PATH-FUNCTIONAL DEPENDENCIES AND THE TWO-VARIABLE GUARDED FRAGMENT WITH COUNTING

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2017

By
Georgios Kourtis
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

PATH-FUNCTIONAL DEPENDENCIES AND THE TWO-VARIABLE GUARDED FRAGMENT WITH COUNTING

Georgios Kourtis

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2017

We examine how logical reasoning in the two-variable guarded fragment with counting quantifiers can be integrated with databases in the presence of certain integrity constraints, called path-functional dependencies. In more detail, we establish that the problems of satisfiability and finite satisfiability for the two-variable guarded fragment with counting quantifiers, a database, and binary path-functional dependencies are EXPTIME-complete; we also establish that the data complexity of these problems is NP-complete. We establish that query answering for the above fragment (with a database and binary path-functional dependencies) is 2-EXPTIME-complete with respect to arbitrary models, and provide a 2-EXPTIME upper bound for finite models. Finally, we establish that the data complexity of query answering is coNP-complete, both with respect to arbitrary and finite models.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First of all, I would like to thank my supervisor, Ian Pratt-Hartmann, for his excellent supervision, our constructive meetings, and for always pointing me to the right direction. I also wish to thank Jonathan Shapiro, Barry Cheetham, and Joshua Knowles for giving me the opportunity to study here. I would like to thank all my friends (you know who you are) for their company and for their general support. Finally, I would like to thank my sister Despoina and the rest of my family because I know they will enjoy being thanked.

1 | Introduction

At the beginning of the twentieth century, one of the most important open problems in mathematics was the so-called *decision problem.* It was originally proposed by David Hilbert as part of his famous program for the formalization of mathematics. The decision problem asks for an *algorithm* which given an arbitrary sentence of first-order logic decides whether it is satisfiable (i.e. whether it has a model). The discovery of such an algorithm would be of great value, as it would allow—at least in theory—an automatic derivation for a large part of mathematics. Unfortunately it was shown in 1936, independently by Church and Turing, that no such algorithm exists.

This negative result did not discourage mathematicians. Various results were produced thereafter, showing that certain restricted versions of the decision problem are solvable. These versions correspond to certain classes of first-order sentences, referred to as *decidable fragments* of first-order logic, which arise by setting syntactic restrictions to first-order logic. (A class of sentences is said to be *decidable* if there is an algorithm to decide whether any given sentence in it is satisfiable; otherwise, it is said to be *undecidable.*) The main focus was initially, for historic reasons, on fragments which arise by imposing restrictions on the order of quantifiers of sentences in prenex normal form, like the $\exists^*\forall^*$ class and the $\exists^*\forall\exists^*$ class.

During the last three decades the interest shifted toward fragments with a bounded number of variables and, later, the guarded fragment. This was mainly due to important applications in areas like hardware and software verification, database theory, and descriptive complexity theory. In the case of fragments with a bounded number of variables, the line between decidability and undecidability is drawn quickly: first-order logic (with equality) with one or two variables is decidable; three or more variables make it undecidable. Two-variable first-order logic retains its decidability even with the addition of counting quantifiers, i.e. quantifiers of the form $\exists_{\leq C}$, $\exists_{=C}$, and $\exists_{\geq C}$, where C is a (natural) number. Using such quantifiers one can succinctly express requirements about the *number* of objects having a certain property, which makes them very useful in a knowledge representation setting, where one might be interested to write statements like 'every professor supervises at most ten students'.

The guarded fragment was an attempt to explain and generalize the nice computational properties of modal logic, which is the starting point for many interesting logics with important applications (e.g. description logics can be viewed as variants of modal logics). The guarded fragment dictates that quantifiers be relativised (guarded) by atoms, i.e. quantifiers can only appear in the form $\forall \bar{y}\, \alpha(\bar{x}, \bar{y}) \rightarrow \varphi(\bar{x}, \bar{y})$ or $\exists \bar{y}\, \alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y})$, where $\bar{x}, \bar{y}$ are tuples of variables and α is an atom. (One gets quantified formulas in this exact form, albeit with only two variables, when translating from modal to first-order logic.) This fragment is decidable and remains decidable after particularly powerful extensions (e.g. with fixpoint operators). Adding counting quantifiers to the guarded fragment with three or more variables makes it undecidable. However, the *two*-variable guarded fragment with counting is decidable. The latter fragment properly contains the description logic $\mathcal{ALCQI}$, which makes it important in a knowledge representation setting.

Guarded fragments have an interesting model theory. Most notably, models of guarded fragments are 'tree-like', in the sense that if a sentence has a model, then it has a model of bounded tree-width. Models of the two-variable guarded fragment with counting quantifiers are 'locally tree-shaped', in the sense that they can be taken to contain no cycles of length less than a fixed number Ω. The latter fact is crucial to our thesis: it allows us to succinctly 'describe' certain configurations in models using guarded two-variable formulas with counting quantifiers. (This is not possible when cycles of arbitrary length are allowed to exist.) Roughly speaking, our aim is to take advantage of this property to study the interaction between the two-variable guarded fragment with counting and certain concepts from database theory (in particular certain types of integrity constraints).

Logic, in general, plays a central role in database theory. Among other things, it is useful for establishing the computational complexity of queries, studying the expressive limitations of query languages, comparing the expressive capabilities of different query languages, and reasoning about data integrity. In fact, SQL, the most popular query language of the last few decades, is first-order logic in disguise. New generation information systems also allow querying data in the presence of a given *logical theory* (often referred to as a *background theory*), which codifies implicit information about the data. Description logics are perhaps the most popular logics used currently in such systems. Because the two-variable guarded fragment with counting contains $\mathcal{ALCQI}$ and this logic is very common, our work is relevant to the theory and implementation of such systems.

Data in an information system are rarely unrestricted: various forms of constraints are imposed upon them. An important type of constraints, both in theory and in practice, are *functional dependencies.* Functional dependencies dictate that certain values in a database record determine other values in that record. For example, in the database table $\mathsf{employee}(\mathsf{name}, \mathsf{surname}, \mathsf{nin}, \mathsf{tel})$ recording the name, surname, national insurance number, and telephone number of a company's employees, we may assume that no two distinct entries (employees) have the same national insurance number. That is, if a and b are entries (employees) in the table, then

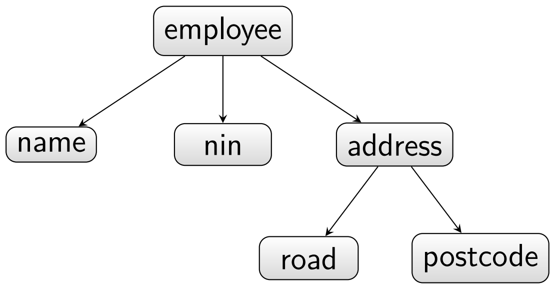
$$\mathsf{a.nin} = \mathsf{b.nin} \implies \mathsf{a} = \mathsf{b}. \tag{1.1}$$

(1.1) is an example of a *unary* functional dependency. Of course, we may wish to write a constraint like

$$\mathsf{a.nin} = \mathsf{b.nin} \;\&\; \mathsf{a.tel} = \mathsf{b.tel} \implies \mathsf{a} = \mathsf{b}, \tag{1.2}$$

which is an example of a *binary* functional dependency. In general, if the are n conjuncts in the antecedent of a functional dependency like the above, then this dependency is said to be *n-ary* or to have *arity n.*

Many applications require the representation of data into complex hierarchies of objects. For example, we may wish to represent the data for a company's employees in the following form:



In addition, we may want to enforce the condition that the national insurance number and the postcode of the address of an employee fully determine the employee, that is,

$$\mathsf{a.nin} = \mathsf{b.nin} \ \& \ \mathsf{a.address.postcode} = \mathsf{b.address.postcode} \implies \mathsf{a} = \mathsf{b}. \tag{1.3}$$

The above constraint is an example of a *path-functional dependency.* As with functional dependencies, the number of conjuncts in the antecedent of a dependency like (1.3) determines its arity. For example, (1.3) has arity 2, i.e. it is a binary path-functional dependency.

In this thesis, we study path-functional dependencies in the context of first-order logic and, in particular, the two-variable guarded fragment with counting quantifiers. In that case, because we are unable to write expressions like (1.3), we introduce abbreviations for path-functional dependencies and annotate our formulas with such abbreviations. Then, when interpreting the annotated formulas, we make sure to enforce the constraints that the dependencies dictate. For example, instead of (1.3) we write

$$\mathrm{PFD}[\mathsf{nin},\ \mathsf{address.postcode}].$$

To state that every employee has at most two supervisors under the constraint that employees are fully determined by their national insurance number and the postcode of their address, we write

$$\forall x(\mathsf{employee}(x) \to \exists_{\leq 2} y\, \mathsf{supervises}(y, x)) : \mathrm{PFD}[\mathsf{nin},\ \mathsf{address.postcode}].$$

We say that an expression like the above is (finitely) satisfiable if there exists a (finite) model satisfying the formula on the left side that does not violate the constraints imposed by the dependency on the right side. We also speak of (finite) satisfiability in the context of a database. For our purposes, a *database* is a finite set of ground literals—i.e. atomic statements or negated atomic statements featuring only constants as arguments. For example, in the context of the above examples, a subset of a company's database might be

$$\{\mathsf{empoyee}(\mathsf{Alice}), \mathsf{empoyee}(\mathsf{Bob}), \mathsf{supervises}(\mathsf{Alice}, \mathsf{Bob})\}.$$

In general, we say that an expression of the form $\Delta, \psi : \wp_1, \ldots, \wp_k$, where Δ is a database, ψ is a guarded two-variable sentence with counting quantifiers, and $\wp_1, \ldots, \wp_k$ are path-functional dependencies, is *(finitely) satisfiable* if there exists a (finite) model that interprets the constants in Δ and satisfies each literal in Δ, satisfies ψ, and does not violate the constraints dictated by $\wp_1, \ldots, \wp_k$. Satisfiability is the most important problem for our purposes, because we can reduce query answering to it.

The literature on path-functional dependencies takes two directions: The first is akin to the study of functional dependencies in classical database theory and involves, among other things, querying data (organized in a hierarchical form as in the example that we gave earlier) under the restrictions imposed by sets of path-functional dependencies, *without* considering logical (background) theories. The second revolves around integrating path-functional dependencies into knowledge representation systems based on description logics (and thus may involve background theories, something common in reasoning with description logics).

Aims of the thesis

Our aim is to study the computational complexity of reasoning for the two-variable guarded fragment with counting quantifiers in the presence of path-functional dependencies and a database. Our main focus will be on *binary* path-functional dependencies. As we will see in Chapter 6, unary path-functional dependencies are quite simple to deal with. Binary path-functional dependencies, on the other hand, present serious difficulties that require a much more involved approach. We will also see in Chapter 8 (Section 'Future work'), that the binary case provides a template to handle path-functional dependencies of arbitrary arity (again, the proof is quite involved and tedious, so we will just give a proof sketch due to lack of time). In particular:

- We establish the complexity of satisfiability and finite satisfiability for the two-variable guarded fragment with counting quantifiers in the presence of a database (without path-functional dependencies).

- We use the above result to establish the complexity of satisfiability and finite satisfiability for the two-variable guarded fragment with counting quantifiers in the presence of binary path-functional dependencies and a database.

- We establish the (combined) complexity of conjunctive query answering (with respect to both finite and arbitrary models) in the presence of binary path-functional dependencies and a logical (background) theory of guarded two-variable sentences with counting quantifiers. We also establish the data complexity of the above problem.

Structure of the thesis

The thesis is organized as follows. Chapters 2 – 4 present some background material. In Chapter 2 we mention relevant notions from logic (including first-order logic and description logics) and the theory of computation. In Chapter 3 we mention related work from computational logic, database theory, and descriptive complexity (mainly expressiveness of query languages). In Chapter 4 we formally introduce the guarded fragment and the two-variable guarded fragment with counting, we present two results relevant to the model theory of those fragments (the first, due to Grädel, mostly for completeness; the second, due to Pratt-Hartmann, because it provides the foundation for our further work), and discuss the relationship between the two-variable guarded fragment with counting quantifiers and the description logic $\mathcal{ALCQI}$.

Chapters 5 – 7 contain our contributions. In Chapter 5 we extend the proof due to Pratt-Hartmann [PH07] that the (finite) satisfiability problem for the guarded two-variable fragment with counting quantifiers is EXPTIME-complete so as to accommodate the presence of a database. We were unable to find a way to use Pratt-Hartmann's result directly, so his proof had to be adapted. The original proof is by reduction of the (finite) satisfiability problem for the above fragment to the existence of a solution for a set of linear equations/inequalities. We show how to add to the original system (which we leave untouched) a set of inequalities that take the database into consideration and then prove that those inequalities have the desired effect.

In Chapter 6 we study (finite) satisfiability for the two-variable guarded fragment with counting quantifiers in the presence of a database and (binary) path-functional dependencies. We show how to systematically eliminate the dependencies, leaving us with a 'plain' (finite) satisfiability problem for the above fragment with a database (which can be solved using the methods in Chapter 5). The approach in this chapter is based on the fact that models of the two-variable guarded fragment with counting quantifiers can be assumed to contain no small cycles.

Using this observation, one can identify putative violations of each given dependency with certain kinds of acyclic graph walks. These walks can be decomposed into components described by guarded two-variable formulas with counting quantifiers, and can either be forbidden or checked within the database.

In Chapter 7 we first establish a 2-EXPTIME upper bound for the (combined) complexity of query answering (with respect to both infinite and finite models) for the two-variable guarded fragment with counting quantifiers in the presence of path-functional dependencies (and a database). We do that by working through Pratt-Hartmann's proof in [PH09], which establishes an upper (and lower) bound for the *data* complexity of query answering for the above fragment (without path-functional dependencies). The main approach is exactly the same—it reduces query answering to satisfiability—so we just work out some subtleties having to do with path-functional dependencies, and carefully examine the new bounds (in the original result the query was fixed, but we allow it to vary). We obtain a lower bound for query answering with respect to arbitrary models from a result due to Lutz [LST05], concerning query answering for the description logic $\mathcal{ALCQI}$. We then establish that the data complexity of (finite) satisfiability for the guarded fragment with counting quantifiers in the presence of path-functional dependencies and a database is NP-complete. Finally, we use the latter result to obtain that the data complexity for our first problem in the chapter is coNP-complete, again based on the work from [PH09].

2 | Preliminaries

In this chapter we introduce some important notions from mathematical logic and the theory of computation.

2.1 First-order logic

A signature provides names for constants, relations, and functions, in a domain of interest.

Definition 2.1. A *signature* or *vocabulary* is a triple $\sigma = \langle \mathsf{Const}; \mathsf{Rel}; \mathsf{Func} \rangle$, where Const is a collection of *constant symbols* (denoted $c_1, \ldots, c_\ell$), Rel is a collection of *relation symbols* (denoted $p_1, \ldots, p_m$), and Func is a collection of *function symbols* (denoted $f_1, \ldots, f_n$). With each signature is also associated a function

$$\alpha : \mathsf{Rel} \cup \mathsf{Func} \to \{1, 2, \ldots\},$$

assigning to each relation and function symbol an *arity*.

A signature is said to be *relational* if it contains no function symbols. All the signatures that we deal with in this thesis are relational. A signature containing neither any constants nor function symbols is said to be *purely relational*.

Structures are used for interpreting the language of first-order logic.

Definition 2.2. Let σ be a signature. A *σ-structure* $\mathfrak{S} = \langle S, \{c_i^{\mathfrak{S}}\}, \{p_j^{\mathfrak{S}}\}, \{f_k^{\mathfrak{S}}\} \rangle$ consists of a universe S, and an interpretation of

- each constant symbol c_i from σ as an element $c_i^{\mathfrak{S}} \in S$;
- each relation symbol p_j from σ as a relation $p_j^{\mathfrak{S}} \subseteq S^{\alpha(p_j)}$;
- each function symbol f_k from σ as a function $f_k^{\mathfrak{S}} : S^{\alpha(f_k)} \to S$.

For example, a directed graph $G = (V, E)$ is a structure over the purely relational signature $\sigma = \langle E \rangle$. By convention we use fraktur letters $(\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \mathfrak{D}, \ldots)$ for structures and roman letters $(A, B, C, D, \ldots)$ for universes correspondingly. A structure is said to be *finite* if its universe is finite. If $\mathfrak{A}$ is a σ-structure and $A_0 \subseteq A$, we define $\mathfrak{A}|_{A_0}$ to be the σ-structure $\mathfrak{B}$ whose universe is the set $B = A_0 \cup \{c^{\mathfrak{A}} \mid c \text{ is a constant symbol in } \sigma\}$, with $c^{\mathfrak{B}} = c^{\mathfrak{A}}$ for every constant symbol c in σ, and $p^{\mathfrak{B}}, f^{\mathfrak{B}}$ being the restriction to B of $p^{\mathfrak{A}}, f^{\mathfrak{A}}$ respectively, for each relation symbol p and function symbol f in σ.

We now define the language of first-order logic (with equality), which is actually a family of languages, one for each signature σ.

Definition 2.3. The *alphabet* of *first-order logic* comprises, for each signature σ, the symbols in σ, and, in addition:

(i) the *propositional symbols* $\neg$ (not), $\wedge$ (and), $\vee$ (or);
(ii) the *quantifiers* $\forall$ (for all) and $\exists$ (there exists);
(iii) the *punctuation symbols* '(' and ')';
(iv) a set of countably many *variable symbols* $\mathsf{Vars} = \{x_0, x_1, x_2, \ldots\}$.

Definition 2.4. The set of *terms* of *first-order logic*, over a signature σ, are the expressions t generated by the grammar

$$t := x \mid c \mid f(t_1, \ldots, t_n),$$

where x is a variable, c is a constant symbol in σ, f is a function symbol in σ of arity n, and $t_1, \ldots, t_n$ are terms.

Definition 2.5. The set of *formulas* of *first-order logic*, over a signature σ, are the expressions χ generated by the grammar

$$\chi := p(t_1, \ldots, t_n) \mid s = t \mid \neg(\varphi) \mid (\varphi) \wedge (\psi) \mid (\varphi) \vee (\psi) \mid \forall x(\varphi) \mid \exists x(\varphi),$$

where p is a relation symbol in σ of arity n, $t_1, \ldots, t_n$ and s, t are terms, φ, ψ are formulas, and x is a variable.

We usually omit the parentheses when no confusion arises. We also use various letters (possibly with subscripts and usually from the end of the roman alphabet) other than x to denote variables. We may also use symbols other than c, p, and f for constant, relation, and function symbols respectively; the type of such symbols will always be clear from context. Finally, we use the following abbreviations for convenience: $\varphi \to \psi$ for $\neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi$ for $(\varphi \to \psi) \wedge (\psi \to \varphi)$, where φ and ψ are formulas.

A formula is said to be *quantifier-free* if it contains no quantifiers. A formula is said to be in *prenex normal form* if it is of the form $\mathsf{Q}x_1 \dots \mathsf{Q}x_n\psi$, where Q is $\forall$ or $\exists$, and ψ is a quantifier-free formula. An *atom* or *atomic formula* is a formula of the form $p(t_1, \dots, t_n)$, where p is a relation symbol and $t_1, \dots, t_n$ are terms; an atom is *ground* if it contains no variables, and function-free if it contains no function symbols.

Definition 2.6. The *free variables* of a term or formula are defined recursively as follows:

- Constant terms have no free variables.
- The (only) free variable of a term x is x.
- The free variables of $t_1 = t_2$ are the free variables of t_1 and t_2.
- The free variables of $p(t_1, \dots, t_n)$ are the free variables of $t_1, \dots, t_n$.
- The free variables of $f(t_1, \dots, t_n)$ are the free variables of $t_1, \dots, t_n$.
- The free variables of $\neg\varphi$ are the free variables of φ.
- The free variables of $\varphi \wedge \psi$ or $\varphi \vee \psi$ are the free variables of φ and ψ.
- The free variables of $\forall x\varphi$ or $\exists x\varphi$ are the free variables of φ, except x.

A *sentence* is a formula having no free variables. Variables that are not free are called *bound*. If $\bar{x}$ is the tuple of all free variables appearing in a formula φ, we write $\varphi(\bar{x})$.

We are now ready to define the semantics of first-order logic. Let σ be a signature and $\mathfrak{A}$ be a σ-structure. A *valuation* for $\mathfrak{A}$ is a function $\pi : \mathsf{Vars} \to A$; if π is a valuation, $a \in A$, and $x \in \mathsf{Vars}$, we define $\pi^a_x : \mathsf{Vars} \to A$ as

$$\pi^a_x(y) = \begin{cases} \pi(y) & \text{if } y \neq x, \\ a & \text{otherwise.} \end{cases}$$

The *value* of a term t (over σ) in $\mathfrak{A}$ under the valuation π, denoted $[\![t]\!]^{\mathfrak{A}}_{\pi}$, is defined recursively as follows:

- $[\![c]\!]^{\mathfrak{A}}_{\pi} = c^{\mathfrak{A}}$, for each constant symbol c in σ.
- $[\![x]\!]^{\mathfrak{A}}_{\pi} = \pi(x)$, where $x \in \mathsf{Vars}$.
- $[\![f(t_1, \dots, t_n)]\!]^{\mathfrak{A}}_{\pi} = f^{\mathfrak{A}}([\![t_1]\!]^{\mathfrak{A}}_{\pi}, \dots, [\![t_n]\!]^{\mathfrak{A}}_{\pi})$, for each function symbol f in σ.

The satisfiability relation for first-order logic is defined as follows:

Definition 2.7. Let σ be a signature, $\mathfrak{A}$ be a σ-structure, and π be a valuation for $\mathfrak{A}$; let φ be a formula over σ. We define $(\mathfrak{A}, \pi) \models \varphi$ recursively as follows:

- if φ is $p(t_1, \ldots, t_n)$, then $(\mathfrak{A}, \pi) \models \varphi$ iff $\langle [\![t_1]\!]^{\mathfrak{A}}_{\pi}, \ldots, [\![t_n]\!]^{\mathfrak{A}}_{\pi} \rangle \in p^{\mathfrak{A}}$;
- if φ is $s = t$, then $(\mathfrak{A}, \pi) \models \varphi$ iff $[\![s]\!]^{\mathfrak{A}}_{\pi} = [\![t]\!]^{\mathfrak{A}}_{\pi}$;
- if φ is $\neg\psi$, then $(\mathfrak{A}, \pi) \models \varphi$ iff $(\mathfrak{A}, \pi) \models \psi$ does not hold;
- if φ is $\psi_1 \wedge \psi_2$, then $(\mathfrak{A}, \pi) \models \varphi$ iff $(\mathfrak{A}, \pi) \models \psi_1$ and $(\mathfrak{A}, \pi) \models \psi_2$;
- if φ is $\psi_1 \vee \psi_2$, then $(\mathfrak{A}, \pi) \models \varphi$ iff $(\mathfrak{A}, \pi) \models \psi_1$ or $(\mathfrak{A}, \pi) \models \psi_2$;
- if φ is $\forall x \psi$, then $(\mathfrak{A}, \pi) \models \varphi$ iff $(\mathfrak{A}, \pi^a_x) \models \psi$, for all $a \in A$;
- if φ is $\exists x \psi$, then $(\mathfrak{A}, \pi) \models \varphi$ iff $(\mathfrak{A}, \pi^a_x) \models \psi$, for some $a \in A$.

If $(\mathfrak{A}, \pi) \models \varphi$ we say that $\mathfrak{A}$ *satisfies* φ under the valuation π.

Notice that when a structure $\mathfrak{A}$ satisfies a formula φ under a valuation π, the satisfaction of φ depends only on the values that π assigns to φ's free variables. In other words, if $(\mathfrak{A}, \pi) \models \varphi$, then, for any valuation π' that agrees with π on the values of φ's free variables, $(\mathfrak{A}, \pi') \models \varphi$. For this reason, we often write $\mathfrak{A} \models \varphi(\bar{a})$, where $\bar{a}$ is a tuple of elements in $\mathfrak{A}$, same in number as φ's free variables, say $\bar{x}$, to denote that $(\mathfrak{A}, \pi) \models \varphi$ for any valuation π that assigns the elements in $\bar{a}$ to the corresponding free variables in $\bar{x}$. It also follows from the above that the satisfaction of any sentence is independent of any valuation; thus, if φ is a sentence, we write $\mathfrak{A} \models \varphi$ to denote that $(\mathfrak{A}, \pi) \models \varphi$, for any valuation π.

If φ is a sentence, a structure $\mathfrak{A}$ such that $\mathfrak{A} \models \varphi$ is said to be a *model* of φ; φ is said to be *satisfiable* if it has a model and *finitely satisfiable* if it has a finite model. If Γ is a set of sentences, we say that $\mathfrak{A}$ is a model of Γ if $\mathfrak{A} \models \gamma$ for each $\gamma \in \Gamma$. If Γ is a set of sentences and φ is a sentence, we say that Γ *entails* φ, denoted $\Gamma \models \varphi$, if every model of Γ is also a model of φ; we say that Γ *finitely entails* φ, denoted $\Gamma \models_{\text{fin}} \varphi$, if every finite model of Γ is also a model of φ.

A *fragment of first-order logic*[1] is a formal language whose syntax is a restriction of the syntax (i.e. the definition of terms and formulas) of first-order logic. (Trivially, first-order logic itself is a fragment of first-order logic.) For instance, the two-variable fragment of first-order logic allows the use of only two variables,

[1]The word 'fragment' on its own also appears in the literature, referring to a formal language updating the syntax *and* semantics of some logic. As a matter of fact, the two-variable fragment with counting *and* with path-functional dependencies introduced later is not a fragment of first-order logic, as it extends its syntax with annotations for path-functional dependencies *and* provides semantics for these annotations, in addition to the semantics of first-order logic.

say x and y, and the fragment $\exists^*\forall^*$ (known as the Bernays-Schönfinkel class) allows only formulas of the form $\exists\bar{x}\forall\bar{y}\,\varphi(\bar{x},\bar{y})$, where φ is quantifier-free. (We remark that the semantics for fragments of first-order logic remains the same.) A fragment has the *finite model property* if any satisfiable sentence in it is also finitely satisfiable.

For any given fragment $\mathcal{F}$ of first-order logic, we are interested in the following problems:

- Sat($\mathcal{F}$): given a sentence φ in $\mathcal{F}$, is it satisfiable?
- FinSat($\mathcal{F}$): given a sentence φ in $\mathcal{F}$, is it finitely satisfiable?

We refer to Sat($\mathcal{F}$) as the *satisfiability problem* and to FinSat($\mathcal{F}$) as the *finite satisfiability problem* for $\mathcal{F}$. In the context of databases (where a *database* is simply a set of ground, function-free atoms or negations of atoms—see Section 5.1), we are also interested in the following satisfiability problems:

- DSat($\mathcal{F}$): given a database Δ and a sentence φ in $\mathcal{F}$, is the sentence $\bigwedge(\Delta \cup \{\varphi\})$ satisfiable?
- DFinSat($\mathcal{F}$): given a database Δ and a sentence φ in $\mathcal{F}$, is the sentence $\bigwedge(\Delta \cup \{\varphi\})$ finitely satisfiable?

We refer to DSat($\mathcal{F}$) as the *data satisfiability problem* and to DFinSat($\mathcal{F}$) as the *data finite satisfiability problem* for $\mathcal{F}$.

With respect to databases, we are also interested in the following problems:

- DQAns($\mathcal{F}$): given a database Δ, a sentence φ in $\mathcal{F}$, a formula $\psi(\bar{x})$, and a tuple of constants $\bar{a}$ from Δ, same in length with $\bar{x}$, does $\Delta, \varphi \models \psi(\bar{a})$?
- DFinQAns($\mathcal{F}$): given a database Δ, a sentence φ in $\mathcal{F}$, a formula $\psi(\bar{x})$, and a tuple of constants $\bar{a}$ from Δ, same in length with $\bar{x}$, does $\Delta, \varphi \models_{\text{fin}} \psi(\bar{a})$?

We refer to DQAns($\mathcal{F}$) as the *query answering problem* and to DFinQAns($\mathcal{F}$) as the *finite query answering problem* for $\mathcal{F}$. The sentence φ is called a *background theory* and the formula ψ is called a *query*. Perhaps the most important type of queries are conjunctive queries, since they arise very often in practice; a *conjunctive query* is a formula of the form $\exists\bar{y}\,\eta(\bar{x},\bar{y})$, where $\eta(\bar{x},\bar{y})$ is a conjunction of function-free atoms. A conjunctive query is *boolean* if it contains no free variables. In this thesis, we are only interested in conjunctive queries so, without mention, we will take ψ in the problems DQAns($\mathcal{F}$) and DFinQAns($\mathcal{F}$) above to be a conjunctive query. The problem of (finite) query answering for boolean conjunctive queries is referred to as *(finite) query entailment*. In the sequel we consider, for simplicity, only boolean conjunctive queries.

We remark on the importance of background theories in modern query answering. A background theory adds semantics to the data in a given database, and thus allows inferences based on implicit facts about the data. For example, imagine that a university provides either a laptop or a desktop to each of its students, and that it wants to know how many computers it has distributed in total. Then, a background theory may say that 'a computer is either a laptop or a desktop' and the query to compute the required results would be 'return the number of computers belonging to students in the database'. This example may seem contrived, but in practical situations (e.g. in knowledge representation with ontologies) background theories may comprise hundreds of thousands of sentences, and their implications can be entirely non-trivial.

One often wishes to consider versions of DSat($\mathcal{F}$) and DFinSat($\mathcal{F}$) where the background theory is fixed, and versions of DQAns($\mathcal{F}$) and DFinQAns($\mathcal{F}$) where the background theory and the query is fixed. For, in practice, the background theory and the query are significantly smaller than the corresponding database, and it is only the varying size of the database that affects the performance of query execution. We use the same notation for these new versions of the above problems, but with the fixed quantities as subscripts: $\mathrm{DSat}_{\varphi}(\mathcal{F})$, $\mathrm{DFinSat}_{\varphi}(\mathcal{F})$, $\mathrm{DQAns}_{\varphi,\psi}(\mathcal{F})$ and $\mathrm{DFinQAns}_{\varphi,\psi}(\mathcal{F})$.

We also use the same notation in the context of the fragment $\mathcal{GC}^2\mathcal{DP}^2$, i.e. the two-variable guarded-fragment with counting and binary path-functional dependencies, which, strictly speaking, is not a fragment of first-order logic, but it does provide an analogous notion of satisfiability. Thus, Sat($\mathcal{GC}^2\mathcal{DP}^2$) is the satisfiability problem for $\mathcal{GC}^2\mathcal{DP}^2$, FinSat($\mathcal{GC}^2\mathcal{DP}^2$) is its finite satisfiability problem, and DSat($\mathcal{GC}^2\mathcal{DP}^2$), DFinSat($\mathcal{GC}^2\mathcal{DP}^2$) are the 'data' analogues of the latter two problems. In addition, DQAns($\mathcal{GC}^2\mathcal{DP}^2$) and DFinQAns($\mathcal{GC}^2\mathcal{DP}^2$) are the query and finite query answering problem for $\mathcal{GC}^2\mathcal{DP}^2$ respectively. Later, when we define the notion of a path-functional dependency and wish to discuss the last four problems with a fixed background theory, query, *and* a fixed path-functional dependency $\wp$, we also add it as a subscript, that is, we write $\mathrm{DSat}_{\varphi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$, $\mathrm{DFinSat}_{\varphi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$, $\mathrm{DQAns}_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ and $\mathrm{DFinQAns}_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ correspondingly.

The computational complexity of the problem DQAns($\mathcal{F}$), for a fragment $\mathcal{F}$, is referred to as its *combined complexity*; the computational complexity of the problem DQAns$_{\varphi,\psi}$($\mathcal{F}$), for a fragment $\mathcal{F}$, is referred to as its *data complexity.* We also use these descriptions for the corresponding problems of $\mathcal{GC}^2\mathcal{DP}^2$, and also for their finite counterparts (depending on context).

2.2 Description logics

Description logics are a very popular family of logics used in industrial strength knowledge representation systems [Baa03]. Notably, they are important in relation to the Semantic Web, as they underly the Web Ontology Language (OWL) [G$^+$09]. From a logical perspective, description logics are fragments of first-order logic; they can also be viewed as variants of modal logics [Sch91] and are, thus, relevant to the guarded fragment—and the other way around [Gra98]. We are mainly interested in the description logic $\mathcal{ALCQI}$, used in Section 7.1.

The basic blocks for building expressions for a description logic are *atomic concepts* and *atomic roles.* Atomic concepts are analogous to unary predicates in first-order logic (e.g. Human, $\mathsf{Elephant}$, Fox) and atomic roles are analogous to binary predicates (e.g. $\mathsf{childOf}$, $\mathsf{parentOf}$). One can construct more general concepts using atomic concepts and roles, and various boolean constructors such as *conjunction* ($\sqcap$), *disjunction* ($\sqcup$), and *negation* ($\neg$), as well as two *restriction constructors*—the *existential* ($\exists\mathsf{r}.\mathsf{C}$) and the *universal* ($\forall\mathsf{r}.\mathsf{C}$), where r is an atomic role and C is a (not necessarily atomic) concept.

We now proceed to define the syntax and semantics for the logic $\mathcal{ALC}$, which is the most basic description logic.

Definition 2.8 ($\mathcal{ALC}$ syntax)**.** Let C be a set of *concept names* and R be a set of *role names.* Then, $\mathcal{ALC}$*-concepts* are defined recursively:

(i) $\top$, $\bot$, and every concept name $\mathsf{C} \in C$ is an $\mathcal{ALC}$-concept;

(ii) if C, D are $\mathcal{ALC}$-concepts and $\mathsf{r} \in R$, then $\mathsf{C} \sqcup \mathsf{D}$, $\mathsf{C} \sqcap \mathsf{D}$, $\neg\mathsf{C}$, $\exists\mathsf{r}.\mathsf{C}$, and $\forall\mathsf{r}.\mathsf{C}$ are $\mathcal{ALC}$-concepts.

Definition 2.9 ($\mathcal{ALC}$ semantics)**.** An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ mapping each $\mathcal{ALC}$-concept to a subset of $\Delta^{\mathcal{I}}$, and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, such that, for all $\mathcal{ALC}$-concepts C, D and all role names r,

$$\begin{gathered}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}}, \quad \bot^{\mathcal{I}} = \emptyset, \\
(\mathsf{C} \sqcup \mathsf{D})^{\mathcal{I}} = \mathsf{C}^{\mathcal{I}} \cup \mathsf{D}^{\mathcal{I}}, \quad (\mathsf{C} \sqcap \mathsf{D})^{\mathcal{I}} = \mathsf{C}^{\mathcal{I}} \cap \mathsf{D}^{\mathcal{I}}, \quad \neg \mathsf{C}^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus \mathsf{C}^{\mathcal{I}}, \\
(\exists \mathsf{r}.\mathsf{C})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{for some } y \in \Delta^{\mathcal{I}}, (x,y) \in \mathsf{r}^{\mathcal{I}} \text{ and } y \in \mathsf{C}^{\mathcal{I}}\}, \\
(\forall \mathsf{r}.\mathsf{C})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{for all } y \in \Delta^{\mathcal{I}} \text{ s.t. } (x,y) \in \mathsf{r}^{\mathcal{I}}, y \in \mathsf{C}^{\mathcal{I}}\}.
\end{gathered}$$

In general, a knowledge base is made up of two parts: a terminological and an assertional one. The assertional part, called the ABox, is a set of assertions like $\mathsf{PhD} : \mathsf{Degree}$ and $(\mathsf{Alice}, \mathsf{Bob}) : \mathsf{sisterOf}$; it is analogous to the body of data in a database. The terminological part, called the TBox, is a set of concept hierarchies like $\mathsf{Panda} \sqsubseteq \mathsf{Mammal}$; it is analogous to a database schema.

Definition 2.10. A *concept inclusion* is an expression of the form $\mathsf{C} \sqsubseteq \mathsf{D}$, where C, D are $\mathcal{ALC}$-concepts. An interpretation $\mathcal{I}$ is a *model* of a concept inclusion $\mathsf{C} \sqsubseteq \mathsf{D}$ if $\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$. A finite set of concept inclusions is called a *TBox*. An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ if it is a model of every concept inclusion in $\mathcal{T}$. When $\mathsf{C} \sqsubseteq \mathsf{D}$ and $\mathsf{D} \sqsubseteq \mathsf{C}$, for two $\mathcal{ALC}$-concepts C, D, we write $\mathsf{C} \equiv \mathsf{D}$.

Definition 2.11. An *assertional axiom* is an expression of the form $\mathsf{x} : \mathsf{C}$ or $(\mathsf{x}, \mathsf{y}) : \mathsf{r}$, where C is a concept and r is a role. An interpretation $\mathcal{I}$ is a *model* of an assertional axiom $\mathsf{x} : \mathsf{C}$ if $\mathsf{x}^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$, and of $(\mathsf{x}, \mathsf{y}) : \mathsf{r}$ if $(\mathsf{x}, \mathsf{y})^{\mathcal{I}} \in \mathsf{r}^{\mathcal{I}}$. A finite set of assertional axioms is called an *ABox*. An interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ if it is a model of every assertional axiom in $\mathcal{A}$.

Definition 2.12. A *knowledge base* is a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is a *model* of a knowledge base $(\mathcal{T}, \mathcal{A})$, if it is a model of $\mathcal{T}$ and a model of $\mathcal{A}$.

If an interpretation $\mathcal{I}$ is a model of Υ (where Υ is a concept inclusion, assertional axiom, TBox, ABox, or a knowledge base), we write $\mathcal{I} \models \Upsilon$.

There are various ways to extend the (limited for many applications) expressive capabilities of $\mathcal{ALC}$. Two of those ways that are of interest to us lead to the description logic $\mathcal{ALCQI}$. Formally, $\mathcal{ALCQI}$ extends $\mathcal{ALC}$ by allowing the following features:

Qualified number restrictions Two restriction operators $\leqslant n\, \mathsf{r}.\mathsf{C}$ and $\geqslant n\, \mathsf{r}.\mathsf{C}$, where n is a (natural) number (encoded in binary), r is a role, and C is an

$\mathcal{ALCQI}$-concept, interpreted as follows:

$$(\leqslant n\,\mathsf{r}.\mathsf{C})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} : |\{y \in \Delta^{\mathcal{I}} : (x,y) \in \mathsf{r}^{\mathcal{I}} \text{ and } y \in \mathsf{C}^{\mathcal{I}}\}| \leq n\},$$
$$(\geqslant n\,\mathsf{r}.\mathsf{C})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} : |\{y \in \Delta^{\mathcal{I}} : (x,y) \in \mathsf{r}^{\mathcal{I}} \text{ and } y \in \mathsf{C}^{\mathcal{I}}\}| \geq n\}.$$

Inverse roles If r is a role, then r^- (the *inverse* of r) is also a role, interpreted as follows:

$$(\mathsf{r}^-)^{\mathcal{I}} = \{(x,y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : (y,x) \in \mathsf{r}^{\mathcal{I}}\}.$$

The definitions of a TBox, ABox, and knowledge base for $\mathcal{ALCQI}$ remain unchanged, except for using $\mathcal{ALCQI}$-concepts instead of $\mathcal{ALC}$-concepts and allowing inverse roles.

As an example of what can be expressed in $\mathcal{ALCQI}$, consider the TBox that consists of the following concept inclusions:

$$\mathsf{Student} \sqsubseteq \mathsf{Undergraduate} \sqcup \mathsf{Graduate}$$
$$\mathsf{Student} \sqsubseteq \geqslant 1\,\mathsf{attends}.\mathsf{Course} \sqcap \leqslant 6\,\mathsf{attends}.\mathsf{Course}$$
$$\mathsf{Student} \sqsubseteq \exists \mathsf{supervises}^-.\mathsf{Professor}$$

It states that a student is either undergraduate or postgraduate, attends from one to six courses, and is supervised by a professor.

The main reasoning tasks associated with a description logic are:

Satisfiability A concept C is *satisfiable* with respect to a TBox $\mathcal{T}$ if there exists an model $\mathcal{I}$ of $\mathcal{T}$ such that $\mathsf{C}^{\mathcal{I}}$ is non-empty.

Subsumption A concept C is *subsumed* by a concept D with respect to a TBox $\mathcal{T}$, denoted $\mathsf{C} \sqsubseteq_{\mathcal{T}} \mathsf{D}$, if $\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{T}$.

Equivalence Two concepts C and D are *equivalent* with respect to a TBox $\mathcal{T}$, denoted $\mathsf{C} \equiv_{\mathcal{T}} \mathsf{D}$, if $\mathsf{C}^{\mathcal{I}} = \mathsf{D}^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{T}$.

Disjointness Two concepts C and D are *disjoint* with respect to a TBox $\mathcal{T}$ if $\mathsf{C}^{\mathcal{I}} \cap \mathsf{D}^{\mathcal{I}} = \emptyset$, for every model $\mathcal{I}$ of $\mathcal{T}$.

The following problem is more relevant for our purposes [GHLS08]:

Query entailment Let V be a countably infinite set of variable names. An *atom* is an expression $\mathsf{C}(v)$ or $\mathsf{r}(v, v')$, where C is a concept name, r is a role name, and $v, v' \in V$. A *conjunctive query* is a non-empty, finite set of atoms, meant to be viewed as the conjunction of its elements. Let $\mathsf{Var}(q)$ denote the set of variables occurring in the query q. Let $\mathcal{I}$ be an interpretation, q a conjunctive query, and $\pi : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ a total function. We write

- $\mathcal{I} \models^{\pi} \mathsf{C}(v)$ if $\pi(v) \in \mathsf{C}^{\mathcal{I}}$;
- $\mathcal{I} \models^{\pi} \mathsf{r}(v, v')$ if $(\pi(v), \pi(v')) \in \mathsf{r}^{\mathcal{I}}$.

If $\mathcal{I} \models^{\pi} \alpha$, for all $\alpha \in q$, we write $\mathcal{I} \models^{\pi} q$ and call π a *match* for $\mathcal{I}$ and q. We say that $\mathcal{I}$ *satisfies* q and write $\mathcal{I} \models q$ if there is a match π for $\mathcal{I}$ and q. If $\mathcal{I} \models q$ for all models $\mathcal{I}$ of a knowledge base $\mathcal{K}$, we write $\mathcal{K} \models q$ and say that $\mathcal{K}$ *entails* q. A related problem to query entailment is *query answering*, where one asks for the tuples (assignments) that satisfy a query q. It is known that these problems are mutually reducible [CDGL98, HT00].

