation. If \mathcal{F}^Z is a labeled free derivation path constructed from the rules of a symmetric (j, k) -Datalog program and $\mathfrak{E}(\mathcal{F}^Z)$ contains a path p then any IDB-line can be “touched” by at most j bridges. By touching we mean either that the bridge crosses that IDB line or has an endpoint on that IDB line. In addition, each edge layer can contain at most $k - 1$ EDBs of p .

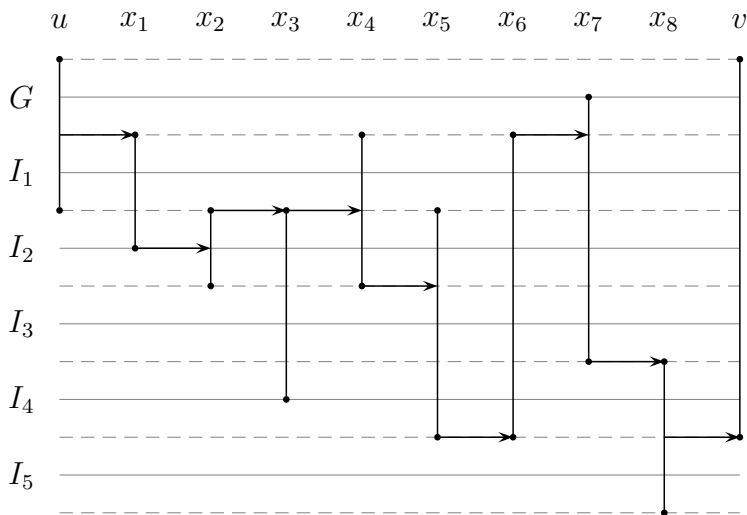


Figure 5.4: Following a uv -path in a free derivation path.

5.4 Disconnecting an Isolated UV-Path

5.4.2 The Disconnecting Lemma

Lemma 5.17. *Let \mathcal{F}^Z be a labeled free derivation path originating from a symmetric Datalog program in which the goal IDB has two variables u and v . Let $p = E(x_0 = u, x_1), \dots, E(x_{n-2}, x_{n-1} = v)$ be a path in $\mathfrak{E}(\mathcal{F}^Z)$ from u to v of length at least ℓ , where ℓ is specified in the proof. Then there exists a consecutive sequence of IDBs in \mathcal{F}^Z such that:*

1. *Let $I(\bar{\alpha})$ be the topmost and $J(\bar{\beta})$ be the bottommost rule in the sequence. There exist two edges of p $E(x_{i-1}, x_i)$ and $E(x_j, x_{j+1})$, $i < j$, such that the following holds. $E(x_{i-1}, x_i)$ appears in the body of a rule whose head is strictly above I and $E(x_j, x_{j+1})$ appears in the body of a rule whose head is J or below J . (The case when $E(x_{i-1}, x_i)$ is below and $E(x_j, x_{j+1})$ is above is similar.);*
2. *Let $E(x_h, x_{h+1})$ be an edge of p such that $i \leq h \leq j - 1$. Then $E(x_h, x_{h+1})$ is in the body of a rule whose head IDB is I or below I but strictly above J ;*
3. *For any variable x_h , $i \leq h \leq j$, $x_h \in \bar{\alpha} \rightarrow x_h \notin \bar{\beta}$ and $x_h \in \bar{\beta} \rightarrow x_h \notin \bar{\alpha}$.*

For example, the IDBs I_1 , I_2 and I_3 in Figure 5.4 satisfy the above conditions where the two edges in condition 1 are $E(u, x_1)$ $E(x_5, x_6)$.

Proof. Observe that satisfying the conditions of the lemma corresponds to finding a consecutive sequence of IDB lines in the uv -path following diagram such that:

1. Let I_t be the topmost and I_b be the bottommost IDB-line in the sequence. There exists two edges $E(x_{i-1}, x_i)$ and $E(x_j, x_{j+1})$, $i < j$, such that the following holds. $E(x_{i-1}, x_i)$ appears in an edge layer above I_t and $E(x_j, x_{j+1})$ appears in an edge layer below I_b . (The case when $E(x_{i-1}, x_i)$ is below and $E(x_j, x_{j+1})$ is above is similar.);

5.4 Disconnecting an Isolated UV-Path

2. Let $E(x_h, x_{h+1})$ be any edge of p such that $i \leq h \leq j - 1$. Then the edge layer containing $E(x_h, x_{h+1})$ is between I_t and I_b ;
3. For any h , $i \leq h \leq j$, the bridge corresponding to x_h has at least one of its endpoints between I_t and I_b .