2.3 Computability and complexity

In this section, we overview some standard concepts; for more details see [Pap03, AB09]. The model of computation we use is the Turing machine:

Definition 2.13. A *(deterministic) Turing machine* is a tuple $\langle Q, \Sigma, \text{␣}, \delta, q_0, F\rangle$, say M, where

- Q is a finite, non-empty set of *states*;
- Σ is a finite, non-empty set of symbols, called the *alphabet* of M;
- ␣ is the *blank symbol*, allowed to occur infinitely often on the tape;[2]
- $\delta : (Q \setminus F) \times \Sigma \to Q \times \Sigma \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}$ is the *transition function* of M;
- $q_0 \in Q$ is the *initial state*;
- $F = \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\} \subseteq Q$ is the set of *final states*: the *accepting state* (q_{acc}) and the *rejecting state* (q_{rej});

A Turing machine requires for its operation: (i) a *tape* divided into cells; (ii) a *head* that can read or write symbols on the tape (one symbol at each cell) and move left or right (one cell at a time); and (iii) a *state register* that stores the state of the machine (among the states in Q).

Before its operation, a Turing machine M is initialized with an input $x \in \Sigma^*$ on the tape (with blanks everywhere else), its head on the first symbol of x,

[2]We assume that ␣ does not occur in Σ.

and its state register set to q_0. From that point onward, the operation proceeds according to the transition function δ. Suppose that the machine is at state q and the head is at a cell containing the symbol σ. If $q \in F$, then the machine *halts*. Otherwise, it performs a *step*: let $\delta(q, \sigma) = (q', \sigma', d)$; M updates its state register to q' from q, the cell where the head is located to σ' from σ, and moves the head according to d: left if $d = \mathsf{L}$, right if $d = \mathsf{R}$, and does not move at all if $d = \mathsf{S}$. Note that it is not necessary for M to halt. If, for a given input x, M halts and its state is q_{acc}, we say that x was *accepted*; if it halts and its state is q_{rej}, we say that x was *rejected*. (The terms *'succeed'* and *'fail'* also appear in the literature.) By convention, in the former case we take M's *output* $M(x)$ to be 1, and in the latter we take $M(x)$ to be 0.

Without loss of generality, we take $\Sigma = \{0, 1\}$. A *language* is a subset of $\{0, 1\}^*$. We say that a Turing machine M *decides* a language L when, for all $x \in \{0, 1\}^*$, $M(x) = 1 \Leftrightarrow x \in L$ and $M(x) = 0 \Leftrightarrow x \notin L$. A language is called *decidable* if there is a Turing machine that decides it. The canonical undecidable language is the one corresponding to Turing's *halting problem* [Tur36]. Others include the languages corresponding to tiling problems [Wan60, Wan65], and, of course, Hilbert's decision problem [Chu36a, Chu36b, Tur36]. The notion of decidability easily generalizes to functions: a function $f : \{0, 1\}^* \to \{0, 1\}^*$ is said to be *computable* if the relation $L = \{\langle x, y\rangle \mid f(x) = y\}$ is decidable.

To measure the complexity of operation for a given Turing machine M one proposes so-called *complexity measures*, two of the most common being *running time* (i.e. the number of steps M performs until it halts) and *space* (i.e. the maximum number of cells that are updated, possibly more than once, with a symbol from M's alphabet). The running time and space of a Turing machine are measured as a function (from $\mathbb{N}$ to $\mathbb{N}$) of its input size.[3] Thus arises a classification of decidable languages into classes, called *complexity classes*, according to the running time or space of the Turing machine that decides them:

- For a given function $f : \mathbb{N} \to \mathbb{N}$, $\mathsf{TIME}(f(n))$ is the class of languages decidable by a deterministic Turing machine whose running time is at most $f(n)$, where n is the size of the input.

[3] We mention that not all functions from $\mathbb{N}$ to $\mathbb{N}$ are suitable for measuring complexity: one usually focuses on *time* or *space constructible* functions. A function $f : \mathbb{N} \to \mathbb{N}$ is time (resp. space) constructible if it is non-decreasing and computable by a Turing machine whose running time (resp. space used) on an input n is at most $c \cdot f(n)$, for some constant $c \in \mathbb{N}$. The functions used in complexity theory are almost always time or space constructible, so one usually suppresses the requirement for time or space constructibility.

- For a given function $f : \mathbb{N} \to \mathbb{N}$, $\mathsf{SPACE}(f(n))$ is the class of languages decidable by a deterministic Turing machine using space at most $f(n)$, where n is the size of the input.

So-called *linear speedup* theorems are known for the classes $\mathsf{TIME}(f(n))$ and $\mathsf{SPACE}(f(n))$, for all 'reasonable' functions $f(n)$: roughly speaking, the above classes remain the same, up to a constant multiple of the function $f(n)$.

The following abbreviations are common:

- $\mathsf{P} = \mathsf{PTIME} = \bigcup_{k\geq 1} \mathsf{TIME}(n^k)$: deterministic polynomial time.
- $\mathsf{EXPTIME} = \bigcup_{k\geq 1} \mathsf{TIME}(2^{n^k})$: deterministic exponential time.
- $2\text{-}\mathsf{EXPTIME} = \bigcup_{k\geq 1} \mathsf{TIME}(2^{2^{n^k}})$: deterministic double exponential time.
- $\mathsf{L} = \mathsf{LOGSPACE} = \mathsf{SPACE}(\log(n))$: deterministic logarithmic space.[4]
- $\mathsf{PSPACE} = \bigcup_{k\geq 1} \mathsf{SPACE}(n^k)$: deterministic polynomial space.

Another important notion is that of a non-deterministic Turing machine. This model of computation does not contribute any additional benefit to deterministic Turing machines in terms of the languages they can decide—in fact, any non-deterministic Turing machine can be simulated, at a cost, by a deterministic one. Nevertheless, it enables a significant simplification in defining various problems that arise very often in practice. A *non-deterministic Turing machine* is defined exactly like a deterministic one, but with the following difference:

$$\delta \subseteq ((Q \setminus F) \times \Sigma) \times (Q \times \Sigma \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\})$$

is no longer a function but a relation, the *transition relation.*

After the initialization of a non-deterministic Turing machine N with an input x and its state register set to q_0, its operation is dictated by its transition relation. Instead of taking one step at a time like its deterministic analogue, N takes at each point *all* the steps dictated by δ *simultaneously.* N halts simply if any sequence of steps leads to a final state. However, if it halts, the condition of acceptance and rejection is much different: N accepts an input x if *any* sequence of steps during N's computation leads to the accepting state q_{acc}; it rejects if *all* the sequences of steps lead to the rejecting state q_{rej}. The *running time* of N is defined to be the largest sequence of steps leading to a final state, and the *space*

[4]The space of the input is not counted.

used by N is defined to be the largest amount of space used in any sequence of steps leading to a final state.

The classes $\mathsf{NTIME}(f(n))$ and $\mathsf{NSPACE}(f(n))$ are the non-deterministic counterparts of the classes $\mathsf{TIME}(f(n))$ and $\mathsf{SPACE}(f(n))$ respectively. Linear speedup theorems also exist for such non-deterministic classes, similarly to the deterministic ones. Some common abbreviations are:

- $\mathsf{NP} = \bigcup_{k\geq 1} \mathsf{NTIME}(n^k)$: non-deterministic polynomial time.
- $\mathsf{NEXPTIME} = \bigcup_{k\geq 1} \mathsf{NTIME}(2^{n^k})$: non-deterministic exponential time.
- $\mathsf{NL} = \mathsf{NLOGSPACE} = \mathsf{NSPACE}(\log(n))$: deterministic logarithmic space.[5]
- $\mathsf{NPSPACE} = \bigcup_{k\geq 1} \mathsf{NSPACE}(n^k)$: non-deterministic polynomial time.

The following relations among deterministic and non-deterministic complexity classes are all well-known:

- $\mathsf{TIME}(f(n)) \subseteq \mathsf{NTIME}(f(n))$;
- $\mathsf{SPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$;
- $\mathsf{NTIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$;
- $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(c^{\log n + f(n)})$, for some constant c.

Savitch's theorem [Sav70] tells us that non-determinism does not add any significant advantage with respect to space; for example, it follows from it that $\mathsf{PSPACE} = \mathsf{NPSPACE}$. However, whether this is the case for time too remains one of the biggest questions of complexity theory (e.g., does $\mathsf{P} = \mathsf{NP}$?).

The following inclusions are well known (we remark that $\mathsf{L} = \mathsf{NL}$ does *not* follow from Savitch's theorem):

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME} \subseteq \mathsf{2\text{-}EXPTIME}.$$

One of the early goals of complexity theory was to establish equivalence among problems in a given complexity class $\mathcal{C}$ (most notably NP). To do that, one needs the notion of a reduction:

Definition 2.14. Let L_1 and L_2 be two languages. We say that L_1 *reduces*[6] to L_2, denoted $L_1 \leq L_2$, if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$, computable in polynomial time, such that $x \in L_1 \Leftrightarrow f(x) \in L_2$, for all $x \in \{0,1\}^*$.

[5]The space of the input is not counted.

[6]There are various types of reductions. The reductions we use are known as many-to-one polynomial-time reductions, also known as Karp reductions.

Let $\mathcal{C}$ be a complexity class. A language L is said to be $\mathcal{C}$-*hard* if, for any language L' in $\mathcal{C}$, $L' \leq L$. If, in addition, L is in $\mathcal{C}$, L is said to be $\mathcal{C}$-*complete*.[7] Perhaps the most extensively studied class in connection with its complete languages is NP. Hundreds of languages are known to be NP-complete [GJ79], some of which are:

- `SATISFIABILITY`: contains all the pairs $\langle \psi, \alpha \rangle$, where ψ is (encodes) a propositional formula in conjunctive normal form and α is (encodes) a truth assignment that satisfies ψ.
- `3COLOURING`: contains all the pairs $\langle G, \chi \rangle$, where G is (encodes) an undirected graph and χ is an assignment of three colours, say 0, 1, and 2, to the vertices of G such that no two adjacent vertices have the same colour.
- `INTEGER-LINEAR-PROGRAMMING`: contains all the pairs $\langle A, \boldsymbol{b} \rangle$, where A is an $m \times n$ integer matrix and $\boldsymbol{b}$ is an m-vector, such that there exists a non-negative integer vector $\boldsymbol{x}$ for which $A\boldsymbol{x} \leq \boldsymbol{b}$.

Finally, we mention that there exist other computational models used in finite model theory, e.g. automata and circuits, but none of these is of use to us. Circuits are briefly mentioned in Section 3.4; for more details see [Vol13].

For the sake of brevity, no problem or algorithm mentioned in the sequel is stated in terms of Turing machines; this is, however, possible in principle.

[7] To define hardness and completeness for L and NL, one needs a different type of reduction, called a log-space reduction.

3 | Related Work

In this chapter we overview some results relevant to our research. Of particular interest are Sections 3.1 and 3.3.

3.1 Computational logic

After the results by Church [Chu36a, Chu36b] and Turing [Tur36] on the unsolvability of Hilbert's decision problem, i.e. the impossibility of finding an algorithm for determining the satisfiability of arbitrary first-order sentences, the original question was revised: are there any syntactic classes of first-order sentences for which satisfiability is algorithmically decidable? Originally, the main focus was on prefix classes, i.e. classes of formulas in prenex normal form with restrictions on the order of quantifiers. A full classification of such classes with respect to decidability [DGL79, Lew79, Gol84, Für81] and computational complexity [Grä90, KV90a, Lew80, BGG01] ended up taking a few decades.

Later the focus shifted towards finite-variable fragments, mainly because of their use in finite model theory and descriptive complexity [CFI92, DLW95, Imm82, Imm91, IK89, KV90a, KV90b, Var95]. The most important fragment in this category is the two-variable fragment, because its satisfiability problem is decidable and NEXPTIME-complete [Mor75, Lew80, Für84, GKV97], but the satisfiability problem for the three-variable fragment is undecidable, as the latter fragment properly contains the undecidable class $\forall\exists\forall$ [GK72, KMW62, BGG01]. In this regard, the two-variable fragment draws the boundary between decidability and undecidability for finite-variable fragments. The two-variable fragment has the finite-model property, so its satisfiability and finite satisfiability problems coincide. The satisfiability problem for the two-variable fragment with counting is also known to be decidable and NEXPTIME-complete, both with respect to arbitrary models and finite models [GOR97, PST97, PH05]. The latter fragment does not have the finite model property; indeed, the sentence $\forall x \exists_{=1} y\, r(x, y) \wedge \forall x \exists_{\leq 1} y\, r(y, x) \wedge \exists x \forall y\, \neg r(y, x)$ is satisfiable only in infinite models.

Modal logic [BDRV01, BvBW06] is another important and thoroughly studied branch of mathematical logic. It extends propositional logic with two modal operators $\Box$ (box) and $\Diamond$ (diamond), originally intended to capture the notions of necessity and possibility. The introduction of relational semantics for modal logic by Kripke [Kri59, Kri63, Kri71] precipitated a flurry of interest, with modal logics being used in many areas to reason about time, knowledge, belief, and computation. That was partly due to their good computational behaviour: satisfiability for modal logic is decidable, and it remains decidable for many powerful extensions of the basic language, e.g. with path quantifiers or fixed-point operators. (In terms of complexity, satisfiability for basic modal logic is PSPACE-complete [Lad77], and for the latter two extensions it is EXPTIME-complete [EH85, EJ88].) From the perspective of first-order logic, (basic) modal logic embeds in the two-variable fragment, and this is a partial explanation for its decidability—a better one being its tree-model property [Var96].

The guarded fragment of first-order logic was introduced by Andréka, Neméti, and van Benthem [ANvB98] as a generalization of modal logic and an attempt to explain its good computational properties. The main insight for this project was that, when translated to first-order logic, modal formulas always lead to quantifiers relativised (guarded) by atoms. This was thought to be the feature causing the good behaviour of modal logic, as it also leads to a 'tree-like-model property' [Grä99], compatible with Vardi's view on the matter [Var96]. The guared fragment has the finite-model property and its (finite) satisfiability problem is EXPTIME-complete for predicates of bounded arity and 2-EXPTIME-complete for arbitrary predicates [Grä99]. These bounds remain true even after the addition of fixed-point operators [GW99] (as with modal logic). Conjunctive query answering *without* a background theory is 2-EXPTIME-complete [BGO10], with respect to both finite and arbitrary models.

As already mentioned, description logics can be viewed as variants of modal logics [Sch91], and thus exhibit a similar computational behaviour. We are mainly interested in the description logic $\mathcal{ALCQI}$, which is used in the sequel. Reasoning for $\mathcal{ALCQI}$ is known to be EXPTIME-complete, both with respect to finite models [LST05] and arbitrary models [Gia95, DGL96]. The complexity of query entailment (and answering) for $\mathcal{ALCQI}$ is 2-EXPTIME-complete [Lut08]. (We use the latter result in Section 7.1.)

Of particular interest is the two-variable guarded fragment with counting, as it is a proper superset of the description logic $\mathcal{ALCQI}$ (see Section 4.3). This fragment does not have the finite model property, so its satisfiability and finite satisfiability problems do not coincide. Kazakov [Kaz04] showed that satisfiability for the two-variable fragment with counting is EXPTIME-complete by providing a translation to the three-variable guarded fragment. Later, Pratt-Hartmann [PH07], whose approach we employ in Chapter 5, showed that finite satisfiability for the above fragment is also EXPTIME-complete; his method can be generalized to handle infinite models too. (We mention that adding counting quantifiers to the *three*-variable guarded fragment leads to undecidability.) The complexity of (finite) conjunctive query answering for the above fragment in the presence of a background theory is unknown. (We provide a 2-EXPTIME algorithm for this problem in Chapter 7.) There is, however, a bound for the data complexity of the latter problem, which is coNP-complete [PH09].

Finally, it is worth mentioning that ten Cate and Segoufin [tCS11, StC13] gave an alternative explanation for the good computational properties of modal logic, by introducing the unary negation fragment of first-order logic, which only allows the negation of formulas with one free variable. An even bigger fragment that contains both the guarded fragment (and, thus, modal logic) and the unary negation fragment is the guarded negation fragment [BtCS11, BtCS15]. The latter fragment dictates that negation, and not quantifiers, be guarded by atoms. The satisfiability problem for the unary negation and the guarded negation fragment is 2-EXPTIME-complete, both with respect to finite and arbitrary models.

3.2 The relational model

During the early years of database systems, data was stored in an unordered manner using the means provided by the underlying operating system. It was soon realized that this approach obscured the process of querying the data, and that it was necessary to have *data models*, which would abstract the logical structure of the data from the underlying implementation details. Various data models were proposed; the one that became most popular, however, was Codd's *relational model* [Cod70].

COUNTRY				
name	continent	capital	population	currency
Colombia	S. America	Bogotá	47.12m	Col. Peso
United States	N. America	Washington	318.9m	U.S. Dollar
United Kingdom	Europe	London	64.1m	Pound
France	Europe	Paris	66.03m	Euro
Germany	Europe	Berlin	80.62m	Euro
Nigeria	Africa	Abuja	173.6m	Naira
China	Asia	Beijing	1.357b	Renminbi
Japan	Asia	Tokyo	127.3m	Yen
Australia	Australia	Canberra	23.13m	Aus. Dollar

Table 3.1: A table containing information about some countries.

With the relational model, data is stored in *relations*. A relation is similar to a mathematical relation, but it has named attributes that take values from a given domain. In more detail, to each relation R is associated a *relation schema*, specifying its *name*, its *arity* $\alpha(R)$ (this is the number of its attributes), and a set of attributes $A_1, \dots, A_{\alpha(R)}$. (In the sequel we often speak of attributes in a relation instead of its schema for simplicity.) A relation R, then, consists of a set of *tuples* $\langle d_1, \dots, d_{\alpha(R)} \rangle$, where each d_i $(1 \leq i \leq \alpha(R))$, corresponding to the attribute A_i, is a *constant* (or *data value*) from a countably infinite domain $\mathcal{D}_i$. By convention, relations are represented as tables, see Table 3.1 for instance.

A *database schema* is a finite set of relation schemas, where no two relations are allowed to have the same name. A *database instance* (or, simply, a *database*) Δ over a database schema Σ is a collection of relations, one for each relation schema in Σ. The set of constants that appear in a database instance is called its *active domain*.

The purpose of having data inside a database is to execute *queries* against it, i.e. to ask questions about its contents. For example, some of the questions that one can ask with respect to the relation in Table 3.1 are:

(i) What is the population of Nigeria?
(ii) Are there any countries in Europe with different currency?
(iii) Which continents (in the table) have more than one country?
(iv) Which countries have a currency containing the word ‘Dollar’?

The corresponding answers to these questions are:

(i) 173.6m.

(ii) Yes.
(iii) Europe, Asia.
(iv) United States, Australia.

Notice that queries (i), (iii), and (iv) above require a set of answers, and query (ii) requires just a 'yes' or 'no' answer. Queries of the latter type are called *boolean*.

One needs to be careful about the semantics of the queries executed against a database. Take, for example, query (i) above, about the population of Nigeria. Assuming that the information in the database is correct, one may take the result of the query as a fact about the world, i.e. that the population of Nigeria is 173.6m. Similarly, if one is looking for a train line connecting Liverpool to the Isle of Man and none exists in the National Rail database, then one can conclude that none exists in reality. Such queries are said to be under the *closed-world assumption*, which is the assumption that a statement is true if its truth follows from the database. (Conversely, everything that does not follow from the database is false.) Most commercial database systems follow this assumption.

In contrast, the result of queries like (iii) above does not represent a state of the world, but a state of *knoweledge* about the world. Indeed, the fact that Table 3.1 contains a single country from South America does not mean that South America contains a single country. Similarly, if one does not find Patrick Fluff in Wikipedia, it does not follow that Patrick Fluff does not exist. Queries that are interpreted that way are said to be under the *open-world assumption*. This assumption is more commonly followed by knowledge representation systems.

Of course, queries are meant to be executed by machines, thus it is necessary to have *query languages* to formally define the queries. The most popular query language, on which many practical query languages like SQL are based, is the *relational algebra*, introduced by Codd [Cod70]. It uses a small number of primitive operations on relations, and provides ways of combining them.

Let R be a relation with attributes $A_1, \dots, A_k$. If $s = \langle d_1, \dots, d_k \rangle$ is an R-tuple, and $B \subseteq \{A_1, \dots, A_k\}$, we denote by $s[B]$ the restriction of s on B, i.e. the result of deleting from s the constants corresponding to any attribute not in B.

The primitive operations of relational algebra are the following:

- If R is a relation with attributes $A_1, \dots, A_k$, then its *projection* on a set of attributes $B \subseteq \{A_1, \dots, A_k\}$, denoted $\pi_B(R)$, is the relation $\{s[B] : s \in R\}$.

LANGUAGE	
cname	lang
Colombia	Spanish
United States	English
United Kingdom	English
France	French
Germany	German
Nigeria	English
China	Chinese
Japan	Japanese
Australia	English

Table 3.2: The official languages of the countries in Table 3.1.

- If R_1 is a relation with attributes $A_1, \ldots, A_k, B_1, \ldots, B_m$ and R_2 is a relation with attributes $B_1, \ldots, B_m, C_1, \ldots, C_n$, then the *natural join* of R_1 and R_2, denoted $R_1 \bowtie R_2$, is the relation $\{s \cup t : s \in R_1 \text{ and } t \in R_2\}$ with attributes $A_1, \ldots, A_k, B_1, \ldots, B_m, C_1, \ldots, C_n$, where $s \cup t$ denotes the $(R_1 \bowtie R_2)$-tuple whose components are the corresponding constants of the R_1-tuple s and the R_2-tuple t, *without* repeating common attributes.

- If R_1 and R_2 are two relations with attributes $A_1, \ldots, A_k$ and having the same domain for the values corresponding to each attribute, then the *union* of R_1 and R_2, denoted $R_1 \cup R_2$, is simply the set-theoretic union of R_1 and R_2, i.e. $R_1 \cup R_2 = \{s : s \in R_1 \text{ or } s \in R_2\}$.

- If R_1 and R_2 are two relations with attributes $A_1, \ldots, A_k$ and having the same domain for the values corresponding to each attribute, then the *difference* of R_1 and R_2, denoted $R_1 - R_2$, is simply the set-theoretic difference of R_1 and R_2, i.e. $R_1 - R_2 = \{s : s \in R_1 \text{ and } s \notin R_2\}$.

- If R is a relation with attributes $A_1, \ldots, A_k$ and $A, B \in \{A_1, \ldots, A_k\}$, a *selection* on R by $A = B$, denoted $\sigma_{A=B}(R)$, is the relation $\{s \in R : s[A] = s[B]\}$. We often abuse the previous definition by using binary relations other than $=$ (e.g. $\leq$, $\geq$, $\neq$) and/or a constant in place of B.

- If R is a relation with attributes $A_1, \ldots, A_k$, $A \in \{A_1, \ldots, A_k\}$, and B is a new attribute name, the *renaming* in R of A to B, denoted $\varrho_{A/B}(R)$ is a relation exactly the same as R, except that attribute A is renamed to B.

Definition 3.1. Let Σ be a database schema comprising the relations $R_1, \ldots, R_k$. The set of *relational algebra expressions* over Σ are the expressions E generated by the following grammar:

$$E = R_1 \mid \cdots \mid R_k \mid \pi_A(E) \mid E \bowtie E \mid E \cup E \mid E - E \mid \sigma_{A=B}(E) \mid \varrho_{A/B}(E).$$

We often refer to relational algebra expressions simply as *queries*. The *evaluation* of a relational algebra expression for a given database Δ is the process of computing the relation it specifies according to the operations defined above.

As an example, in addition to Table 3.1, consider Table 3.2 with the official language of each country in Table 3.1. Then, the expression

$$\pi_{\mathsf{name,\,lang}}(\sigma_{\mathsf{continent}=\mathrm{Europe}}(\text{COUNTRY} \bowtie \varrho_{\mathsf{cname/name}}(\text{LANGUAGE})))$$

returns the names and official languages of the European countries (in the above tables), i.e.

$$\{\langle \text{United Kingdom}, \text{English}\rangle, \langle \text{France}, \text{French}\rangle, \langle \text{Germany}, \text{German}\rangle\}.$$

Viewing each of the above relations as a set of ground atoms over the relational signature $\langle \text{COUNTRY}(\cdot,\cdot,\cdot,\cdot,\cdot), \text{LANGUAGE}(\cdot,\cdot)\rangle$, we can write the above expression as a conjunctive query (recall Section 2.1). Indeed, let

$$\varphi(x,y) := \exists z_1 \exists z_2 \exists z_3 (\text{COUNTRY}(x, \text{Europe}, z_1, z_2, z_3) \wedge \text{LANGUAGE}(x,y)).$$

It is clear that the set $\{\langle x,y\rangle \mid x,y \in \mathcal{D} \text{ and } \varphi(x,y)\}$, where $\mathcal{D}$ is the set of constants appearing in Tables 3.1 and 3.2, is exactly the set returned by the above relational algebra expression. This is not a coincidence:

Theorem 3.2 (Codd [Cod72])**.** Relational algebra and relational calculus (i.e. first-order logic) have the same expressive power.

Furthermore, the translation of relational algebra to and from relational calculus can be performed in polynomial time. This is important because it enables the use of logical methods in the theory of databases.

3.3 Integrity constraints

One serious limitation of the relational model is its lack of semantics [SS75, Ken79]. A way to remedy this limitation is by extending it to capture more semantics [Cod88]; another way is by specifying integrity constraints known as *data dependencies*. Data dependencies are, in simple terms, specifications that identify certain database states as illegal. Some extensively studied types of data dependencies are functional dependencies [Cod70, Arm74], multivalued dependencies [Zan76, Fag77], join dependencies [Ris78], and tuple- or equality-generating dependencies [BV81, Fag82]. We give a brief description for each of the above.

Functional dependencies Given relation R from a relation schema Σ, we say that a set of attributes X in R *functionally determines* a set of attributes Y in R, denoted $X \to Y$, if, for each pair of tuples $s, t \in R$,

$$\pi_X(s) = \pi_X(t) \Longrightarrow \pi_Y(s) = \pi_Y(t).$$

In words, two tuples having the same values of X must have the same values of Y. This kind of dependencies is related to the Boyce-Codd normal form (BCNF), used in database normalization.

Multivalued dependencies Given relation R from a relation schema Σ, we say that a set of attributes X in R *multi-determines* a set of attributes Y in R, denoted $X \twoheadrightarrow Y$, if, for each pair of tuples $s, t \in R$ such that $\pi_X(s) = \pi_X(t)$:

(i) there exists a tuple $u \in R$ with $\pi_X(u) = \pi_X(s)$, $\pi_Y(u) = \pi_Y(s)$, and $\pi_Z(u) = \pi_Z(t)$, for all attributes in $R \setminus (X \cup Y)$;

(ii) there exists a tuple $v \in R$ with $\pi_X(v) = \pi_X(s)$, $\pi_Y(v) = \pi_Y(t)$, and $\pi_Z(v) = \pi_Z(s)$, for all attributes in $R \setminus (X \cup Y)$.

This kind of dependencies is related to the fourth normal form (4NF), used in database normalization.

Join dependencies A *join dependency* is an expression $\bowtie[X_1, \ldots, X_k]$, where $X_1, \ldots, X_k$ is a partition of the set of attributes in a relation R (from a relational schema Σ). We say that R *satisfies* $\bowtie[X_1, \ldots, X_k]$ if $R = \bowtie_{i=1}^{k} \pi_{X_i}(R)$. According to the above, multivalued dependencies are a special case of join dependencies: $X \twoheadrightarrow Y$ is equivalent to $\bowtie[XY, X(R \setminus Y)]$.

Tuple- and equality-generating dependencies This kind of dependencies provides a unifying framework, based on mathematical logic, for many types of dependencies (including the above). Their definition comes from the observation that dependencies usually require the existence of certain tuples or components of tuples in a relation (hence the term 'generating') given the existence of other tuples. These required tuples can be new (as with multivalued or join dependencies) or already in the relation (as with functional dependencies); they can either be obtained from the existing tuples (as in all the above cases) or not (*cf.* inclusion dependencies [AHV95]). *Tuple-generating dependencies* are first-order logic sentences of the form

$$\forall x_1 \ldots \forall x_n [\varphi(x_1, \ldots, x_n) \rightarrow \exists z_1, \ldots, z_k\, \psi(y_1, \ldots, y_m)],$$

where $\{z_1, \ldots, z_k\} = \{y_1, \ldots, y_m\} \setminus \{x_1, \ldots, x_n\}$, φ is a (possibly empty) conjunction of atoms, and ψ a non-empty conjunction of non-equality atoms. An *equality-generating dependency* is exactly like a tuple-generating dependency, but with ψ being a non-empty conjunction of equality atoms.

To clarify the above let us give some examples. For functional dependencies, suppose that we have a database containing a relation

$$\mathsf{student}(\mathsf{name}, \mathsf{surname}, \mathsf{address}, \mathsf{mark}, \mathsf{honours}),$$

and that the degree classification (honours) is determined from the mark as described in the following table:

mark	honours
above 70%	1st
60% – 70%	2:1
50% – 60%	2:2
40% – 50%	3rd

Then, we have the dependency $\mathsf{mark} \rightarrow \mathsf{honours}$. Notice, in that case, that we cannot have two tuples with the same mark but different degree classification.

For multivalued dependencies, suppose that we have a database containing the relation $\mathsf{apply}(\mathsf{id}, \mathsf{college}, \mathsf{hobby})$ of students applying to colleges. Then, the existence of some tuples in the database may imply the existence of other tuples, as seen in the following tables:

id	college	hobby
73	Oxford	piano
73	Cambridge	violin

$\Rightarrow$

id	college	hobby
73	Oxford	piano
73	Cambridge	violin
73	Oxford	violin
73	Cambridge	piano

Indeed, if the student is allowed to declare more than one hobby, we may fill in undeclared hobbies for each college, based on the ones declared elsewhere. Thus, in the example above, we have $\mathsf{college} \twoheadrightarrow \mathsf{hobby}$. (Such operations are not always legal, e.g. for security reasons.) Join dependencies are similar, but include more attributes.

Notice that the first example above is equivalent to the equality-generating dependency

$$\forall x_1 \cdots x_5 y_1 \cdots y_3 y_5 [(\mathsf{student}(x_1, x_2, x_3, x_4, x_5) \wedge \\ \mathsf{student}(y_1, y_2, y_3, x_4, y_5)) \to x_5 = y_5],$$

and the second to the tuple-generating dependency

$$\forall x_1 x_2 x_3 y_2 y_3 [(\mathsf{apply}(x_1, x_2, x_3) \wedge \mathsf{apply}(x_1, y_2, y_3)) \to \\ (\mathsf{apply}(x_1, x_2, y_3) \wedge \mathsf{apply}(x_1, y_2, x_3))].$$

One of the most valuable tools for studying dependencies is an algorithm called *the chase*. It originated in the work of Aho, Beeri, and Ullman [ABU79], and Meier, Mendelzon, and Sagiv [MMS79]; see also [Fag82, BV84, DNR08]. The chase has applications in such diverse areas as query implication [MMS79, MSY81], query containment [CM77, ASU79, JK82, CDGL02, MHF03], and data integration and exchange [Len02, FKMP05, Her10]. For applications to query answering under integrity constraints see [CLR03, Ros06, Ros11, CGK08].

Path-functional dependencies [Wed89, Wed92, IW94, vBW94] are a generalization of functional dependencies, better suited for semantic data models [SFL83, Bor85, Cod88, MBW80, SS80, Zan83] and object-oriented databases [BDK92, CBB$^+$97, KLW95, VDBVGAG97, AK89]. Whereas with functional dependencies we have (sets of) attributes determining other (sets of) attributes, with path-functional dependencies we have (sets of) *paths* of attributes determining other (sets of) paths of attributes. For example, the dependency

$$\mathsf{student.hall.address.postcode} \rightarrow \mathsf{student.college.city}$$

is a path-functional dependency stating that the postcode of the address of a student's hall of residence determines the city of the college they are attending. *Key path-functional dependencies* are a special class of path-functional dependencies, with which one is able to express that a set of attribute paths fully determines an object. For example, the dependency

$$\mathsf{student.name},\ \mathsf{student.surname},\ \mathsf{student.birthday} \rightarrow \mathsf{id}$$

is a key path-functional dependency, stating that a student is fully identified by their name, surname, and birthday. In addition to semantic data models and object-oriented databases, path-functional dependencies have been studied in the context of description logics [KTW01, TW04, TW05, TW08].

Our work generalizes the research on path-functional dependencies in two ways: (i) it establishes the complexity of (satisfiability and) query answering with path-functional dependencies *and* a background theory (which was a missing feature in the original research); and (ii) it extends the results related to description logics (which take into account a background theory in the form of an ontology) to a more general logical setting.

3.4 Expressiveness and complexity

The equivalence of relational algebra and relational calculus (Theorem 3.2) has enabled the use of many techniques from logic and (finite) model theory for the mathematical study of relational algebra. For any query language, in general, two properties are of particular interest: its expressiveness (i.e. the types of queries that it can express) and its complexity of evaluation. These two quantities are related by the empirical law that the more expressive a language becomes, the hardest it is to evaluate queries in it.

Both the expressiveness and complexity of relational calculus are well understood. For example, it can be shown by a simple compactness argument that relational calculus cannot express the reachability relation; that is, it cannot express that two nodes in a graph are connected by a path of arbitrary length. (It can, however, express that two nodes are connected by a path of length n, for any fixed natural number n.) In general, it cannot define relations that require some sort of recursion. One of the most useful tools for proving inexpressibility results are Ehrenfeucht-Fraïssé games [Fra50, Fra84, Ehr61]. We mention that many of the techniques used in model theory fail in the context of finite models (Ehrenfeucht-Fraïssé games being one of the few that 'survive'), thus *finite* model theory has been developed to deal with finite models. A central object of study in this theory is expressiveness of various languages (e.g. first-order logic, infinitary logics, fixed-point logics). See Libkin [Lib13] for more details.

In regard to complexity, it is known that the data complexity of evaluating (arbitrary) first-order queries is in AC^0 [Imm89], and thus in LOGSPACE. The combined complexity of evaluating (arbitrary) first-order queries is PSPACE-complete [Sto74, Var82]. (In practice, the complexity of evaluating queries, e.g. in SQL, is not as bad: usually the queries are small compared to the database, so the data complexity is a better measure; in addition, the analysis that led to the above bound for combined complexity is *worst case*.) Allowing counting does not change much: the data complexity of query evaluation for first-order logic with counting is in TC^0 [BIS90], and thus in LOGSPACE; the combined complexity is PSPACE-complete (the upper bound is straightforward and the hardness follows from the hardness of the same problem for plain first-order logic, discussed above).

Things get better when we restrict our attention to conjunctive queries: the data complexity remains the same, but the combined complexity becomes NP-complete [CM77]. These results remain true if one allows *unions* of conjunctive queries; see [SY80, vdM97, KMT98] for other related extensions. The combined complexity of query evaluation for first-order logic with k-variables ($k \geq 2$) is PTIME-complete [Var95].

4 | Guarded Fragments

In this chapter we introduce the guarded fragment and the two-variable guarded fragment with counting. Fundamental to our thesis is the central role of trees in the model theory of guarded fragments: (i) models of the guarded fragment are 'tree-like' [Grä99]; and (ii) models of the two-variable guarded fragment with counting (finite or infinite) are 'locally' tree-shaped, in the sense that they can be taken to have no cycles of length less than an arbitrarily large, fixed number Ω [PH09]. (We mention that a tree-model property is more straightforward with respect to infinite models for the two-variable guarded fragment with counting, as demonstrated by Kazakov [Kaz04].) The latter fact allows us to succinctly 'describe' certain configurations that we want to detect or avoid in models, and is used extensively throughout this thesis. For this reason, we repeat the proof of fact (ii) and, for completeness, also the proof of fact (i). Finally, we discuss the relationship between the description logic $\mathcal{ALCQI}$ and the guarded two-variable fragment with counting.

4.1 The guarded fragment

We now formally introduce the guarded fragment. For a given formula γ, free(γ) denotes the set of free variables that appear in γ.

Definition 4.1. The *guarded fragment* of first-order logic, denoted $\mathcal{GF}$, is defined inductively as follows:

- Atomic formulas are in $\mathcal{GF}$.
- $\mathcal{GF}$ is closed under boolean combinations.
- If $\bar{x}, \bar{y}$ are tuples of variables, $\alpha(\bar{x}, \bar{y})$ is atomic, and $\varphi(\bar{x}, \bar{y})$ is a formula in $\mathcal{GF}$ such that $\text{free}(\varphi) \subseteq \text{free}(\alpha) = \{\bar{x}, \bar{y}\}$, then the formulas $\exists \bar{y}(\alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y}))$ and $\forall \bar{y}(\alpha(\bar{x}, \bar{y}) \rightarrow \varphi(\bar{x}, \bar{y}))$ are in $\mathcal{GF}$.

To reduce clutter, we write formulas of the form $\exists\bar{y}(\alpha(\bar{x},\bar{y}) \wedge \varphi(\bar{x},\bar{y}))$ as $(\exists\bar{y}.\alpha)\varphi(\bar{x},\bar{y})$ and $\forall\bar{y}(\alpha(\bar{x},\bar{y}) \to \varphi(\bar{x},\bar{y}))$ as $(\forall\bar{y}.\alpha)\varphi(\bar{x},\bar{y})$.

It is straightforward to establish a normal form for $\mathcal{GC}^2$-formulas, that is, for every guarded sentence φ we can compute (in polynomial time) an equisatisfiable sentence

$$\psi \; := \; \exists\bar{x}C(\bar{x}) \wedge \bigwedge_{i=1}^{m} (\forall\bar{x}_i.\alpha_i)\exists\bar{y}_i\, \psi_i(\bar{x}_i,\bar{y}_i), \tag{4.1}$$

where C is a new relation symbol and ψ_i, for each i $(1 \le i \le m)$, is quantifier-free. (See [Grä99] for more details.) Henceforth, we restrict our attention to sentences of the above form.

We first introduce some standard concepts. The term 'type' is quite common in model theory (see, for example, [Mar06]), where it usually refers to a more general notion than the one we use here (the notion of a type that we use is often referred to as an atomic type).

Definition 4.2. Let σ be a relational signature. A *σ-literal* is an atomic formula or the negation of an atomic formula over σ. A *k-type* is a maximal consistent set of σ-literals in the variables $x_1,\dots,x_k$. (It is often useful to view a k-type as the conjunction of the formulas that constitute it.) For a given σ-structure $\mathfrak{A}$ and a tuple $(a_1,\dots,a_k) \in A^k$, we denote by $\mathrm{tp}_{\mathfrak{A}}[a_1,\dots,a_k]$ the unique k-type τ such that $\mathfrak{A} \models \tau(a_1,\dots,a_k)$. In this case, we say that τ is *realized* by $(a_1,\dots,a_k)$ or that $(a_1,\dots,a_k)$ *realizes* τ.

We now establish a 'tree-like-model property' for $\mathcal{GF}$. The rest of this section basically repeats [Grä99], but is included for completeness. As already mentioned, an important fact for our thesis is that models of the two-variable guarded fragment with counting (to be introduced later) are 'locally' tree-shaped, i.e. they can be taken to contain no small cycles. Grädel's result shows that—even in a more general logical setting with predicates of arity greater than two—models of guarded sentences can be taken to be 'tree-like', in the sense that if a sentence is satisfiable then it is satisfiable in a model that resembles (in a way that will be made precise) a tree.

Definition 4.3. The *size* of a k-type is the number of distinct components in any tuple that realizes it. (The size of a k-type may be smaller than k if the type contains equalities.) A $(k+\ell)$-type τ' *extends* a k-type τ if $\tau \subseteq \tau'$. Further, τ' extends τ *by m new elements* if τ' extends τ, and m is the difference between the sizes of τ' and τ. A k-type τ is a *reduction* of an m-type τ', $k \leq m$, if there exists a substitution $\rho : \{1, \ldots, k\} \to \{1, \ldots, m\}$ such that, for any tuple $(a_1, \ldots, a_m)$ that realizes τ', $(a_{\rho(1)}, \ldots, a_{\rho(k)})$ realizes τ.

Definition 4.4. Let ψ be a guarded sentence and let n be the number of distinct variables in ψ. We call a *witness* for the satisfiability of ψ the tuple $(T, \tau_0, \mathrm{ext}_1, \ldots, \mathrm{ext}_m)$, where:

- $T = \bigcup_{k \leq n} T^{(k)}$, where for all $k \leq n$, $T^{(k)}$ is a set of k-types, and T is closed under reductions, i.e. if $\tau \in T$ and τ^- is a reduction of τ, then $\tau^- \in T$.
- $\tau_0 \in T$ is such that $\tau_0(\bar{x}) \models C(\bar{x})$.
- For each conjunct $(\forall \bar{x}_i.\alpha_i)\exists \bar{y}_i\, \psi_i(\bar{x}_i, \bar{y}_i)$, where $\bar{x}_i = (x_1, \ldots, x_{k_i})$ and $\bar{y}_i = (y_1, \ldots, y_{\ell_i})$, $\mathrm{ext}_i : T^{(k_i)} \to T^{(k_i+\ell_i)}$ is an extension function such that

  (i) $\mathrm{ext}_i(\tau)$ extends τ;
  (ii) $\mathrm{ext}_i(\tau) \models \alpha_i(\bar{x}) \to \psi_i(\bar{x}, \bar{y})$.[1]

We now prove that the existence of a witness is necessary for satisfiability.

Lemma 4.5. Let ψ be a guarded sentence. If ψ is satisfiable, then it has a witness.

Proof. Let $\mathfrak{A}$ be a model of ψ. For all $k \leq n$, let $T^{(k)}$ be the set of all k-types realized in $\mathfrak{A}$; set $T = \bigcup_{k \leq n} T^{(k)}$. Choose a tuple $\bar{a}_0$ such that $\mathfrak{A} \models C(\bar{a}_0)$ and set $\tau_0 = \mathrm{tp}_{\mathfrak{A}}[\bar{a}_0]$. Now, for the definition of the extension functions, let $\bar{a} = (a_1, \ldots, a_{k_i}) \in A^{k_i}$ be a tuple such that $\mathfrak{A} \models \alpha_i(\bar{a})$. Clearly, since $\mathfrak{A} \models \psi$, which implies $\mathfrak{A} \models (\forall \bar{x}_i.\alpha_i)\exists \bar{y}_i\, \psi_i(\bar{x}_i, \bar{y}_i)$, there exists a tuple $\bar{a}' = (a_1, \ldots, a_{\ell_i}) \in A^{\ell_i}$ such that $\mathfrak{A} \models \psi_i(\bar{a}, \bar{a}')$. Define $\mathrm{ext}_i : T^{(k_i)} \to T^{(k_i+\ell_i)}$, as

$$\mathrm{ext}_i(\tau) = \begin{cases} \mathrm{tp}_{\mathfrak{A}}[\bar{a}, \bar{a}'] & \text{if } \tau = \mathrm{tp}_{\mathfrak{A}}[\bar{a}], \\ \tau & \text{otherwise.} \end{cases}$$

One can easily verify that each ext_i $(1 \leq i \leq m)$ satisfies Definition 4.4. □

[1] That is, given a structure $\mathfrak{A}$ and a tuple $\bar{a} = (a_1, \ldots, a_{k_i}) \in A^{k_i}$ such that $\mathfrak{A} \models \alpha_i(\bar{a})$ and $\mathrm{tp}_{\mathfrak{A}}[\bar{a}] = \tau$, $\mathrm{ext}_i(\tau)$ gives us the type τ' that any tuple $\bar{a}' = (a_1, \ldots, a_{k_i}, a'_1, \ldots, a'_{\ell_i}) \in A^{k_i+\ell_i}$ extending $\bar{a}$ should realize in order for it to satisfy ψ_i, i.e. for $\mathfrak{A} \models \psi_i(\bar{a}')$ to hold. Clearly, in such a case, τ' must extend τ.

We are now ready to prove that the guarded fragment has the tree model property; that is, every satisfiable guarded sentence is satisfiable in a tree-like model. However, we must first make precise what we mean by ‘tree’ or ‘tree-like’ in this case, as predicates of arbitrary arity are allowed, thus we are usually unable to have trees in the pure graph-theoretic sense. Another notion from graph theory plays a more important role here: the notion of tree-width. Intuitively, this measures how close a graph is to being a tree. We generalize that for an arbitrary relational structure.

Definition 4.6. Let σ be a relational vocabulary and $\mathfrak{A}$ be a σ-structure. We say that $\mathfrak{A}$ is a *k-tree* if there exists a tree $S = (V, E)$ and a function $F : V \to \{X \subseteq A : |X| \leq k\}$, assigning to every node v of S a set $F(v)$ of at most k elements of A, such that:

(i) For every σ-atom $\alpha(x_1, \dots, x_r)$ and every tuple $(a_1, \dots, a_r) \in A^r$ such that $\mathfrak{A} \models \alpha(a_1, \dots, a_r)$, there exists a node v of S such that $\{a_1, \dots, a_r\} \subseteq F(v)$.

(ii) For every element a of $\mathfrak{A}$, the set of nodes $\{v \in V \mid a \in F(v)\}$ is connected.

Further, we say that $\mathfrak{A}$ has *bounded degree* if, for every node of S, the number of its neighbours is bounded by some fixed $c \in \mathbb{N}$.

Theorem 4.7. Let ψ be an arbitrary guarded sentence with k variables. If ψ is satisfiable, then there exists a k-tree with bounded degree which is a model of ψ.

Proof. To begin with, we may assume, Lemma 4.5, that ψ has a witness

$$(T, \tau_0, \mathrm{ext}_1, \dots, \mathrm{ext}_m).$$

Our goal is to construct a model $\mathfrak{B}$ of ψ that is a k-tree. We will simultaneously construct the tree $S = (V, E)$ and the function F (according to Definition 4.6), as well as the substructure $\mathfrak{F}(v) \subseteq \mathfrak{B}$, $v \in V$, induced by $F(v)$. For the universe of $\mathfrak{B}$, we set $B = \bigcup_{v \in V} F(v)$.

We start with the root λ of S. Our intention here is to make $\mathfrak{F}(\lambda) \models \exists \bar{x} C(\bar{x})$ hold. Let r be the arity of C. We set $F(\lambda) = \{\lambda_1, \dots, \lambda_r\}$ and set $\mathrm{tp}_{\mathfrak{B}}[\lambda_1, \dots, \lambda_r] = \tau_0$. $\mathfrak{F}(\lambda)$ is now fully specified and clearly $\mathfrak{F}(\lambda) \models \exists \bar{x} C(\bar{x})$, as intended.

For the construction of the rest of the nodes in S, the idea is the following: Suppose the tree has been (inductively) constructed up to level l and let u be a leaf of S. To make sure that there is a substructure $\mathfrak{F}(w) \subseteq \mathfrak{B}$, for some w in S, such that $\mathfrak{F}(w) \models \alpha_i(\bar{x}_i) \to \exists \bar{y}_i \, \psi_i(\bar{x}_i, \bar{y}_i)$, let k_i, ℓ_i be the lengths of $\bar{x}_i$, $\bar{y}_i$ respectively. For every tuple $\bar{b} = (b_1, \ldots, b_{k_i}) \in F(u)^{k_i}$ such that $\mathfrak{F}(u) \models \alpha_i(\bar{b})$ and it is not already the case that $\mathfrak{F}(u) \models \alpha_i(\bar{b}) \to \exists \bar{y}_i \, \psi_i(\bar{b}, \bar{y}_i)$, create a new node w in S, connected by an edge with u, and set $F(w) = \{b_1, \ldots, b_{k_i}, b'_1, \ldots, b'_{\ell_i}\}$, where $b'_1, \ldots, b'_{\ell_i}$ are *new elements*,[2] i.e. they are not already in B. Note that, at this point, $\mathfrak{F}(u)$ is already fully specified, thus we can use ext_i which, given the k_i-type of $(b_1, \ldots, b_{k_i})$, will return the $(k_i + \ell_i)$-type to which $(b_1, \ldots, b_{k_i}, b'_1, \ldots, b'_{\ell_i})$ must be set.

In more detail, for each i, $1 \leq i \leq m$, and each tuple $\bar{b} = (b_1, \ldots, b_{k_i}) \in F(u)^{k_i}$, we create a new node w in S, connected by an edge with u, if and only if:

(i) at least one b_j, $1 \leq j \leq k_i$, is a new element at u;
(ii) $\mathfrak{F}(u) \models \alpha_i(\bar{b})$;
(iii) if $\tau = \text{tp}_{\mathfrak{F}(u)}[\bar{b}]$ and $\tau' = \text{ext}_i(\tau)$, then τ' extends τ by at least one element.

Further, if these conditions are satisfied, we set $F(w) = \{b_1, \ldots, b_{k_i}, b'_1, \ldots, b'_{\ell_i}\}$, as described previously, and set

$$\text{tp}_{\mathfrak{F}(w)}[b_1, \ldots, b_{k_i}, b'_1, \ldots, b'_{\ell_i}] = \text{ext}_i(\tau) = \text{ext}_i\Big(\text{tp}_{\mathfrak{F}(u)}[b_1, \ldots, b_{k_i}]\Big).$$

At this point, $\mathfrak{F}(w)$ is fully specified. Note that, for all v in S, the number of elements in $\mathfrak{F}(v)$ is less than or equal to k, since ψ has at most k variables.

This inductive construction generates an infinite tree. So far, though, we have only specified what happens for tuples $(b_1, \ldots, b_r)$ whose elements are subsets of $F(v)$, i.e. $\{b_1, \ldots, b_r\} \subseteq F(v)$, for some v in S. Call these tuples *local*. Given that conditions (i) – (iii) are satisfied, we have described how to extend all local tuples using the ext_i functions. To complete our construction, we impose that $\mathfrak{B} \models \neg R(\bar{b})$, for all non-local tuples $\bar{b}$ and all relation symbols R in ψ. This ensures that no guard α_i will be made true by a non-local tuple in $\mathfrak{B}$, thus we can completely disregard these cases. Guardedness is very convenient in that respect.

We now have to prove that $\mathfrak{B}$ is indeed a model of ψ. As mentioned above, we have $\mathfrak{F}(\lambda) \models \exists \bar{x} C(\bar{x})$, where λ is the root of S, thus $\mathfrak{B} \models \exists \bar{x} C(\bar{x})$, since $\mathfrak{F}(\lambda) \subseteq \mathfrak{B}$.

[2]Note that $b'_1, \ldots, b'_{\ell_i}$ need not necessarily be distinct. This will only be the case if the type that ext_i returns, extends the type of $(b_1, \ldots, b_{k_i})$ by exactly ℓ_i elements. However, for simplicity and without loss of generality, we will assume here that these elements are distinct.

Let $\bar{b} = (b_1, \ldots, b_{k_i}) \in B^{k_i}$, $1 \leq i \leq m$. If $\mathfrak{B} \models \neg\alpha_i(\bar{b})$, there is nothing to prove. Hence, suppose that $\mathfrak{B} \models \alpha_i(\bar{b})$. Clearly, the elements of $\bar{b}$ must be contained in some node of S, otherwise it would be the case that $\mathfrak{B} \models \neg\alpha_i(\bar{b})$. Let u be the first node in S such that $\{b_1, \ldots, b_{k_i}\} \subseteq F(u)$. Then, at least one of $b_1, \ldots, b_{k_i}$ must be a new element in $F(u)$, otherwise there would be a parent node of u containing all of them. Conditions (i) and (ii) are thus satisfied at $F(u)$.

Regarding condition (iii), let $\tau = \mathrm{tp}_{\mathfrak{F}(u)}[\bar{b}]$ and $\tau' = \mathrm{ext}_i(\tau)$. If τ' does not extend τ by any new elements, then τ' must already be realized in $\mathfrak{F}(u)$, since τ is maximally consistent (by definition) and it already fully specifies $\bar{b}$. If τ' does extend τ by at least one new element, then condition (iii) is also satisfied at $F(u)$, which means that u must have a successor node w in S. Then, it must be the case that $\mathfrak{F}(w) \models \alpha_i(\bar{x}_i) \rightarrow \exists\bar{y}_i\, \psi_i(\bar{x}_i, \bar{y}_i)$, thus τ' is realized in w. In any case, τ' is realized in $\mathfrak{B}$, therefore $\mathfrak{B} \models \alpha_i(\bar{x}_i) \rightarrow \exists\bar{y}_i\, \psi_i(\bar{x}_i, \bar{y}_i)$, $1 \leq i \leq m$. It follows that $\mathfrak{B} \models (\forall\bar{x}_i.\alpha_i)\exists\bar{y}_i\, \psi_i(\bar{x}_i, \bar{y}_i)$, $1 \leq i \leq m$, whence $\mathfrak{B} \models \psi$. $\square$

This establishes a 'tree-like-model property' for $\mathcal{GF}$. This property allows one to establish the finite model property for $\mathcal{GF}$. We will also provide a proof of the latter result (this is, again, from [Grä99]); the uninterested reader may skip the rest of this section. The proof is based on an important result due to Herwig [Her95], which extends a result due to Hrushovski [Hru92]. Of critical importance here is the notion of a partial isomorphism, which is a first-order notion analogous to modal bisimulation [VB91, dR95].

Definition 4.8. Fix a pure relational vocabulary σ and let $\mathfrak{A}$, $\mathfrak{B}$ be two σ-structures. Further, let $p : A_0 \rightarrowtail B_0$, where $A_0 \subsetneq A$ and $B_0 \subsetneq B$, be an injective map from a proper subset of A to a proper subset of B. p is called a *partial isomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ if:

- For all $a_1, a_2 \in A_0$, $a_1 = a_2$ iff $p(a_1) = p(a_2)$.
- For all n-ary relation symbols R in σ and all $a_1, \ldots, a_n \in A_0$,

$$R^{\mathfrak{A}}(a_1, \ldots, a_n) \quad \text{iff} \quad R^{\mathfrak{B}}(p(a_1), \ldots, p(a_n)).$$

An *automorphism* of a structure $\mathfrak{A}$ is a bijective partial isomorphism from $\mathfrak{A}$ to itself.

Theorem 4.9 (Herwig [Her95])**.** Let $\mathfrak{A}$ be a finite structure with finite relational vocabulary. Then, there exists a finite extension $\mathfrak{A}^+$ (called a *Herwig extension*) of $\mathfrak{A}$ such that:

(i) Every partial isomorpism of $\mathfrak{A}$ extends to an automorphism of $\mathfrak{A}^+$.

(ii) Let $\bar{a} = (a_1, \dots, a_m)$ be a tuple such that $\mathfrak{A} \models \beta(a_1, \dots, a_m)$ for some atomic formula $\beta(x_1, \dots, x_m)$ in which all of $x_1, \dots, x_m$ occur. Then, there exists an automorphism g of $\mathfrak{A}^+$ such that $g(\bar{a})$ is in $\mathfrak{A}$.

Theorem 4.10. Every satisfiable guarded sentence has a finite model.

Proof. Let ψ be a satisfiable guarded sentence in the form of (4.1), and let $\mathfrak{B}$ be the tree model constructed in the proof of Theorem 4.7. Recall that $S = (V, E)$ is a (maybe infinite) tree of bounded degree and F is a function, assigning to each node in S a set of k elements in B, where k is the number of variables in ψ.

Observe, initially, that because ψ has a fixed, finite vocabulary, and for each node v in S we have $|F(v)| \leq k$, there is a finite number of substructures $\mathfrak{F}(v) \subseteq \mathfrak{B}$ induced by the sets $F(v)$, up to isomorphism. Consequently, there exists a finite subtree S' of S that contains all nodes in S, up to isomorphism. That is, for every node v in S there exists a node u in S', such that $\mathfrak{F}(v)$ and $\mathfrak{F}(u)$ are isomorphic. Let $\mathfrak{A} \subseteq \mathfrak{B}$ be the substructure induced by $A = \bigcup_{u \in S'} F(u)$, and let $\mathfrak{A}^+$ be a Herwig extension of $\mathfrak{A}$. We will prove that $\mathfrak{A}^+ \models \psi$.

To begin with, $\mathfrak{A} \models \exists \bar{x} C(\bar{x})$ thus $\mathfrak{A}^+ \models \exists \bar{x} C(\bar{x})$, since $\mathfrak{A}$, as explained above, must contain a substructure isomorphic to $\mathfrak{F}(\lambda)$, where λ is the root of S, and $\mathfrak{F}(\lambda) \models \exists \bar{x} C(\bar{x})$.

Regarding conjuncts $(\forall \bar{x}_i . \alpha_i) \exists \bar{y}_i \, \psi_i(\bar{x}_i, \bar{y}_i)$, $1 \leq i \leq m$, let $\bar{a} = (a_1, \dots, a_{k_i}) \in (A^+)^{k_i}$ such that $\mathfrak{A}^+ \models \alpha_i(\bar{a})$. By Herwig's theorem, there exists an automorphism g of $\mathfrak{A}^+$ such that $\bar{a}' = g(\bar{a})$ lies in $\mathfrak{A}$. Since $\bar{a}$ and $\bar{a}'$ are isomorphic, it is also the case that $\mathfrak{A} \models \alpha_i(\bar{a}')$. But, then, there must be a node v in S' such that $\{a_1, \dots, a_{k_i}\} \subseteq F(v)$. Let u be the successor node of v in S—without loss of generality we may assume that there exists one.

By construction, u being a successor of v, it must be the case that $\mathfrak{F}(u) \models \alpha_i(\bar{a}')$ and $\mathfrak{F}(u) \models \exists \bar{y}_i \psi_i(\bar{a}', \bar{y}_i)$. Now, by definition, S' must contain a node w such that $\mathfrak{F}(u)$ and $\mathfrak{F}(w)$ are isomorphic. Note that $\mathfrak{F}(u)$ is not necessarily a substructure of $\mathfrak{A}$ but, since w is in S', $\mathfrak{F}(w) \subseteq \mathfrak{A}$. Let p be an isomorphism from $\mathfrak{F}(u)$ to $\mathfrak{F}(w)$ and let $\bar{a}'' = p(\bar{a}')$. Clearly, $\mathfrak{F}(w) \models \alpha_i(\bar{a}'')$ and $\mathfrak{F}(w) \models \exists \bar{y}_i \psi_i(\bar{a}'', \bar{y}_i)$. But, again by Herwig's theorem, p can be extended to an automorphism f in $\mathfrak{A}^+$. Then, the composition $f \circ g$ is also an automorphism and $(f \circ g)(\bar{a}) = f(g(\bar{a})) = \bar{a}''$. It follows that $\mathfrak{F}(w) \models \exists \bar{y}_i \, \psi_i(\bar{a}, \bar{y}_i)$, thus $\mathfrak{A}^+ \models \exists \bar{y}_i \, \psi_i(\bar{a}, \bar{y}_i)$, since $\mathfrak{F}(w) \subseteq \mathfrak{A} \subseteq \mathfrak{A}^+$. Therefore, $\mathfrak{A}^+ \models (\forall \bar{x}_i . \alpha_i) \exists \bar{y}_i \, \psi_i(\bar{x}_i, \bar{y}_i)$, $1 \leq i \leq m$, hence $\mathfrak{A}^+ \models \psi$. $\square$

4.2 The two-variable guarded fragment with counting

We restrict our attention to first-order languages using the variables x and y, the usual boolean connectives, the quantifiers $\forall$, $\exists$, $\exists_{\leq C}$, $\exists_{\geq C}$ and $\exists_{=C}$ (for all $C > 0$), and the equality predicate (written as $=$). In the logics we consider, function symbols are not allowed. If φ is any formula, the *length* of φ, denoted $\|\varphi\|$, is the number of symbols it contains. We remark that the numbers in counting quantifiers are encoded in binary, and the length of formulas is computed using this binary representation.

Definition 4.11. The guarded two-variable fragment $\mathcal{GC}^2$ is the smallest set of formulas satisfying the following conditions:

- Atomic two-variable formulas are in $\mathcal{GC}^2$.
- Boolean combinations of $\mathcal{GC}^2$-formulas are in $\mathcal{GC}^2$.
- If φ is a $\mathcal{GC}^2$-formula with at most one free variable and u is a variable (i.e. either x or y), then $\forall u\, \varphi$ and $\exists u\, \varphi$ are in $\mathcal{GC}^2$.
- If φ is a $\mathcal{GC}^2$-formula, α is a binary predicate (guard), and u is a variable, then the formulas $\forall u(\alpha \to \varphi)$, $Qu(\alpha \wedge \varphi)$ and $Qu\, \alpha$, where $Q \in \{\exists, \exists_{\leq C}, \exists_{\geq C}, \exists_{=C}\}$, are in $\mathcal{GC}^2$.

Notice that, according to the above definition, constant symbols *cannot* be used in $\mathcal{GC}^2$-formulas: they are reserved only for use inside databases. In the sequel, formulas (not involving constants) that are obviously logically equivalent to a $\mathcal{GC}^2$-formula will typically be counted as $\mathcal{GC}^2$-formulas by courtesy; this relaxation allows certain formulas to be written in a more natural way than the above syntax strictly demands.

The following lemma introduces a normal form for $\mathcal{GC}^2$-formulas, analogous to that originally defined in [Sco62].

Lemma 4.12. Let ψ be a $\mathcal{GC}^2$-formula. We can compute in time polynomial in $\|\psi\|$, a sentence

$$\begin{aligned}\varphi \;:=\; \forall x\, \alpha \wedge \bigwedge_{1\leq j\leq n} & \forall x \forall y(e_j(x,y) \to (\beta_j \vee x = y)) \\ & \wedge \bigwedge_{1\leq i\leq m} \forall x \exists_{=C_i}\, y(f_i(x,y) \wedge x \neq y),\end{aligned}$$

such that: (i) α is a quantifier-free formula in one variable x, not involving equality; (ii) n, m are positive integers; (iii) e_j is a binary predicate different from $=$; (iv) β_j is a quantifier-free formula in the variables x, y, not involving equality; (v) C_i is a positive integer; (vi) f_i is a binary predicate other than $=$; and (vii) φ is finitely satisfiable if and only if ψ is.

Proof. Omitted, see [PH07]. □

Referring to the normal form in Lemma 4.12, the predicates f_i will be called *counting predicates* (in the context of a given φ in normal form).

Remark 4.13. It was shown in [Kaz04] that the satisfiability problem for $\mathcal{GC}^2$ is EXPTIME-complete, and in [PH07] that the finite satisfiability problem for $\mathcal{GC}^2$ is EXPTIME-complete. It is important in this regard that constants are not allowed to appear in $\mathcal{GC}^2$-formulas. It is easy to show (by encoding a grid of exponential size) that if even a single constant is allowed, one immediately gets an NEXPTIME lower bound for (finite) satisfiability.

For the rest of this section we fix a signature σ of unary and binary predicates and a $\mathcal{GC}^2$-sentence φ over σ in normal from as in Lemma 4.12. Referring to the normal form of φ, we let $C = \max_{1 \leq i \leq m}\{C_i\}$.

We now introduce some conventions regarding the use of k-types (recall Definition 4.2). Since the fragment we are concerned with only allows two variables, we only ever use 1- and 2-types. For convenience we fix the following notation: we use the letters π and τ, possibly with subscripts, to denote 1- and 2-types respectively. Let τ be a 2-type; we denote by τ^{-1} the 2-type which is the result of transposing the variables x and y in τ.

Let τ be a 2-type, $\mathfrak{A}$ be a structure, and let $a, b \in A$ be two distinct elements. If $\text{tp}_{\mathfrak{A}}[a, b] = \tau$, we define $\text{tp}_1(\tau) = \text{tp}_{\mathfrak{A}}[a]$ and $\text{tp}_2(\tau) = \text{tp}_{\mathfrak{A}}[b]$. That is, any 2-type τ induces two 1-types $\text{tp}_1(\tau)$ and $\text{tp}_2(\tau)$: the 1-type of the 'starting endpoint' of τ and the 1-type of the 'terminal endpoint' of τ. Note that $\text{tp}_2(\tau) = \text{tp}_1(\tau^{-1})$.

The notions of a message-type (invertible or non-invertible) and chromaticity are from [PH05].

Definition 4.14. Let τ be a 2-type. We say that τ is a *message-type* if $f_i(x, y) \in \tau$, for some counting predicate f_i ($1 \leq i \leq m$). For τ a message-type, if τ^{-1} is also a message-type, we say that τ is *invertible*; otherwise, τ is *non-invertible*. Finally, if τ is a 2-type such that neither τ nor τ^{-1} is a message-type, we say that τ is *silent*.

If π is a 1-type, involving (only) the variable x, we denote by $\pi[y/x]$ the result of replacing all occurrences of x in π by y.

Definition 4.15. Let π and π' be 1-types over σ. The *vacuous* type corresponding to π and π', denoted $\pi \times \pi'$, is the 2-type

$$\pi \cup \pi'[y/x] \cup \{\neg r(x,y), \neg r(y,x) \mid r \text{ binary predicate in } \sigma\}.$$

Lemma 4.16. Suppose $\mathfrak{A} \models \varphi$ and consider the structure $\mathfrak{A}^*$, obtained by replacing any silent 2-type in $\mathfrak{A}$ by its corresponding vacuous type, i.e. having

$$\mathrm{tp}_{\mathfrak{A}^*}[a,b] = \begin{cases} \mathrm{tp}_{\mathfrak{A}}[a] \times \mathrm{tp}_{\mathfrak{A}}[b] & \text{if } \mathrm{tp}_{\mathfrak{A}}[a,b] \text{ is silent,} \\ \mathrm{tp}_{\mathfrak{A}}[a,b] & \text{otherwise.} \end{cases}$$

Then $\mathfrak{A}^* \models \varphi$.

Proof. Evident. □

The concept of chromaticity, introduced in the following definition, provides an important simplification for the structures considered later.

Definition 4.17. Let $\mathfrak{A}$ be a structure. We say that $\mathfrak{A}$ is *chromatic* if the following are true:

- For all $a, b \in \mathfrak{A}$ such that $a \neq b$ and $\mathrm{tp}_{\mathfrak{A}}[a,b]$ is an invertible message-type, we have $\mathrm{tp}_{\mathfrak{A}}[a] \neq \mathrm{tp}_{\mathfrak{A}}[b]$.
- For all pairwise distinct $a, b, c \in \mathfrak{A}$ such that $\mathrm{tp}_{\mathfrak{A}}[a,b]$ and $\mathrm{tp}_{\mathfrak{A}}[b,c]$ are invertible message-types, we have $\mathrm{tp}_{\mathfrak{A}}[a] \neq \mathrm{tp}_{\mathfrak{A}}[c]$. (Note that, because of the above, $\mathrm{tp}_{\mathfrak{A}}[a]$, $\mathrm{tp}_{\mathfrak{A}}[b]$ and $\mathrm{tp}_{\mathfrak{A}}[c]$ must be pairwise distinct.)

In other words, a structure $\mathfrak{A}$ is chromatic if, for all $a \in A$, all elements $a' \in A$ that are reachable from a by a path of 1 or 2 invertible message-types have distinct 1-types (and different from the 1-type of a). Given any $\mathcal{GC}^2$-formula φ, the following lemma shows that any model of φ can be converted to a chromatic model by interpreting not-too-many new unary predicates.

Lemma 4.18. Given a formula φ in normal form, if φ has a model then it has a chromatic model over the same domain.

Proof. The following repeats [PH07]. Let $\mathfrak{A}$ be a model of φ. Define the graph $G = (A, E_1 \cup E_2)$, where

$$E_1 = \{(a,b) \in A^2 \mid a \neq b \text{ and } \mathrm{tp}_{\mathfrak{A}}[a,b] \text{ is an invertible message-type}\},$$
$$E_2 = \{(a,c) \in A^2 \mid a \neq c \text{ and there exists } b \in A \text{ s.t. } (a,b),(b,c) \in E_1\}.$$

That is, G is the graph on A that has edges between any two distinct elements $a, a' \in A$ that are connected by a path of 1 or 2 invertible message types in $\mathfrak{A}$.

It is evident (by the normal form) that for each $a \in A$ there are at most mC elements $b \in A$ such that $\mathrm{tp}_{\mathfrak{A}}[a,b]$ is an invertible message-type and for each of those b there are at most mC elements $c \in A$ (or $mC - 1$ if we disregard b) such that $\mathrm{tp}_{\mathfrak{A}}[a,b]$ is also an invertible message-type. Thus, the degree of G is at most $(mC)^2$. Now, G can be coloured with $(mC)^2 + 1$ colours using the standard greedy algorithm, such that no edge joins two nodes of the same colour. Those colours can be encoded using $\lceil (mC)^2 + 1 \rceil$ new predicates to obtain a chromatic model. $\square$

Thus, we can ensure that all the models we are working with are chromatic by using not-too-many new unary predicates.

We now proceed to formally define the notion of cycles. In addition to the standard, graph-theoretic definition we also introduce the notion of a 'strong' cycle. Further, the cycles that we are concerned with may contain distinguished elements, belonging to a given set K—in a database setting, K will be taken to be the active domain of a given database.

Definition 4.19. Let $\mathfrak{A}$ be a structure over a relational signature τ. We call a sequence of distinct elements $a_0, \ldots, a_{n-1}$ $(n \geq 3)$ in A a *cycle* if, for all i with $0 \leq i \leq n-1$, for some r in τ, $\mathfrak{A} \models r(a_i, a_{i+1})$ or $\mathfrak{A} \models r(a_{i+1}, a_i)$—where the addition inside the indices is performed modulo n. If a structure contains no cycles, we call it *acyclic*. The *length* of a cycle $a_0, \ldots, a_{n-1}$ is n—the number of edges that it comprises, when viewed as graph.

Let $\mathfrak{A}$ be a structure over a domain A, and let $K \subseteq A$ be a set of distinguished elements. A cycle in $\mathfrak{A}$ is *strong* if, for any consecutive pair of elements a and b in that cycle, either both $a, b \in K$ or $\mathrm{tp}_{\mathfrak{A}}[a,b]$ is an invertible message-type.

We now lay the foundations for a 'no-small-cycles' lemma, which is fundamental to our thesis: given any model $\mathfrak{A}$ of a $\mathcal{GC}^2$-formula φ (in normal form), we can obtain a new model $\mathfrak{B}$ of φ in which no small cycles—i.e. cycles having length less than a fixed number Ω—appear. The following repeats [PH09].

Let $\mathfrak{A}$ be a structure over a relational signature σ, and k be a positive integer. For each $i \in \{1, \ldots, k\}$ let $\mathfrak{A}_i$ be a copy of $\mathfrak{A}$ with the domains A_i being pairwise disjoint. We denote by $\mathfrak{A}^{\times k}$ the structure with domain $A_1 \cup \ldots \cup A_k$ and interpretations $q^{\mathfrak{A}^{\times k}} = \bigcup_{i=1}^{k} q^{\mathfrak{A}_i}$, for each predicate q in σ. The following simple lemma tells us that if a $\mathcal{GC}^2$-formula has a finite model, it has arbitrarily large finite models (and even infinite, in fact).

Lemma 4.20. Let φ be a $\mathcal{GC}^2$-formula and $\mathfrak{A}$ be a structure over the signature of φ, such that $\mathfrak{A} \models \varphi$. Then, for all positive integers k, $\mathfrak{A}^{\times k} \models \varphi$.

Proof. Immediate. (Guardedness is the key here.) □

For the rest of this section, we suppress (for convenience) all references to Lemma 4.20: if we assume $\mathfrak{A} \models \varphi$, we will take for granted that $\mathfrak{A}^{\times k} \models \varphi$, for any positive integer k.

Lemma 4.21. Suppose $\mathfrak{A} \models \varphi$ and that B, B' are disjoint subsets of A, such that $|B| \geq (mC)^2 + mC + 1$ and $|B'| \geq mC + 1$. Then, there exist elements $b \in B$ and $b' \in B'$ such that $\mathrm{tp}_{\mathfrak{A}}[b, b']$ is silent.

Proof. Pick any $B'_0 \subseteq B'$, such that $|B'_0| = mC + 1$. By our normal form, no more than $mC(mC + 1)$ elements in B can receive messages originating from B'_0. By the cardinality of B, let $b \in B$ be an element that receives no message from any element of B'_0. Now, there can be at most mC elements of B'_0 that receive a message from b. By the cardinality of B'_0, let $b' \in B'_0 \subseteq B'$ be an element that receives no message from b. Evidently, $\mathrm{tp}_{\mathfrak{A}}[b, b']$ is silent. □

Suppose $\mathfrak{A} \models \varphi$ and let $K \subseteq A$ be a set of distinguished elements. For any pair of elements $a, b \in A$ we say that b is *directly accessible* from a if either (i) $a = b$, (ii) $\mathrm{tp}_{\mathfrak{A}}[a, b]$ is a message-type, or (iii) a and b are both in K.[3] Further, for any $a, b \in A$, we say that b is *accessible in ℓ steps* from a if there exists a sequence of elements $a_0, \ldots, a_\ell$ of A such that $a_0 = a$, $a_\ell = b$, and for all i $(0 \leq i < \ell)$, a_{i+1} is directly accessible from a_i.

[3]In future, K will be taken to be the set of constants appearing inside a given database. For this reason, we may assume that each element of K is connected with (or accessible from) any other element of K—if not we can ensure this by interpreting a new predicate.

The following lemma tells us that, if $\mathfrak{A} \models \varphi$, by 'copying' $\mathfrak{A}$ enough times, we can be sure to find for any pair of elements $a, b \in A$ such that $\mathrm{tp}_{\mathfrak{A}}[a, b]$ is invertible, (i) an 'inaccessible', 'identical' with a, b pair of elements c, d, and (ii) two elements e, f whose 1-types are the same with a, b respectively, their 2-type is silent, and d does not send a message to e. Such elements c, d, e, f are used later to expand any strong cycle whose first two elements are a and b.

Lemma 4.22. Suppose $\mathfrak{A} \models \varphi$ and let $K \subseteq A$ be a set of distinguished elements in A. Let $\Omega \geq 4$ and $N \geq 2(|K| + 1)((mC)^{\Omega} - 1))/(mC - 1) + 2$ be two natural numbers, and consider the structure $\mathfrak{A}^{\times N}$ with domain $A^{\times N}$, as described above. Let $a, b \in A^{\times N}$, such that $\mathrm{tp}_{\mathfrak{A}^{\times N}}[a, b] = \mu$ is an invertible message-type. Then, there exist distinct elements $c, d, e, f \in A^{\times N} \setminus K$, such that:

(i) $\mathrm{tp}_{\mathfrak{A}^{\times N}}[c, d] = \mu$;

(ii) neither c nor d is accessible from either a or b in $\Omega - 2$ steps;

(iii) $\mathrm{tp}_{\mathfrak{A}^{\times N}}[e] = \mathrm{tp}_{\mathfrak{A}^{\times N}}[a]$ and $\mathrm{tp}_{\mathfrak{A}^{\times N}}[f] = \mathrm{tp}_{\mathfrak{A}^{\times N}}[b]$;

(iv) $\mathrm{tp}_{\mathfrak{A}^{\times N}}[e, f]$ is silent;

(v) $\mathrm{tp}_{\mathfrak{A}^{\times N}}[d, e]$ is not a message-type.

Proof. According to the normal form in Lemma 4.12, the number of accessible elements from either a or b in $\Omega - 1$ steps is bounded by

$$2(|K| + 1) \sum_{i=0}^{\Omega-1} (mC)^i = ((mC)^{\Omega} - 1))/(mC - 1) < N.$$

Then, by the construction of $\mathfrak{A}^{\times N}$, we can choose an element $c \in A^{\times N} \setminus K$ such that c sends a message of type μ, and c is not accessible from either a or b in $\Omega - 1$ steps. Let d be the 'recipient' of the message that c sends, i.e. $\mathrm{tp}_{\mathfrak{A}^{\times N}}[c, d] = \mu$. It follows that d is not accessible from a or b in $\Omega - 2$ steps.

For the rest of the proof, refer to Figure 4.1. Let E be the set of elements in $A^{\times N} \setminus K$ having the same 1-type as a, and let F be the set of elements in $A^{\times N} \setminus K$ having the same 1-type as b. It is clear by the construction of $\mathfrak{A}^{\times N}$, that $|E|, |F| \geq N$, from which, because $\Omega \geq 4$, it follows that

$$\begin{aligned} |E|, |F| &\geq 2((mC)^4 - 1)/(mC - 1) + 2 \\ &= 2((mC)^3 + (mC)^2 + mC + 2). \end{aligned}$$

Now, observe that

$$|E \setminus \{a,b,c,d\}| \;\geq\; |E| - 4 \;=\; 2mC((mC)^2 + mC + 1),$$

since, at worst, $E = F$ and $a,b,c,d \in E$. Similarly,

$$|F \setminus \{a,b,c,d\}| \;\geq\; 2mC((mC)^2 + mC + 1).$$

Thus, we can select subsets $E_1, \ldots, E_{mC}$ and $F_1, \ldots, F_{mC}$ of $E \setminus \{a,b,c,d\}$ and $F \setminus \{a,b,c,d\}$ respectively, all of which are pairwise disjoint and, for all i ($1 \leq i \leq mC$), $|E_i|, |F_i| \geq (mC)^2 + mC + 1$. (Note that this is possible even if $E = F$.) By Lemma 4.21, we can select for each i ($0 \leq i \leq mC$) a pair of elements $e_i \in E_i$ and $f_i \in F_i$ such that $\text{tp}_{\mathfrak{A}^{\times N}}[e_i, f_i]$ is silent.

Finally, notice (due to our normal form) that d cannot send more than mC messages. Because d already sends a message to c, it can send at most $mC - 1$ messages whose 'recipient' is inside one of the sets E_i ($1 \leq i \leq mC$). Thus, there exists an i ($1 \leq i \leq mC$) such that no element of E_i receives a message from d; hence, we can pick e to be the element $e_i \in E_i$ and f to be the corresponding element $f_i \in F_i$. (In Figure 4.1, for example, $e = e_1$ and $f = f_1$.) It is clear, then, that the elements c, d, e, and f have the required properties. □

In the following lemma we show how one can remove all 'short' strong cycles from a model of φ.

Lemma 4.23. Suppose $\mathfrak{A} \models \varphi$ and let $K \subseteq A$ be a set of distinguished elements in A; let Ω be a positive integer. We can find a model $\mathfrak{B} \models \varphi$, such that: (i) $K \subseteq B$, (ii) $\mathfrak{A}|_K = \mathfrak{B}|_K$, and (iii) $\mathfrak{B}$ contains no strong cycles of length less than Ω. Moreover, if $\mathfrak{A}$ is finite, we can ensure that $\mathfrak{B}$ is finite.

Proof. We assume without loss of generality that $\Omega \geq 4$ and $mC > 1$. Consider the structure $\mathfrak{A}^{\times N}$, for $N = 2(|K|+1)((mC)^\Omega - 1))/(mC - 1) + 2$. Now, suppose that $\gamma = a_0, a_1, \ldots, a_{\ell-1}\,(, a_\ell)$ is a strong cycle in $\mathfrak{A}^{\times N}$; note that the length of γ is ℓ. Assume, without loss of generality, that $a_0 \notin K$. We show how to destroy γ, ensuring that only new larger cycles are created in the process.

Let $a = a_0$, $b = a_1$, and $\mu = \text{tp}_{\mathfrak{A}^{\times N}}[a, b]$ (μ is an invertible message-type). Let c, d, e, and f be elements of $A^{\times N}$, having the properties guaranteed by Lemma 4.22. We wish to modify $\mathfrak{A}^{\times N}$ so as to ensure that the 2-type connecting

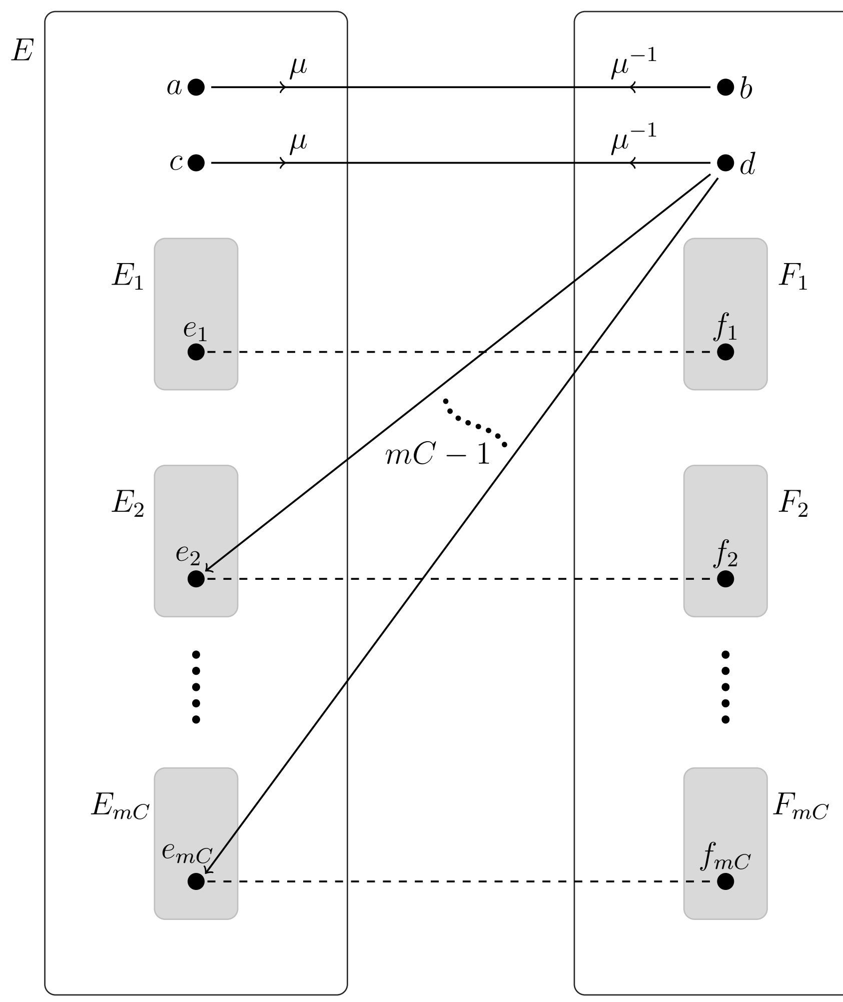


Figure 4.1: The sets E, F and (some of the) messages sent by their elements, used in the proof of Lemma 4.22. The lines with two arrows represent invertible message-types, the lines with one arrow represent non-invertible message-types, and the dashed lines represent silent 2-types.

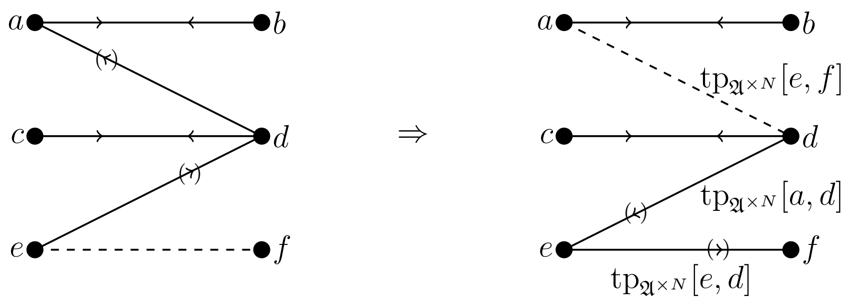

a
b
c
d
e
f
$\Rightarrow$
$\mathrm{tp}_{\mathfrak{A}^{\times N}}[e,f]$
$\mathrm{tp}_{\mathfrak{A}^{\times N}}[a,d]$
$\mathrm{tp}_{\mathfrak{A}^{\times N}}[e,d]$

Figure 4.2: The transformation of $\mathfrak{A}^{\times N}$ (left part) to $\mathfrak{B}'$ (right part), to ensure that the 2-type of a and d is silent. The lines with arrows and dashed lines are as in Figure 4.1. The arrows in parenthesis indicate a 2-type which may or may not be a message-type.

a and d is silent. To this end, we define a new structure $\mathfrak{B}'$, which the same as $\mathfrak{A}^{\times N}$, except that

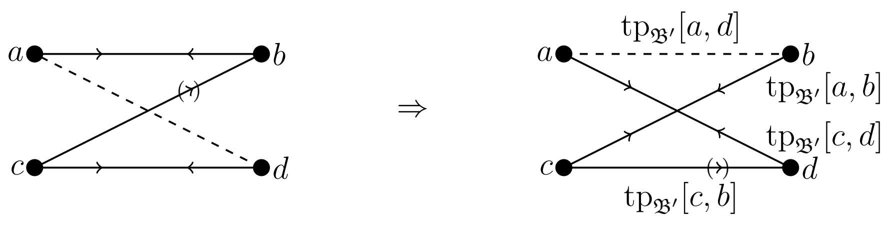
$$\begin{aligned}\mathrm{tp}_{\mathfrak{B}'}[a,d] &= \mathrm{tp}_{\mathfrak{A}^{\times N}}[e,f],\\ \mathrm{tp}_{\mathfrak{B}'}[e,d] &= \mathrm{tp}_{\mathfrak{A}^{\times N}}[a,d],\\ \mathrm{tp}_{\mathfrak{B}'}[e,f] &= \mathrm{tp}_{\mathfrak{A}^{\times N}}[e,d].\end{aligned}$$

This transformation is depicted in Figure 4.2. It is clear that $\mathfrak{B}' \models \varphi$, since the above assignments just rearrange 2-types, without changing the total number of messages sent (and received).