We will define a function $L_k(m)$ and show by induction on m the following. If a symmetric Datalog program has width (j, k) and it derives a path p of length (at least) $L_k(j)$ then in the uv -path following diagram corresponding to the labeled free derivation path of p we can find a sequence of IDB-lines satisfying conditions 1,2 and 3. ℓ in the statement of the Lemma is set to $L_k(j)$.

Set $L_k(1) = 3(k - 1)$ and $L_k(m) \geq 4L_k(m - 1) + 6$. For the base case, observe that if any IDB-line can be crossed by at most one bridge and we have edges in at least three different edge layers (because an edge layer can contain at most $k - 1$ edges) then it is trivial to pick a consecutive sequence of IDB-lines that satisfy conditions 1,2 and 3.

Assume that the path has length at least $L_k(m)$. Let L be the lowest edge layer or IDB-line in the uv -path following diagram that contains the bottom of a bridge corresponding to a variable of p . Similarly, let H be the highest edge layer or IDB-line that contains the top of a bridge corresponding to a variable of p .

Case 1: Assume that there are 3 or less bridges which together cover the vertical gap between L and H . Remove these bridges together with the (at most 6) edges incident on them to get at most 4 subpaths. Now there is at least one subpath which has length at least $\frac{L_k(j)-6}{4} \geq L_k(j - 1)$ and at most $j - 1$ bridges cross any IDB-lines. Therefore by the inductive hypothesis we can pick a consecutive sequence of IDB-lines that satisfy conditions 1,2 and 3.

Case 2: Assume that no set of 3 or less bridges cover the gap between L and H (see Figure 5.5). Choose two bridges \mathcal{B}_1 with bottom b_1 and top t_1 , and \mathcal{B}_2 with bottom b_2 and top t_2 such that \mathcal{B}_2 is to the right of \mathcal{B}_1 . Without

5.4 Disconnecting an Isolated UV-Path

loss of generality assume that t_1 touches H and b_2 touches L . (The case when t_2 touches H and b_1 touches L is similar.) Let a be the edge leaving \mathcal{B}_1 to the right and b be the edge coming into \mathcal{B}_2 from the left. Notice that b must be below a , otherwise \mathcal{B}_1 and \mathcal{B}_2 cover the gap between L and H .

Follow the edges of p from a towards b until an edge a' is reached such that all the edges between a' and b are strictly below a' . Similarly, follow the edges of p from b towards a' until an edge b' is reached such that all the edges between b' and a' are strictly above b' . Observe that there must be an edge between a' and b' because if not then the bridge corresponding to the common variable of a' and b' together with \mathcal{B}_1 and \mathcal{B}_2 would cover the gap between L and H .

Let I_t be the IDB-line just below a' and set $E(x_{i-1}, x_i) = a'$. Let I_b be the IDB-line just above b' and set $E(x_j, x_{j+1}) = b'$. Considering that there is an edge between a' and b' , this choice of I_t and I_b satisfies condition 1. Condition 2 is also satisfied because of the way we chose a' and b' . Finally, assume that there is a bridge \mathcal{C} corresponding to x_h , $i \leq h \leq j$, such that none of its endpoints are between I_t and I_b . But then \mathcal{B}_1 , \mathcal{C} and \mathcal{B}_2 would cover the gap between L and H which is a contradiction. Therefore condition 3 is satisfied.

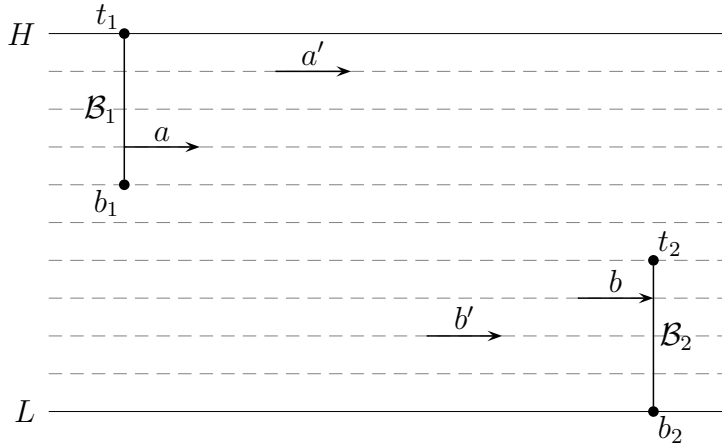


Figure 5.5: Case 2 in the proof of Lemma 5.17.

5.4 Disconnecting an Isolated UV-Path

□

Now we are ready to prove Lemma 5.15.

Proof of Lemma 5.15. Let ℓ be the same as in Lemma 5.17 and find a sequence S of rules in \mathcal{F}^Z which satisfies the conditions of Lemma 5.17. Let a be the label of the topmost IDB and b be the label of the bottommost IDB of S . We claim that for any $r > \frac{2j-1}{2}$, the operator $\mu_{[a,b],r}$ satisfies the condition given in the statement of the lemma.

To show this we construct a more graph theoretic representation of the segment of the uv -path following diagram between the IDB lines I_a and I_b . Let the two edges specified in condition 1 of Lemma 5.17 be $E(x_{i-1}, x_i)$ and $E(x_j, x_{j+1})$. Let $P = \{x_i, x_{i+1}, \dots, x_j\}$ and partition P into three parts as follows. The first set S contains those variables which belong to the IDB I_a and the second set T contains those variables which belong to the IDB I_b . M contains the rest of the variables. Now write the variables in S from left to right as they appear in I_a . Put the variables of M below the variables of S . Write the variables in T from left to right as they appear in I_b , below the variables of M . Finally, add any edge of p which has both of its endpoints in P . For example, the IDBs I_1, I_2 and I_3 in Figure 5.4 satisfy the conditions of Lemma 5.17 and our corresponding representation is shown in Figure 5.6a.

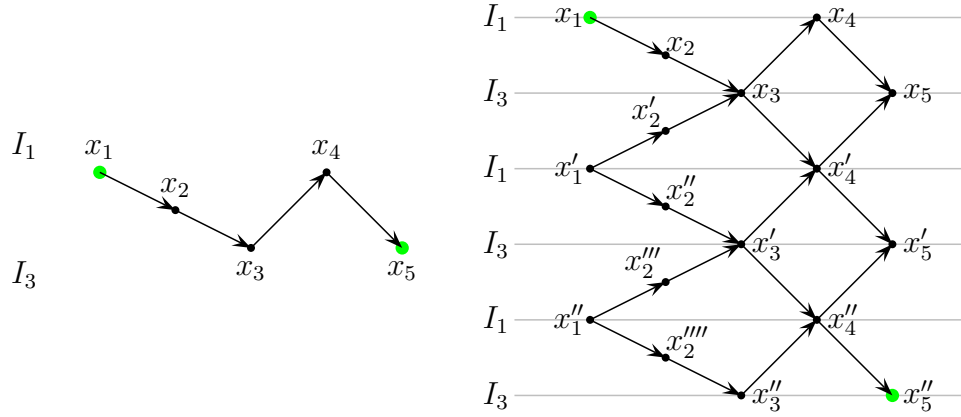
Let \bar{G} denote the vertical mirror image of G and construct G' from G as follows. Insert

$$\underbrace{\bar{G}G\bar{G}\dots G}_{2r}$$

under the original G and between any two consecutive copies of G (one is the mirror image of the other) identify the corresponding variables and rename each variable to a new name whenever possible. For example, see Figure 5.6b. Assume that the following two conditions hold.