We now destroy the strong cycle γ in $\mathfrak{B}'$ by performing more rearrangements of 2-types, as before. Let $\mathfrak{B}$ be the same as $\mathfrak{B}'$, except that

$$\begin{aligned}\mathrm{tp}_{\mathfrak{B}}[a,b] &= \mathrm{tp}_{\mathfrak{B}'}[a,d],\\ \mathrm{tp}_{\mathfrak{B}}[a,d] &= \mathrm{tp}_{\mathfrak{B}'}[a,b],\\ \mathrm{tp}_{\mathfrak{B}}[c,b] &= \mathrm{tp}_{\mathfrak{B}'}[c,d],\\ \mathrm{tp}_{\mathfrak{B}}[c,d] &= \mathrm{tp}_{\mathfrak{B}'}[c,b].\end{aligned}$$

This transformation is depicted in Figure 4.3. Again, it is clear that $\mathfrak{B} \models \varphi$; in addition, any new strong cycle created as a result of the above is longer than γ. (See [PH09] for more details.) By repeating this process for every strong cycle, one eventually obtains the required result. □



a
b
c
d
(γ)
$\Rightarrow$
$\mathrm{tp}_{\mathfrak{B}'}[a,d]$
$\mathrm{tp}_{\mathfrak{B}'}[a,b]$
$\mathrm{tp}_{\mathfrak{B}'}[c,d]$
$\mathrm{tp}_{\mathfrak{B}'}[c,b]$

Figure 4.3: The transformation of $\mathfrak{B}'$ (left part) to $\mathfrak{B}$ (right part), destroying the strong cycle γ whose first two elements are a and b. The lines with arrows and dashed lines are as in Figures 4.1 and 4.2. The arrows in parenthesis indicate a 2-type which may or may not be a message-type.

In the following lemma, we show how to destroy any cycle (strong or otherwise). It is one of the most important tools for this thesis.

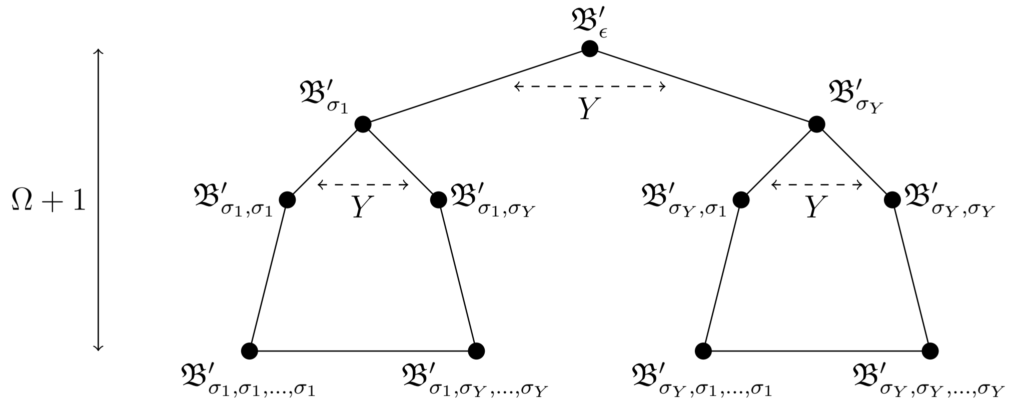
Lemma 4.24. Suppose $\mathfrak{A} \models \varphi$ and let $K \subseteq A$ be a set of distinguished elements in A; let Ω be a positive integer. We can find a model $\mathfrak{B} \models \varphi$, such that: (i) $K \subseteq B$, (ii) $\mathfrak{A}|_K = \mathfrak{B}|_K$, and (iii) $\mathfrak{B}$ contains no cycles of length less than Ω. Moreover, if $\mathfrak{A}$ is finite, we can ensure that $\mathfrak{B}$ is finite.

Proof. We only concern ourselves with finite models, but the proof generalizes to infinite models with minor adjustments. Let $\mathfrak{B}'$ be a finite model of φ, with the properties guaranteed by applying Lemma 4.23 to $\mathfrak{A}$. We create a tree of copies of $\mathfrak{B}'$ that allows us to 'divert' non-invertible messages within a copy of $\mathfrak{B}'$ to elements of another copy of $\mathfrak{B}'$, destroying cycles in the process. Let

$$S = \{(a,b) \in B' \times B' \mid a \neq b \text{ and } \mathrm{tp}_{\mathfrak{B}'}[a,b] \text{ is a non-inv. message-type}\},$$

and let $Y = |S|$. We define $S^{*\Omega}$ to be the set of sequences of elements of S of length $\leq \Omega$. We denote the length of any sequence $\sigma \in S^{*\Omega}$ by $\|\sigma\|$; ϵ denotes the empty sequence, and the concatenation of two sequences $\sigma, \tau \in S^{*\Omega}$ is written as σ, τ or simply $\sigma\tau$. For convenience, sequences of $S^{*\Omega}$ having length 1 are identified with the corresponding elements of S.

Let $\mathfrak{B}'_\epsilon = \mathfrak{B}'$, and, for each $\sigma \in S^{*\Omega} \setminus \{\epsilon\}$, let $\mathfrak{B}'_\sigma$ be a new copy of $\mathfrak{B}'$, with domain B'_σ; for any $a \in B'$, denote by a_σ the corresponding element of B'_σ. We assume that all the sets B'_σ in this definition are pairwise disjoint. Let $\mathfrak{B}''$ be the structure such that $\mathfrak{B}'' = \bigcup_{\sigma \in S^{*\Omega}} \mathfrak{B}'_\sigma$ and $q^{\mathfrak{B}''} = \bigcup_{\sigma \in S^{*\Omega}} q^{\mathfrak{B}'_\sigma}$, for each predicate q in the signature of φ. (K remains a subset of $B' = B'_\epsilon$.) It follows from Lemma 4.20 that $\mathfrak{B}'' \models \varphi$—in fact, $\mathfrak{B}''$ is $\mathfrak{B}'^{\times N}$ for $N = (Y^{\Omega+1} - 1)/(Y - 1)$. $\mathfrak{B}''$, pictured as a tree of height $\Omega + 1$, can be seen in Figure 4.4.



$\mathfrak{B}'_\epsilon$
$\mathfrak{B}'_{\sigma_1}$
$\mathfrak{B}'_{\sigma_Y}$
Y
$\Omega+1$
$\mathfrak{B}'_{\sigma_1,\sigma_1}$
Y
$\mathfrak{B}'_{\sigma_1,\sigma_Y}$
$\mathfrak{B}'_{\sigma_Y,\sigma_1}$
Y
$\mathfrak{B}'_{\sigma_Y,\sigma_Y}$
$\mathfrak{B}'_{\sigma_1,\sigma_1,\dots,\sigma_1}$
$\mathfrak{B}'_{\sigma_1,\sigma_Y,\dots,\sigma_Y}$
$\mathfrak{B}'_{\sigma_Y,\sigma_1,\dots,\sigma_1}$
$\mathfrak{B}'_{\sigma_Y,\sigma_Y,\dots,\sigma_Y}$

Figure 4.4: $\mathfrak{B}''$ pictured as a tree. The members of S are numbered, for convenience, as $\sigma_1, \dots, \sigma_Y$.

We now modify the structure $\mathfrak{B}''$ to obtain the required $\mathfrak{B}$. As a simple preprocessing step, we replace (by Lemma 4.16) every silent 2-type connecting two elements a and b, not both in K, with the corresponding vacuous type $\text{tp}_{\mathfrak{A}}[a] \times \text{tp}_{\mathfrak{A}}[b]$. Hence, we may assume that all cycles in $\mathfrak{B}''$ consist of message types (except for their parts inside K). Recall that, by Lemma 4.23, $\mathfrak{B}''$ does not contain any strong cycles of length $< \Omega$; thus, no cycle in $\mathfrak{B}''$ of length $< \Omega$ consists solely of invertible message-types.

Let $a, b \in B'_\epsilon$ be distinct and not both in K. If $\text{tp}_{\mathfrak{B}''}[a, b]$ is a non-invertible message-type, we redirect the message from b to the corresponding element $b_\sigma \in B'_\sigma$, where $\sigma = (a, b)$, at the second tier of the tree. Formally, we set

$$\begin{aligned} \text{tp}_{\mathfrak{B}}[a, b] &= \text{tp}_{\mathfrak{B}''}[a] \times \text{tp}_{\mathfrak{B}''}[b], \\ \text{tp}_{\mathfrak{B}}[a, b_{(a,b)}] &= \text{tp}_{\mathfrak{B}''}[a, b]. \end{aligned}$$

We then move to the second tier of the tree. Let $\sigma \in S^{*\Omega}$ with $\|\sigma\| = 1$, and let $a_\sigma, b_\sigma \in B'_\sigma$ be distinct. If $\text{tp}_{\mathfrak{B}''}[a_\sigma, b_\sigma]$ is a non-invertible message-type, we redirect the message from b_σ to the corresponding element $b_{\sigma'} \in B'_{\sigma'}$, where $\sigma' = \sigma\,(a, b)$, at the third tier of the tree. Formally, we set

$$\begin{aligned} \text{tp}_{\mathfrak{B}}[a_\sigma, b_\sigma] &= \text{tp}_{\mathfrak{B}''}[a_\sigma] \times \text{tp}_{\mathfrak{B}''}[b_\sigma], \\ \text{tp}_{\mathfrak{B}}[a_\sigma, b_{\sigma\,(a,b)}] &= \text{tp}_{\mathfrak{B}''}[a_\sigma, b_\sigma]. \end{aligned}$$

After performing the above reassignments, we move to the third tier, fourth tier, and so forth until we reach the bottom tier.

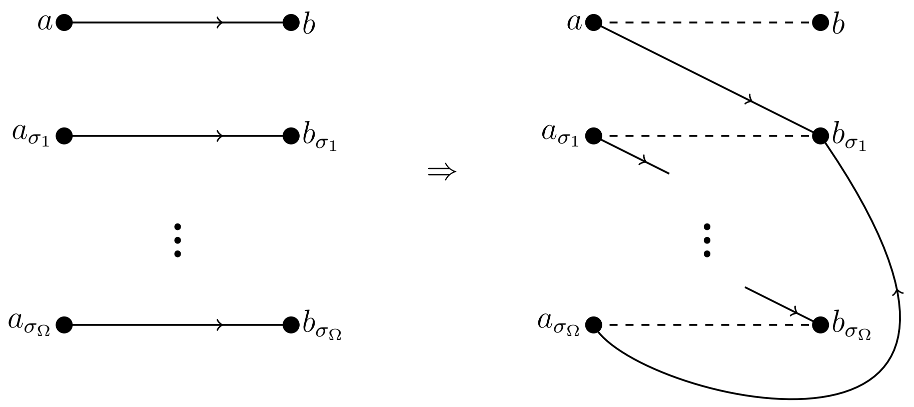


Figure 4.5: The whole series of the transformations described in the proof of Lemma 4.24, for a given pair of elements a and b, not both in K. The dashed lines represent vacuous 2-types.

For the bottom tier, let $\sigma \in S^{*\Omega}$ with $\|\sigma\| = \Omega$, and let $a_\sigma, b_\sigma \in B'_\sigma$ be distinct. If $\mathrm{tp}_{\mathfrak{B}''}[a_\sigma, b_\sigma]$ is a non-invertible message-type, we redirect the message from b_σ to the corresponding element $b_{(a,b)} \in B'_{(a,b)}$, at the *second* tier of the tree. Formally, we set

$$\begin{aligned}
\mathrm{tp}_{\mathfrak{B}}[a_\sigma, b_\sigma] &= \mathrm{tp}_{\mathfrak{B}''}[a_\sigma] \times \mathrm{tp}_{\mathfrak{B}''}[b_\sigma], \\
\mathrm{tp}_{\mathfrak{B}}[a_\sigma, b_{(a,b)}] &= \mathrm{tp}_{\mathfrak{B}''}[a_\sigma, b_\sigma].
\end{aligned}$$

On completion, it is clear that $\mathfrak{B} \models \varphi$, since the numbers of messages have not changed—only their 'recipients'. In addition, there cannot be any cycles of length $< \Omega$ in $\mathfrak{B}$, and the other requirements of the lemma are also satisfied. See Figure 4.5 for an illustration. □

Remark 4.25. When we use Lemma 4.24 in the presence of a database, we always take K to be the active domain of this database, and we never explicitly mention this for the sake of brevity.

4.3 Relation to description logics

We now provide a translation from the description logic $\mathcal{ALCQI}$ to $\mathcal{GC}^2$, which will allow us to use the fact that query entailment for $\mathcal{ALCQI}$ is 2-EXPTIME-complete [Lut08] to obtain a lower bound for the same problem with respect to $\mathcal{GC}^2$ with a database (and path-functional dependencies).

The translation from $\mathcal{ALCQI}$ to $\mathcal{GC}^2$ is done with the following two simultaneous recursive functions (A is an atomic concept):

$$\begin{aligned}
\mathsf{t}_x(\mathsf{A}) &= A(x) & \mathsf{t}_y(\mathsf{A}) &= A(y)\\
\mathsf{t}_x(\neg\mathsf{C}) &= \neg\mathsf{t}_x(\mathsf{C}) & \mathsf{t}_y(\neg\mathsf{C}) &= \neg\mathsf{t}_y(\mathsf{C})\\
\mathsf{t}_x(\mathsf{C}\sqcup\mathsf{D}) &= \mathsf{t}_x(\mathsf{C})\vee\mathsf{t}_x(\mathsf{D}) & \mathsf{t}_y(\mathsf{C}\sqcup\mathsf{D}) &= \mathsf{t}_y(\mathsf{C})\vee\mathsf{t}_y(\mathsf{D})\\
\mathsf{t}_x(\mathsf{C}\sqcap\mathsf{D}) &= \mathsf{t}_x(\mathsf{C})\wedge\mathsf{t}_x(\mathsf{D}) & \mathsf{t}_y(\mathsf{C}\sqcap\mathsf{D}) &= \mathsf{t}_y(\mathsf{C})\wedge\mathsf{t}_y(\mathsf{D})\\
\mathsf{t}_x(\exists\mathsf{r}.\mathsf{C}) &= \exists y\, r(x,y)\wedge\mathsf{t}_y(\mathsf{C}) & \mathsf{t}_y(\exists\mathsf{r}.\mathsf{C}) &= \exists y\, r(x,y)\wedge\mathsf{t}_x(\mathsf{C})\\
\mathsf{t}_x(\exists\mathsf{r}^-.\mathsf{C}) &= \exists y\, r(y,x)\wedge\mathsf{t}_y(\mathsf{C}) & \mathsf{t}_y(\exists\mathsf{r}^-.\mathsf{C}) &= \exists y\, r(y,x)\wedge\mathsf{t}_x(\mathsf{C})\\
\mathsf{t}_x(\forall\mathsf{r}.\mathsf{C}) &= \forall y\, r(x,y)\rightarrow\mathsf{t}_y(\mathsf{C}) & \mathsf{t}_y(\forall\mathsf{r}.\mathsf{C}) &= \forall y\, r(x,y)\rightarrow\mathsf{t}_x(\mathsf{C})\\
\mathsf{t}_x(\forall\mathsf{r}^-.\mathsf{C}) &= \forall y\, r(y,x)\rightarrow\mathsf{t}_y(\mathsf{C}) & \mathsf{t}_y(\forall\mathsf{r}^-.\mathsf{C}) &= \forall y\, r(y,x)\rightarrow\mathsf{t}_x(\mathsf{C})\\
\mathsf{t}_x(\leqslant n\,\mathsf{r}.\mathsf{C}) &= \exists_{\leq n} y\, r(x,y)\wedge\mathsf{t}_y(\mathsf{C}) & \mathsf{t}_y(\leqslant n\,\mathsf{r}.\mathsf{C}) &= \exists_{\leq n} y\, r(x,y)\wedge\mathsf{t}_x(\mathsf{C})\\
\mathsf{t}_x(\geqslant n\,\mathsf{r}.\mathsf{C}) &= \exists_{\geq n} y\, r(x,y)\wedge\mathsf{t}_y(\mathsf{C}) & \mathsf{t}_y(\geqslant n\,\mathsf{r}.\mathsf{C}) &= \exists_{\geq n} y\, r(x,y)\wedge\mathsf{t}_x(\mathsf{C})
\end{aligned}$$

With the above, we can translate a given TBox $\mathcal{T}$ and a given ABox $\mathcal{A}$ as follows (we denote by $\psi[y/x]$ the result of replacing all occurrences of x in ψ by y):

$$\begin{aligned}
\mathsf{t}(\mathcal{T}) &= \bigwedge_{\mathsf{C}\sqsubseteq\mathsf{D}\in\mathcal{T}} \forall x(\mathsf{t}_x(\mathsf{C})\rightarrow\mathsf{t}_x(\mathsf{D})),\\
\mathsf{t}(\mathcal{A}) &= \bigwedge_{\mathsf{a}:\mathsf{C}\in\mathcal{A}} \mathsf{t}_x(\mathsf{C})[a/x] \;\wedge \bigwedge_{(\mathsf{a},\mathsf{b}):\mathsf{r}\in\mathcal{A}} r(a,b).
\end{aligned}$$

For a given knowledge base $\mathcal{K}=(\mathcal{T},\mathcal{A})$, it is clear that the query entailment problem $\mathcal{K}\models q$ is equivalent to the query entailment problem $\Delta,\varphi\models q'$ (of $\mathcal{GC}^2$), where $\Delta=\mathsf{t}(\mathcal{A})$, $\varphi=\mathsf{t}(\mathcal{T})$, and $q'=\exists\bar{v}q$, for $\bar{v}=\mathsf{Var}(q)$.

5 | $\mathcal{GC}^2$ with a Database

In this chapter we establish that (finite) satisfiability for $\mathcal{GC}^2$ with a database, denoted $\mathcal{GC}^2\mathcal{D}$, is EXPTIME-complete. We do that by adapting the techniques used in [PH07], which establishes the same bound for $\mathcal{GC}^2$ (without a database). We reuse a lot of the material in [PH07], making modifications and adding extensions so as to accommodate the presence of a database. In particular, for a given database Δ and a background theory φ, we provide a set of additional inequalities that guarantee (and enforce) the availability of elements that realize the constants in Δ, in any model of Δ, φ (assuming one exists). The hard part is to write exactly the necessary number of inequalities—the naive approach leads to a doubly exponential number. We then show that these inequalities have the desired effect, i.e. that if there is a (finite) model of Δ, φ, then the system of the original equations/inequalities in [PH07] together with our new inequalities has a solution and, conversely, if there is a solution, then there is a (finite) model of Δ, φ.

5.1 Preliminaries

Definition 5.1. Fix a relational signature σ. A *database* is a set (conjunction) of ground (function-free) literals over the signature σ. A database Δ is *consistent* if for any tuple $\bar{a}$ of constants and any predicate symbol r in σ, $\{r(\bar{a}), \neg r(\bar{a})\} \nsubseteq \Delta$. A database Δ is *complete* if for any σ-literal $l(\bar{a}) \notin \Delta$ involving a tuple $\bar{a}$ of constants in σ, $\Delta \cup \{l(\bar{a})\}$ is inconsistent. Given a database Δ, a *completion* for Δ is any complete set $\Delta^* \supseteq \Delta$ of ground (function-free) literals in the same signature. The set of constants that appear in a database is called the *active domain* of the database.

Since we will be working with two-variable fragments, we may assume that the atoms that appear inside a given database are unary or binary. This does not have any effect in the type of relations that we can store in a database, since relations of arbitrary arities can be easily 'emulated' in this model. For example, a tuple $t = (\mathsf{Bob}, \mathsf{Wright}, \mathsf{Essex})$ in the relation $\mathsf{student}(\mathsf{name}, \mathsf{surname}, \mathsf{city})$ corresponds to an entry $\{\mathsf{student}(t), \mathsf{name}(t, \mathsf{Bob}), \mathsf{surname}(t, \mathsf{Wright}), \mathsf{city}(t, \mathsf{Essex})\}$.

Remark 5.2. For the rest of the thesis we adopt the *unique name assumption*, i.e. we assume that distinct constants are interpreted by distinct elements of a given universe. This assumption is very common in the setting of description logics (see, e.g., [Baa03]), though it is not made in OWL [G$^+$09].

Remark 5.3. All the elements in a database Δ are assumed to be neigbours, in the sense that for any two elements $a, b \in D$, where D is the active domain of Δ, there is an atomic predicate p such that either $p(a, b)$ or $p(b, a)$ is in Δ. For, if that is not the case, we can introduce a new binary predicate q and extend the database with the statements $\{q(a, b) \mid a, b \in D\}$.

We slightly abuse the notion of a k-type—which is defined with respect to a structure $\mathfrak{A}$, in Definition 4.2—to talk about 1- and 2-types of elements in a database Δ. Assuming Δ is complete, for each constant c in its active domain, the 1-type of c in Δ, denoted $\mathrm{tp}_\Delta[c]$, is the set $\mathrm{tp}_\Delta[c]$ containing all the unary predicates (say in the variable x) that hold for c in Δ (including $c(x)$). The definition of the 2-type for two constants $c_1, c_2 \in \Delta$, denoted $\mathrm{tp}_\Delta[c_1, c_2]$, is analogous.

For the rest of this chapter we fix a relational signature σ, a sentence φ over σ in the normal form of Lemma 4.12, and a database Δ over σ; we denote the active domain of Δ by D. As in the previous chapter, we let, referring to the normal form of φ, $C = \max_{1 \le i \le m}\{C_i\}$. For convenience, we often equivocate between constants and their denotations. Thus, depending on context, D may also denote the set of elements in a model that interpret the constants in Δ. Further, we often speak of the satisfiability of $\Delta, \psi_0, \ldots, \psi_{k-1}$ (where $\psi_0, \ldots, \psi_{k-1}$ are arbitrary $\mathcal{GC}^2$-sentences), when formally we mean the satisfiability of $\bigwedge(\Delta \cup \{\psi_0, \ldots, \psi_{k-1}\})$.

Let $\Pi = \pi_0, \ldots, \pi_{P-1}$ be an enumeration of the 1-types over σ. It is clear that P is a power of 2, thus $p = \log P$ is an integer. For reasons that will become clear later, we will need to index the previous sequence using bit-strings as follows: Let ϵ denote the empty string and define $\Pi_\epsilon = \Pi$ to be the whole sequence. Now, let $\mathtt{s}$ be a bit-string such that $0 \le |\mathtt{s}| < p$ and let $\Pi_\mathtt{s} = \pi_j, \ldots, \pi_{k-1}$ be the sub-sequence of Π indexed by $\mathtt{s}$. Define (recursively) $\Pi_{\mathtt{s0}}$ and $\Pi_{\mathtt{s1}}$ to be the first and second halves respectively of the sub-sequence indexed by $\mathtt{s}$; that is,

$$\Pi_{\mathtt{s0}} = \pi_j, \ldots, \pi_{(j+k)/2-1} \quad \text{and} \quad \Pi_{\mathtt{s1}} = \pi_{(j+k)/2}, \ldots, \pi_{k-1}.$$

When $|\mathtt{s}| = p$, it is clear that $\Pi_{\mathtt{s}}$ corresponds to one exactly 1-type π_j. Sometimes for convenience we will denote this type by $\pi_{\mathtt{s}}$. Note that, in this case, $\mathtt{s}$ is the binary representation of the subscript j. Moreover, we sometimes use the notation $\pi \in \Pi_{\mathtt{s}}$ meaning that the 1-type π is in the sequence indexed by $\mathtt{s}$.

We now explain how to index 2-types in a similar manner. We first fix some notation for convenience. From now on, we will use the letter $\mathtt{s}$ to denote an arbitrary bit-string indexing a sequence of invertible message-types and $\mathtt{t}$ similarly for non-invertible message-types or silent 2-types.

We index (sequences of) invertible message-types according to their terminal 1-types using bit-strings as follows: Let Λ be the set of all invertible message-types (over σ). Fix any 1-type π, let $\mathtt{s}$ be any bit-string with $0 \leq |\mathtt{s}| \leq p$, and define

$$\Lambda_{\pi,\mathtt{s}} = \{\lambda \in \Lambda \mid \mathrm{tp}_1(\lambda) = \pi \text{ and } \mathrm{tp}_2(\lambda) \in \Pi_{\mathtt{s}}\},$$

i.e. $\Lambda_{\pi,\mathtt{s}}$ is the set of all invertible message-types 'starting' from an element of 1-type π and 'ending' on an element of 1-type indexed by $\mathtt{s}$.

Remark 5.4. Each of the sets $\Lambda_{\pi,\mathtt{s}}$ will usually contain more than one 2-type. This is true even when $|\mathtt{s}| = p$, as there are several ways one could 'connect' an element of 1-type π with another element of (possibly different) 1-type π'. Recall, however, that we have restricted our attention to *chromatic* models. In that case (by definition) we are guaranteed that no element sends an invertible message-type to an element of the same 1-type and any element a of 1-type π can send an invertible message to no more than one element b of type π' ($\neq \pi$). Thus, when $|\mathtt{s}| = p$, there can be at most one element $b \in A \setminus \{a\}$ such that $\mathrm{tp}_{\mathfrak{A}}[a, b] \in \Lambda_{\pi,\mathtt{s}}$.

In a similar way, we use bit-strings to index sequences of 2-types that are not invertible message-types. Fix any 1-type π and let

$$M_\pi \; = \; \mu_{\pi,0}, \ldots, \mu_{\pi,Q-1}$$

be an enumeration of all 2-types τ with $\mathrm{tp}_1(\tau) = \pi$ that are either non-invertible message-types or silent 2-types. In other words, M_π is an enumeration of all 2-types τ with $\mathrm{tp}_1(\tau) = \pi$ such that τ^{-1} is not a message-type. It follows, then, that Q is a power of 2, thus $q = \log Q$ is an integer. Now, define $M_{\pi,\epsilon} = M$, and for any bit-string $\mathtt{t}$ with $|\mathtt{t}| < q$, recursively define

$$M_{\pi,\mathtt{t0}} = \mu_j, \ldots, \mu_{(j+k)/2-1} \quad \text{and} \quad M_{\pi,\mathtt{t1}} = \mu_{(j+k)/2}, \ldots, \mu_{k-1}.$$

Note that if $|\mathtt{t}| = q$, then $M_{\pi,\mathtt{t}}$ contains a single 2-type $\mu_{\pi,j}$, which we often write as $\mu_{\pi,\mathtt{t}}$ for convenience.

5.2 Spectra and tallies

To motivate the introduction of spectra and tallies we now give an overview of our approach. Our goal is to provide an exponential-time algorithm for the finite satisfiability of $\mathcal{GC}^2$ with a database. We do that by 'translating' our input into a system of exponentially many linear equations/inequalities that specify how often certain (local) configurational properties are realized in a model. These configurational properties are characterized in terms of spectra and tallies. We then show how to 'recover' a model given only the solutions to that system.

By *vector* we understand an m-dimensional vector over $\mathbb{N}$. (Recall the normal form of Lemma 4.12; m is defined there.) We denote the vector $(C_1, \ldots, C_m)$ by $\boldsymbol{C}$ and the m-dimensional zero vector by $\mathbf{0}$. For any two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, we write $\boldsymbol{v} \leq \boldsymbol{w}$ if every component of $\boldsymbol{v}$ is less than or equal to the corresponding component of $\boldsymbol{w}$; similarly for $<$, $\geq$ and $>$. For the rest of this chapter, we fix $C = \max_{1 \leq i \leq m}\{C_i\}$. The number of vectors $\boldsymbol{u}$ such that $\boldsymbol{u} \leq \boldsymbol{C}$ is evidently bounded by $(C+1)^m$, and thus by an exponential function of $\|\varphi\|$.

Definition 5.5. Let $\mathfrak{A}$ be a structure interpreting σ and let $a \in A$ be an element of 1-type π, i.e. $\pi = \mathrm{tp}_{\mathfrak{A}}[a]$. Let $\mathtt{s}$ be any bit-string of length at most p and define the $\mathtt{s}$-*spectrum* of a, denoted by $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a]$, to be the m-element vector $\boldsymbol{v} = (v_1, \ldots, v_m)$ where, for $1 \leq i \leq m$,

$$v_i = |\{b \in A \setminus \{a\} : \mathfrak{A} \models f_i(a,b) \text{ and } \mathrm{tp}_{\mathfrak{A}}[a,b] \in \Lambda_{\pi,\mathtt{s}}\}|.$$

Definition 5.6. Let $\mathfrak{A}$ be a structure interpreting σ and let $a \in A$ be an element of 1-type π. Let $\mathtt{t}$ be any bit-string with $|\mathtt{t}| < q$ and define the $\mathtt{t}$-*tally* of a, denoted by $\mathrm{tl}^{\mathfrak{A}}_{\mathtt{s}}[a]$, to be the m-element vector $\boldsymbol{w} = (w_1, \ldots, w_m)$ where, for

$1 \le i \le m$,

$$w_i = |\{b \in A \setminus \{a\} : \mathfrak{A} \models f_i(a,b) \text{ and } \mathrm{tp}_{\mathfrak{A}}[a,b] \in M_{\pi,\mathtt{t}}\}|.$$

Informally, $\mathrm{sp}^{\mathfrak{A}}_{\epsilon}[a]$ records the number of outgoing f_i arrows ($1 \le i \le m$) from a that are part of an invertible message-type. The same applies to $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a]$, but here we only count f_i arrows that are part of invertible message-types in $\Lambda_{\pi,\mathtt{s}}$. Tallies are defined in a similar way, but with respect to non-invertible message-types.

Let $\mathtt{s}$ be any bit-string with $|\mathtt{s}| < p$ and fix a 1-type π. For a given structure $\mathfrak{A}$, each vector $\boldsymbol{v}$ specifies a set of elements of $\mathfrak{A}$, i.e. the set of elements of type π that have $\mathtt{s}$-spectrum $\boldsymbol{v}$. The following lemma encapsulates the observation that this set can also be characterized as the union of sets of elements of type π whose $\mathtt{s0}$- and $\mathtt{s1}$-spectra add up to $\boldsymbol{v}$. The same idea applies to tallies. This provides an important intuition, used in Section 5.3 when transforming our formula to a system of equations/inequalities.

Lemma 5.7. Let $\mathfrak{A}$ be a model of Δ, φ and let $a \in A$ with $\mathrm{tp}_{\mathfrak{A}}[a] = \pi$. Let $\mathtt{s}$, $\mathtt{t}$ be any bit-strings with $|\mathtt{s}| < p$ and $|\mathtt{t}| < q$. Then, the following equations hold:

$$\mathrm{sp}^{\mathfrak{A}}_{\epsilon}[a] + \mathrm{tl}^{\mathfrak{A}}_{\epsilon}[a] = \boldsymbol{C} \tag{5.1}$$

$$\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s0}}[a] + \mathrm{sp}^{\mathfrak{A}}_{\mathtt{s1}}[a] = \mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] \tag{5.2}$$

$$\mathrm{tl}^{\mathfrak{A}}_{\mathtt{t0}}[a] + \mathrm{tl}^{\mathfrak{A}}_{\mathtt{t1}}[a] = \mathrm{tl}^{\mathfrak{A}}_{\mathtt{t}}[a] \tag{5.3}$$

Proof. Equation (5.1) is immediate from the definition of spectra and tallies and the normal form in Lemma 4.12. For Equation (5.2), notice that the set $\Lambda_{\pi,\mathtt{s}}$ can be partitioned into two subsets $\Lambda_{\pi,\mathtt{s0}}$ and $\Lambda_{\pi,\mathtt{s1}}$, and this induces a partition of the outgoing f_i arrows from a ($1 \le i \le m$); Equation (5.2) is then evident. Likewise for Equation (5.3). □

Let τ be any 2-type. With τ we associate an m-dimensional vector $\boldsymbol{C}_\tau$, whose ith component is given by

$$(\boldsymbol{C}_\tau)_i = \begin{cases} 1 & \text{if } f_i(x,y) \in \tau, \\ 0 & \text{otherwise.} \end{cases}$$

Using this notation, some interesting observations can be made. Let $\mathtt{t}$ be a bit-string with $|\mathtt{t}| = q$ and let $\mathfrak{A}$ be a structure and $a \in A$. Now, consider the (only) 2-type $\mu_{\pi,\mathtt{t}}$ in $M_{\pi,\mathtt{t}}$ and, if $\mu_{\pi,\mathtt{t}}$ is non-silent, let n be the number of messages of type $\mu_{\pi,\mathtt{t}}$ sent by a, i.e. n is the number of elements $b \in A \setminus \{a\}$ such that $\mathrm{tp}_{\mathfrak{A}}[a,b] = \mu_{\pi,\mathtt{t}}$. It is clear, then, that $\mathrm{tl}^{\mathfrak{A}}_{\mathtt{t}}[a] = n\boldsymbol{C}_{\mu_{\pi,\mathtt{t}}}$. On the other hand, if $\mu_{\pi,\mathtt{t}}$ is silent, we have $\mathrm{tl}^{\mathfrak{A}}_{\mathtt{t}}[a] = \boldsymbol{C}_{\mu_{\pi,\mathtt{t}}} = \mathbf{0}$.

In the case of $\mathtt{s}$-spectra with $|\mathtt{s}| = p$, in a *chromatic* model $\mathfrak{A}$, each element $a \in A$ with non-zero $\mathtt{s}$-spectrum induces a (unique) vector $\boldsymbol{C}_\lambda$, where λ is the invertible 2-type indexed by $\mathtt{s}$. Recall by Remark 5.4 that, given that $\mathfrak{A}$ is chromatic, no element may send more than one message whose type is in $\Lambda_{\pi,\mathtt{s}}$.

Lemma 5.8. Let $\mathfrak{A}$ be a chromatic model of φ. Let $a \in A$, π be a 1-type and $\mathtt{s}$ be a bit-string with $|\mathtt{s}| = p$. If $\mathrm{tp}_{\mathfrak{A}}[a] = \pi$ and $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] \neq \mathbf{0}$, then there exists $\lambda \in \Lambda_{\pi,\mathtt{s}}$ with $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] = \boldsymbol{C}_\lambda$ such that a sends a message of type λ to some $b \in A \setminus \{a\}$. Conversely, if there exists $\lambda \in \Lambda_{\pi,\mathtt{s}}$ such that a sends a message of type λ to some $b \in A \setminus \{a\}$, then $\mathrm{tp}_{\mathfrak{A}}[a] = \pi$ and $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] = \boldsymbol{C}_\lambda$.

Proof. Suppose $\mathrm{tp}_{\mathfrak{A}}[a] = \pi$ and $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] \neq \mathbf{0}$. As discussed previously, there exists a unique $b \in A \setminus \{a\}$ such that $\lambda = \mathrm{tp}_{\mathfrak{A}}[a,b] \in \Lambda_{\pi,\mathtt{s}}$. Clearly, then, $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] = \boldsymbol{C}_\lambda$. Conversely, suppose a sends a message of type $\lambda \in \Lambda_{\pi,\mathtt{s}}$ to some element $b \in A \setminus \{a\}$. Evidently, then, $\mathrm{tp}_{\mathfrak{A}}[a] = \pi$ and b is unique, thus $\mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] = \boldsymbol{C}_\lambda$. □

5.3 Translation to linear programming

We now show how to reduce the question for the (finite) satisfiability of Δ, φ to the problem of determining whether a certain system of linear equations/inequalities has a solution over $\mathbb{N}$. The solutions of these equations count how often various local configurations appear in a model. These configurations are:

- Realizations of each invertible message-type λ.
- Elements of 1-type π having $\mathtt{s}$-spectrum $\boldsymbol{u}$, for all vectors $\boldsymbol{u} \leq \boldsymbol{C}$.
- Elements of 1-type π having $\mathtt{t}$-tally $\boldsymbol{u}$, for all vectors $\boldsymbol{u} \leq \boldsymbol{C}$.
- Elements of 1-type π whose $\mathtt{s}$-spectrum $\boldsymbol{u}$ is obtained as the sum of an $\mathtt{s0}$-spectrum $\boldsymbol{v}$ and an $\mathtt{s1}$-spectrum $\boldsymbol{w}$, for $\boldsymbol{v}, \boldsymbol{u}, \boldsymbol{w} \leq \boldsymbol{C}$ and for all $\mathtt{s}$ with $|\mathtt{s}| < p$.
- Elements of 1-type π whose $\mathtt{t}$-tally $\boldsymbol{u}$ is obtained as the sum of a $\mathtt{t0}$-tally $\boldsymbol{v}$ and a $\mathtt{t1}$-tally $\boldsymbol{w}$, for $\boldsymbol{v}, \boldsymbol{u}, \boldsymbol{w} \leq \boldsymbol{C}$ and for all $\mathtt{t}$ with $|\mathtt{t}| < q$.

To each of those configurations we associate a variable which is intended to capture how many times it appears in a model. These variables and the properties

Variable	Intended meaning of its value
x_λ	$\lvert\{a \in A : \text{for some } b \in A \setminus \{a\},\ \mathrm{tp}_{\mathfrak{A}}[a,b] = \lambda\}\rvert$
$y_{\pi,\mathtt{s},\boldsymbol{u}}$	$\lvert\{a \in A_\pi : \mathrm{sp}^{\mathfrak{A}}_{\mathtt{s}}[a] = \boldsymbol{u}\}\rvert$
$z_{\pi,\mathtt{t},\boldsymbol{u}}$	$\lvert\{a \in A_\pi : \mathrm{tl}^{\mathfrak{A}}_{\mathtt{s}}[a] = \boldsymbol{u}\}\rvert$
$\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}}$	$\lvert\{a \in A_\pi : \mathrm{sp}^{\mathfrak{A}}_{\mathtt{s0}}[a] = \boldsymbol{v} \text{ and } \mathrm{sp}^{\mathfrak{A}}_{\mathtt{s1}}[a] = \boldsymbol{w}, \text{ whenever } \lvert\mathtt{s}\rvert < p\}\rvert$
$\hat{z}_{\pi,\mathtt{t},\boldsymbol{v},\boldsymbol{w}}$	$\lvert\{a \in A_\pi : \mathrm{tl}^{\mathfrak{A}}_{\mathtt{t0}}[a] = \boldsymbol{v} \text{ and } \mathrm{tl}^{\mathfrak{A}}_{\mathtt{t1}}[a] = \boldsymbol{w}, \text{ whenever } \lvert\mathtt{t}\rvert < q\}\rvert$

Table 5.1: Variables and their intended meanings, for a finite model $\mathfrak{A}$ of $\Delta \cup \{\varphi\}$. Recall, $A_\pi = \{a \in A \mid \mathrm{tp}_{\mathfrak{A}}[a] = \pi\}$.

that they capture can be seen in Table 5.1. We remark here that these are not single variables but *classes* of variables. Unless specified otherwise, the ranges of the subscripts of these variables are as follows: π ranges over all 1-types in Π, λ ranges over all invertible message-types, $\mathtt{s}$ ranges over all bit-strings with $|\mathtt{s}| \leq p$, $\mathtt{t}$ ranges over all bit-strings with $|\mathtt{t}| \leq q$, and $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ vary over all vectors $\leq \boldsymbol{C}$. (Similarly for their primed counterparts $\pi', \lambda', \mathtt{s}', \mathtt{t}', \boldsymbol{u}', \boldsymbol{v}'$ and $\boldsymbol{w}'$.) That is, we have one variable x_λ for each invertible message-type λ, one variable $y_{\pi,\mathtt{s},\boldsymbol{u}}$ for each possible combination of $\pi \in \Pi$, $\mathtt{s}$ with $|\mathtt{s}| \leq p$ and $\boldsymbol{u} \leq \boldsymbol{C}$, and so on.

Henceforth, given a structure $\mathfrak{A}$, we will denote the set of elements in the universe A of $\mathfrak{A}$ having 1-type π by A_π, that is

$$A_\pi = \{a \in A \mid \mathrm{tp}_{\mathfrak{A}}[a] = \pi\}.$$

We now write several equations/inequalities that a given structure $\mathfrak{A}$ has to satisfy for it to be a model of $\Delta \cup \{\varphi\}$. For convenience we provide these constraints in four stages: the first three capture the requirements that φ imposes and the fourth one ensures that the database is satisfied. Note, again, that each of the following equations/inequalities represents a *class* of equations/inequalities for different values of $\mathtt{s}$, $\mathtt{t}$, $\boldsymbol{v}$, $\boldsymbol{u}$, etc.