1. G' contains no path from x_1 to the bottommost copy of x_m (in the example x_5'');

5.4 Disconnecting an Isolated UV-Path



- (a) A more graph theoretic representation of the segment of the uv -path following diagram between the IDB lines I_1 and I_3 in Figure 5.4.
- (b) The effect of μ on the representation in Figure 5.6a, assuming that $j = 2$. Notice that there is no path from x_1 to x''_5 .

Figure 5.6: The “mirroring” process.

- Let the variables of I_a and I_b which also belong to p be a_1, \dots, a_f and b_1, \dots, b_g , respectively. We say that two variables a_c ($1 \leq c \leq f$) and b_d ($1 \leq d \leq g$) are p -**connected** if there is a path from a_c to b_d or from b_d to a_c using the edges in $\mathfrak{C}(p)$. Note that such a path could have length zero if a_c is the same variable as b_d .

Then if in \mathcal{F}^Z , a_c and b_d are not p -connected then in $\mu(\mathcal{F}^Z)$, a_c and the copy of b_d in the last copy of I_b are not $\mu(p)$ -connected.

If the above two conditions hold then notice that the μ -image of p cannot contain a path from u to v ;

To see that **2** holds assume that in $\mu(\mathcal{F}^Z)$, a_c and the copy of b_d in the last copy of I_b are $\mu(p)$ -connected. Then “folding back” shows that a_c and b_d must be p -connected in \mathcal{F}^Z .

To see that **1** holds assume that there is a path p' from x_1 to the bottommost copy of x_m . This path has length at least $2r + 1$ so it has at least $2r + 2$ vertices. It follows that if r is at least $\frac{2j-1}{2}$ then p' visits at least $2j + 1$

5.5 Directed ST-Connectivity Is Not in Symmetric Datalog

vertices. Now “fold back”

$$\underbrace{\bar{G}G\bar{G}\dots G}_{2r}$$

onto G and let p'' be the image of p' in G . Notice that every edge of p'' is in G . This implies that p'' is a path in G that visits $2j + 1$ vertices. But there are at most $2j$ vertices so we must have a cycle in P and this leads to a contradiction. \square

5.5 Directed ST-Connectivity Is Not in Symmetric Datalog

In this section we show that finding the reflexive transitive closure of a binary relation is essentially the same problem as $\neg\text{CSP}(\Delta)$ or the directed st -connectivity problem, where Δ is defined as follows. Let τ be the vocabulary $\langle S, T, E \rangle$, where S and T are unary relation symbols and E is a binary relation symbol. Let Δ be the τ -structure such that its domain is $\{0, 1\}$, E^Δ is the less-than-or-equal-to relation, i.e. $\{\langle 0, 0 \rangle \langle 0, 1 \rangle \langle 1, 1 \rangle\}$, $S^\Delta = \{1\}$, and $T^\Delta = \{0\}$.

In the st -connectivity problem we are given a vertex set V , an edge relation E and two constants s and t . The task is to decide whether there is a path from s to t . If we allow constants in our structures, we can think of this problem as $\neg\text{CSP}(\Theta)$ where Θ is the structure $\langle \{0, 1\}; \leq, c_1, c_0 \rangle$, c_1 is the constant whose value is 1 and c_0 is the constant whose value is 0.

In addition to the previous definitions, we need to define σ as the vocabulary that contains a binary relation symbol E .

Lemma 5.18. *The following statements are equivalent:*

1. *There exists a symmetric Datalog program \mathcal{D} such that for any σ -structure \mathbf{A} , $G^{\mathbf{A}^\mathcal{D}}(u, v)$ is the reflexive transitive closure of $E^\mathbf{A}$.*
2. *$\neg\text{CSP}(\Delta)$ is definable in symmetric Datalog;*

5.5 Directed ST-Connectivity Is Not in Symmetric Datalog

3. $\neg\text{CSP}(\Theta)$ is definable in symmetric Datalog;

Proof. We show the equivalence of 1 and 2. The equivalence of 1 and 3 can be proved similarly. Recall that by Lemma 3.4, a Datalog program for $\neg\text{CSP}(\Delta)$ must accept precisely those τ -structures which contain a path from a vertex in their S relation to a vertex in their T relation.