Let $\mathcal{E}_1$ be the following classes of constraints, where π, $\boldsymbol{u}$, etc., vary as described previously, and for the bit-strings $\mathtt{s}$ and $\mathtt{t}$ we also require that $|\mathtt{s}| < p$ and $|\mathtt{t}| < q$:

$$y_{\pi,\epsilon,\boldsymbol{u}} = z_{\pi,\epsilon,\boldsymbol{C}-\boldsymbol{u}} \tag{5.4}$$

$$y_{\pi,\mathtt{s},\boldsymbol{u}} = \sum\{\hat{y}_{\pi,\mathtt{s},\boldsymbol{v}',\boldsymbol{w}'} \mid \boldsymbol{v}' + \boldsymbol{w}' = \boldsymbol{u}\} \tag{5.5}$$

$$\begin{aligned}
z_{\pi,\mathtt{t},\boldsymbol{u}} &= \sum\{\hat{z}_{\pi,\mathtt{t},\boldsymbol{v}',\boldsymbol{w}'} \mid \boldsymbol{v}' + \boldsymbol{w}' = \boldsymbol{u}\} && (5.6)\\
y_{\pi,\mathtt{s0},\boldsymbol{v}} &= \sum\{\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}'} \mid \boldsymbol{v} + \boldsymbol{w}' \leq \boldsymbol{C}\} && (5.7)\\
y_{\pi,\mathtt{s1},\boldsymbol{w}} &= \sum\{\hat{y}_{\pi,\mathtt{s},\boldsymbol{v}',\boldsymbol{w}} \mid \boldsymbol{v}' + \boldsymbol{w} \leq \boldsymbol{C}\} && (5.8)\\
z_{\pi,\mathtt{t0},\boldsymbol{v}} &= \sum\{\hat{z}_{\pi,\mathtt{t},\boldsymbol{v},\boldsymbol{w}'} \mid \boldsymbol{v} + \boldsymbol{w}' \leq \boldsymbol{C}\} && (5.9)\\
z_{\pi,\mathtt{t1},\boldsymbol{w}} &= \sum\{\hat{z}_{\pi,\mathtt{t},\boldsymbol{v}',\boldsymbol{w}} \mid \boldsymbol{v}' + \boldsymbol{w} \leq \boldsymbol{C}\} && (5.10)\\
1 &\leq \sum\{y_{\pi',\epsilon,\boldsymbol{u}'} \mid \pi' \text{ a 1-type, } \boldsymbol{u}' \leq \boldsymbol{C}\} && (5.11)
\end{aligned}$$

Lemma 5.9. Let $\mathfrak{A}$ be a finite model of φ. The constraints $\mathcal{E}_1$ are satisfied when the variables take the values specified in Table 5.1.

Proof. Omitted; see [PH07]. □

Before continuing we need some extra terminology. Let τ be any 2-type; we say that τ is *forbidden* if the following formula is unsatisfiable (referring to the normal form in Lemma 4.12):

$$\bigwedge \tau \wedge \alpha(x) \wedge \alpha(y) \wedge \bigwedge_{1\leq j\leq n} \Big((e_j(x,y) \rightarrow (\beta_j(x,y)) \wedge (e_j(y,x) \rightarrow (\beta_j(y,x))\Big).$$

A forbidden 2-type cannot be assigned to any pair of elements in a model, as it directly conflicts with the requirements imposed by the normal form. Such a 2-type, however, can easily be identified (since the above formula is purely boolean) in time bounded by an exponential function of $\|\varphi\|$.

Let $\mathcal{E}_2$ be the following classes of constraints, where π and λ vary as described previously, $\boldsymbol{u} \neq \mathbf{0}$, and for the bit-strings $\mathtt{s}$ and $\mathtt{t}$ also require that $|\mathtt{s}| = p$ and $|\mathtt{t}| = q$:

$$\begin{aligned}
y_{\pi,\mathtt{s},\boldsymbol{u}} &= \sum\{x_{\lambda'} \mid \lambda' \in \Lambda_{\pi,\mathtt{s}} \text{ and } \boldsymbol{C}_{\lambda'} = \boldsymbol{u}\} && (5.12)\\
x_{\lambda^{-1}} &= x_\lambda && (5.13)\\
x_\lambda &= 0, \quad \text{whenever } \mathrm{tp}_1(\lambda) = \mathrm{tp}_2(\lambda) && (5.14)\\
x_\lambda &= 0, \quad \text{whenever } \lambda \text{ is forbidden} && (5.15)\\
z_{\pi,\mathtt{t},\boldsymbol{u}} &= 0, \quad \text{whenever } \mu_{\pi,\mathtt{t}} \text{ is forbidden} && (5.16)\\
z_{\pi,\mathtt{t},\boldsymbol{u}} &= 0, \quad \text{whenever } \boldsymbol{u} \text{ is not a scalar multiple of } \boldsymbol{C}_{\mu_{\pi,\mathtt{t}}} && (5.17)
\end{aligned}$$

Lemma 5.10. Let $\mathfrak{A}$ be a finite, chromatic model of φ. The constraints $\mathcal{E}_2$ are satisfied when the variables take the values specified in Table 5.1.

Proof. Omitted; see [PH07]. $\square$

Let $\mathcal{E}_3$ be the following classes of constraints, where π varies as described, $\mathtt{t}$ varies over all bit-strings with $|\mathtt{t}| = q$, and $\boldsymbol{u} \neq \mathbf{0}$:

$$z_{\pi,\mathtt{t},\boldsymbol{u}} > 0 \;\Rightarrow\; \sum\{y_{\pi',\epsilon,\boldsymbol{u}'} \mid \pi' = \mathrm{tp}_2(\mu_{\pi,\mathtt{t}}) \text{ and } \boldsymbol{u}' \leq \boldsymbol{C}\} > 0 \tag{5.18}$$

Lemma 5.11. Let $\mathfrak{A}$ be a finite model of φ. The constraints $\mathcal{E}_3$ are satisfied when the variables take the values specified in Table 5.1.

Proof. Omitted; see [PH07]. $\square$

We now turn our attention to the constraints that our database enforces. We first introduce, for each constant $c \in D$, the sets

$$\Lambda_c = \{\lambda \in \Lambda : \mathrm{tp}_\Delta[c, c'] = \lambda, \text{ for some } c' \in \Delta \setminus \{c\}\},$$

and

$$M_c = \{\mu \in M_{\mathrm{tp}_\Delta[c]} : \mathrm{tp}_\Delta[c, c'] = \mu, \text{ for some } c' \in \Delta \setminus \{c\}\}.$$

That is, Λ_c is the set of invertible message-types 'connecting' c to another element in Δ; similarly, M_c is the set of non-invertible message-types or silent 2-types 'connecting' c to another element in Δ. Assuming that Δ is complete, Λ_c and M_c can be read from Δ. It should be obvious that if Δ, φ is to be satisfied, any model of Δ, φ has to contain an element that (at the very least) sends all the messages in $\Lambda_c \cup M_c$.

Fix a constant $c \in D$. By the indexing of invertible message-types described in Section 5.1, to each $\lambda \in \Lambda_c$ corresponds a bit-string $\mathtt{s}$ with $|\mathtt{s}| = p$ such that $\lambda \in \Lambda_{\mathrm{tp}_\Delta[c],\mathtt{s}}$; for the sake of brevity, we often abuse the notation for set membership, writing $\mathtt{s} \in \Lambda_c$, to mean that $\lambda \in \Lambda_c$. Notice that no confusion arises from this notation, because we are working with chromatic models; hence each constant $c \in \Delta$ can send at most one invertible message whose type is in $\Lambda_{\mathrm{tp}_\Delta[c],\mathtt{s}}$, for any given $\mathtt{s}$ with $|\mathtt{s}| = p$ (recall Remark 5.4). We also write $\mathtt{t} \in M_c$, where $\mathtt{t}$ is a bit-string with $|t| = q$, to mean that the non-invertible message-type or silent 2-type $\mu_{\mathrm{tp}_\Delta[c],\mathtt{t}}$ is in M_c.

Now, fix a finite model $\mathfrak{A}$ of Δ, φ. For each constant $c \in D$, let $c^{\mathfrak{A}} \in A$ be the element interpreting c, and let $\pi = \mathrm{tp}_{\mathfrak{A}}[c^{\mathfrak{A}}]$ ($= \mathrm{tp}_{\Delta}[c]$, assuming Δ is complete). We use $\gamma^c_{\pi,\mathtt{s}}$ to denote the s-spectrum of $c^{\mathfrak{A}}$ in $\mathfrak{A}$, for all $\mathtt{s}$ with $|\mathtt{s}| \leq p$, and $\delta^c_{\pi,\mathtt{t}}$ to denote its $\mathtt{t}$-tally, for all $\mathtt{t}$ with $|\mathtt{t}| \leq q$. The meaning of these symbols will remain *fixed* for the rest of the chapter. According to Lemma 5.7, $\gamma^c_{\pi,\epsilon} + \delta^c_{\pi,\epsilon} = \boldsymbol{C}$, $\gamma^c_{\pi,\mathtt{s}} = \gamma^c_{\pi,\mathtt{s0}} + \gamma^c_{\pi,\mathtt{s1}}$ (for all $\mathtt{s}$ with $|\mathtt{s}| < p$), and $\delta^c_{\pi,\mathtt{t}} = \delta^c_{\pi,\mathtt{t0}} + \delta^c_{\pi,\mathtt{t1}}$ (for all $\mathtt{t}$ with $|\mathtt{t}| < q$). It is also clear (since $c^{\mathfrak{A}}$ realizes c) that if $\lambda \in \Lambda_c$ and $\mathtt{s}$ is such that $\lambda \in \Lambda_{\pi,\mathtt{s}}$, then $\gamma^c_{\pi,\mathtt{s}} = \boldsymbol{C}_\lambda$; further, if $\mu = \mu_{\pi,\mathtt{t}} \in M_c$, there exists $n \geq |\{c' \in D \setminus \{c\} : \mathrm{tp}_\Delta[c, c'] = \mu\}|$ such that $\delta^c_{\pi,\mathtt{t}} = n \cdot \boldsymbol{C}_\mu$.

For a given bit-string $\mathtt{x}$, we denote by $\mathrm{seg}(\mathtt{x})$ the set of all *proper* prefixes of $\mathtt{x}$. We also define $\mathrm{seg}^+(\mathtt{x}) = \mathrm{seg}(\mathtt{x}) \cup \{\mathtt{x}\}$. For example,

$$\mathrm{seg}(\mathtt{1101}) = \{\epsilon, \mathtt{1}, \mathtt{11}, \mathtt{110}\},$$
$$\mathrm{seg}^+(\mathtt{1101}) = \{\epsilon, \mathtt{1}, \mathtt{11}, \mathtt{110}, \mathtt{1101}\}.$$

For each invertible type λ, let η_λ be the number of elements in the database that send an invertible message of type λ (to another database element). For the next set of constraints, the subscripts range as follows:

- $\pi \in \Pi^\Delta$, where $\Pi^\Delta = \{\mathrm{tp}_\Delta[c] : c \in D\}$;
- $\mathtt{s} \in \bigcup_{c \in D}\{\mathrm{seg}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$;
- $\mathtt{t} \in \bigcup_{c \in D}\{\mathrm{seg}(\mathtt{t}') : \mathtt{t}' \in M_c\}$.

Let $\mathcal{E}_4$ be the following classes of constraints, where λ varies as usual and the other subscripts vary as described above:

$$x_\lambda \geq \eta_\lambda \tag{5.19}$$
$$y_{\pi,\epsilon,\boldsymbol{u}} \geq 1, \quad \text{when } \boldsymbol{u} = \gamma^c_{\pi,\epsilon} \text{ and } \pi = \mathrm{tp}_\Delta[c] \tag{5.20}$$
$$z_{\pi,\epsilon,\boldsymbol{u}} \geq 1, \quad \text{when } \boldsymbol{u} = \delta^c_{\pi,\epsilon} \text{ and } \pi = \mathrm{tp}_\Delta[c] \tag{5.21}$$
$$\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}} \geq 1, \quad \text{when } \boldsymbol{v} = \gamma^c_{\pi,\mathtt{s0}},\ \boldsymbol{w} = \gamma^c_{\pi,\mathtt{s1}}, \text{ and } \pi = \mathrm{tp}_\Delta[c] \tag{5.22}$$
$$\hat{z}_{\pi,\mathtt{t},\boldsymbol{v},\boldsymbol{w}} \geq 1, \quad \text{when } \boldsymbol{v} = \delta^c_{\pi,\mathtt{t0}},\ \boldsymbol{w} = \delta^c_{\pi,\mathtt{t1}}, \text{ and } \pi = \mathrm{tp}_\Delta[c] \tag{5.23}$$

Lemma 5.12. Let $\mathfrak{A}$ be a finite model of Δ, φ, with Δ complete. The constraints $\mathcal{E}_4$ are satisfied when the variables take the values specified in Table 5.1.

Proof. The constraints (5.19), (5.20) and (5.21) are evident. For (5.22), let $c \in D$ be a constant and let $\pi = \mathrm{tp}_\Delta[c]$. Pick any $\mathtt{s} \in \bigcup\{\mathrm{seg}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$. By definition,

$\gamma^c_{\pi,\mathtt{s0}}$ and $\gamma^c_{\pi,\mathtt{s1}}$ are the $\mathtt{s0}$- and $\mathtt{s1}$-spectrum (respectively) of an element $a \in A$ that realizes c. But then, referring to Table 5.1, a is one of the elements 'recorded' by the variable $\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}}$, for $\boldsymbol{v} = \gamma^c_{\pi,\mathtt{s0}}$ and $\boldsymbol{w} = \gamma^c_{\pi,\mathtt{s1}}$. Consequently, $\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}} \geq 1$, when $\boldsymbol{v} = \gamma^c_{\pi,\mathtt{s0}}$ and $\boldsymbol{w} = \gamma^c_{\pi,\mathtt{s1}}$. Thus, the constraints (5.22) are satisfied. Similarly, the constraints (5.23) are also satisfied. □

For any bit-string $\mathtt{x}$, let

$$\text{flip}(\mathtt{x}) = \text{seg}^+(\mathtt{x}) \cup \{\mathtt{y}\bar{\mathtt{a}} : \mathtt{ya} \in \text{seg}^+(\mathtt{x}) \text{ and } |\mathtt{a}| = 1\},$$

where $\bar{\mathtt{a}}$ is the complement of the bit $\mathtt{a}$, i.e. $\bar{\mathtt{0}} = \mathtt{1}$ and $\bar{\mathtt{1}} = \mathtt{0}$. For example,

$$\text{flip}(\mathtt{1101}) = \{\epsilon, \mathtt{1}, \mathtt{11}, \mathtt{110}, \mathtt{1101}\} \cup \{\mathtt{0}, \mathtt{10}, \mathtt{111}, \mathtt{1100}\}.$$

Notice that, for any bit-string $\mathtt{x}$, $|\text{flip}(\mathtt{x})| = 2 \cdot |\mathtt{x}| + 1$. Now, observe that, for each constant $c \in D$, the vectors $\gamma^c_{\pi,\mathtt{s}}$ ($\pi = \text{tp}_\Delta[c]$) used in $\mathcal{E}_4$ are exactly those with $\mathtt{s} \in \bigcup\{\text{flip}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$; similarly, the vectors $\delta^c_{\pi,\mathtt{t}}$ ($\pi = \text{tp}_\Delta[c]$) in $\mathcal{E}_4$ are exactly those with $\mathtt{t} \in \bigcup\{\text{flip}(\mathtt{t}') : \mathtt{t}' \in M_c\}$. Thus, the number of vectors $\gamma^c_{\pi,\mathtt{s}}$ and $\delta^c_{\pi,\mathtt{t}}$ required to write $\mathcal{E}_4$ is *linear* in the size $|\Delta|$ of Δ.

Let $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3 \cup \mathcal{E}_4$. Note that the size $\|\mathcal{E}\|$ of $\mathcal{E}$ is bounded above by an exponential function of $\|\varphi\|$.

Lemma 5.13. Let Δ and φ be as above, with Δ complete. If Δ, φ is finitely satisfiable, then $\mathcal{E}$ has a solution over $\mathbb{N}$.

Proof. Lemmas 5.9 – 5.11, and 5.12. □

We remark that the vectors $\gamma^c_{\pi,\mathtt{s}}$ and $\delta^c_{\pi,\mathtt{t}}$ (for all possible values of $\mathtt{s}$ and $\mathtt{t}$) are defined in terms of a putative finite model $\mathfrak{A}$ of Δ, φ. Lemma 5.13 tells us that if such a model exists, then $\mathcal{E}$ has a solution over $\mathbb{N}$. However, the values of these vectors cannot be obtained from Δ for $\mathcal{E}_4$ to be written in the first place. This issue is easily fixed: since only the vectors $\gamma^c_{\pi,\mathtt{s}}$, for $\mathtt{s} \in \bigcup\{\text{flip}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$, and $\delta^c_{\pi,\mathtt{t}}$, for $\mathtt{t} \in \bigcup\{\text{flip}(\mathtt{t}') : \mathtt{t}' \in M_c\}$, are required (for each constant $c \in D$, where $\pi = \text{tp}_\Delta[c]$), all possible values of these vectors can be tried. In more detail, we need to find vectors $\gamma^c_{\pi,\mathtt{s}}$, for each $\mathtt{s} \in \bigcup\{\text{flip}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$, and $\delta^c_{\pi,\mathtt{t}}$, for each $\mathtt{t} \in \bigcup\{\text{flip}(\mathtt{t}') : \mathtt{t}' \in M_c\}$, such that

$$\gamma^c_{\pi,\epsilon} + \delta^c_{\pi,\epsilon} = \boldsymbol{C} \tag{5.24}$$

$$\gamma^c_{\pi,\mathtt{s0}} + \gamma^c_{\pi,\mathtt{s1}} = \gamma^c_{\pi,\mathtt{s}}, \quad \text{for } \mathtt{s} \in \{\mathrm{seg}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\} \tag{5.25}$$

$$\delta^c_{\pi,\mathtt{t0}} + \delta^c_{\pi,\mathtt{t1}} = \delta^c_{\pi,\mathtt{t}}, \quad \text{for } \mathtt{t} \in \{\mathrm{seg}(\mathtt{t}') : \mathtt{t}' \in M_c\} \tag{5.26}$$

and, in addition,

$$\gamma^c_{\pi,\mathtt{s}} = \boldsymbol{C}_\lambda, \tag{5.27}$$

if $\lambda \in \Lambda_c$ and $\mathtt{s}$ is such that $\lambda \in \Lambda_{\pi,\mathtt{s}}$, and

$$\delta^c_{\pi,\mathtt{t}} = n \cdot \boldsymbol{C}_\mu, \tag{5.28}$$

for some $n \geq |\{c' \in D \setminus \{c\} : \mathrm{tp}_\Delta[c, c'] = \mu\}|$, when $\mu = \mu_{\pi,\mathtt{t}} \in M_c$. (Observe that all the bit-strings that appear as subscripts in Equations 5.24 – 5.28 are either in $\bigcup\{\mathrm{flip}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$ or in $\bigcup\{\mathrm{flip}(\mathtt{t}') : \mathtt{t}' \in M_c\}$.) Recalling that all the vectors we are dealing with are $\leq \boldsymbol{C}$, the required vectors can be found in time exponential with respect to $\|\varphi\|$. Finally, it is clear that if such vectors do not exist, Δ, φ is not finitely satisfiable. (See the proof of Theorem 5.18 for more details.)

5.4 Obtaining a model from the solutions

To construct a model given a solution of $\mathcal{E}$ over $\mathbb{N}$, we will start with sets of elements A_π of the right cardinality, for each 1-type π, and gradually build the message-types (or silent-types for that matter) that those elements 'want' to send. Fix some 1-type π and let A_π be a set with cardinality

$$|A_\pi| = \sum\{y_{\pi,\epsilon,\boldsymbol{u}'} \mid \boldsymbol{u}' \leq \boldsymbol{C}\}.$$

Think of A_π as the set of elements that 'want' to have 1-type π. We now define the functions $\mathbf{f}_{\pi,\mathtt{s}}$ and $\mathbf{g}_{\pi,\mathtt{t}}$ that give us the spectra and tallies for each element $a \in A_\pi$. Think of $\mathbf{f}_{\pi,\mathtt{s}}(a)$ as the $\mathtt{s}$-spectrum that a 'wants' to have and $\mathbf{g}_{\pi,\mathtt{t}}(a)$ as the $\mathtt{t}$-tally that a 'wants' to have (when a model is eventually built). For those functions to agree with the solutions of the previous system of constraints, we need to ensure that

$$|\mathbf{f}^{-1}_{\pi,\mathtt{s}}(\boldsymbol{u})| = y_{\pi,\mathtt{s},\boldsymbol{u}}, \tag{5.29}$$

$$|\mathbf{g}^{-1}_{\pi,\mathtt{t}}(\boldsymbol{u})| = z_{\pi,\mathtt{t},\boldsymbol{u}}. \tag{5.30}$$

Furthermore, we need to make sure that, for all $a \in A_\pi$,

$$\mathbf{f}_{\pi,\epsilon}(a) + \mathbf{g}_{\pi,\epsilon}(a) = \boldsymbol{C}, \tag{5.31}$$

$$\mathbf{f}_{\pi,\mathtt{s0}}(a) + \mathbf{f}_{\pi,\mathtt{s1}}(a) = \mathbf{f}_{\pi,\mathtt{s}}(a), \tag{5.32}$$

$$\mathbf{g}_{\pi,\mathtt{t0}}(a) + \mathbf{g}_{\pi,\mathtt{t1}}(a) = \mathbf{g}_{\pi,\mathtt{t}}(a). \tag{5.33}$$

Finally, for each set A_π where $\pi = \mathrm{tp}_\Delta[c]$, we wish to enforce the existence of an element $b_c \in A_\pi$ such that, for all $\mathtt{s} \in \Lambda_c$ and all $\mathtt{t} \in M_c$,

$$\mathbf{f}_{\pi,\mathtt{s}}(b_c) = \gamma^c_{\pi,\mathtt{s}}, \tag{5.34}$$

$$\mathbf{g}_{\pi,\mathtt{t}}(b_c) = \delta^c_{\pi,\mathtt{t}}. \tag{5.35}$$

Such an element b_c can be used to realize the constant c, when a model is eventually built.

The following lemma guarantees that the above requirements can be satisfied.

Lemma 5.14. Suppose that x_λ, $y_{\pi,\mathtt{s},\boldsymbol{u}}$, $z_{\pi,\mathtt{t},\boldsymbol{u}}$, $\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}}$, $\hat{z}_{\pi,\mathtt{t},\boldsymbol{v},\boldsymbol{w}}$ are (classes of) natural numbers satisfying the constraints $\mathcal{E}$ given above. Fix any 1-type $\pi \in \Pi$, and let A_π be a set of cardinality $\sum\{y_{\pi,\epsilon,\boldsymbol{u}'} \mid \boldsymbol{u}' \le \boldsymbol{C}\}$. Then there exists a system of functions on A_π

$$\mathbf{f}_{\pi,\mathtt{s}} : A_\pi \to \{\boldsymbol{u} \mid \boldsymbol{u} \le \boldsymbol{C}\} \quad \text{and} \quad \mathbf{g}_{\pi,\mathtt{t}} : A_\pi \to \{\boldsymbol{u} \mid \boldsymbol{u} \le \boldsymbol{C}\},$$

for each bit-string $\mathtt{s}$ with $|\mathtt{s}| \le p$ and $\mathtt{t}$ with $|\mathtt{t}| \le q$, satisfying the following conditions:

(i) Equations (5.29) and (5.30) hold for all vectors $\boldsymbol{u} \le \boldsymbol{C}$.

(ii) If $|\mathtt{s}| < p$ and $|\mathtt{t}| < q$, Equations (5.31) – (5.33) hold for all $a \in A_\pi$.

In addition, the above functions $\mathbf{f}_{\pi,\mathtt{s}}$ and $\mathbf{g}_{\pi,\mathtt{t}}$ can be chosen such that if $c \in D$ is a constant and $\pi = \mathrm{tp}_\Delta[c]$, there exists an element $b_c \in A_\pi$ satisfying Equations (5.34) and (5.35) (for each $\mathtt{s} \in \Lambda_c$ and $\mathtt{t} \in M_c$).

Proof. Decompose A_π into pairwise disjoint sets $A_{\boldsymbol{u}}$ of cardinality $|A_{\boldsymbol{u}}| = y_{\pi,\epsilon,\boldsymbol{u}}$, for each vector $\boldsymbol{u} \le \boldsymbol{C}$. Note that, since $y_{\pi,\epsilon,\boldsymbol{u}}$ might be zero, some of those sets could be empty. However, if $\pi = \mathrm{tp}_\Delta[c]$, for some $c \in D$, we have by the constraints (5.20) that $A_{\boldsymbol{u}} \neq \emptyset$ for $\boldsymbol{u} = \gamma^c_{\pi,\epsilon}$.

We construct the functions $\mathbf{f}_{\pi,\mathtt{s}}$, where $0 < |\mathtt{s}| \le p$ by induction on $\mathtt{s}$. Suppose $\mathtt{s} = \epsilon$; for each $\boldsymbol{u} \le \boldsymbol{C}$, and for all $a \in A_{\boldsymbol{u}}$, set $\mathbf{f}_{\pi,\epsilon}(a) = \boldsymbol{u}$ and $\mathbf{g}_{\pi,\epsilon}(a) =$

$\boldsymbol{C} - \boldsymbol{u}$. These assignments clearly satisfy (5.29) and (5.30), keeping in mind the constraints (5.4). Now, assume that $\mathbf{f}_{\pi,\mathtt{s}}$ has been defined, for some $\mathtt{s}$ with $0 \leq |\mathtt{s}| < p$. For every vector $\boldsymbol{u} \leq \boldsymbol{C}$, decompose the set $\mathbf{f}^{-1}_{\pi,\mathtt{s}}(\boldsymbol{u})$ into subsets $A_{\boldsymbol{v},\boldsymbol{w}}$ with cardinality $|A_{\boldsymbol{v},\boldsymbol{w}}| = \hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}}$, for all $\boldsymbol{v}$, $\boldsymbol{w}$ such that $\boldsymbol{v} + \boldsymbol{w} = \boldsymbol{u}$. This is possible from the constraints (5.5) and Equation (5.29). For all $a \in A_{\boldsymbol{v},\boldsymbol{w}}$, set

$$\mathbf{f}_{\pi,\mathtt{s0}}(a) = \boldsymbol{v} \quad \text{and} \quad \mathbf{f}_{\pi,\mathtt{s1}}(a) = \boldsymbol{w}.$$

Notice that Equation (5.32) holds as required.

That $\mathbf{f}_{\pi,\mathtt{s0}}$ and $\mathbf{f}_{\pi,\mathtt{s1}}$ both satisfy Equation (5.29) is guaranteed by the constraints (5.7) and (5.8). Clearly, $\mathbf{f}_{\pi,\mathtt{s0}}(a) = \boldsymbol{v}$ if and only if for some vector $\boldsymbol{w}'$ such that $\boldsymbol{v} + \boldsymbol{w}' \leq \boldsymbol{C}$, $a \in A_{\boldsymbol{v},\boldsymbol{w}'}$. Thus, we have

$$|\mathbf{f}^{-1}_{\pi,\mathtt{s0}}(\boldsymbol{v})| = \left|\bigcup\{A_{\boldsymbol{v},\boldsymbol{w}'} : \boldsymbol{v} + \boldsymbol{w}' \leq \boldsymbol{C}\}\right| = y_{\pi,\mathtt{s0},\boldsymbol{v}}.$$

Similarly, we have

$$|\mathbf{f}^{-1}_{\pi,\mathtt{s1}}(\boldsymbol{w})| = \left|\bigcup\{A_{\boldsymbol{v}',\boldsymbol{w}} : \boldsymbol{v} + \boldsymbol{w}' \leq \boldsymbol{C}\}\right| = y_{\pi,\mathtt{s1},\boldsymbol{w}}.$$

This completes the induction for the functions $\mathbf{f}_{\pi,\mathtt{s}}$. The construction of the functions $\mathbf{g}_{\pi,\mathtt{t}}$ is completely analogous.

Finally, suppose that $\pi = \mathrm{tp}_\Delta[c]$, for some constant $c \in D$. In that case, we have to ensure the existence of an element $b_c \in A_\pi$ that satisfies Equation (5.34), for all $\mathtt{s} \in \Lambda_c$. During the initial decomposition of A_π into sets $A_{\boldsymbol{u}}$, b_c is chosen to be any element in the (non-empty) set $A_{\boldsymbol{u}}$, where $\boldsymbol{u} = \gamma^c_{\pi,\epsilon}$. To ensure that b_c satisfies (5.34) for all $\mathtt{s} \in \Lambda_c$, the stronger condition that b_c satisfies (5.34) for all $\mathtt{s} \in L = \bigcup\{\mathrm{flip}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$ can be enforced. Notice that Equation (5.34) can be written in the (more convenient for our purpose) form

$$b_c \in \mathbf{f}^{-1}_{\pi,\mathtt{s}}(\gamma^c_{\pi,\mathtt{s}}); \tag{5.36}$$

thus, we show how to enforce (5.36), for all $\mathtt{s} \in L$.

Notice, initially, that, by the above construction of the functions $\mathbf{f}_{\pi,\mathtt{s}}$, $\mathbf{f}_{\pi,\epsilon}(b_c)$ will be set to $\gamma^c_{\pi,\epsilon}$, i.e. b_c will be in the set $\mathbf{f}^{-1}_{\pi,\epsilon}(\gamma^c_{\pi,\epsilon})$. Now suppose that, by inductive hypothesis, b_c is in the set $\mathbf{f}^{-1}_{\pi,\mathtt{s}}(\gamma^c_{\pi,\mathtt{s}})$, for some $\mathtt{s} \in L$ with $|\mathtt{s}| < p$. When decomposing the set $\mathbf{f}^{-1}_{\pi,\mathtt{s}}(\gamma^c_{\pi,\mathtt{s}})$ into subsets $A_{\boldsymbol{v},\boldsymbol{w}}$ as described above, the constraints (5.22) guarantee that, when $\boldsymbol{v} = \gamma^c_{\pi,\mathtt{s0}}$ and $\boldsymbol{w} = \gamma^c_{\pi,\mathtt{s1}}$, $A_{\boldsymbol{v},\boldsymbol{w}}$ will be non-empty. Thus, when $\boldsymbol{v} = \gamma^c_{\pi,\mathtt{s0}}$ and $\boldsymbol{w} = \gamma^c_{\pi,\mathtt{s1}}$, $A_{\boldsymbol{v},\boldsymbol{w}}$ can be chosen such that b_c is in it. Then, by the above construction of the functions $\mathbf{f}_{\pi,\mathtt{s}}$, $\mathbf{f}_{\pi,\mathtt{s0}}(b_c)$ will be set to $\gamma^c_{\pi,\mathtt{s0}}$ and $\mathbf{f}_{\pi,\mathtt{s1}}(b_c)$ to $\gamma^c_{\pi,\mathtt{s1}}$. Equivalently, b_c will be in the set $\mathbf{f}^{-1}_{\pi,\mathtt{s0}}(\gamma^c_{\pi,\mathtt{s0}})$ and the set $\mathbf{f}^{-1}_{\pi,\mathtt{s1}}(\gamma^c_{\pi,\mathtt{s1}})$. This (inductively) establishes (5.36)—and thus (5.34)—for all $\mathtt{s} \in L$. A similar argument establishes Equation (5.35) for b_c. □

Lemma 5.15. Let the functions $\mathbf{f}_{\pi,\mathtt{s}}$ and $\mathbf{g}_{\pi,\mathtt{t}}$ be constructed as in Lemma 5.14. Then, for all $a \in A_\pi$, we have

$$\sum\{\mathbf{f}_{\pi,\mathtt{s}'}(a) : |\mathtt{s}'| = p\} + \sum\{\mathbf{g}_{\pi,\mathtt{t}'}(a) : |\mathtt{t}'| = q\} = \boldsymbol{C}.$$

Proof. Simple induction; see [PH07]. □

We are now ready to prove the converse of Lemma 5.13.

Lemma 5.16. Let Δ, φ and $\mathcal{E}$ be as defined above. Assuming Δ is complete, if $\mathcal{E}$ has a solution over $\mathbb{N}$ then Δ, φ is finitely satisfiable.

Proof. Let x_λ, $y_{\pi,\mathtt{s},\boldsymbol{u}}$, $z_{\pi,\mathtt{t},\boldsymbol{u}}$, $\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}}$, $\hat{z}_{\pi,\mathtt{t},\boldsymbol{v},\boldsymbol{w}}$ be (classes of) natural numbers satisfying $\mathcal{E}$. Notice that, for all positive integers k, the (classes of) natural numbers kx_λ, $ky_{\pi,\mathtt{s},\boldsymbol{u}}$, $kz_{\pi,\mathtt{t},\boldsymbol{u}}$, $k\hat{y}_{\pi,\mathtt{s},\boldsymbol{v},\boldsymbol{w}}$, $k\hat{z}_{\pi,\mathtt{t},\boldsymbol{v},\boldsymbol{w}}$ also satisfy $\mathcal{E}$. Thus, we may assume that all solutions are greater than or equal to $3mC$.

We start by defining the universe A of our model $\mathfrak{A}$. Let

$$A = \bigcup\{A_\pi \mid \pi \text{ is any 1-type over } \sigma\},$$

where each set A_π has cardinality

$$|A_\pi| = \sum\{y_{\pi,\epsilon,\boldsymbol{u}'} \mid \boldsymbol{u}' \leq \boldsymbol{C}\}$$

and the sets A_π are pairwise disjoint. Think of A_π as the elements of A that 'want' to have 1-type π. Note that $A \neq \emptyset$, by the constraint (5.11).

Let the functions $\mathbf{f}_{\pi,\mathtt{s}}$, $\mathbf{g}_{\pi,\mathtt{t}}$, and the elements b_c (for each $c \in D$) be as constructed in Lemma 5.14. Think of $\mathbf{f}_{\pi,\mathtt{s}}(a)$ as the $\mathtt{s}$-spectrum that a 'wants' to have and $\mathbf{g}_{\pi,\mathtt{t}}(a)$ as the $\mathtt{t}$-tally that a 'wants' to have; we are only interested in the values of these functions for $|\mathtt{s}| = p$ and $|\mathtt{t}| = q$. Think of b_c (for each $c \in D$) as the element that 'wants' to realize the constant c. Decompose each A_π into sets $\mathbf{f}^{-1}_{\pi,\mathtt{s}}(\boldsymbol{u})$, for each vector $\boldsymbol{u}$ with $\mathbf{0} < \boldsymbol{u} \leq \boldsymbol{C}$. By the constraints (5.12) and Equation (5.29), decompose each of those $\mathbf{f}^{-1}_{\pi,\mathtt{s}}(\boldsymbol{u})$ into pairwise disjoint (possibly empty) sets A_λ with $|A_\lambda| = x_\lambda$, for all invertible message-types $\lambda \in \Lambda_{\pi,\mathtt{s}}$ with $\boldsymbol{C}_\lambda = \boldsymbol{u}$. Think of A_λ as the set of elements in A_π that 'want' to send a single invertible message of type λ. The decompositions into sets A_λ must be performed *with the following restriction:* if c sends an invertible message of type λ to another database element, b_c must be assigned to the set A_λ. By the constraints (5.19), the cardinality of A_λ is such that the latter assignment of b_c to A_λ is valid. The above process is repeated for all possible different values of $\mathtt{s}$ (with $|\mathtt{s}| = p$), and each decomposition should be thought of as independent of each other. Analogously, A_π is decomposed into pairwise disjoint sets $\mathbf{g}^{-1}_{\pi,\mathtt{t}}(\boldsymbol{u})$ and, again, those decompositions should be thought of as independent of each other.

Based on the above decompositions, we specify for each $a \in A_\pi$ a 'mosaic piece' and show how to assemble these pieces into a model of φ. A mosaic piece is, informally, a collection of the messages that a 'wants' to send. This collection might contain more than one message of each type (or zero for that matter). Let $a \in A_\pi$; the mosaic piece corresponding to a contains:

(i) a single message labeled $\lambda_{a,\mathtt{s}}$ for each bit-string $\mathtt{s}$ with $|\mathtt{s}| = p$ if $\mathbf{f}_{\pi,\mathtt{s}}(a) \neq \mathbf{0}$, where $\lambda_{a,\mathtt{s}}$ is the (unique) 2-type λ for which $a \in A_\lambda$;

(ii) $n_{a,\mathtt{t}}$ messages labeled $\mu_{\pi,\mathtt{t}}$ for each bit-string $\mathtt{t}$ with $|\mathtt{t}| = q$, where $n_{a,\mathtt{t}}$ is the (unique) natural number such that $\mathbf{g}_{\pi,\mathtt{t}}(a) = n_{a,\mathtt{t}} \cdot \boldsymbol{C}_{\mu_{\pi,\mathtt{t}}}$. Notice that if $\mathbf{g}_{\pi,\mathtt{t}}(a) = 0$ then $n_{a,\mathtt{t}} = 0$, otherwise $n_{a,\mathtt{t}}$ exists by the constraints (5.17) and Equation (5.30).

The mosaic piece corresponding to a is depicted in Figure 5.1.

Let $a \in A$ and define $\boldsymbol{C}_a$ to be the vector $(C_{a,1}, \ldots, C_{a,m})$ whose ith coordinate $C_{a,i}$ records the number of messages in the mosaic piece of a containing an outgoing f_i arrow—i.e. messages having label ν for which $f_i(x,y) \in \nu$. Clearly (see Figure 5.1),

$$\boldsymbol{C}_a = \sum\{\mathbf{f}_{\pi,\mathtt{s}'}(a) : |\mathtt{s}'| = p\} + \sum\{\mathbf{g}_{\pi,\mathtt{t}'} : |\mathtt{t}'| = q\}$$

and, by Lemma 5.15,

$$\boldsymbol{C}_a = \boldsymbol{C}. \tag{5.37}$$

We now build $\mathfrak{A}$ in four steps as follows.

Step 1 (Fixing the 1-types) For all 1-types π and all $a \in A_\pi$, set $\mathrm{tp}_{\mathfrak{A}}[a] = \pi$.

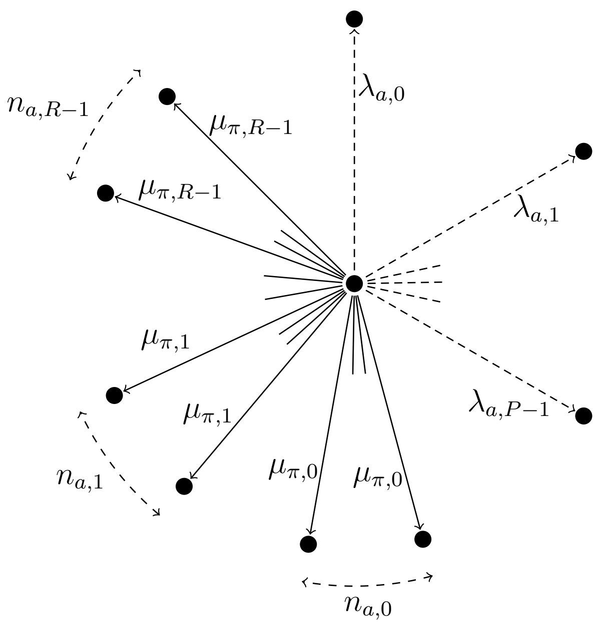


Figure 5.1: The messages sent by $a \in A_\pi$. For each j $(0 \leq j < P)$, a may or may not send a message labeled $\lambda_{a,j}$ (hence the dotted lines); if it does, then $\lambda_{a,j} \in \Lambda_{\pi,j}$. For each k $(0 \leq k < R)$, a sends $n_{a,k}$ messages labeled $\mu_{\pi,k}$; but the numbers $n_{a,k}$ can be zero.

Since the sets A_π are pairwise disjoint, no clashes arise. For each $c \in D$, let $c^{\mathfrak{A}} = b_c$.

Step 2 (Fixing the invertible message-types) We first assign the invertible message-types for all pairs $c_0^{\mathfrak{A}}, c_1^{\mathfrak{A}} \in A$ (corresponding to constants $c_0, c_1 \in D$ respectively), as dictated by D. For, if c_0 and c_1 are connected by λ in Δ, then $c_0^{\mathfrak{A}}$ will 'want' to have spectrum $\boldsymbol{C}_\lambda$ (by Lemma 5.14) and will have been assigned to A_λ; similarly, $c_1^{\mathfrak{A}}$ will 'want' to have spectrum $\boldsymbol{C}_{\lambda^{-1}}$ and will have been assigned to $A_{\lambda^{-1}}$. We then put, by the constraints (5.13), all other λ-labeled messages and all λ^{-1}-labeled messages in one-to-one correspondence, for each invertible message of type λ. Thus, if a sends a λ-labeled message and b 'wants to receive it' (i.e. sends a λ^{-1}-labeled message), we set $\mathrm{tp}_{\mathfrak{A}}[a, b] = \lambda$. To ensure that each assignment $\mathrm{tp}_{\mathfrak{A}}[a, b]$ $(a, b \in A)$ is valid, we need only check that a and b are distinct. But, since $x_\lambda > 0$, by the constraints (5.14) we must have $\mathrm{tp}_{\mathfrak{A}}[a] \neq \mathrm{tp}_{\mathfrak{A}}[b]$, hence, by construction, a and b belong to the pairwise distinct sets $A_{\mathrm{tp}_{\mathfrak{A}}[a]}$ and $A_{\mathrm{tp}_{\mathfrak{A}}[b]}$. Moreover, since every element sends at most one invertible message of each type, no conflicts with the present assignment will arise in future assignments. Finally, all the elements $c^{\mathfrak{A}} \in A$ (corresponding to constants $c \in D$) are distinct, because they belong to distinct sets A_π (where $\pi = \mathrm{tp}_\Delta[c]$).

Step 3 (Fixing the non-invertible message-types) Start by decomposing each set A_π into three pairwise disjoint (possibly empty) sets $A_{\pi,0}$, $A_{\pi,1}$ and $A_{\pi,2}$ having at least mC elements each, if $|A_\pi| \geq 3mC$, and with the following restriction: if $c^{\mathfrak{A}} \in A_\pi$, for some constant $c \in D$ (i.e. $\pi = \mathrm{tp}_\Delta[c]$), pick these three sets in a way such that $c^{\mathfrak{A}} \in A_{\pi,0}$.