Let \mathbf{A} be a σ -structure. Assume that there exists a symmetric Datalog program \mathfrak{D} with goal predicate $G(u, v)$ such that $G(u, v)^{\mathbf{A}^\mathfrak{D}}$ is the reflexive transitive closure of $E^{\mathbf{A}}$. Construct a new goal predicate G' and add the rule $G' \leftarrow G(x, y); S(x), T(y)$ to \mathfrak{D} . Clearly, this program is symmetric and given a τ -structure \mathbf{B} , G' is nonempty if and only if there is a path from a vertex in $S^{\mathbf{B}}$ to a vertex in $T^{\mathbf{B}}$.

Conversely, let \mathfrak{D} be a symmetric Datalog program that decides $\neg\text{CSP}(\Delta)$. First, we produce a program \mathfrak{D}' from \mathfrak{D} that decides $\neg\text{CSP}(\Delta)$ such that every rule of \mathfrak{D}' contains at most one occurrence of a unary EDB. To obtain \mathfrak{D}' , identify the variables that occur in a rule in the EDB S , and similarly for the EDB T . For example the rule

$$I(x, y, z) \leftarrow J(x, y), S(x), S(z), T(y), T(w)$$

becomes

$$I(x, y, x) \leftarrow J(x, y), S(x), T(y).$$

Let \mathcal{C} be the set of structures accepted by \mathfrak{D} and \mathcal{C}' be the set of structures accepted by \mathfrak{D}' . Identification of the variables can be thought of as imposing equality relations on the variables. Therefore the set of conditions in a rule of \mathfrak{D}' is a superset of the set of conditions in a rule of \mathfrak{D} so $\mathcal{C}' \subseteq \mathcal{C}$.

Conversely, assume that \mathfrak{D} accepts a τ -structure \mathbf{B} . Because \mathfrak{D} accepts \mathbf{B} , there is a path from a vertex $s \in S^{\mathbf{B}}$ to a vertex $t \in T^{\mathbf{B}}$. Construct \mathbf{B}' from \mathbf{B} by removing any edge from $E^{\mathbf{B}}$ which does not belong to this path

5.5 Directed ST-Connectivity Is Not in Symmetric Datalog

and observe that by the definition of \mathfrak{D} , \mathbf{B}' is also accepted by \mathfrak{D} . In addition, there is a homomorphism from \mathbf{B}' to \mathbf{B} . Observe that $|S^{\mathbf{B}'}| = |T^{\mathbf{B}'}| = 1$ so in a derivation path \mathcal{T}' for \mathbf{A}' all variables appearing in S must be instantiated to the same element and similarly, all variables appearing in T must be instantiated to the same element and this way we can get a derivation tree for \mathfrak{D}' over \mathbf{B}' . Using the fact that $\mathbf{B}' \rightarrow \mathbf{B}$, \mathfrak{D}' also accepts \mathbf{B} . Therefore $\mathcal{C} \subseteq \mathcal{C}'$ and it follows that $\mathcal{C} = \mathcal{C}'$.

Now create a new program \mathfrak{D}'' from \mathfrak{D}' as follows.

- To each IDB of \mathfrak{D}' , including the goal predicate, add two variables (call them u and v);
- In every rule that contains an occurrence of S , rename the variable in S to u . Similarly, in every rule that contains an occurrence of T , rename the variable in T to v .
- Remove any occurrence of the EDBs S and T .

We claim that given a σ -structure \mathbf{A} , $G(u, v)^{\mathbf{A}^{\mathfrak{D}''}}$ is the reflexive transitive closure of $E^{\mathbf{A}}$. This clearly follows from the following claim.

Claim. Let \mathbf{A} be a σ -structure and let \mathbf{A}' be the τ -structure obtained from \mathbf{A} by adding the unary relations $S = \{a\}$ and $T = \{b\}$. Then $\langle a, b \rangle \in G(u, v)^{\mathbf{A}^{\mathfrak{D}''}}$ if and only if \mathbf{A}' is accepted by \mathfrak{D}' .