Let $\mu_{\pi,\mathtt{t}}$ be any non-invertible message type, with $\pi = \mathrm{tp}_1(\mu_{\pi,\mathtt{t}})$ and $\rho = \mathrm{tp}_2(\mu_{\pi,\mathtt{t}})$ being its starting and terminal 1-types. Let $a \in A$ be an element that sends $n_{a,\mathtt{t}} > 0$ messages of type $\mu_{\pi,\mathtt{t}}$. Clearly, then, $a \in A_\pi$ and there is a vector $\boldsymbol{u} > \mathbf{0}$ such that $\mathbf{g}_{\pi,\mathtt{t}}(a) = \boldsymbol{u}$, hence $\mathbf{g}^{-1}_{\pi,\mathtt{t}}(\boldsymbol{u})$ is non-empty. As a result, $z_{\pi,\mathtt{t},\boldsymbol{u}} = |\mathbf{g}^{-1}_{\pi,\mathtt{t}}(\boldsymbol{u})| > 0$, thus, by the constraints (5.18), $\sum\{y_{\rho,\epsilon,\boldsymbol{u}'} \mid \boldsymbol{u}' \leq \boldsymbol{C}\} > 0$. This implies that A_ρ is non-empty since, clearly, $|A_\rho| = \sum\{y_{\rho,\epsilon,\boldsymbol{u}'} \mid \boldsymbol{u}' \leq \boldsymbol{C}\}$, hence $|A_\rho| \geq 3mC$ by our choice of solution. This means that, by our preliminary decomposition, A_ρ has been partitioned into three sets $A_{\rho,0}$, $A_{\rho,1}$ and $A_{\rho,2}$ having at least mC elements each.

Suppose, for the moment, that a is not equal to the interpretation of any constant. By our initial decomposition A_π has also been partitioned into three sets $A_{\pi,0}$, $A_{\pi,1}$ and $A_{\pi,2}$; since $a \in A_\pi$, let j be such that $a \in A_{\pi,j}$, $0 \leq j \leq 2$. Let $k = j+1 \pmod 3$ and select $n_{a,\mathtt{t}}$ elements b from $A_{\rho,k}$ that have not already been chosen to receive any messages (invertible or non-invertible) and set, for each one of those, $\mathrm{tp}_{\mathfrak{A}}[a,b] = \mu_{\pi,\mathtt{t}}$. Note that there are enough elements in $A_{\rho,k}$ to choose from, as a can send at most mC messages (of invertible or non-invertible type).

On the other hand, suppose that $a = c_1^{\mathfrak{A}}$, for some constant $c_1 \in D$. Further, suppose that there is a constant $c_2 \in D$ to which c_1 sends (in Δ) a message of type $\mu_{\pi,\mathtt{t}}$, i.e. $\mathrm{tp}_\Delta[c_1, c_2] = \mu_{\pi,\mathtt{t}}$. Notice that $c_2^{\mathfrak{A}}$ must lie in A_ρ: in Step 1, to each constant c we assigned an element $c^{\mathfrak{A}} = b_c \in A_{\mathrm{tp}_\Delta[c]}$; hence, $c_2^{\mathfrak{A}} \in A_{\mathrm{tp}_\Delta[c_2]} = A_\rho$. (In fact, by the decompositions at the beginning of this step, $c_2^{\mathfrak{A}} \in A_{\rho,0}$.) Thus, we may set $\mathrm{tp}_{\mathfrak{A}}[c_1^{\mathfrak{A}}, c_2^{\mathfrak{A}}] = \mu_{\pi,\mathtt{t}}$. Now, decrement the value $n_{a,\mathtt{t}}$ to take account of the fact that a non-invertible message of type $\mu_{\pi,\mathtt{t}}$ has been dealt with, and proceed as before.

It is clear, at this point, that no clashes arise with any of the assignments in Steps 1 or 2. Furhter, no clashes arise in future assignments in this step due to the incremental (modulo 3) reuse of the partitioned sets: an element a sending a non-invertible message of type μ to another element b cannot accidentally receive a non-invertible message of type μ' from b. For example, if $a \in A_{\pi,0}$, then $b \in A_{\rho,1}$

and b's messages will be sent to elements $c \in A_{\pi,2}$. Moreover, suppose that $a = c_1^{\mathfrak{A}} \in A_\pi$, for some $c_1 \in D$, sends a non-invertible message to an element $b = c_2^{\mathfrak{A}} \in A_\rho$, for some $c_2 \in D$. Due to our initial decomposition, $c_1^{\mathfrak{A}}$ and $c_2^{\mathfrak{A}}$ belong to $A_{\pi,0}$ and $A_{\rho,0}$ respectively; thus, no clash can arise with future assignments by accidentally setting $c_1^{\mathfrak{A}}$ to be the recipient of a non-invertible message sent by $c_2^{\mathfrak{A}}$, because such a message will be sent to $A_{\pi,1}$.

Step 4 (Fixing the remaining 2-types) Taking advantage of guardedness here, if $\mathrm{tp}_{\mathfrak{A}}[a, b]$ has not already been defined, set it to be the 2-type

$$\pi \cup \rho[y/x] \cup \{\neg\gamma \mid \gamma \text{ is a guard-atom not involving } =\},$$

where $\pi = \mathrm{tp}_{\mathfrak{A}}[a]$, $\rho = \mathrm{tp}_{\mathfrak{A}}[b]$ and $\rho[y/x]$ is the result of replacing x by y in ρ. Note that, since $C_1, \ldots, C_m$ are by assumption positive integers, a and b certainly send some messages and, thus, the constraints (5.15) and (5.16) ensure that both $\alpha \wedge \bigwedge \pi$ and $\alpha \wedge \bigwedge \rho$ are satisfiable.

This completes the definition of $\mathfrak{A}$ and we now show that $\mathfrak{A} \models \Delta, \varphi$. Referring to the normal form in Lemma 4.12, notice that none of the above steps violates the conjuncts

$$\forall x\, \alpha \wedge \bigwedge_{1 \le j \le n} \forall x \forall y (e_j(x, y) \to (\beta_j \vee x = y)).$$

Furthermore, the conjuncts

$$\bigwedge_{1 \le i \le m} \forall x \exists_{=C_i}\, y (f_i(x, y) \wedge x \neq y)$$

are all satisfied taking into account Equation (5.37) and the fact that none of the 2-types assigned in Step 4 is a message type. The database is satisfied due to Lemma 5.14 and our assignment of the sets A_λ, at the beginning of this proof. For each constant $c \in D$, the element that realizes c in $\mathfrak{A}$ is $c^{\mathfrak{A}}$ (chosen at the beginning of the proof). □

Now, observe that all the constraints in $\mathcal{E}$ have the forms

$$\begin{aligned} x_1 + \ldots + x_n &= x, \\ x_1 + \ldots + x_n &\geq c, \\ x &= 0, \end{aligned}$$

$$\begin{aligned} x_1 &= x_2, \\ x &\geq c, \\ x > 0 \Rightarrow x_1 + \ldots + x_n &> 0, \end{aligned}$$

where x, $x_1, \ldots x_n$ are variables and c is a constant. Recall that the size $\|\mathcal{E}\|$ of $\mathcal{E}$ is exponential in the size of $\|\Delta \cup \{\varphi\}\|$. Our goal is to find a solution of $\mathcal{E}$ in $\mathbb{N}$.

The following lemma shows that we can transform a system of the above form into an integer programming problem which, in turn, can be regarded as a linear programming problem. This is important because linear programming is in P, whereas integer programming is in NP. The original version is due to Calvanese [Cal96].

Lemma 5.17. Let Δ, φ and $\mathcal{E}$ as above. An algorithm exists to determine whether $\mathcal{E}$ has a solution over $\mathbb{N}$ in time bounded by a polynomial function of $\|\mathcal{E}\|$, and hence by an exponential function of $\|\Delta, \varphi\|$.

Proof. Omitted; similar to Lemma 15 in [PH07]. $\square$

We have now arrived to our main results.

Theorem 5.18. FinSat($\mathcal{GC}^2\mathcal{D}$) is in EXPTIME.

Proof. Let Δ and ψ be given. Convert ψ into a formula φ in normal form (as discussed in Lemma 4.12) and search for a completion Δ^* of Δ (this can be done in exponential time). Guess, for each constant $c \in D$, vectors $\gamma^c_{\pi,\mathtt{s}}$, for $\mathtt{s} \in \bigcup\{\text{flip}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$, and $\delta^c_{\pi,\mathtt{t}}$, for $\mathtt{t} \in \bigcup\{\text{flip}(\mathtt{t}') : \mathtt{t}' \in M_c\}$ satisfying Equations (5.24) – (5.28). These vectors are polynomially many (with respect to the size of Δ) and each is $\leq \boldsymbol{C}$, therefore all possible guesses can be tried in exponential time. If such vectors do not exist, fail. Otherwise, write the system $\mathcal{E}$ as in Section 5.3. If $\mathcal{E}$ has a solution over $\mathbb{N}$ succeed, otherwise fail. The existence (or not) of a model based on whether $\mathcal{E}$ has a solution (or not, respectively) is due to Lemmas 5.13 and 5.16. The time bound follows from Lemma 5.17. $\square$

The hardness for FinSat($\mathcal{GC}^2\mathcal{D}$) follows from the fact that $\mathcal{GC}^2$ is EXPTIME-hard. Thus, FinSat($\mathcal{GC}^2\mathcal{D}$) is EXPTIME-complete.

We now turn to 'general' satisfiability, and show that Sat($\mathcal{GC}^2\mathcal{D}$) is EXPTIME-complete. The approach is the same as in [PH07], so we just give a quick sketch for completeness.

The main goal here is to reduce $\mathcal{E}$ into a satisfiability problem for a system (conjunction) of (exponentially many) Horn clauses, as such systems are known to be solvable in time polynomial in the size of the input—thus leaving us in EXPTIME. To reduce $\mathcal{E}$ into this 'binary' system, we first view it as a system in the set $\mathbb{N} \cup \{\aleph_0\}$—where addition and multiplication are extended in the obvious way, i.e. $\aleph_0 + \aleph_0 = \aleph_0 \cdot \aleph_0 = \aleph_0$, $n + \aleph_0 = \aleph_0 + n = \aleph_0$, for all $n \in \mathbb{N}$, etc. We then observe that any solution to $\mathcal{E}$ remains a solution if all of the non-zero values that it assigns (to variables of $\mathcal{E}$) are replaced with $\aleph_0$. But, in that case, $\mathcal{E}$ becomes a system over the set $\{0, \aleph_0\}$. Such a system is essentially boolean, so its constraints can be viewed as formulas of propositional logic; and, with a little care, they can be written as Horn clauses. For instance, identifying 0 with 'false' and $\aleph_0$ with 'true', the constraints corresponding to the database ($\mathcal{E}_4$), which are of the form $x \geq c$ (where c is a non-zero constant), become $\top \rightarrow x$—i.e. x must be non-zero. It is easily seen that all the relevant lemmas remain valid in the 'infinite' case. This establishes membership of Sat($\mathcal{GC}^2\mathcal{D}$)—and, thus, Sat($\mathcal{GC}^2\mathcal{DP}^2$)—in EXPTIME, and the hardness follows as before.

6 | Satisfiability for $\mathcal{GC}^2\mathcal{DP}^2$

In this chapter we establish that (finite) satisfiability for the two-variable guarded fragment with counting quantifiers in the presence of *binary* path-functional dependencies and a database, denoted $\mathcal{GC}^2\mathcal{DP}^2$, is EXPTIME-complete. (We also provide a quick proof of the same result but for the two-variable guarded fragment with counting quantifiers in the presence of *unary* path-functional dependencies and a database, denoted $\mathcal{GC}^2\mathcal{DP}^1$. This result is almost trivial compared to the one for $\mathcal{GC}^2\mathcal{DP}^2$.) The proof is by (a rather complex) reduction to (finite) satisfiability for $\mathcal{GC}^2\mathcal{D}$, which was shown to be EXPTIME-complete in the previous chapter.

Central to our approach is the notion of a *path* and the closely related notion of a *tour*. The notion of a path is analogous to the widely used notion of a *walk* in graph theory; a walk on a graph G is simply a sequence of vertices such that each consecutive pair of these vertices is connected by an edge in G. (Walks are also referred to as *chains* or *paths* in the graph-theoretic literature, although the term 'path' is usually reserved for walks that contain no repeated vertices.) The notion of a tour is analogous to that of a *closed walk* in graph theory. (Closed walks are also referred to as *cycles* in the graph-theoretic literature, although the term 'cycle' is usually reserved for closed walks containing no repeated vertices except their first vertex.)

Our approach is based on the identification of path-functional dependency violations with certain kinds of tours. That is, if a dependency is violated in a given structure $\mathfrak{A}$, then we are sure to find in $\mathfrak{A}$ a tour whose shape is among ten possible shapes that only depend on the dependency. Each of those ten shapes corresponds to a different way to decompose a given tour into tree-shaped subtours called *fans* or larger configurations called *isthmuses* (plus its part inside the database, if the tour includes database elements). Although the term 'fan' is new, the use of trees is extensive in the literature of the guarded fragment and/or query answering; trees are used among other things to obtain decidability results (see, e.g., [Lad77, Kaz04]), decision procedures (see, e.g., [SSS91, BBH96, DGHP13]), and for query rewriting (see, e.g., [PH09]). The notion of an isthmus, on the other hand, is completely new. An isthmus is a versatile technical device enabling us to express that a certain property holds at an element arbitrarily far from a given element.

It is useful, at this point, to discuss a possible alternative for the above approach using conjunctive queries. Since, as mentioned above, checking path-functional dependency violations amounts to the detection (exclusion) of certain tours in any given structure, one may naturally consider expressing such tours as conjunctive queries and express the requirement that those tours do not occur in any model of a database Δ and a $\mathcal{GC}^2$-sentence φ as a conjunctive query non-entailment. In mode detail, for any given tour, say $\bar{r}$, comprising the predicates $r_1, \ldots, r_k$ one can write a conjunctive query

$$\kappa_{\bar{r}} := \exists x_1 \cdots \exists x_k \, r_1(x_1, x_2) \wedge r_2(x_2, x_3) \wedge \cdots \wedge r_{k-1}(x_{k-1}, x_k),$$

such that any structure $\mathfrak{A}$ that satisfies $\kappa_{\bar{r}}$ contains at least one such tour—and vice versa. Then, to decide whether Δ, φ has a model that contains no '$\bar{r}$-tours' one need only decide whether $\Delta, \varphi \not\models \kappa_{\bar{r}}$. (It is easily checked that $\Delta, \varphi, \neg\kappa_{\bar{r}}$ is unsatisfiable if and only if $\Delta, \varphi \models \kappa_{\bar{r}}$.)

This approach fails for the following reason: we will see later on that the violation of a given path-functional dependency requires not only the existence of certain tours, but also that the first three elements of any such tour be *distinct*. To express this requirement as a conjunctive query one needs to use inequalities; however, it is known that conjunctive query answering with even one inequality is undecidable [GBIGKK13].

6.1 Preliminaries

Before focusing on path-functional dependencies, let us give a simple—and maybe surprising—lemma about replacing three-variable guarded formulas of a certain form (arising often in the sequel) with equivalent two-variable guarded formulas with counting quantifiers. For the rest of this section, let $\mathsf{dst}(x, y, z)$ (for distinct) be an abbreviation for the formula $x \neq y \wedge y \neq z \wedge x \neq z$.

Lemma 6.1. Consider the formulas

$$\begin{aligned}
\varphi_1(x) &:= \exists y \exists z(\mathsf{dst}(x,y,z) \wedge \alpha(x,y) \wedge \beta(y,z)),\\
\varphi_2(y) &:= \exists x \exists z(\mathsf{dst}(x,y,z) \wedge \alpha(x,y) \wedge \beta(y,z)),
\end{aligned}$$

where $\alpha(x,y)$ and $\beta(y,z)$ are conjunctions of literals, containing at least one atom, with free variables x, y and y, z respectively. Then, we can compute in polynomial time $\mathcal{GC}^2$-formulas $\varphi_1^*(x)$ and $\varphi_2^*(x)$ that are logically equivalent to $\varphi_1(x)$ and $\varphi_2(x)$ respectively.

Proof. Let

$$\begin{aligned}
\varphi_1^*(x) &:= \forall x\, \neg\alpha(x,x) \wedge \forall x\, \neg\beta(x,x) \wedge\\
&\qquad (\exists y(\alpha(x,y) \wedge \neg\beta(y,x) \wedge \exists x \beta(y,x))\\
&\qquad\quad \vee \exists y(\alpha(x,y) \wedge \beta(y,x) \wedge \exists_{\geq 2} x \beta(y,x))),
\end{aligned}$$

$$\begin{aligned}
\varphi_2^*(y) &:= \forall x\, \neg\alpha(x,x) \wedge \forall x\, \neg\beta(x,x) \wedge\\
&\qquad [(\exists x(\alpha(x,y) \wedge \neg\beta(y,x)) \wedge \exists x \beta(y,x))\\
&\qquad\quad \vee \exists x(\alpha(x,y) \wedge \beta(y,x)) \wedge \exists_{\geq 2} x \beta(y,x)].
\end{aligned}$$

It is evident that, for any structure $\mathfrak{A}$ interpreting the signature of φ_1, and any $a \in A$, $\mathfrak{A} \models \varphi_1(a)$ if and only if $\mathfrak{A} \models \varphi_1^*(a)$; and, likewise, for $\varphi_2(x)$ and $\varphi_2^*(x)$. □

In the sequel, we consider signatures featuring a (possibly empty) distinguished subset of binary predicates, which we refer to as *functional predicates*. Functional predicates are to be interpreted as graphs of irreflexive functions. That is, if f is a functional predicate, then, in any structure $\mathfrak{A}$ interpreting it, we require $\mathfrak{A} \models \forall x \exists_{\leq 1} y\, f(x,y)$ and $\mathfrak{A} \models \forall x\, \neg f(x,x)$. Notice that the previous definition allows functional predicates to be interpreted as partial functions. To simplify the ensuing discussion, we take functional predicates to denote only *total* functions. This restriction comes at no cost: partial functions can be 'simulated' with total functions over domains featuring new 'dummy' objects. The requirement for functional predicates to be irreflexive, however, is not so easily eliminated, but it is (in most cases) natural. For example, suppose that we use the binary predicate $\mathsf{age}(\cdot,\cdot)$ to assign ages to individuals in a database. That is, we write $\mathsf{age}(e,n)$ to denote that the entity (constant) e has age n. In that case, it is sensible to require age to be a non-reflexive functional predicate, because nobody has more than one age, and nobody is identical to their age.

In addition, for each functional predicate f in a signature τ, we assume that τ contains new binary predicate f^{-1}, referred to as the *inverse* of f, subject to the requirement that $\mathfrak{A} \models \forall x \forall y(f(x,y) \leftrightarrow f^{-1}(y,x))$, for any structure τ-$\mathfrak{A}$. (Observe that this formula is in $\mathcal{GC}^2$.) Observe, incidentally, that, if f is a functional predicate, f^{-1} need not be a functional predicate; however, it must be irreflexive.

Notation 6.2. Let $\bar{f} = f_0, \ldots, f_{k-1}$ and $\bar{f}' = f'_0, \ldots, f'_{\ell-1}$ be two sequences of functional predicates. The concatenation $f_0, \ldots, f_{k-1}, f'_0, \ldots, f'_{\ell-1}$ of $\bar{f}$ and $\bar{f}'$ is denoted by $\bar{f}\bar{f}'$. The empty sequence is denoted by ϵ. If $\bar{f} = f_0, \ldots, f_{k-1}$ is a sequence of functional predicates, then its *inverse*, denoted $\bar{f}^{-1}$, is the sequence $f^{-1}_{k-1}, \ldots, f^{-1}_0$.

In the following parts, we use some standard graph-theoretic terminology. A directed graph G is a set $\{V, E\}$, where V is a set of vertices and $E = \{R_1, \ldots, R_n\}$ is a family of sets of edges connecting elements of V, i.e. $R_i \subseteq V \times V$, for $1 \leq i \leq n$. A directed graph, then, is a structure over the relational signature $\langle R_1, \ldots, R_n \rangle$. For this reason, we often use graph theoretic terminology when it makes our presentation more intuitive. (For instance, we may refer to elements of a structure as vertices and predicates as edges.)

Referring to Definition 4.19, the elements that constitute a cycle are distinct. (Recall from Definition 4.19 that a cycle consists of *at least three* elements.) We now define the more general notion of a tour—as we will see later, violations of path-functional dependencies can be identified with certain kinds of tours. A tour (like a cycle) is a sequence of elements in a structure that 'revisits' its origin. The elements of a tour, however, need not be distinct. Thus, we can have tours on acyclic structures (but no cycles, of course); and, when the underlying structure is acyclic, we can succinctly 'capture' violations of path-functional dependencies by utilizing certain $\mathcal{GC}^2$-formulas (to be introduced later).

Definition 6.3. Let $\mathfrak{A}$ be a structure and let $\bar{r} = r_0, \ldots, r_{\ell-1}$, $\ell \geq 0$, be a sequence of binary predicates. An *$\bar{r}$-path* in $\mathfrak{A}$ is a sequence $\bar{a} = a_0, \ldots, a_\ell$ of elements in A such that $\mathfrak{A} \models r_i(a_i, a_{i+1})$, for $0 \leq i < \ell$. The *length* of the above path is ℓ, which is the number of edges that it contains. A *simple* path is a path with no repeated vertices. An *$\bar{r}$-tour* in $\mathfrak{A}$ is an $\bar{r}$-path whose endpoints are the same. That is, an $\bar{r}$-path $\bar{a} = a_0, \ldots, a_\ell$ is a tour if and only if $a_0 = a_\ell$. A path (or tour) $a_0, \dots, a_\ell$ is *tree-shaped* if the graph $(\{a_0, \dots, a_\ell\}, \{\{a_i, a_{i+1}\} \mid 0 \le i < \ell\})$ is acyclic.

From the above definition, any single vertex a is the start (and end) of a tour, the ϵ-tour, where ϵ denotes the empty sequence.

Convention 6.4. Given a tour $\bar{a} = a_0, \dots, a_{\ell-1}, a_\ell$, with $a_0 = a_\ell$, we omit the last vertex, a_ℓ, writing $\bar{a}$ as $a_0, \dots, a_{\ell-1}$ instead. Thus, the length of the *tour* $a_0, \dots, a_{\ell-1}$ is ℓ and its last element is a_ℓ $(= a_0)$.

Convention 6.5. When talking about sequences of predicates, paths or tours, we typically omit the commas for readability. Thus, we might talk about the sequence $t_1t_2t_3$ of predicates when we actually mean the sequence t_1, t_2, t_3 or the path $a_1a_2a_3a_4$ when we actually mean the path a_1, a_2, a_3, a_4.

Let us now turn our attention to the study of path-functional dependencies. We first need the notion of path convergence. Let $\bar{h} = h_0, \dots, h_{k-1}$ be a sequence of functional predicates. We say that two $\bar{h}$-paths $a_0, \dots, a_k$ and $b_0, \dots, b_k$ *converge* if there is an $i \le k$ such that $a_i = b_i$. When two paths $\bar{a}$ and $\bar{b}$ converge, we write $\bar{a} \bowtie \bar{b}$.

As a warm-up, we discuss unary path-functional dependencies first. The guarded two-variable fragment with counting quantifiers, a database, and *unary* path-functional dependencies—denoted $\mathcal{GC}^2\mathcal{DP}^1$—is basically $\mathcal{GC}^2\mathcal{D}$, extended with syntactic annotations for specifying unary path-functional dependencies. In more detail, if Δ is a database and ψ is a $\mathcal{GC}^2$-formula, $\mathcal{GC}^2\mathcal{DP}^1$ allows the formation of the expression

$$\Delta, \psi : \mathrm{PFD}[\bar{f}]$$

where $\bar{f}$ is a non-empty sequence of functional predicates, possibly appearing in Δ and/or ψ, over a relational signature τ.[1]

[1]A few remarks on our notation (which also apply to the binary case below): The use of a colon separating the database and background theory from the dependency resembles the way dependencies are specified in [TW08]. A different approach would be to allow path-functional dependencies to be regular formulas, interpreted in a particular way. The second approach would make them more familiar, but would allow them to appear anywhere in arbitrarily nested formulas, when real-world applications only require them to be used at the top level, next to Δ and ψ. One can provide a lemma showing that dependencies can be 'extracted' to the top level, but this is a level of indirection that we felt was aesthetically unsuitable. For this reason we chose the above notation, which enforces the dependencies to be specified at the top level, together with Δ and ψ, at the cost of a slightly unfamiliar syntax.

To define the semantics of such an expression, we introduce the following notation: for a given τ-structure $\mathfrak{A}$, if $\bar{h}$ is a sequence of functional predicates, we denote by $\bar{h}^{\mathfrak{A}}(a)$ the $\bar{h}$-path starting at $a \in A$. (Recall that, for simplicity, we assume that functional predicates are interpreted as graphs of *total* functions, hence there is a unique $\bar{h}$-path starting at a.) We say that Δ, ψ is *satisfiable under* the unary path-functional dependency $\mathrm{PFD}[\bar{f}]$ if there exists a τ-structure $\mathfrak{A}$, such that $\mathfrak{A} \models \Delta$, $\mathfrak{A} \models \psi$, and, for all $a, b \in A$, $\bar{f}^{\mathfrak{A}}(a) \bowtie \bar{f}^{\mathfrak{A}}(b)$ implies $a = b$; in that case, we write $\mathfrak{A} \models \Delta, \psi : \mathrm{PFD}[\bar{f}]$. We speak of *finite* satisfiability when in the above definition we impose the extra condition that $\mathfrak{A}$ be finite. If, for a given τ-structure $\mathfrak{A}$, there are two *distinct* elements $a, b \in A$ such that $\bar{f}^{\mathfrak{A}}(a) \bowtie \bar{f}^{\mathfrak{A}}(b)$, we say that the path-functional dependency $\mathrm{PFD}[\bar{f}]$ is *violated* in $\mathfrak{A}$. Each such pair of elements a, b is called a *violating pair* for $\mathrm{PFD}[\bar{f}]$.

$\mathcal{GC}^2\mathcal{DP}^1$ also allows the specification of multiple dependencies. If, for a positive integer m, $\bar{f}_1, \dots, \bar{f}_m$ are pairs of sequences of functional predicates, the expression

$$\Delta, \psi : \mathrm{PFD}[\bar{f}_1], \dots, \mathrm{PFD}[\bar{f}_m]$$

is in $\mathcal{GC}^2\mathcal{DP}^1$, and it dictates that Δ, ψ be satisfiable under each path-functional dependency $\mathrm{PFD}[\bar{f}_i]$ $(1 \le i \le m)$ simultaneously. We will not concern ourselves with this case—it is trivial to generalize the following approach to handle multiple dependencies.

Let τ be a relational signature and let $\bar{f}$ be a non-empty sequence of functional predicates from τ. Suppose that we want to decide whether $\Delta, \psi : \mathrm{PFD}[\bar{f}]$ is (finitely) satisfiable. Let τ' consist of τ together with the predicates $\mathsf{path}_1\langle\bar{g}\rangle(\cdot)$ and $\mathsf{path}_2\langle\bar{g}\rangle(\cdot)$, one for each *contiguous* subsequence of $\bar{f}$. Abbreviating $\mathrm{PFD}[\bar{f}]$ as $\wp$, and letting $\bar{h} \lhd \bar{h}'$ denote that $\bar{h}$ is a contiguous subsequence of $\bar{h}'$, we define the sets

$$\begin{aligned} P^1_{\wp} = {} & \{\forall x\, \mathsf{path}_1\langle\epsilon\rangle(x)\} \cup \\ & \{\forall x(\mathsf{path}_1\langle g\bar{g}\rangle(x) \leftrightarrow \exists y(x \neq y \wedge g(x,y) \wedge \mathsf{path}_1\langle\bar{g}\rangle(y))) \mid g\bar{g} \lhd \bar{f}\} \end{aligned}$$

and

$$\begin{aligned} P^2_{\wp} = {} & \{\forall x\, \mathsf{path}_2\langle\epsilon\rangle(x)\} \cup \\ & \{\forall x(\mathsf{path}_2\langle\bar{g}g\rangle(x) \leftrightarrow \exists y(x \neq y \wedge \mathsf{path}_2\langle\bar{g}\rangle(y) \wedge g(y,x))) \mid \bar{g}g \lhd \bar{f}\}. \end{aligned}$$

Note that the size of the sets $P^1_\wp$ and $P^2_\wp$ is polynomial in the length of $\bar{f}$. Intuitively, in the presence of $P^1_\wp$ and $P^2_\wp$, $\mathsf{path}_1\langle\bar{g}\rangle(x)$ is to be read as stating that x is the start of a $\bar{g}$-path and $\mathsf{path}_2\langle\bar{g}\rangle(x)$ as stating that x is at the end of a $\bar{g}$-path. Now, notice that PFD$[\bar{f}]$ ($= \wp$) is violated in a given structure $\mathfrak{A}$ (satisfying $P^1_\wp$ and $P^2_\wp$) if and only if $\mathfrak{A} \models \exists y\, \chi(y)$, where

$$\begin{aligned}\chi(y) := \bigvee_{\bar{f}=\bar{f}'f\bar{f}''} \exists x \exists z(&\mathsf{dst}(x,y,z) \wedge \mathsf{path}_2\langle\bar{f}'\rangle(x) \wedge f(x,y)\\ &\wedge \mathsf{path}_2\langle\bar{f}'\rangle(z) \wedge f(z,y) \wedge \mathsf{path}_1\langle\bar{f}''\rangle(y)).\end{aligned}$$

Indeed, if for some element b in a structure $\mathfrak{A}$ (satisfying the $P^1_\wp$ and $P^2_\wp$) we have $\mathfrak{A} \models \chi(b)$ then there exist two distint elements $a, c \in A$ (and also both different from b) such that there is an ingoing $\bar{f}'$-path to a, an outgoing f-edge from a to b, an ingoing $\bar{f}'$-path to c, an outgoing f-edge from c to b, and an outgoing $\bar{f}''$-path from b. This configuration constitutes, according to our definition, a violation of PFD$[\bar{f}]$. The converse is evident.

Note that the size of $\chi(y)$ is polynomial in the length of $\bar{f}$. Of course, $\chi(y)$ does not belong in $\mathcal{GC}^2$, but, by Lemma 6.1, it can be replaced by an equivalent $\mathcal{GC}^2$-formula $\chi^*(y)$. Thus, (finite) satisfiability for $\Delta, \psi :$ PFD$[\bar{f}]$ reduces to (finite) satisfiability for the tuple

$$\Delta, \psi, \neg\exists y\, \chi^*(y), \bigwedge(P^1_\wp \cup P^2_\wp).$$

We now focus our attention to our main object of study, the guarded two-variable fragment with counting quantifiers, a database, and *binary* path-functional dependencies—denoted $\mathcal{GC}^2\mathcal{DP}^2$. Like $\mathcal{GC}^2\mathcal{DP}^1$, $\mathcal{GC}^2\mathcal{DP}^2$ is basically $\mathcal{GC}^2\mathcal{D}$, extended with syntactic annotations for specifying binary path-functional dependencies. In more detail, if Δ is a database and ψ is a $\mathcal{GC}^2$-formula, $\mathcal{GC}^2\mathcal{DP}^2$ allows the formation of the expression

$$\Delta, \psi : \text{PFD}[\bar{f}, \bar{g}]$$

where $\bar{f}$ and $\bar{g}$ are non-empty sequences of functional predicates, possibly appearing in Δ and/or ψ, over a relational signature τ.

Recall from above that, for a given τ-structure $\mathfrak{A}$, if $\bar{h}$ is a sequence of functional predicates, we denote by $\bar{h}^{\mathfrak{A}}(a)$ the $\bar{h}$-path starting at $a \in A$. (Assuming, again, that functional predicates are interpreted as graphs of *total* functions, there is a unique $\bar{h}$-path starting at a.) We say that Δ, ψ is *satisfiable under* the path-functional dependency $\mathrm{PFD}[\bar{f}, \bar{g}]$ if there exists a τ-structure $\mathfrak{A}$, such that $\mathfrak{A} \models \Delta$, $\mathfrak{A} \models \psi$ and, for all $a, b \in A$, $\bar{f}^{\mathfrak{A}}(a) \bowtie \bar{f}^{\mathfrak{A}}(b)$ and $\bar{g}^{\mathfrak{A}}(a) \bowtie \bar{g}^{\mathfrak{A}}(b)$ implies $a = b$; in that case, we write $\mathfrak{A} \models \Delta, \psi : \mathrm{PFD}[\bar{f}, \bar{g}]$. We speak of *finite* satisfiability when in the above definition we impose the extra condition that $\mathfrak{A}$ be finite. If, for a given τ-structure $\mathfrak{A}$, there are two *distinct* elements $a, b \in A$ such that $\bar{f}^{\mathfrak{A}}(a) \bowtie \bar{f}^{\mathfrak{A}}(b)$ *and* $\bar{g}^{\mathfrak{A}}(a) \bowtie \bar{g}^{\mathfrak{A}}(b)$, we say that the path-functional dependency $\mathrm{PFD}[\bar{f}, \bar{g}]$ is *violated* in $\mathfrak{A}$. Each such pair of elements a, b is called a *violating pair* for $\mathrm{PFD}[\bar{f}, \bar{g}]$.

$\mathcal{GC}^2\mathcal{DP}^2$ also allows the specification of multiple dependencies. If, for a positive integer m, $\bar{f}_1, \bar{g}_1, \ldots, \bar{f}_m, \bar{g}_m$ are pairs of sequences of functional predicates, the expression

$$\psi : \mathrm{PFD}[\bar{f}_1, \bar{g}_1], \ldots, \mathrm{PFD}[\bar{f}_m, \bar{g}_m]$$

is in $\mathcal{GC}^2\mathcal{DP}^2$, and it dictates that ψ be satisfiable under each path-functional dependency $\mathrm{PFD}[\bar{f}_i, \bar{g}_i]$ $(1 \leq i \leq m)$ simultaneously. That is,

$$\mathfrak{A} \models \psi : \mathrm{PFD}[\bar{f}_1, \bar{g}_1], \ldots, \mathrm{PFD}[\bar{f}_m, \bar{g}_m]$$

if there are no two *distinct* elements $a, b \in A$ such that, for some i $(1 \leq i \leq m)$, $\bar{f}_i^{\mathfrak{A}}(a) \bowtie \bar{f}_i^{\mathfrak{A}}(b)$ and $\bar{g}_i^{\mathfrak{A}}(a) \bowtie \bar{g}_i^{\mathfrak{A}}(b)$. We do not concern ourselves with violations of multiple dependencies: it is straightforward to adapt the apparatus we develop in the sequel for (single) dependencies $\mathrm{PFD}[\bar{f}, \bar{g}]$ so that multiple dependencies are taken into account.

Suppose that a path-functional dependency $\mathrm{PFD}[\bar{f}, \bar{g}]$ is violated in a structure $\mathfrak{A}$, and let $a, b \in A$ be a violating pair for $\mathrm{PFD}[\bar{f}, \bar{g}]$. Thus, $\bar{f}^{\mathfrak{A}}(a) \bowtie \bar{f}^{\mathfrak{A}}(b)$ and $\bar{g}^{\mathfrak{A}}(a) \bowtie \bar{g}^{\mathfrak{A}}(b)$, i.e. the $\bar{f}$-paths $a_0 (= a), \ldots, a_k$ and $b_0 (= b), \ldots, b_k$ converge; and, similarly, for $\bar{g}(a)$ and $\bar{g}(b)$. Then, there exists (according to our definition) a smallest $i \leq k$ such that $a_i = b_i$. (Clearly $i > 0$, since a and b are distinct.) We say that the violating pair a, b is *critical* if $i = k$, i.e. when $\bar{f}(a)$ and $\bar{f}(b)$ converge only at their last element. We say that a path-functional dependency $\mathrm{PFD}[\bar{f}, \bar{g}]$ is *critically violated* in a structure $\mathfrak{A}$ if it is violated in $\mathfrak{A}$ and at least one of its violating pairs is critical. Note that after the two paths converge, their 'behaviour' is identical. See Figure 6.1 for an illustration.

Remark 6.6. There is no requirement that all the elements of the paths mentioned above be distinct, so the graph that those sequences specify might not

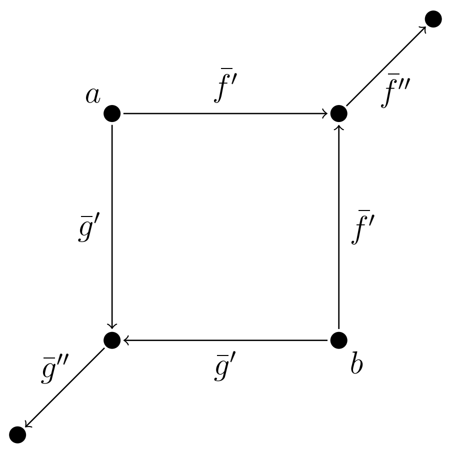


Figure 6.1: A violating pair a, b for a path-functional dependency PFD$[\bar{f}, \bar{g}]$. Note that the arrows denote sequences of functional predicates and not single (directed) edges. In that context, $\bar{f} = \bar{f}'\bar{f}''$ and $\bar{g} = \bar{g}'\bar{g}''$. Note, also, that $\bar{f}''$ and $\bar{g}''$ can be empty. The pair a, b is critical just in case $\bar{f}'' = \epsilon$.

necessarily have cycles. Thus, the illustration in Figure 6.1 is an abstraction and should not be conflated with the actual structure to which the elements of these sequences belong.

We claim that a method for deciding whether a path-functional dependency is *critically* violated in a structure $\mathfrak{A}$, is sufficient for deciding if a given dependency PFD$[\bar{f}, \bar{g}]$ is violated in $\mathfrak{A}$. Indeed, suppose $\bar{f} = f_1, \ldots, f_k$ and denote by $\bar{f}_{1\,..\,i}$ ($i \leq k$) the prefix $f_1, \ldots, f_i$ of $\bar{f}$. It is easily seen that if PFD$[\bar{f}, \bar{g}]$ is violated, then PFD$[\bar{f}_{1\,..\,i}, \bar{g}]$ is critically violated, for at least one i ($0 < i \leq k$). Then, to check if PFD$[\bar{f}, \bar{g}]$ is violated in a structure $\mathfrak{A}$, one checks if PFD$[\bar{f}_{1\,..\,i}, \bar{g}]$ is critically violated in $\mathfrak{A}$, for each i ($0 < i \leq k$). If none of the dependencies PFD$[\bar{f}_{1\,..\,i}, \bar{g}]$ is critically violated, it is clear that PFD$[\bar{f}, \bar{g}]$ is not violated. Conversely, if any of the dependencies PFD$[\bar{f}_{1\,..\,i}, \bar{g}]$ is critically violated, then PFD$[\bar{f}, \bar{g}]$ is violated (recall from the beginning of this section that functional predicates are always interpreted as graphs total functions). For this reason, we will be concerned *only* with the detection of critical violations.

Suppose that the path-functional dependency PFD$[\bar{f}f, \bar{g}]$[2] is critically violated in a structure $\mathfrak{A}$, and let $a, b \in A$ be any violating pair for PFD$[\bar{f}f, \bar{g}]$. Then, a, b are distinct, there exist two converging $\bar{f}f$-paths starting at a, b respectively, and two converging $\bar{g}$-paths starting, again, at a, b respectively. Let d be the

[2]Since, by definition, the sequences involved in a path-functional dependency are non-empty, we may assume that each dependency is of this form, where $\bar{f}$ is allowed to be the empty sequence ϵ.

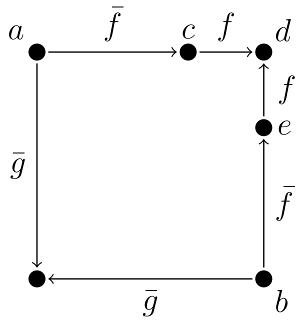


Figure 6.2: A critical violating tour of PFD$[\bar{f}f, \bar{g}]$. Note that the arrows labeled with $\bar{f}$ or $\bar{g}$ denote sequences of functional predicates and not single (directed) edges. The elements c, d and e are *distinct.*

element on which the two $\bar{f}f$-paths converge; let c be the penultimate element of the $\bar{f}f$-path $a, \ldots, d$ and e be the penultimate element of the $\bar{f}f$-path $b, \ldots, d$. Observe that the elements c, d, and e must be distinct, otherwise the violation would not be critical. See Figure 6.2 for an illustration.

Thus, referring again to Figure 6.2, each critical violation of PFD$[\bar{f}f, \bar{g}]$ in $\mathfrak{A}$ can be identified with an $\bar{f}ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}$-tour $a \cdots cde \cdots b \cdots a$, where $a \cdots c$ is an $\bar{f}$-path in $\mathfrak{A}$, $\mathfrak{A} \models f(c,d)$, $\mathfrak{A} \models f^{-1}(d,e)$, $e \cdots b$ is an $\bar{f}^{-1}$-path in $\mathfrak{A}$, and $b \cdots a$ is a $\bar{g}\bar{g}^{-1}$-path in $\mathfrak{A}$—where the elements c, d, and e are distinct. This observation turns out not to be very useful. However, identifying critical violations of PFD$[\bar{f}f, \bar{g}]$ with tours is not a bad idea. The following insight lays the foundation for our approach: any rotation of the above tour still identifies a critical violation of PFD$[\bar{f}f, \bar{g}]$. Moreover, all rotations of the above tour share a common characteristic: they all contain the ff^{-1}-path cde (where c, d, and e are distinct). It looks promising, then, to take the rotation of the above tour *beginning at c* as our starting point. That is, we identify any violation of PFD$[\bar{f}f, \bar{g}]$ with an $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour whose first three elements, say c, d, and e, are distinct. It is clear (keeping in mind Figure 6.2) that if no such tour exists in a given structure $\mathfrak{A}$, then PFD$[\bar{f}f, \bar{g}]$ is not violated in $\mathfrak{A}$. (Notice that the violating pair a, b is irrelevant at this point: the existence of the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour starting with c, d, and e is a necessary condition for a, b to be a violating pair.)

Next, we concern ourselves with the classification of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tours according to their decompositions into smaller, tree-shaped subtours.

6.2 Decompositions of paths and tours

Let G be a directed graph and let $\bar{a} = a_0, \ldots, a_\ell$ ($\ell > 0$) be a tree-shaped path in G. Then, $\bar{a}$ can be decomposed as $b_0\bar{b}_0 \cdots b_k\bar{b}_k$ ($k > 0$), with $b_i\bar{b}_i = a_{\iota(i)}, \ldots, a_{\iota'(i)}$ ($0 \leq i \leq k$), where $a_{\iota(0)} = a_0$, $a_{\iota(i)} = a_{\iota'(i-1)+1}$ when $i > 0$, and $\iota'(i)$ is the *largest* index such that $a_{\iota'(i)} = a_{\iota(i)}$; each $b_i\bar{b}_i$ is a tree-shaped subtour of $\bar{a}$. Intuitively, suppose that $a_0 = v \in G$; then the path 'visits' other vertices of G before (possibly) returning again to v. In that case, $\iota'(0)$ 'records' the last time that the path 'visits' v. Then the path 'moves' to $a_{\iota(1)} = a_{\iota'(0)+1} = u \in G$ (which is different to v); similarly, $\iota'(1)$ 'records' the last time that the path visits u. Then the path moves to a different vertex $w \in G$, and so on.

For the above decomposition of $\bar{a}$, we call the sequence $b_0 \cdots b_k$ (i.e. $b_i = a_{\iota(i)}$ for $0 \leq i \leq k$) the *backbone* of $\bar{a}$. It is clear that all the vertices in a backbone are distinct, thus the backbone of any path is a simple path. It is also clear that the backbone of $\bar{a}$ is the shortest subpath of $\bar{a}$ connecting a_0 to a_ℓ and it is unique. See Figure 6.3 for an illustration.

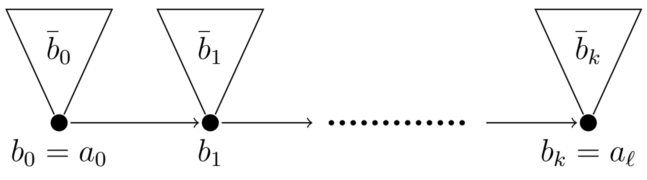


Figure 6.3: A decomposition of a path $a_0, \ldots, a_\ell$ into its backbone $b_0, \ldots, b_k$, (possibly) with a tree-shaped subtour at each b_i, $0 \leq i \leq k$.

Lemma 6.7. Let G be a directed graph and $\bar{a} = a_0, \ldots, a_{\ell-1}$ ($\ell > 1$) be a tree-shaped tour in G. Let $\bar{a}' = a_0, \ldots, a_t$ and $\bar{a}'' = a_t, \ldots, a_\ell$ ($= a_0$), where $a = a_t$ is any vertex in $\bar{a}$ not equal to a_0. Let $\bar{p}' = b'_0, \ldots, b'_m$ be the backbone of $\bar{a}'$ and let $\bar{p}'' = b''_0, \ldots, b''_n$ be the backbone of $\bar{a}''$. Then $b''_i = b'_{m-i}$, for all $0 \leq i \leq m$ (whence $m = n$).

Proof. Suppose that the conclusion of the lemma does not hold. We claim that the longer of the two backbones must introduce a cycle. Without loss of generality, we assume that $m < n$. Let $i \geq 0$ be the smallest index such that $b''_i = b'_{m-i}$ and $b''_{i+1} \neq b'_{m-(i+1)}$. Let $j \geq i + 1$ be the smallest index such that b''_j belongs to $\bar{p}'$ and, for all k with $j > k > i$, b''_k is not in $\bar{p}'$; the set of such b''_k elements is empty when $j = i + 1$. (Such a j exists since $b''_n = b'_0$.) Let s be such that $b''_j = b'_s$; i.e. s

is the corresponding index of b''_j in $\bar{p}'$. First, note that s cannot be greater than or equal to $m - i$ because $\bar{p}''$, being a backbone, is a simple path—hence, it cannot 'revisit' any vertices. Furthermore, if $j = i + 1$ we have $s \neq m - (i + 1)$, because we assumed that $b''_{i+1} \neq b'_{m-(i+1)}$. But, then, in any case, the tour contains the cycle $b''_i, \ldots, b'_s\,(= b''_j), \ldots, b'_{m-i}\,(= b''_i)$; hence it is not tree-shaped, which is a contradiction. See Figure 6.4 for an illustration. □

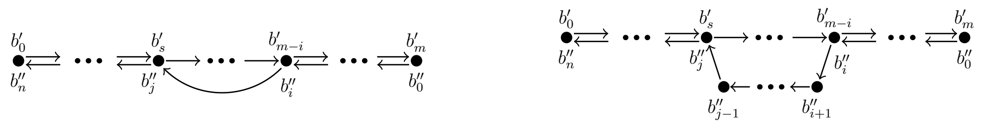


Figure 6.4: Two possible ways that cycles are introduced if the conclusion of Lemma 6.7 does not hold. In both cases $b''_i, \ldots, b'_s\,(= b''_j), \ldots, b'_{m-i}\,(= b''_i)$ is a cycle, which conflicts with the assumption that the tour is tree-shaped.

If $\bar{a}$ is a tree-shaped tour going through a fixed vertex a, we can write $\bar{a} = \bar{a}'\bar{b}\bar{a}''$, where $\bar{b}$ is a tree-shaped tour starting (and ending) at a, and a does not appear in $\bar{a}'$ or $\bar{a}''$. Now, $\bar{a}'a$ is the path from a_0 to a, and $\bar{b}\,\bar{a}''$ is the path from a back to a_0. As discussed earlier, we can write

$$\bar{a}'a = b'_0\bar{b}'_0 \cdots b'_m\bar{b}'_m, \tag{6.1}$$

for some $m > 0$, such that: $b'_i\bar{b}'_i = a_{\iota(i)}, \ldots, a_{\iota'(i)}$ $(0 \leq i \leq m)$, where $a_{\iota(0)} = a_0$, $a_{\iota(i)} = a_{\iota'(i-1)+1}$ when $i > 0$, and $\iota'(i)$ is the largest index such that $a_{\iota'(i)} = a_{\iota(i)}$. Notice that $\bar{b}'_m$ in this decomposition is *empty*, since a does not appear in $\bar{a}'$ and a is the last element of the path $\bar{a}'a$ (i.e. $b'_m = a$). Similarly, we can write

$$\bar{b}\,\bar{a}'' = b''_0\bar{b}''_0 \cdots b''_n\bar{b}''_n, \tag{6.2}$$

for some $n > 0$, such that: $b''_j\bar{b}''_j = a_{\eta(j)}, \ldots, a_{\eta'(j)}$ $(0 \leq i \leq n)$, where $a_{\eta(0)} = b'_m = a$, $a_{\eta(j)} = a_{\eta'(j-1)+1}$ when $j > 0$, and $\eta'(j)$ is the largest index such that $a_{\eta'(j)} = a_{\eta(j)}$. By Lemma 6.7, $m = n$ and $b''_i = b'_{m-i}$ $(0 \leq i \leq m)$, which leads us to the configuration illustrated in Figure 6.5.

The above decompositions have the following simple but very useful consequence. (Keep in mind Figure 6.5.)

Lemma 6.8. Let G be a directed graph and $\bar{a} = a_0, \ldots, a_{\ell-1}$ be a tree-shaped tour in G. Let a be any vertex in $\bar{a}$ not equal to a_0 and let i, j $(0 < i \leq \ell$,

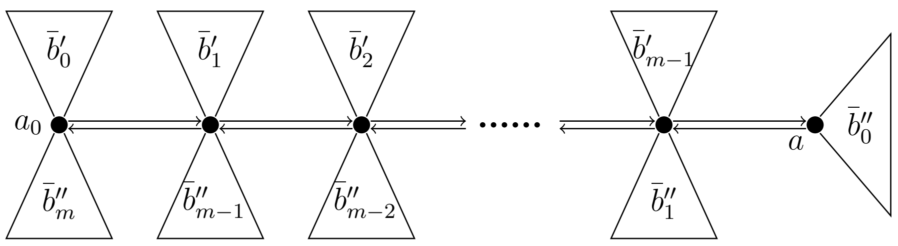


Figure 6.5: The shape of a tree-shaped tour (starting at a_0) that goes through a. The triangles at each node represent a tree-shaped subtour starting (and ending) at this particular node.

$i \leq j < \ell$) be the smallest and the largest index respectively such that $a_i = a_j = a$. Then $a_{i-1} = a_{j+1}$.

Proof. Decompose $\bar{a}$ into $\bar{a}'\,\bar{b}\,\bar{a}''$, where $\bar{b} = a_i, \ldots, a_j$ is a tree-shaped subtour of $\bar{a}$, starting and ending at a. Notice that, by assumption, a does not appear outside $\bar{b}$. Now, decompose the path $\bar{a}'a$ into $b'_0\bar{b}'_0 \cdots b'_m\bar{b}'_m$ as in Equation (6.1). Recall that, in this decomposition, $b'_m = a$ and $\bar{b}'_m$ is empty. Thus, because a does not appear in $\bar{a}'$, it must be $b'_m = a_i$. It follows that $b'_{m-1} = a_{\iota(m-1)} = a_{\iota'(m-1)} = a_{i-1}$—for the last equality, recall that by definition $a_i = a_{\iota'(m-1)+1}$, hence $a_{\iota'(m-1)} = a_{i-1}$. On the other hand, by decomposing $\bar{b}\bar{a}''$ into $b''_0\bar{b}''_0 \cdots b''_n\bar{b}''_n$ as in Equation (6.2), $\bar{b} = a_i, \ldots, a_j = b''_0\bar{b}''_0$. By definition, then, $b''_1 = a_{j+1}$. But, by Lemma 6.7, $m = n$ and $b''_k = b'_{m-k}$ (for all $0 \leq k \leq m$); in particular $b'_{m-1} = b''_1$, thus $a_{i-1} = a_{j+1}$. □

We now wish to consider paths or tours in structures involving constants from a given database. We first introduce some terminology to distinguish the elements that interpret a database constant from those that do not. For the rest of this section, we fix a database Δ, and let D be its active domain. We assume that all the structures we will be working with interpret Δ. Since Δ is fixed, it is in some sense 'locked' or 'hidden' from us; on the other hand, elements that do not interpret constants in D are 'visible', hence the following definition:

Definition 6.9. Let $\mathfrak{A}$ be a structure. An element $a \in A \setminus D$ is called an *observable.* A cycle is *observable* if at least one of its elements is an observable. Otherwise (when all its elements are in D) the cycle is *non-observable.*

We refer to tours that contain database elements (i.e. elements of D) as tours that go inside the database. For such tours we have decompositions similar to the ones above, but before discussing these decompositions we need the following two lemmas. Intuitively, the first lemma states that when a tour that contains no observable cycles goes inside the database, the point of 'entry' must also be the point of 'exit'. The second lemma states that whenever a tour that contains no observable cycles goes out of the database and then in again, the point of 'exit' and the point of 're-entry' are the same. It easy to see that a violation of any of these two lemmas would introduce observable cycles while the tour by assumption cannot contain any.

Lemma 6.10. Let $\mathfrak{A}$ be a structure and $\bar{a} = a_0, \ldots, a_{\ell-1}$, with $a_0 \notin D$ but $\bar{a} \cap D \neq \emptyset$, be a tour in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Let $i > 0$ be the smallest index such that $a_i \in D$. Let j be the largest index such that $a_j \in D$ (but $a_{j+1} \notin D$). Then $a_i = a_j$.

Proof. Since all elements of the database are connected (see Remark 5.3), this is an easy consequence of the fact that $\mathfrak{A}$ contains no observable cycles of length $\leq \ell$. □

Lemma 6.11. Let $\mathfrak{A}$ be a structure and let $\bar{a} = a_0, \ldots, a_{\ell-1}$ be a tour in $\mathfrak{A}$ with $\bar{a} \cap D \neq \emptyset$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Let i be such that $a_i \in D$ (but $a_{i+1} \notin D$). Suppose that there exists $j > i+1$ such that $a_j \in D$ and consider the *smallest* such index. Then $a_i = a_j$.

Proof. Same as Lemma 6.10. □

Let $\bar{a}$ be a tour that goes inside the database, starting at a_0. If $a_0 \notin D$, then there exists an element $a \in D$ through which $\bar{a}$ enters and leaves the database. If $a_0 \in D$, then by definition the tour returns to a_0. Thus, in any case, $\bar{a}$ can be decomposed into $\bar{a}'\,\bar{b}\,\bar{a}''$ (where $\bar{a}', \bar{a}''$ are empty if $a_0 \in D$), where $\bar{b}$ is the part of the tour inside the database, starting (and ending) at a. That is, no database element appears in $\bar{a}'$ or $\bar{a}''$. We decompose $\bar{b}$ into $b_0\bar{b}_0 \cdots b_k\bar{b}_k$ ($k > 0$) as with paths/tours outside the database, but with a small twist: we insist that each $b_i \in D$ ($0 \leq i \leq k$), but no database element appears in any $\bar{b}_i$ ($0 \leq i \leq k$).

Lemma 6.12. Let $\mathfrak{A}$ be a structure and $\bar{a} = a_0, \ldots, a_{\ell-1}$, with $a_0 \in D$, be a tour in $\mathfrak{A}$ (i.e. $a_\ell \in D$). Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Then $\bar{a}$ can be decomposed into $b_0\bar{b}_0 \cdots b_k\bar{b}_k$ (where $b_0 = a_0$ and $b_k = a_\ell$), for some $k \leq \ell$, where each $b_i\bar{b}_i$ ($0 \leq i \leq k$), is a tree-shaped subtour of $\bar{a}$ and $b_i \in D$.

Proof. Refer to Figure 6.6 for an illustration. Let $\bar{p} = b_0, \ldots, b_k$ ($k \geq 1$) be the sequence of elements in $\bar{a}$ that belong to D, ordered by their index in $\bar{a}$

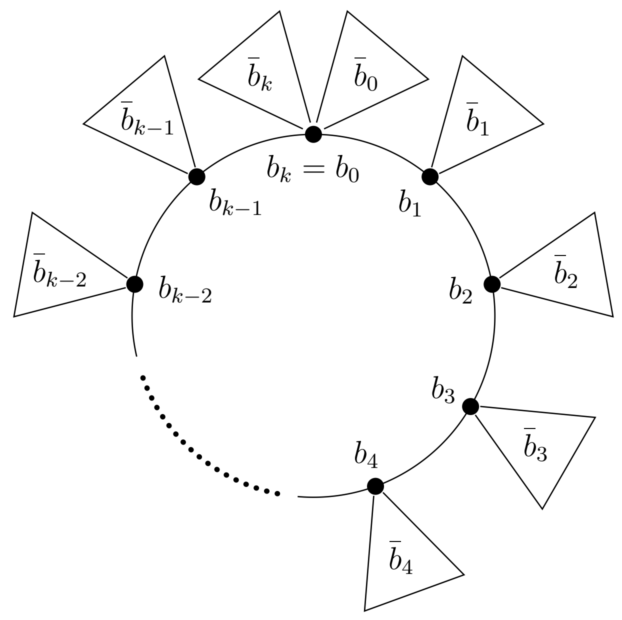


Figure 6.6: The shape of a tour in the database, starting (and ending) at b_0. Each b_i $(0 \leq i \leq k)$ belongs to D; each $b_i\bar{b}_i$ $(0 \leq i \leq k)$ is a tree-shaped tour, with $\bar{b}_i$ possibly being empty.

(ascending)—i.e. $b_0 = a_0$, $b_1 = a_i$, where $i > 0$ is the smallest index such that $a_i \in D$, etc. The elements of $\bar{p}$ need not be distinct. To each b_j $(0 \leq j \leq k)$ we associate the subtour $b_j\bar{b}_j = a_{\iota(j)}, \ldots, a_{\iota'(j)}$, where $\iota(j)$ is the corresponding index to b_j in $\bar{a}$; and, if $a_{\iota(j)+1} \notin D$, $\iota'(j) > \iota(j)$ is the smallest index such that $a_{\iota'(j)} \in D$—otherwise $\bar{b}_j$ is empty. Note that if $a_{\iota(j)+1} \notin D$, since $a_\ell \in D$, an index $\iota'(j) > \iota(j)$ such that $a_{\iota'(j)} \in D$ must exist, and $a_{\iota'(j)} = b_j \, (= a_{\iota(j)})$ by Lemma 6.11. Informally, $b_j\bar{b}_j$ $(0 \leq j \leq k)$ is a tree-shaped 'detour' that starts (and ends) at b_j; it is easily seen that each $b_j\bar{b}_j$ is tree-shaped, as $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. □

Note that exactly the same decomposition can be achieved for paths in the database; hence, Lemma 6.12 holds even if $a_0 \neq a_\ell$.

If $a_0 \notin D$, we obtain a decomposition similar to the one in Figure 6.5:

Lemma 6.13. Let $\mathfrak{A}$ be a structure and $\bar{a} = a_0, \ldots, a_{\ell-1}$, with $a_0 \notin D$ be a tour in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Then $\bar{a}$ can be decomposed into

$$b_0\bar{b}_0 \cdots b_k\bar{b}_k \, \bar{b} \, b_k\bar{b}'_0 \cdots b_0\bar{b}'_k,$$

for some k $(0 < k \leq \ell)$, where each $b_i\bar{b}_i, b_i\bar{b}'_{k-i}$ $(0 \leq i \leq k)$, is a tree-shaped subtour of $\bar{a}$ and $\bar{b}$ is the part of the tour in the database, starting (and ending) at $a \in D$—i.e. no database elements appear outside $\bar{b}$.

Proof. Refer to Figure 6.7 for an illustration. To begin with, write $\bar{a}$ as $\bar{a}'\,\bar{b}\,\bar{a}''$, where $\bar{b}$ is a tour starting (and ending) at $a \in D$, and no database element appears in $\bar{a}'$ or $\bar{a}''$. Now, $\bar{a}'a$ and $a\bar{a}''$ are regular paths, from a_0 to a and from a to a_0 respectively. Further, because $\mathfrak{A}$ has no observable cycles of length $\leq \ell$, these paths must be tree-shaped. Thus, we can decompose them as described earlier into $b_0\bar{b}_0 \cdots b_k\bar{b}_k$ and $b'_0\bar{b}'_0 \cdots b'_{k'}\bar{b}'_{k'}$ respectively, for some $k, k' > 0$ (since $a_0 \notin D$). Then, by Lemma 6.7, $k = k'$ and $b'_i = b_{k-i}$, which yields the required decomposition. $\square$

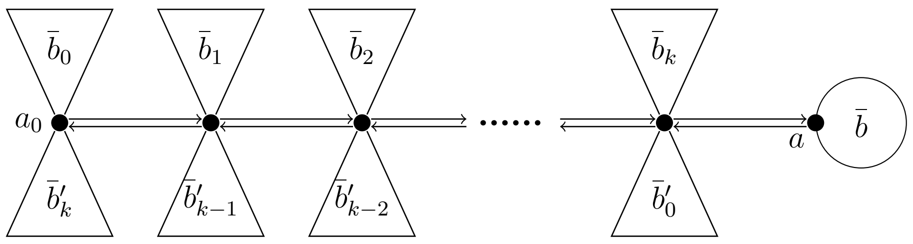


Figure 6.7: A tour starting at a_0 and going inside the database through a, as described in Lemma 6.13. $\bar{b}$ is further decomposed as in Figure 6.6.

Notice that we left $\bar{b}$ untouched, since we showed how to decompose it in Lemma 6.12.

We now know how to perform all the necessary decompositions of tours and paths. Now, notice that the decomposition of paths that we described above in terms of their vertices, also induce decompositions in terms of their edges. Indeed, suppose that we have a $\bar{h}$-path (or tour) $\bar{a} = a_0, \ldots, a_\ell$, where $\bar{h} = h_0, \ldots, h_{\ell-1}$ is a sequence of functional predicates, in a structure $\mathfrak{A}$; that is, $\mathfrak{A} \models h_i(a_i, a_{i+1})$, for all i ($0 \leq i < \ell$). If $\bar{a}$ is decomposed as $b_0\bar{b}_0 \cdots b_k\bar{b}_k$, for some $k \geq 0$, as discussed previously, then each $b_j\bar{b}_j$ ($0 \leq j \leq k$) is a subtour $a_{\iota(j)}, \ldots, a_{\iota'(j)}$ of $\bar{a}$ starting (and ending) at b_j. To each such subtour corresponds the sequence of predicates $\bar{t}_j = h_{\iota(j)}, \ldots, h_{\iota'(j)-1}$ ($0 \leq j \leq k$), i.e. the sequence of predicates that, starting at b_j, lead back to b_j. Note that $\bar{t}_j = \epsilon$ when $\bar{b}_j$ is empty. Then, $t_j = h_{\iota'(j)}$ ($0 \leq j < k$) is the predicate leading from b_j to b_{j+1}, i.e. $\mathfrak{A} \models t_j(b_j, b_{j+1})$. Thus, in terms of predicates, $\bar{a}$ (or $\bar{h}$) is decomposed as $\bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$.

This alternative way of referring to a decomposition of a path or tour is more useful for our purposes, because we are interested in the structure of decompositions (of paths or tours) rather than the individual elements that comprise them.

Notation 6.14. We often abbreviate predicate sequences of the form $\bar{t}_0 t_0 \cdots \bar{t}_k t_k$ ($k \geq 0$) as $\{\bar{t}_i t_i\}_{i=0}^{k}$.

Similarly, a tour starting at an element of D, can be decomposed (in terms of predicates) as $\{\bar{t}_i t_i\}_{i=0}^{k-1}\bar{t}_k$, for some $k \geq 0$, where each $\bar{t}_i$ ($0 \leq i \leq k$) specifies a tree-shaped tour. Further, a tour staring at element not in D but which goes into the database, can be decomposed as $\bar{t}t\,\bar{s}\,t'\bar{t}'$, where $\bar{s}$ is a tour starting at a database element (the 'point of entry') and $\bar{t}t, t'\bar{t}'$ are regular paths outside the database (and, thus, can be decomposed normally, leading to a configuration like the one in Figure 6.7). Then, every tour that goes inside the database (regardless of whether its first element is in D or not) can be decomposed as $\bar{t}t\,\bar{s}\,t'\bar{t}'$—where $\bar{t}t, t'\bar{t}'$ are empty if the tour starts (and ends) at an element of D.

6.3 Predicates related to tours

We now show how to encode in $\mathcal{GC}^2$ two important types of tours: regular tree-shaped tours in a structure $\mathfrak{A}$ (called 'fans'), and tours in $\mathfrak{A}$ that go through a fixed element $a \in A$ for which $\mathfrak{A} \models \omega(a)$, where $\omega(x)$ is a $\mathcal{GC}^2$-formula (such configurations are called 'isthmuses'). Using the $\mathcal{GC}^2$-formulas for fans and isthmuses as 'building blocks', we can detect all possible ways in which a given path-functional dependency may be violated.

Let $\wp = \mathrm{PFD}[\bar{f}f, \bar{g}]$ be a path-functional dependency and σ be a relational signature featuring (possibly among other) the predicates appearing in $\wp$. Let $\sigma^1_\wp$ consist of σ together with the unary predicates $\mathsf{fan}\langle\bar{h}\rangle(\cdot)$, one for each *contiguous* subsequence $\bar{h}$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$. Let

$$\begin{aligned} F_\wp \;=\; \Big\{\forall x\, \mathsf{fan}\langle\epsilon\rangle(x)\Big\} \,\cup\, \Big\{\forall x \Big(&\mathsf{fan}\langle\bar{h}\rangle(x) \leftrightarrow \\ &\bigvee_{\bar{h}=r\bar{r}s\bar{s}} \exists y\,(x \neq y \wedge r(x,y) \wedge \mathsf{fan}\langle\bar{r}\rangle(y) \wedge s(y,x) \wedge \mathsf{fan}\langle\bar{s}\rangle(x))\Big) \\ &\quad \mid \bar{h} \text{ is a } \textit{contiguous} \text{ subsequence of } ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}\Big\}. \end{aligned}$$

Note that the size of F is polynomial in the size of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$: there are $O(n^2)$ contiguous subsequences $\bar{h}$, and, for each of those, $O(n)$ ways to decompose $\bar{h}$

into $r\bar{r}s\bar{s}$, where n is the length of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$.

Lemma 6.15. Let $\mathfrak{A}$ be a structure, let $a \in A$, and suppose that $\mathfrak{A} \models \bigwedge F$. Let $\bar{h} = h_0, \dots, h_{\ell-1}$ be a sequence of binary predicates. If $\mathfrak{A} \models \mathsf{fan}\langle \bar{h} \rangle(a)$ then a is the start of an $\bar{h}$-tour in $\mathfrak{A}$. Conversely, if $\mathfrak{A}$ has no observable cycles of length $\leq \ell$ and a is the start of an $\bar{h}$-tour then $\mathfrak{A} \models \mathsf{fan}\langle \bar{h} \rangle(a)$.

Proof. We prove by induction on the length of $\bar{h}$ that if $\mathfrak{A} \models \mathsf{fan}\langle \bar{h} \rangle(a)$, then a is the start of an $\bar{h}$-tour in $\mathfrak{A}$. If $\bar{h} = \epsilon$, then the result is evident (recall that every single element is the start and of the ϵ-tour). Now, if $\bar{h} \neq \epsilon$, by the conditions in F, we can write $\bar{h} = r\bar{r}s\bar{s}$, for some $\bar{r}$ and $\bar{s}$, such that there exists an $a' \in A$ with $a \neq a'$, $\mathfrak{A} \models r(a, a')$, $\mathfrak{A} \models s(a', a)$, $\mathfrak{A} \models \mathsf{fan}\langle \bar{r} \rangle(a')$ and $\mathfrak{A} \models \mathsf{fan}\langle \bar{s} \rangle(a)$. Thus, by inductive hypothesis, a' is the start (and end) of an $\bar{r}$-tour and a is the start (and end) of an $\bar{s}$-tour. Clearly, then, a is the start (and end) of the tour specified by the above decomposition of $\bar{h}$.

For the converse, suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. We prove again by induction on the length of $\bar{h}$ that if $\bar{a} = a_0, \dots, a_{\ell-1}$ is an $\bar{h}$-tour in $\mathfrak{A}$, then $\mathfrak{A} \models \mathsf{fan}\langle \bar{h} \rangle(a_0)$. (Recall that we omit a_ℓ, which is equal to a_0, by convention.) If $\bar{h} = \epsilon$ (i.e. $\ell = 0$) the required result holds because $\mathfrak{A} \models \mathsf{fan}\langle \epsilon \rangle(a)$, for all $a \in A$, by the conditions in F. Now, suppose that $\bar{h} \neq \epsilon$ (i.e. $\ell > 0$) and let $\bar{a} = a_0, \dots, a_{\ell-1}$ be an $\bar{h}$-tour in $\mathfrak{A}$. Then, $\mathfrak{A} \models h_0(a_0, a_1)$. Let j be the largest index such that $a_j = a_1$. By Lemma 6.8, $a_{j+1} = a_0$; thus, $\mathfrak{A} \models h_j(a_1, a_0)$. Let $\bar{r} = h_1, \dots, h_{j-1}$ be the sequence of predicates that $a_1, \dots, a_{j-1}$ induces. Then, by inductive hypothesis, $\mathfrak{A} \models \mathsf{fan}\langle \bar{r} \rangle(a_1)$. Similarly, let $\bar{s} = h_{j+1}, \dots, h_{\ell-1}$ be the predicates constitute the subtour $a_{j+1}, \dots, a_{\ell-1}$. By inductive hypothesis again, we have $\mathfrak{A} \models \mathsf{fan}\langle \bar{s} \rangle(a_0)$. Evidently, then, by the conditions in F, $\mathfrak{A} \models \mathsf{fan}\langle \bar{h} \rangle(a_0)$, where $\bar{h}$ is decomposed as $h_0\,\bar{r}\,h_j\,\bar{s}$. □

An $(\bar{h}, \omega)$-*isthmus* is a tree-shaped $\bar{h}$-tour $\bar{a}$ in a structure $\mathfrak{A}$, starting at an element $a_0 \in A$ and passing through a fixed $a \in A$ at which a given $\mathcal{GC}^2$-formula $\omega(x)$ holds, i.e. $\mathfrak{A} \models \omega(a)$. Then, similarly with previous decompositions, $\bar{a}$ can be decomposed into

$$b_0\bar{b}_0 \cdots b_{k-1}\bar{b}_{k-1}\,\bar{b}\,b'_1\bar{b}'_1 \cdots b'_k\bar{b}'_k,$$

where $\bar{b}$ is a tree-shaped tour starting (and ending) at a, where a does not appear outside $\bar{b}$, and $b'_{i+1} = b_{k-(i+1)}$ $(0 \leq i < k)$. This decomposition induces on $\bar{h}$ the decomposition

$$\{\bar{r}_i r_i\}_{i=0}^{k-1}\bar{r}_k\,\{\bar{s}_i s_i\}_{i=0}^{k-1}\bar{s}_k,$$

where each $\bar{r}_i, \bar{s}_i$ $(0 \leq i \leq k)$ specifies a tree-shaped subtour, $r_0 \cdots r_{k-1}$ are the predicates constituting the backbone of the path from a_0 to a, and $s_0 \cdots s_{k-1}$ are the predicates constituting the backbone of the path from a back to a_0. Note that we split the sequence of predicates that comprise the tree-shaped subtour $\bar{b}$ into two smaller parts $\bar{r}_k$ and $\bar{s}_0$ for symmetry; if such a split is impossible one can take $\bar{r}_k$ or $\bar{s}_0$ to be ϵ. Thus, we have the decomposition illustrated in Figure 6.8, where $\bar{r}_i$ and $\bar{s}_{i+1}$ $(0 \leq i < k)$ correspond to $b_i\bar{b}_i$ and $b'_{i+1}\bar{b}'_{i+1}$ respectively, and $\bar{r}_k\bar{s}_0$ corresponds to $\bar{b}$.

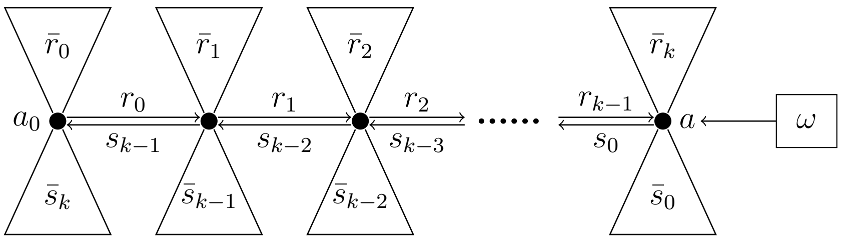


Figure 6.8: An isthmus starting at a_0 and going through a. The box with ω inside it denotes that $\omega(x)$ holds at a.

For a given any $\mathcal{GC}^2$-formula $\omega(x)$, let $\sigma^2_{\wp,\omega}$ consist of $\sigma^1_{\wp}$ together with the predicates $\mathsf{isth}\langle\bar{r}, \omega, \bar{s}\rangle(\cdot)$, one for each pair of distinct *contiguous* subsequences $\bar{r}$ and $\bar{s}$ of $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$. In the following definition (for brevity) we write $\bar{r}, \bar{s} \lhd ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$ to denote that $\bar{r}$ and $\bar{s}$ are distinct contiguous subsequences of $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$. Let

$$\begin{aligned} I_{\wp,\omega} \;=\; \Big\{\forall x\Big[&\mathsf{isth}\langle\bar{r},\omega,\bar{s}\rangle(x) \leftrightarrow \Big((\mathsf{fan}\langle\bar{r}\rangle(x) \wedge \omega(x) \wedge \mathsf{fan}\langle\bar{s}\rangle(x)) \vee \\ &\bigvee_{\substack{\bar{r}=\bar{r}'r\bar{r}'',\\ \bar{s}=\bar{s}''s\bar{s}'}} \Big(\mathsf{fan}\langle\bar{r}'\rangle(x) \wedge \mathsf{fan}\langle\bar{s}'\rangle(x) \wedge \exists y(x \neq y \wedge r(x,y) \wedge s(y,x) \\ &\qquad \wedge\, \mathsf{isth}\langle\bar{r}'',\omega,\bar{s}''\rangle(y))\Big)\Big)\Big] \;:\; \bar{r},\bar{s} \lhd ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}\Big\}. \end{aligned}$$

Note that, for any given ω, the size of $I_{\wp,\omega}$ is polynomial in the size of $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$: there are $O(n^2)$ contiguous subsequences $\bar{r}$ and $O(n^2)$ contiguous subsequences $\bar{r}$; for each pair of those, $O(n)$ ways to decompose $\bar{r}$ into $\bar{r}'r\bar{r}''$ and $O(n)$ ways to decompose $\bar{s}$ into $\bar{s}'s\bar{s}''$—where n is the length of $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$. Thus, the total size of $I_{\wp,\omega}$ is $O(n^6)$, n being the length of $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$.

For the following lemma, it helps to keep in mind Figure 6.8.

Lemma 6.16. Let $\mathfrak{A}$ be a structure, let $a_0 \in A$, $\omega(x)$ be a $\mathcal{GC}^2$-formula, and suppose that $\mathfrak{A} \models \bigwedge(F \cup I_\omega)$. Let $\bar{r}, \bar{s} \lhd ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$, as in the definition of $I_{\wp,\omega}$.

If $\mathfrak{A} \models \mathsf{isth}\langle\bar{r}, \omega, \bar{s}\rangle(a_0)$, then a_0 is the start of an $\bar{r}\bar{s}$-tour $\bar{a}$ in $\mathfrak{A}$ that goes through an element $a \in A$ such that $\mathfrak{A} \models \omega(a)$ and $\bar{a} = \bar{a}'a\bar{a}''$, where $\bar{a}'a$ is an $\bar{r}$-path from a_0 to a and $a\bar{a}''$ is an $\bar{s}$-path from a to a_0.

Conversely, suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Let $\bar{a} = a_0 \cdots a \cdots a_\ell$ be an $\bar{r}\bar{s}$-tour, such that $\bar{a}' = a_0 \cdots a$ is an $\bar{r}$-path and $\bar{a}'' = a \cdots a_\ell$ is an $\bar{s}$-path. Suppose, further, that $\bar{r}$ can be decomposed as $\{\bar{r}_i r_i\}_{i=0}^{k-1}\bar{r}_k$, where $\bar{r}_i$ $(0 \leq i \leq k)$ is a tree-shaped subtour of $\bar{a}'$ and $r_0 \cdots r_{k-1}$ is the backbone of $\bar{a}'$; and $\bar{s}$ can be decomposed as $\{\bar{s}_i s_i\}_{i=0}^{k-1}\bar{s}_k$, where $\bar{s}_i$ $(0 \leq i \leq k)$ is a tree-shaped subtour of $\bar{a}''$ and $s_0 \cdots s_{k-1}$ is the backbone of $\bar{a}''$. Finally, suppose that $\mathfrak{A} \models \omega(a)$. Then, $\mathfrak{A} \models \mathsf{isth}\langle\bar{r}, \omega, \bar{s}\rangle(a_0)$.

Proof. For the first part, we use induction on the structure of $\bar{r}\bar{s}$, based on the conditions in $I_{\wp,\omega}$. Suppose that $\mathfrak{A} \models \mathsf{isth}\langle\bar{r}, \omega, \bar{s}\rangle(a_0)$. If $\mathfrak{A} \models \mathsf{fan}\langle\bar{r}\rangle(a_0) \wedge \omega(a_0) \wedge \mathsf{fan}\langle\bar{s}\rangle(a_0)$ (base case), we have (by Lemma 6.15) an $\bar{r}$-tour $\bar{a}'a_0$ and an $\bar{s}$-tour $a_0\bar{a}''$ starting and ending at a_0. Then, the required result holds trivially, for $a = a_0$. Now, if we are not in the previous case, we can write (by the conditions in I_ω) $\bar{r} = \bar{r}'r\bar{r}''$ and $\bar{s} = \bar{s}''s\bar{s}'$, such that $\mathfrak{A} \models \mathsf{fan}\langle\bar{r}'\rangle(a_0) \wedge \mathsf{fan}\langle\bar{s}'\rangle(a_0)$ and there exists an $a_1 \in A$ with $\mathfrak{A} \models r(a_0, a_1) \wedge s(a_1, a_0)$ and $\mathfrak{A} \models \mathsf{isth}\langle\bar{r}'', \omega, \bar{s}''\rangle(a_1)$. By inductive hypothesis, a_1 is the start (and end) of a tour $\bar{b} = \bar{b}'a\bar{b}''$, where $\mathfrak{A} \models \omega(a)$, $\bar{b}'a$ is an $\bar{r}''$-path, and $a\bar{b}''$ is an $\bar{s}''$-path. Using $\bar{b}$, it is easy to construct the required $\bar{r}\bar{s}$-tour starting (and ending) at a_0, where $\bar{r}\bar{s}$ is decomposed as $\bar{r}'r\bar{r}''\bar{s}''s\bar{s}'$.

For the converse, we proceed by induction on k. If $k = 0$ then $\{\bar{r}_i r_i\}_{i=0}^{k-1}$ and $\{\bar{s}_i s_i\}_{i=0}^{k-1}$ are empty, i.e. $\bar{r}\bar{s} = \bar{r}_0\bar{s}_0$. Thus, $a_0 \cdots a$ is an $\bar{r}_0$-tour and $a \cdots a_0$ is an $\bar{s}_0$ tour, both starting (and ending) at a_0 (i.e. $a = a_0$). Then, by Lemma 6.15, $\mathfrak{A} \models \mathsf{fan}\langle\bar{r}_0\rangle(a_0)$ and $\mathfrak{A} \models \mathsf{fan}\langle\bar{s}_0\rangle(a_0)$; whence, because $\mathfrak{A} \models \omega(a)$, $\mathfrak{A} \models \mathsf{isth}\langle\bar{r}, \omega, \bar{s}\rangle(a_0)$.

Now, suppose that $k > 0$. It is useful to refer to the predicates of $\bar{r}\bar{s}$ individually, so let $\bar{r}\bar{s} = t_0, \ldots, t_{\ell-1}$. Let m be such that $\bar{r}_0 = t_0, \ldots, t_{m-1}$ and n be such that $\bar{s}_k = t_n, \ldots, t_{\ell-1}$. Thus, $a_0, \ldots, a_m$ (with $a_m = a_0$) is the subtour of $\bar{a}$ specified by $\bar{r}_0$, and a_{m+1}—such that $\mathfrak{A} \models r_0(a_m, a_{m+1})$—is the second element of the backbone of the path from a_0 to a. Similarly, $a_n, \ldots, a_\ell$ (with $a_n = a_\ell = a_0$) is the subtour of $\bar{a}$ specified by $\bar{s}_k$, and a_{n-1}—such that $\mathfrak{A} \models s_{k-1}(a_{n-1}, a_n)$—is the penultimate element of the backbone of the path from a to a_0. Then, by Lemma 6.7, $a_{m+1} = a_{n-1}$. Thus, $\bar{a}$ is decomposed as $a_0 \cdots a_m a_{m+1} \cdots a \cdots a_{n-1} a_n \cdots a_{\ell-1} \,(, a_\ell)$, where $a_0 \cdots a_m$ is an $\bar{r}_0$-tour (whence, from Lemma 6.15, $\mathfrak{A} \models \mathsf{fan}\langle\bar{r}_0\rangle(a_0)$), $\mathfrak{A} \models r_0(a_0, a_{m+1})$, $a_{m+1} \cdots a_{n-1}$ is an $\bar{r}''\bar{s}''$-tour (for $\bar{r}''\bar{s}'' = \{\bar{r}_i r_i\}_{i=1}^{k-1}\bar{r}_k\{\bar{s}_i s_i\}_{i=0}^{k-2}\bar{s}_{k-1}$), $\mathfrak{A} \models s_{k-1}(a_{n-1}, a_n)$, and $a_n \cdots a_\ell$ is an

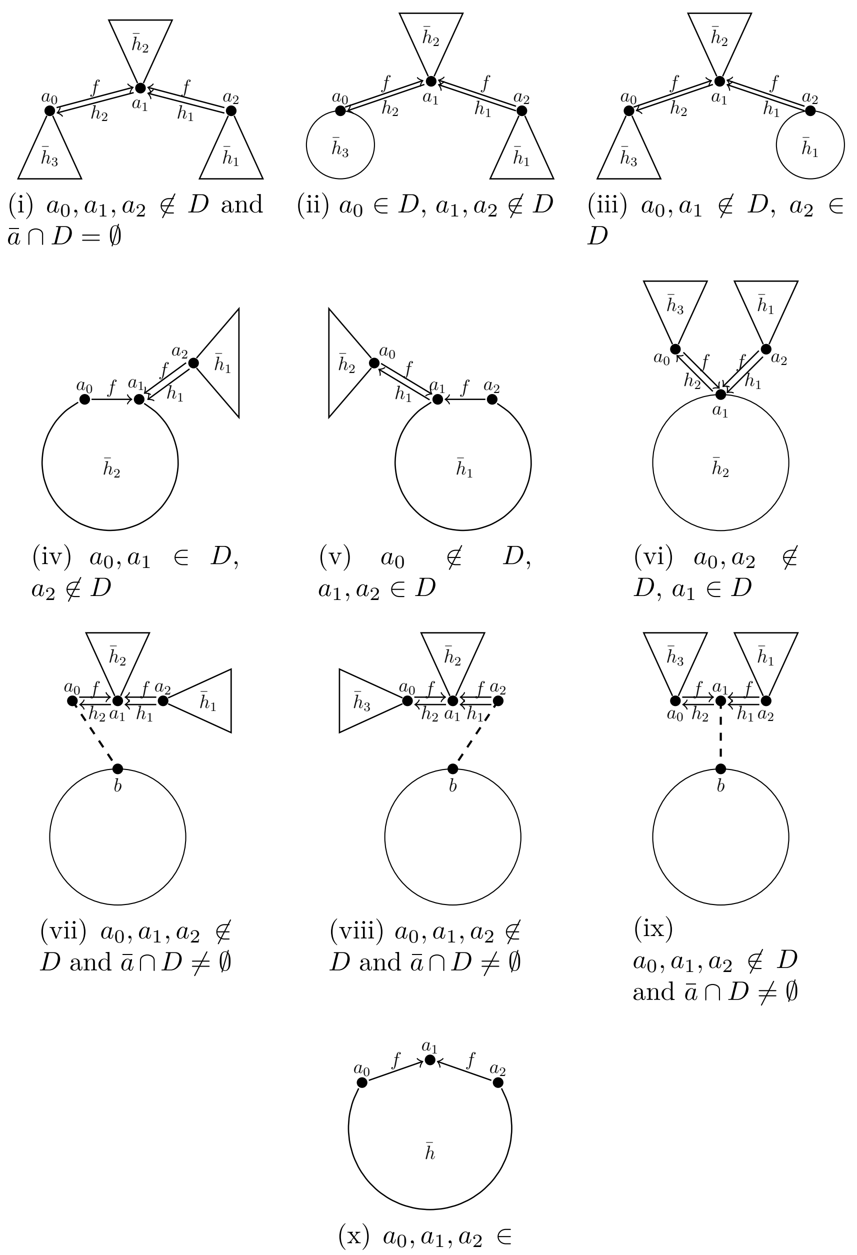
$\bar{s}_k$-tour (whence, from Lemma 6.15, $\mathfrak{A} \models \mathsf{fan}\langle \bar{s}_k \rangle(a_n)$). By inductive hypothesis, we have $\mathfrak{A} \models \mathsf{isth}\langle \bar{r}'', \omega, \bar{s}'' \rangle(a_{m+1})$—it is easily checked that $a_{m+1} \cdots a_{n-1}$ satisfies all the necessary requirements. Then, it follows from the conditions in I_ω that $\mathfrak{A} \models \mathsf{isth}\langle \bar{r}, \omega, \bar{s} \rangle(a_0)$, where $\bar{r} = \bar{r}_0 r_0 \bar{r}''$ and $\bar{s} = \bar{s}'' s_{k-1} \bar{s}_k$. □

6.4 Detection of violations

We now show how to detect violations of path-functional dependencies, using the apparatus developed so far. Let $\wp = \mathrm{PFD}[\bar{f}f, \bar{g}]$ be a dependency, Δ be a database whose active domain is D, and let σ be a relational signature featuring (possibly among other) the predicates appearing in $\wp$ and the constants in D. Recall from Section 6.1 that any critical violation of $\wp$ in a structure $\mathfrak{A}$ is identified with an $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour with three *distinct* initial elements $c, d, e \in A$. Depending on which (if any) of c, d, e belong to the active domain D, the tour can be decomposed in different ways (see Figure 6.9). Thus, by ensuring that none of these possible decompositions occurs in a structure $\mathfrak{A}$, one can be certain that $\mathrm{PFD}[\bar{f}, \bar{g}]$ is not critically violated in $\mathfrak{A}$.