Proof of Claim. Suppose that \mathbf{A}' is accepted by \mathfrak{D}' and let \mathcal{T}' be the corresponding derivation path. Modify \mathcal{T}' to obtain a derivation path for \mathfrak{D} over \mathbf{A} by simply assigning the value a to every occurrence of a variable u and the value b to every occurrence of a variable v . Obviously, this is a valid derivation path because the only element of $S^{\mathbf{A}'}$ is a and the only element of $T^{\mathbf{A}'}$ is b .

Now assume that $\langle a, b \rangle \in G(u, v)^{\mathbf{A}^{\mathfrak{D}''}}$ and let \mathcal{T} be the corresponding derivation path. We claim that its obvious modification is a derivation path for \mathfrak{D}' over \mathbf{A}' . First observe that by construction, $\langle a, b \rangle$ is the only pair that

5.5 Directed ST-Connectivity Is Not in Symmetric Datalog

occurs in the two extra variables u and v added to each IDB. Now simply add back the unaries $S(a)$ and $T(b)$ and erase the additional variables u and v of the IDBs to obtain a derivation path for \mathfrak{D}' over \mathbf{A}' . \square

We have proved the following theorem.

Theorem 5.19. *Let τ , σ , Δ and Θ be as above. Then*

1. *There is no symmetric Datalog program \mathfrak{D} that defines $\neg\text{CSP}(\Delta)$;*
2. *There is no symmetric Datalog program \mathfrak{D} with goal predicate $G(u, v)$ such that for every σ -structure \mathbf{A} , $G(u, v)^{\mathbf{A}^{\mathfrak{D}}}$ is the reflexive transitive closure of $E^{\mathbf{A}}$;*
3. *There is no symmetric Datalog program \mathfrak{D} that defines $\neg\text{CSP}(\Theta)$. In other words, the directed st-connectivity problem is not definable in symmetric Datalog.*

We note that it is easy to extend Theorem 5.19 for symmetric Datalog(\neg, \neq).

Chapter 6

Conclusion

6.1 Overview

In this work we introduced symmetric Datalog, a new restriction of the database-inspired logic programming language Datalog, and proved upper and lower bounds for it.

In Chapter 1 and 2 we provided the necessary definitions and surveyed cutting-edge results related to the pivotal dichotomy conjecture of Feder and Vardi. In Chapter 3 we showed that symmetric Datalog programs can be evaluated in L and we established a connection between symmetric Datalog and CSPs. An important observation is that CSPs whose complements are definable in symmetric Datalog are in L. We also showed that in the presence of the successor relation, symmetric Datalog(\neg, \neq) captures L. In Chapter 4 we gave a combinatorial description of a large class of CSPs definable in symmetric Datalog and therefore lying in L. In Chapter 5 we proved lower bounds for symmetric Datalog. In particular, we proved that directed *st*-connectivity, a related CSP and the reflexive transitive closure of a binary relation cannot be expressed in symmetric Datalog. Because undirected *st*-connectivity can be expressed in symmetric Datalog, our lower bound sheds new light on the computational differences between the L-complete undirected *st*-connectivity

6.2 Possibility for Future Work

and the NL-complete directed *st*-connectivity problems.

6.2 Possibility for Future Work

It is shown in [14] that over the two-element domain and under a standard complexity-theoretic assumption, the set of CSPs in L is precisely the set of CSPs whose complement is definable in symmetric Datalog. The results in this thesis and [14] provide preliminary evidence that symmetric Datalog may be a unifying explanation for families of CSPs in L .

Conjecture 6.1. *CSP(\mathbf{B}) is in L if and only if \neg CSP(\mathbf{B}) can be defined in symmetric Datalog.*

We note that this conjecture is similar to a conjecture of Dalmau stating that CSP(\mathbf{B}) is in NL if and only if \neg CSP(\mathbf{B}) is definable in linear Datalog [10]. Conjecture 6.1 together with Theorem 5.19 in Chapter 5 obviously implies that L is different from NL. Therefore it becomes meaningless if $L = \text{NL}$.

On the other hand, results that relate Γ or the algebra associated with Γ to the expressibility of CSP(Γ) in a given fragment of Datalog can be independent of complexity-theoretic assumptions. In [29] the following conjecture attempts to characterize symmetric Datalog (for CSPs) algebraically.

Conjecture 6.2. *\neg CSP(Γ) is in symmetric Datalog if and only if the algebra associated with Γ generates a variety of pure type 3.*

Defining varieties of pure type 3 is beyond the scope of this thesis and details can be found in [29]. One direction is established in Theorem 4.2 of [29] and this theorem crucially relies on Theorem 5.19 in Chapter 5. Whether the other direction holds is unknown but a recent result of Dalmau and Larose shows the following.

6.2 Possibility for Future Work

Theorem 6.3 ([28]). *If $\neg\text{CSP}(\Gamma)$ is definable in Datalog and the algebra associated with Γ contains a Mal'tsev term then $\neg\text{CSP}(\Gamma)$ is in symmetric Datalog*

This is an initial step towards proving Conjecture 6.2. Note the interesting combination of the algebraic (Chapter 2) and the logical approaches (Chapter 3) in Conjecture 6.2 and Theorem 6.3.

Finally, to better understand the expressive power of symmetric Datalog, it would be interesting to give a different proof of Theorem 5.19 in Chapter 5. Our proof assumes that we have a symmetric Datalog program \mathfrak{D} for reflexive transitive closure and then the program must also accept some inputs it should reject. In other words, if \mathfrak{D} is complete then it is unsound.

We suspect that there could be a different proof that uses a “dual strategy” and by this we mean the following. Find the canonical symmetric Datalog program¹ \mathfrak{D} for $\text{CSP}(\Delta)$ where Δ is defined in Section 5.5 of Chapter 5. Now show that \mathfrak{D} does not accept some structures it should, i.e. if \mathfrak{D} is sound then it is incomplete.

¹The canonical symmetric Datalog program is obtained from the canonical program by removing any recursive rule that does not have its symmetric pair.

Bibliography

- [1] M. Ajtai and R. Fagin. Reachability is harder for directed than for undirected finite graphs. *J. Symb. Log.*, 55(1):113–150, 1990. 58
- [2] M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. In *Proceedings of Computer Science Logic and the 8th Kurt Gödel Colloquium*, volume 2803 of *Lecture Notes in Computer Science*, pages 44–57. Springer-Verlag, 2003. 5
- [3] A. Bulatov. Mal'tsev constraints are tractable. Technical Report PRG-RR-02-05, Computing Laboratory, University of Oxford, Oxford, UK, 2002. 2, 19
- [4] A. Bulatov and V. Dalmau. A simple algorithm for mal'tsev constraints. *SIAM J. Comput.*, 36(1):16–27, 2006. 2, 19
- [5] A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005. 1, 6, 8, 9, 17, 18
- [6] A. A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 321, Washington, DC, USA, 2003. IEEE Computer Society. 19
- [7] A. A. Bulatov, A. A. Krokhin, and P. Jeavons. Constraint satisfaction problems and finite algebras. In *ICALP '00: Proceedings of the 27th*

BIBLIOGRAPHY

- International Colloquium on Automata, Languages and Programming*, pages 272–282, London, UK, 2000. Springer-Verlag. 21
- [8] D. Cohen, P. Jeavons, P. Jonsson, and M. Koubarakis. Building tractable disjunctive constraints. *J. ACM*, 47(5):826–853, 2000. 1
- [9] M. C. Cooper, D. A. Cohen, and P. G. Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65:347–361, 1994. 6
- [10] V. Dalmau. Linear Datalog and bounded path duality of relational structures. *Logical Methods in Computer Science*, 1(1), 2005. 6, 37, 88
- [11] V. Dalmau. Generalized majority-minority operations are tractable. *Logical Methods in Computer Science*, 2(4), 2006. 20
- [12] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. 34(1):1–38, 1988. 1
- [13] K. Denecke and S. L. Wismath. *Universal algebra and its applications in theoretical computer science*. Chapman & Hall/CRC Press, 2002. 9, 11
- [14] L. Egri, B. Larose, and P. Tesson. Symmetric datalog and constraint satisfaction problems in logspace. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 193–202, 2007. 22, 42, 57, 88
- [15] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1999. 6, 8, 31
- [16] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 5
- [17] E. Grädel. Capturing complexity classes by fragments of second-order logic. *Theor. Comput. Sci.*, 101(1):35–57, 1992. 40

BIBLIOGRAPHY

- [18] M. Gyssens, P. G. Jeavons, and D. A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66(1):57–89, 1994. [9](#)
- [19] P. Idziak, P. Markovic, R. McKenzie, M. Valeriote, and R. Willard. Tractability and learnability arising from algebras with few subpowers. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 213–224, Washington, DC, USA, 2007. IEEE Computer Society. [20](#)
- [20] N. Immerman. *Descriptive complexity*. Graduate Texts in Computer Science. Springer, 1999. [40](#)
- [21] P. Jeavons. On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.*, 200(1-2):185–204, 1998. [9](#), [13](#), [16](#), [21](#)
- [22] P. Jeavons, D. Cohen, and M. C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2):251–265, 1998. [8](#), [19](#)
- [23] P. Jeavons, D. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In *CP '95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 276–291, London, UK, 1995. Springer-Verlag. [8](#)
- [24] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997. [8](#), [16](#), [19](#)
- [25] P. G. Jeavons and M. C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995. [8](#)
- [26] L. M. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence*, 64(1):147–160, 1993. [6](#)
- [27] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992. [1](#)

BIBLIOGRAPHY

- [28] B. Larose. Personal communication, 2007. [89](#)
- [29] B. Larose and P. Tesson. Universal algebra and hardness results for constraint satisfaction problems. In *ICALP*, pages 267–278, 2007. [10](#), [43](#), [88](#)
- [30] D. Lesaint, N. Azarmi, R. Laithwaite, and P. Walker. Engineering dynamic scheduler for work manager. *BT Technology Journal*, 16(3):16–29, 1998. [1](#)
- [31] L. Libkin. *Elements of finite model theory*. Springer, 2004. [41](#)
- [32] P. Lincoln and J. C. Mitchell. Algorithmic aspects of type inference with subtypes. In *POPL '92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on principles of programming languages*, pages 293–304, New York, NY, USA, 1992. ACM Press. [1](#)
- [33] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977. [1](#)
- [34] P. Meseguer. Constraint satisfaction problem: An overview. *AICOM*, 2:3–16, 1989. [1](#)
- [35] J. C. Mitchell. Coercion and type inference. In *POPL '84: Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 175–185, New York, NY, USA, 1984. ACM Press. [1](#)
- [36] U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7(2):95–132, 1974. Also Tech.Rep., Carnegie Mellon University, 1970. [1](#)
- [37] C. M. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994. [4](#), [5](#)

BIBLIOGRAPHY

- [38] R. Pöschel and L. Kalužnin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979. [12](#), [13](#)
- [39] V. Pratt and J. Tiuryn. Satisfiability of inequalities in a poset. *Fundam. Inf.*, 28(1-2):165–182, 1996. [1](#)
- [40] L. Purvis and J. P. Constraint tractability theory and its application to the product development process for a constraint-based scheduler. In *Proceedings 1st International Conference on the Practical Applications of Constraint Technologies and Logic Programming, PACLP'99*, pages 63–79, 1999. [1](#)
- [41] O. Reingold. Undirected ST-connectivity in log-space. pages 376–385, 2005. [7](#), [34](#), [40](#)
- [42] I. Rosenberg. Minimal clones I: the five types. In *Lectures in Universal Algebra (Proc. Conf. Szeged 1983)*, volume 43 of *Colloq. Math. Soc. János Bolyai*, pages 405–427. North-Holland, 1986. [20](#)
- [43] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. [8](#)
- [44] T. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing, STOC'78*, pages 216–226, 1978. [4](#), [5](#), [21](#)
- [45] A. Szendrei. *Clones in Universal Algebra*, volume 99 of *Seminaires de Mathematiques Superieures*. University of Montreal, 1986. [20](#)
- [46] M. Wand and P. O'Keefe. On the complexity of type inference with coercion. In *FPCA '89: Proceedings of the fourth international conference on functional programming languages and computer architecture*, pages 293–298, New York, NY, USA, 1989. ACM Press. [1](#)