In particular we distinguish ten cases: one when the violating tour never intersects the database (Figure 6.9 (i)), one when the violating tour intersects the database and all of its three initial elements are in D (Figure 6.9 (x)), two when the violating tour instersects the database and two of its three initial elements are in D (Figure 6.9 (iv) and (v)), three when the violating tour intersects the database and one of its three initial elements is in D (Figure 6.9 (ii), (iii), and (vi)), and finally three when the violating tour intersects the database and none of its three initial elements is in D (Figure 6.9 (vii), (viii), and (iv)).

It should be noted here that the last three cases are generalizations of the three cases before them. In more detail, Figure 6.9 (vii) is a generalization of Figure 6.9 (ii), Figure 6.9 (viii) is a generalization of Figure 6.9 (iii), and Figure 6.9 (ix) is a generalization of Figure 6.9 (vi). Indeed, each of these figures illustrates a connected component of three elements—the three initial elements of the tour with two tree-shaped subtours on two of these three elements—either attached to the database directly or connected to it through a (tree-shaped) path (illustrated by a *dotted line* in the relevant figures). This is where the concept of an isthmus is applied: we use it to say that the connected component mentioned above exists at an arbitrary distance from the element b (the element where the tour enters and exits the database).



(i) $a_0, a_1, a_2 \notin D$ and $\bar{a} \cap D = \emptyset$ (ii) $a_0 \in D$, $a_1, a_2 \notin D$ (iii) $a_0, a_1 \notin D$, $a_2 \in D$

(iv) $a_0, a_1 \in D$, $a_2 \notin D$ (v) $a_0 \notin D$, $a_1, a_2 \in D$ (vi) $a_0, a_2 \notin D$, $a_1 \in D$

(vii) $a_0, a_1, a_2 \notin D$ and $\bar{a} \cap D \neq \emptyset$ (viii) $a_0, a_1, a_2 \notin D$ and $\bar{a} \cap D \neq \emptyset$ (ix) $a_0, a_1, a_2 \notin D$ and $\bar{a} \cap D \neq \emptyset$

(x) $a_0, a_1, a_2 \in D$

Figure 6.9: Possible configurations of a violating tour whose three initial (and distinct) elements are a_0, a_1 and a_2. Each dotted line represents an isthmus; where we use a dotted line, a predicate $\mathsf{isth}\langle \bar{r}, \omega, \bar{s} \rangle(\cdot)$ holds at the endpoint labeled b with the sentence ω capturing the configuration at the other endpoint.

Lemma 6.17. Let $\mathfrak{A}$ be a structure and consider the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Further, suppose that $a_i \notin D$ $(0 \le i \le \ell)$ and that the first three elements a_0, a_1 and a_2 of $\bar{a}$ are distinct. Then there is a decomposition $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$ of $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$ such that $\bar{h}_1$, $\bar{h}_2$ and $\bar{h}_3$ specify tree-shaped subtours of $\bar{a}$.

Proof. Refer to Figure 6.9 (i). Let $\bar{r} = r_0, \ldots, r_{\ell-1}$ be the sequence of predicates that constitute $\bar{a}$. Thus, we have $r_0 = f$ and $r_1 = f^{-1}$. Now, consider the subtour starting at a_2. Let $i \ge 2$ be the largest index such that $a_i = a_2$. Write $\bar{h}_1 = r_2, \ldots, r_{i-1}$; clearly $\bar{h}_1$ specifies a tree—recall that the tour does not enter the database. By Lemma 6.8 we now have that $a_{i+1} = a_1$; write $h_1 = r_i$. Similarly, let j be the largest index such that $a_j = a_1$. Write $\bar{h}_2 = r_{i+1}, \ldots, r_{j-1}$; evidently, again, $\bar{h}_2$ is a tree. By Lemma 6.8, again, we have that $a_{j+1} = a_0$; write $h_2 = r_j$. Finally, write $\bar{h}_3 = r_{j+1}, \ldots, r_{\ell-1}$. This completes the tour (going back to a_0); it is clear $\bar{h}_3$ is also a tree. □

Remark 6.18. In subsequent lemmas, the expression 'tour inside the database' refers to tours that include database elements but, of course, can have tree-shaped subtours outside of the database as discussed in Section 6.2 (recall, in particular, Figure 6.6); and likewise for the expression 'path inside the database'.

Lemma 6.19. Let $\mathfrak{A}$ be a structure and consider the $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\le \ell$. Further, suppose that a_0, a_1 and a_2 are distinct with $a_0 \in D$ but $a_1, a_2 \notin D$. Then there is a decomposition $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$ of $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$ such that $\bar{h}_1$ and $\bar{h}_2$ specify tree-shaped subtours, and $\bar{h}_3 = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ $(k \le \ell)$ is a tour inside the database, where each $\bar{t}_i$ $(0 \le i \le k)$ specifies a tree-shaped subtour.

Proof. Refer to Figure 6.9 (ii). Let $\bar{r} = r_0, \ldots, r_{\ell-1}$ be the sequence of predicates that constitute $\bar{a}$. Thus, we have $r_0 = f$ and $r_1 = f^{-1}$. Let $i \ge 2$ be the largest index such that $a_i = a_2$ and $a_j \notin D$, for all $2 < j < i$. Write $\bar{h}_1 = r_2, \ldots, r_{i-1}$; clearly, since no observable cycles of length $\le \ell$ exist, $\bar{h}_1$ defines a tree. Let $\nu > i$ be the smallest index such that $a_\nu = a_0$. (Such an index must exist, since, by Lemma 6.11, $\bar{a}$ must 're-enter' the database through a_0.) Notice that the tour $a_0, \ldots, a_{\nu-1}$ is tree-shaped, since it contains no element in D other than a_0, and $\mathfrak{A}$, by assumption, contains no observable cycles of length $\le \ell$. By Lemma 6.8, applied to $a_0, \ldots, a_{\nu-1}$, we must have $a_{i+1} = a_1$; write $h_1 = r_i$. Similarly, let i' be the largest index such that $a_{i'} = a_1$ and $a_{j'} \notin D$, for all $i + 1 < j' < i'$. Write $\bar{h}_2 = r_{i+1}, \ldots, r_{i'-1}$; evidently $\bar{h}_2$ defines a tree. Let $\nu' > i'$ be the smallest index such that $a_{\nu'} = a_0$. (Again, such an index must exist due to Lemma 6.11.) By Lemma 6.8, applied to the tour $a_0, \ldots, a_{\nu'-1}$ (which is tree-shaped by the same reasoning as $a_0, \ldots, a_{\nu-1}$ above), we have $a_{i'+1} = a_0$; write $h_2 = r_{i'}$. The remaining part of $\bar{a}$, i.e. $a_{i'+1}, \ldots, a_{\ell-1}$ $(, a_\ell)$, is decomposed as in Lemma 6.12 into $b_0\bar{b}_0 \cdots b_k\bar{b}_k$. Correspondingly, $\bar{h}_3 = r_{i'+1}, \ldots, r_{\ell-1}$ is decomposed into $\bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$, as described in Section 6.2. □

Lemma 6.20. Let $\mathfrak{A}$ be a structure and consider the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Further, suppose that a_0, a_1 and a_2 are distinct with $a_0, a_1 \notin D$ and $a_2 \in D$. Then there is a decomposition $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$ such that $\bar{h}_2$ and $\bar{h}_3$ specify tree-shaped subtours, and $\bar{h}_1 = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ $(k \leq \ell)$ is a tour inside the database, where each $\bar{t}_i$ $(0 \leq i \leq k)$ specifies a tree-shaped subtour.

Proof. Analogous to the proof of Lemma 6.19. Refer to Figure 6.9 (iii). □

Lemma 6.21. Let $\mathfrak{A}$ be a structure and consider the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Further, suppose that a_0, a_1 and a_2 are distinct with $a_0, a_1 \in D$ but $a_2 \notin D$. Then there is a decomposition $ff^{-1}\bar{h}_1 h_1 \bar{h}_2$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$ such that $\bar{h}_1$ specifies a tree-shaped subtour, and $\bar{h}_2 = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ $(k \leq \ell)$ is a path inside the database whose final element is a_0, where each $\bar{t}_i$ $(0 \leq i \leq k)$ specifies a tree-shaped subtour.

Proof. Refer to Figure 6.9 (iv). Let $\bar{r} = r_0, \ldots, r_{\ell-1}$ be the sequence of predicates that constitute $\bar{a}$. Thus, we have $r_0 = f$ and $r_1 = f^{-1}$. Let $i \geq 2$ be the largest index such that $a_i = a_2$ and $a_j \notin D$, for all $2 < j < i$. Write $\bar{h}_1 = r_2, \ldots, r_{i-1}$ and clearly, since no observable cycles of length $\leq \ell$ exist, $\bar{h}_1$ defines a tree. Let $\nu > i$ be the smallest index such that $a_\nu = a_1$. (Such an index must exist, since, by Lemma 6.11, $\bar{a}$ must 're-enter' the database through a_1.) Notice that the tour $a_1, \ldots, a_{\nu-1}$ is tree-shaped, since it contains no element in D other than a_1, and $\mathfrak{A}$, by assumption, contains no observable cycles of length $\leq \ell$. By Lemma 6.8, applied to $a_1, \ldots, a_{\nu-1}$, we must have $a_{i+1} = a_1$; write $h_1 = r_i$. We now have the $\bar{h}_2$-path $a_{i+1}, \ldots, a_\ell$ inside the database, where $\bar{h}_2 = r_{i+1}, \ldots, r_{\ell-1}$. This path is decomposed as in Lemma 6.12, to obtain $\bar{h}_2 = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ as required in the statement of this lemma. □

Lemma 6.22. Let $\mathfrak{A}$ be a structure that has no observable cycles of length $\leq \ell$. Let $\bar{a} = a_0, \ldots, a_{\ell-1}$ be an $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour in $\mathfrak{A}$. Further, suppose that a_0, a_1 and a_2 are distinct with $a_1, a_2 \in D$ and $a_0 \notin D$. Then there is a decomposition

$ff^{-1}\bar{h}_1 h_1 \bar{h}_2$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$ such that $\bar{h}_2$ specifies a tree-shaped subtour, and $\bar{h}_1 = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ $(k \leq \ell)$ is a path inside the database whose final element is a_0, where each $\bar{t}_i$ $(0 \leq i \leq k)$ specifies a tree-shaped subtour.

Proof. Analogous to the proof of Lemma 6.21. Refer to Figure 6.9 (v). □

Lemma 6.23. Let $\mathfrak{A}$ be a structure and consider the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Further, suppose that a_0, a_1 and a_2 are distinct with $a_1 \in D$ but $a_0, a_2 \notin D$. Then there is a decomposition $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$ such that $\bar{h}_1$ and $\bar{h}_3$ specify tree-shaped subtours, and $\bar{h}_2 = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ $(k \leq \ell)$ is a tour inside the database, where each $\bar{t}_i$ $(0 \leq i \leq k)$ specifies a tree-shaped subtour.

Proof. Refer to Figure 6.9 (vi). Let $\bar{r} = r_0, \ldots, r_{\ell-1}$ be the sequence of predicates that constitute $\bar{a}$. Thus, we have $r_0 = f$ and $r_1 = f^{-1}$. Similarly with previous proofs, write $\bar{h}_1 = r_2, \ldots, r_{i-1}$, where $a_i = a_2$ and i is the largest index such that $a_i = a_2$ and $a_j \notin D$, for all $2 < j < i$. Evidently, $\bar{h}_1$ is tree-shaped, as $\mathfrak{A}$ contains no cycles of length $\leq \ell$. Let $\nu > i$ be the smallest index such that $a_\nu = a_1$. (Such an index must exist, since, by Lemma 6.11, $\bar{a}$ must 're-enter' the database through a_1.) Notice that the tour $a_1, \ldots, a_{\nu-1}$ is tree-shaped, since it contains no element in D other than a_1, and $\mathfrak{A}$, by assumption, contains no observable cycles of length $\leq \ell$. By Lemma 6.8, applied to $a_1, \ldots, a_{\nu-1}$, $a_{i+1} = a_1$; write $h_1 = r_i$. (It follows that $\nu = i+1$.) The tour is now inside the database and, by Lemma 6.10, it has to exit at a_1. Let $a_{i+1}, \ldots, a_j$ be the subtour of $\bar{a}$ inside the database, i.e. with $a_j = a_{i+1}$ $(= a_1)$ and $a_{j'} \notin D$ for all $j' \geq j$; let $\bar{h}_2 = r_{i+1}, \ldots, r_{j-1}$ for the corresponding sequence of predicates. The subtour $a_{i+1}, \ldots, a_{j-1}$ is decomposed into $b_0 \bar{b}_0 \cdots b_k \bar{b}_k$ as in Lemma 6.12, and, correspondingly, $\bar{h}_2$ is decomposed into $\bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ (having the required properties) as described in Section 6.2. Thus, we have decomposed $\bar{a}$ as $a_0 \cdots a_\nu \cdots a_j \cdots a_{\ell-1}$, where the subtour $a_\nu, \ldots, a_{j-1}$ is the part of $\bar{a}$ inside the database—that is, $a_\nu = a_j = a_1$ and no element of D appears outside this subtour. We saw that the subtour $a_0, \ldots, a_{\nu-1}$ is tree-shaped, and, by the same reasoning, the subtour $a_j, \ldots, a_{\ell-1}$ is also tree-shaped. Towards applying Lemma 6.8 to show that $a_{j+1} = a_0$, we can disregard the subtour inside the database, obtaining a new tour $\bar{a}' = a_0 \cdots a_\nu a_{j+1} \cdots a_{\ell-1}$ that is tree-shaped. Then, by Lemma 6.8, applied to $\bar{a}'$, it must be $a_{j+1} = a_0$; write $h_2 = r_j$. Set $\bar{h}_3 = r_{j+1}, \ldots, r_{\ell-1}$; as already said, $\bar{h}_3$ is tree-shaped, by the assumption that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. □

Lemma 6.24. Let $\mathfrak{A}$ be a structure and consider the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Further, suppose that a_0, a_1 and a_2 are distinct with $a_0, a_1, a_2 \notin D$ but $\bar{a} \cap D \neq \emptyset$. Then there is a decomposition $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$ such that one of $\bar{h}_1, \bar{h}_2, \bar{h}_3$ specifies a subtour that goes inside the database and the other two specify tree-shaped subtours.

Proof. Let $\bar{r} = r_0, \ldots, r_{\ell-1}$ be the sequence of predicates that constitute $\bar{a}$. Thus, we have $r_0 = f$ and $r_1 = f^{-1}$. Since the tour goes into the database at some point, let $b \in D$ be the point of entry—by Lemma 6.10, $\bar{a}'$ will also have to 'exit' the database through b. Thus, $\bar{a}$ is decomposed as $a_0 a_1 a_2 \cdots a_\nu \cdots a_{\nu'} \cdots a_{\ell-1}$, where $\bar{a}' = a_\nu \cdots a_{\nu'-1}$ is the subtour of $\bar{a}$ inside the database, i.e. $a_\nu = a_{\nu'} = b$ and no element of D appears outside $\bar{a}'$. Notice that $\bar{a}'' = a_0 a_1 a_2 \cdots a_\nu a_{\nu'+1} \cdots a_{\ell-1}$ is also a tour, and it is tree-shaped since it only contains one element of D (i.e. b). For the rest of the proof, when we use Lemma 6.8 (which technically requires a tree-shaped tour) it will be applied (without mention, for the sake of brevity) to $\bar{a}''$. Let $\bar{p}$ be the backbone of the path from a_0 to b, and let $\bar{p}'$ be the backbone of the path from b back to a_0. Recall, also, that $\bar{p}$ is the shortest subpath of $\bar{a}$ from a_0 to b and, similarly, $\bar{p}'$ is the shortest subpath of $\bar{a}$ from b to a_0; also, both $\bar{p}$ and $\bar{p}'$ are unique. We distinguish three possibilities:

- $\bar{p}$ goes through a_2 (see Figure 6.9 (viii)): This implies that $\bar{p}$ also goes through a_1, since the shortest path connecting a_0 and a_2 goes through a_1. Let $i > 2$ be the largest index such that $a_i = a_2$ and for all $i' > i$, $a_{i'} \notin D$. We know that such an index exists for, otherwise, there would be a path $\bar{p}'' \neq \bar{p}'$ from b to a_0, and the uniqueness of $\bar{p}'$ would be violated. (In other words, after going out of the database for the last time, the tour has to 'revisit' its origin through $\bar{p}'$, which includes a_2 due to Lemma 6.7.) Thus, the sequence $\bar{h}_1 = r_2, \ldots, r_{i-1}$ constitutes a tour that goes into the database, starting (and ending) at a_2. Now, by Lemma 6.8, $a_{i+1} = a_1$; set $h_1 = r_i$. Similarly with previous proofs, let $\bar{h}_2 = r_{i+1}, \ldots, r_{j-1}$, where j is the largest index such that $a_{i+1} = a_j$ $(= a_1)$; let $h_2 = r_j$. Again, by Lemma 6.8, $a_{j+1} = a_0$; set $\bar{h}_3 = r_{j+1}, \ldots, r_{\ell-1}$. Clearly, $\bar{h}_2$ and $\bar{h}_3$ specify tree-shaped tours, by the assumption that $\mathfrak{A}$ contains no observable cycles of length $\leq \ell$.

- $\bar{p}$ goes through a_1 but not a_2 (see Figure 6.9 (ix)): Let $i \geq 2$ be the largest index such that $a_i = a_2$ and for all $i' < i$, $a_{i'} \notin D$. Such an i exists otherwise we would be in the previous case. Set $\bar{h}_1 = r_2, \ldots, r_{i-1}$; clearly, $\bar{h}_1$ is tree-shaped. By Lemma 6.8, $a_{i+1} = a_1$; set $h_1 = r_i$. Let j be the largest index such that $a_j = a_1$ and for all $j' > j$, $a_{j'} \notin D$. Again, such an index exists for, if it did not, the uniqueness of $\bar{p}'$ would be violated. Thus, the sequence $\bar{h}_2 = r_{i+1}, \ldots, r_{j-1}$ constitutes a tour that goes into the database, starting (and ending) at a_1. By Lemma 6.8, $a_{j+1} = a_0$; set $h_2 = r_j$. Finally, set $\bar{h}_3 = r_{j+1}, \ldots, r_{\ell-1}$ and observe that it specifies a tree-shaped tour.

- $\bar{p}$ does not go through a_1 or a_2 (see Figure 6.9 (vii)): Similarly with the above, let $i \geq 2$ be the largest index such that $a_i = a_2$ and for all $i' < i$, $a_{i'} \notin D$. Set $\bar{h}_1 = r_2, \ldots, r_{i-1}$; clearly, $\bar{h}_1$ is tree-shaped. By Lemma 6.8, $a_{i+1} = a_1$; set $h_1 = r_i$. Let $j \geq i+1$ be the largest index such that $a_j = a_1$ and for all $j' < j$, $a_{j'} \notin D$. Set $\bar{h}_2 = r_{i+1}, \ldots, r_{j-1}$; clearly, $\bar{h}_2$ is tree-shaped. By Lemma 6.8, $a_{j+1} = a_0$; set $h_2 = r_j$. In that case, $\bar{h}_3 = r_{j+1}, \ldots, r_{\ell-1}$ is a tour that goes into the database, starting (and ending) at a_0. □

In the previous lemma, among $\bar{h}_1, \bar{h}_2, \bar{h}_3$, the subtour that goes inside the database can further be decomposed (using Lemmas 6.12 and 6.13) as illustrated in Figure 6.7.

Lemma 6.25. Let $\mathfrak{A}$ be a structure and consider the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ in $\mathfrak{A}$. Suppose that $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. Further, suppose that a_0, a_1 and a_2 are distinct with $a_0, a_1, a_2 \in D$. Then there is a decomposition $ff^{-1}\bar{h}$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$, such that $\bar{h} = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ $(k \leq \ell)$ is a path inside the database from a_2 to a_0, where each $\bar{t}_i$ $(0 \leq i \leq k)$ specifies a tree-shaped subtour.

Proof. Refer to Figure 6.9 (x). Clearly the first predicates of the tour are f and f^{-1}, i.e. $\mathfrak{A} \models f(a_0, a_1)$ and $\mathfrak{A} \models f(a_1, a_2)$. Now, $a_2, \ldots, a_0$ can be decomposed into $b_0 \bar{b}_0 \cdots b_k \bar{b}_k$ as in Lemma 6.12, and, correspondingly, $\bar{h} = r_2, \ldots, r_{\ell-1}$ into $\bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$, as described in Section 6.2. □

Before stating our final results for this section, we write two formulas related to the configuration illustrated in Figure 6.9 (i). Let

$$\omega_1\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(x) := \exists y \exists z \Big(\mathsf{dst}(x, y, z) \wedge f(x, y) \wedge f^{-1}(y, z) \\ \wedge \mathsf{fan}\langle \bar{h}_1\rangle(z) \wedge h_1(z, y) \wedge \mathsf{fan}\langle \bar{h}_2\rangle(y) \wedge h_2(y, x) \wedge \mathsf{fan}\langle \bar{h}_3\rangle(x)\Big),$$

$$\omega_2\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(y) := \exists x \exists z \Big(\mathsf{dst}(x,y,z) \wedge f(x,y) \wedge f^{-1}(y,z) \\ \wedge \mathsf{fan}\langle \bar{h}_1\rangle(z) \wedge h_1(z,y) \wedge \mathsf{fan}\langle \bar{h}_2\rangle(y) \wedge h_2(y,x) \wedge \mathsf{fan}\langle \bar{h}_3\rangle(x)\Big),$$

where $\bar{h}_1 h_1, \bar{h}_2 h_2$, and $\bar{h}_3$ are sequences of functional predicates. It is clear that if $\bar{a}$ is a tree-shaped $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$-tour in a structure $\mathfrak{A}$, $\bar{h}_1$, $\bar{h}_2$ and $\bar{h}_3$ specify tree-shaped subtours of $\bar{a}$ (as in Figure 6.9 (i)), and the first three elements a_0, a_1 and a_2 of $\bar{a}$ are distinct, then $\mathfrak{A} \models \omega_1\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(a_0)$ and $\mathfrak{A} \models \omega_2\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(a_1)$. Of course, ω_1 and ω_2 are not in $\mathcal{GC}^2$, but this is easily fixed using Lemma 6.1.

Recall from Section 6.3, that $\sigma^1_\wp$ consists of the predicates in σ together with one new predicate $\mathsf{fan}\langle \bar{h}\rangle(\cdot)$ for each contiguous subsequence of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$. Further, for any given $\mathcal{GC}^2$-formula $\omega(x)$, $\sigma^2_{\wp,\omega}$ consists of $\sigma^1_\wp$ together with the predicates $\mathsf{isth}\langle \bar{r}, \omega, \bar{s}\rangle(\cdot)$, one for each pair of distinct contiguous subsequences $\bar{r}$ and $\bar{s}$ of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$. Let $\sigma^3_\wp$ be the union of $\sigma^2_{\wp,\omega^*_1}$ and $\sigma^2_{\wp,\omega^*_2}$; the ensuing discussion is over $\sigma^3_\wp$. Note that the size of $\sigma^3_\wp$ is polynomial in the length of $\bar{f}$ and $\bar{g}$.

Let us sum up what we have discussed so far. The violation of PFD$[\bar{f}f, \bar{g}]$ in any structure $\mathfrak{A}$ interpreting Δ that contains no cycles of length $\leq \ell$, where ℓ is the length of the sequence $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$, is identified with the existence of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tours whose first three elements are distinct. (By Lemma 4.24, we may assume that all models that we are concerned with have no cycles of length $\leq \ell$.) Depending on which (if any) of the three initial elements of an $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a}$ belong to D, $\bar{a}$ can be decomposed in different ways, as seen in Figure 6.9. Our goal, then, is to ensure that no such tours occur in any structure of interest. Now, notice that all these decompositions, except the one in Figure 6.9 (i), corresponding to Lemma 6.17, involve elements of D.

Note that if the database is complete with respect to $\sigma^3_\wp$, it contains all the information required to check the existence of violating tours like the ones illustrated in Figure 6.9 (ii) – (x). (This follows from the semantics of the fan and isth predicates—Lemmas 6.15 and 6.16.) In that sense, the violating tours that are decomposed as in Lemmas 6.19 – 6.25, and illustrated in Figure 6.9 (ii) – (x), are independent of the model to which they belong (and only dependent upon the complete database). Thus, we introduce a notion of 'appropriateness' of the database for the dependency PFD$[\bar{f}f, \bar{g}]$, which is irrespective of any model extending Δ and checkable in exponential time (in the size of Δ and the sequences $\bar{f}, \bar{g}$). Before defining this formally, we introduce a few procedures that check the existence of violating tours like the ones in Figure 6.9 (ii) – (x); the checks are performed starting at the point of 'entry' b.

For the following paragraphs, we assume that Δ is *complete* with respect to the signature $\sigma^3_{\wp}$. For convenience, we let $\bar{h} = ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$. To begin with, we need a procedure that checks whether, for two *distinct* elements $a, b \in D$, there is an $\bar{s}$-path $\bar{a}$ inside the database starting at a and ending at b, where $\bar{s}$ is decomposed into $\{\bar{s}_i s_i\}_{i=0}^{n-1}\bar{s}_n$ and each $\bar{s}_i$ $(0 \leq i \leq n)$ specifies a tree-shaped subtour. That is, according to the discussion in Section 6.2, we need to check if $\bar{a}$ can be decomposed as $b_0\bar{b}_0 \dots b_n\bar{b}_n$, where each $b_i\bar{b}_i$ $(0 \leq i \leq n)$ is a tree-shaped $\bar{s}_i$-tour, $b_i \in D$, and $s_j(b_j, b_{j+1}) \in \Delta$ $(0 \leq j < n)$. This straightforward task is performed by the following procedure.

```
1: define DB-PATH($\{\bar{s}_i s_i\}_{i=m}^{n-1}\bar{s}_n$, $a$, $b$)
2:     if $m = n - 1$ and $\mathsf{fan}\langle\bar{s}_m\rangle(a) \wedge s_m(a,b) \wedge \mathsf{fan}\langle\bar{s}_n\rangle(b) \in \Delta$ then
3:         return TRUE
4:     else if $\mathsf{fan}\langle\bar{s}_m\rangle(a) \in \Delta$ and $\exists\, a' \in D \setminus \{a,b\}$ s.t. $s_m(a,a') \in \Delta$ then
5:         return DB-PATH($\{\bar{s}_i s_i\}_{i=m+1}^{n-1}\bar{s}_n$, $a'$, $b$)
6:     else
7:         return FALSE
```

To get the required result, one 'calls' the above procedure with $m = 0$. It is clear that DB-PATH terminates in polynomial time (with respect to its input size), and it produces the required result (this is an easy induction on n).

The following procedure is based on DB-PATH, and checks whether there is an $\bar{s}$-tour inside the database starting (and ending) at a given $b \in D$, and $\bar{s}$ is decomposed into $\{\bar{s}_i s_i\}_{i=0}^{n-1}\bar{s}_n$, where each $\bar{s}_i$ $(0 \leq i \leq n)$ specifies a tree-shaped subtour.

```
1: define DB-TOUR($\{\bar{s}_i s_i\}_{i=0}^{n-1}\bar{s}_n$, $b$)
2:     if $n = 0$ and $\mathsf{fan}\langle\bar{s}_n\rangle(b) \in \Delta$ then
3:         return TRUE
4:     else if $\mathsf{fan}\langle\bar{s}_0\rangle(b) \in \Delta$ and $\exists\, b' \in D \setminus \{b\}$ s.t. $s_0(b,b') \in \Delta$ then
5:         return DB-PATH($\{\bar{s}_i s_i\}_{i=1}^{n-1}\bar{s}_n$, $b'$, $b$)
6:     else
7:         return FALSE
```

Again, it is clear that DB-TOUR terminates in polynomial time and produces the required result.

Consider the following three procedures:

```
1: define VIOLATIONS-II-VII($b$)
2:     for each decomposition $ff^{-1}\bar{h}_1h_1\bar{h}_2h_2\bar{h}_3$ of $\bar{h}$ do
3:         for each decomposition $\{\bar{r}_ir_i\}_{i=0}^{k-1}\bar{r}_kr\,\bar{s}\,t\{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k$ of $\bar{h}_3$ do
4:             for each decomposition $\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n$ of $\bar{s}$ do
5:                 let $\bar{r} = \{\bar{r}_ir_i\}_{i=0}^{k-1}\bar{r}_k$, $\omega = \omega_1^*\langle\bar{h}_1h_1, \bar{h}_2h_2, \epsilon\rangle$, $\bar{t} = \{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k$
6:                 if $\mathsf{isth}\langle t\bar{t}, \omega, \bar{r}r\rangle(b) \in \Delta$ and DB-TOUR($\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n$, $b$) then
7:                     return TRUE
8:     return FALSE
```

```
1: define VIOLATIONS-III-VIII($b$)
2:     for each decomposition $ff^{-1}\bar{h}_1h_1\bar{h}_2h_2\bar{h}_3$ of $\bar{h}$ do
3:         for each decomposition $\{\bar{r}_ir_i\}_{i=0}^{k-1}\bar{r}_kr\,\bar{s}\,t\{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k$ of $\bar{h}_1$ do
4:             for each decomposition $\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n$ of $\bar{s}$ do
5:                 let $\bar{r} = \{\bar{r}_ir_i\}_{i=0}^{k-1}\bar{r}_k$, $\omega = \omega_1^*\langle\bar{h}_3h_2^{-1}), \bar{h}_2h_1^{-1}, \epsilon\rangle$, $\bar{t} = \{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k$
6:                 if $\mathsf{isth}\langle t\bar{t}, \omega, \bar{r}r\rangle(b) \in \Delta$ and DB-TOUR($\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n$, $b$) then
7:                     return TRUE
8:     return FALSE
```

```
1: define VIOLATIONS-VI-IX($b$)
2:     for each decomposition $ff^{-1}\bar{h}_1h_1\bar{h}_2h_2\bar{h}_3$ of $\bar{h}$ do
3:         for each decomposition $\{\bar{r}_ir_i\}_{i=0}^{k-1}\bar{r}_kr\,\bar{s}\,t\{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k$ of $\bar{h}_2$ do
4:             for each decomposition $\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n$ of $\bar{s}$ do
5:                 let $\bar{r} = \{\bar{r}_ir_i\}_{i=0}^{k-1}\bar{r}_k$, $\omega = \omega_2^*\langle\bar{h}_1h_1, \bar{h}_2, \bar{h}_3\rangle$, $\bar{t} = \{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k$
6:                 if $\mathsf{isth}\langle t\bar{t}, \omega, \bar{r}r\rangle(b) \in \Delta$ and DB-TOUR($\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n$, $b$) then
7:                     return TRUE
8:     return FALSE
```

We claim that in any structure $\mathfrak{A}$ interpreting Δ, there is a violating $\bar{h}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ with $\bar{a} \cap D \neq \emptyset$ and $a_1, a_2 \notin D$ if and only if VIOLATION-II-VII(b), VIOLATION-III-VIII(b), or VIOLATION-VI-IX(b) returns TRUE for some $b \in D$. These procedures correspond to the violating tours illustrated in Figure 6.9 (ii) and (vii), (iii) and (viii), and (vi) and (ix) respectively. Note that (ii), (iii), and (vi) illustrate $\bar{a}$ with $a_0 \in D$, and (vii), (viii), and (ix) illustrate $\bar{a}$ with $a_0 \notin D$ respectively. Now, if any of the above procedures returns TRUE for some $b \in D$, the existence of a violating tour like $\bar{a}$ is evident.

We only prove the converse for the procedure VIOLATION-II-VII; the proof for the other two is completely analogous. Suppose that $\bar{a} = a_0, \ldots, a_{\ell-1}$ is a $\bar{h}$-tour in $\mathfrak{A}$ with $\bar{a} \cap D \neq \emptyset$ and $a_1, a_2 \notin D$. Let $b \in D$ be the point at which $\bar{a}$ enters (and leaves, by Lemma 6.10) the database. By Lemma 4.24, we may assume that $\mathfrak{A}$ has no cycles of length $\leq \ell$. If $a_0 \in D$ (i.e. $b = a_0$) then, by Lemma 6.19, $\bar{h}$ can be decomposed into $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$, such that $\bar{h}_1$ and $\bar{h}_2$ specify tree-shaped subtours, and $\bar{h}_3 = \bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$ $(k \leq \ell)$ is a tour inside the database, where each $\bar{t}_i$ $(0 \leq i \leq k)$ specifies a tree-shaped subtour. In particular, referring to Figure 6.9 (ii), $\bar{a}$ is decomposed as $a_0 a_1 a_2 \cdots a_0 \cdots a_0$, where the last part $a_0 \cdots a_0$ is an $\bar{h}_3$-tour inside the database, thus DB-TOUR$(\{\bar{t}_i t_i\}_{i=0}^{k-1} \bar{t}_k, a_0)$ will return TRUE. Further, it is clear that $\mathfrak{A} \models \omega_1 \langle \bar{h}_1 h_1, \bar{h}_2 h_2, \epsilon \rangle (a_0)$, and, by Lemma 6.1, $\mathfrak{A} \models \omega_1^* \langle \bar{h}_1 h_1, \bar{h}_2 h_2, \epsilon \rangle (a_0)$. Thus, by the semantics of isth in Lemma 6.16, we have $\mathfrak{A} \models \mathsf{isth}\langle \bar{r}, \omega, \bar{s} \rangle (a_0)$, for $\bar{r} = \bar{s} = \epsilon$ and $\omega = \omega_1^* \langle \bar{h}_1 h_1, \bar{h}_2 h_2, \epsilon \rangle$. It is then evident that VIOLATION-II-VII(b) will return TRUE for $b = a_0$.

Now, suppose that $a_0 \notin D$ (i.e. $b \neq a_0$), and that the shortest path from a_0 to b does not go through a_1 or a_2, as in Figure 6.9 (vii). Then $\bar{h}$ (and, correspondingly, $\bar{a}$) can be decomposed (as in Lemma 6.24) into $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$, where $\bar{h}_3$ is a subtour, say $\bar{a}'$, of $\bar{a}$ that goes into the database and $\bar{h}_1, \bar{h}_2$ are tree-shaped subtours of $\bar{a}$. Before discussing decompositions of $\bar{a}'$, notice that $\mathfrak{A} \models \omega_1 \langle \bar{h}_1 h_1, \bar{h}_2 h_2, \epsilon \rangle (a_0)$, and, thus, by Lemma 6.1, $\mathfrak{A} \models \omega_1^* \langle \bar{h}_1 h_1, \bar{h}_2 h_2, \epsilon \rangle (a_0)$.

According to Lemma 6.13, $\bar{a}'$ can be decomposed as

$$b_0 \bar{b}_0 \cdots b_k \bar{b}_k \, \bar{b} \, b_k \bar{b}'_0 \cdots b_0 \bar{b}'_k,$$

where $\bar{b}$ is a database tour starting (and ending) at b, and no database element appears outside $\bar{b}$. Let $\bar{s}$ be the sequence of predicates that $\bar{b}$ induces. Thus, we write $\bar{h}_3$ as $\bar{r}r\,\bar{s}\,t\bar{t}$, where $b_0 \bar{b}_0 \cdots b_k \bar{b}_k$ is an $\bar{r}$-path (from b_0 to b_k), $\mathfrak{A} \models r(b_k, b)$, $\bar{s}$ is the part of the tour inside the database, $\mathfrak{A} \models t(b, b_k)$, and $b_k \bar{b}'_0 \cdots b_0 \bar{b}'_k$ is a $\bar{t}$-path. In particular, $\bar{r} = \{\bar{r}_i r_i\}_{i=0}^{k-1} \bar{r}_k$, where each $\bar{r}_i$ $(0 \leq i \leq k)$ corresponds to $b_i \bar{b}_i$ and specifies a tree-shaped subtour, and $\mathfrak{A} \models r_j(b_j, b_{j+1})$, for all j $(0 \leq j < k)$; similarly, we write $\bar{t} = \{\bar{t}_i t_i\}_{i=0}^{k-1} \bar{t}_k$. (It helps here to keep in mind Figure 6.7.) Finally, $\bar{s}$ can be decomposed into $\{\bar{s}_i s_i\}_{i=0}^{n-1} \bar{s}_n$—this follows from the decomposition that Lemma 6.12 guarantees, expressed in terms of the predicates that it induces.

It is now easily seen that DB-TOUR$(\{\bar{s}_i s_i\}_{i=0}^{n-1} \bar{s}_n,\ b)$ will return TRUE, for the above decomposition of $\bar{s}$. Further, by Lemma 6.16, we have $\mathfrak{A} \models \mathsf{isth}\langle t\bar{t}, \omega, \bar{r}r \rangle (b)$, for $\omega = \omega_1 \langle \bar{h}_1 h_1, \bar{h}_2 h_2, \epsilon \rangle (a_0)$. (Notice the reverse order of $t\bar{t}$ and $\bar{r}r$ compared to $\bar{r}r\,\bar{s}\,t\bar{t}$: the isthmus starts at b, whereas $\bar{a}'$ starts at a_0.) It then follows that VIOLATION-II-VII(b) will return TRUE.

Consider the following procedure:

```
1: define VIOLATION-IV($b$)
2:     for each decomposition $ff^{-1}\bar{h}_1h_1\bar{h}_2$ of $\bar{h}$ do
3:         let $\omega = \omega_1^*\langle\bar{h}_1h_1, \epsilon, \epsilon\rangle$
4:         if isth$\langle\epsilon, \omega, \epsilon\rangle(b) \in \Delta$ and $\exists\, b' \in D \setminus \{b\}$ s.t. $f(b, b') \in \Delta$ then
5:             for each decomposition $\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n$ of $\bar{h}_2$ do
6:                 if DB-PATH($\{\bar{s}_is_i\}_{i=0}^{n-1}\bar{s}_n, b, b'$) then
7:                     return TRUE
8:     return FALSE
```

We claim that in any structure $\mathfrak{A}$ interpreting Δ, there is a violating $\bar{h}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ with $a_0, a_1 \in D$ and $a_2 \notin D$ if and only if VIOLATION-IV(b) returns TRUE for some $b \in D$. Indeed, if VIOLATION-IV(b) returns TRUE for some $b \in D$, the existence of such a violating tour is evident.

Conversely, suppose that $\bar{a} = a_0, \ldots, a_{\ell-1}$ is a $\bar{h}$-tour in $\mathfrak{A}$ with $a_0, a_1 \in D$ and $a_2 \notin D$. By Lemma 4.24, we may assume that $\mathfrak{A}$ has no cycles of length $\leq \ell$. Then, by Lemma 6.21, $\bar{a}$ can be decomposed as $ff^{-1}\bar{h}_1h_1\bar{h}_2$, where $\bar{h}_1$ specifies a tree-shaped subtour of $\bar{a}$ and $\bar{h}_2$ is a path inside the database decomposed as $\{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k$, for some $n \leq \ell$. In particular, referring to Figure 6.9 (iv), $\bar{a}$ is decomposed as $a_0a_1a_2\cdots a_1\cdots a_0$, where the last part $a_1\cdots a_0$ is an $\bar{h}_2$-path inside the database, thus DB-PATH($\{\bar{t}_it_i\}_{i=0}^{k-1}\bar{t}_k, a_1, a_0$) will return TRUE. Further, it is clear that $\mathfrak{A} \models \omega_1\langle\bar{h}_1h_1, \epsilon, \epsilon\rangle(a_0)$, and, by Lemma 6.1, $\mathfrak{A} \models \omega_1^*\langle\bar{h}_1h_1, \epsilon, \epsilon\rangle(a_0)$. In that case, by the semantics of isth in Lemma 6.16, we have that $\mathfrak{A} \models \mathsf{isth}\langle\bar{r}, \omega, \bar{s}\rangle(a_0)$, for $\bar{r} = \bar{s} = \epsilon$ and $\omega = \omega_1^*\langle\bar{h}_1h_1, \epsilon, \epsilon\rangle$. It is then evident that VIOLATION-IV(b) will return TRUE for $b = a_0$ (and $b' = a_1$).

Note that the violating tours (iv) and (v) in Figure 6.9 are symmetric, so our approach can be adapted to handle the case where $a_1, a_2 \in D$ and $a_0 \notin D$. Thus, we assume the existence of a similar procedure VIOLATION-V(b), which returns true if and only if in any structure $\mathfrak{A}$ interpreting Δ there exists a violating tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ with $a_1, a_2 \in D$ and $a_0 \notin D$, as illustrated in Figure 6.9 (v).

Consider the following procedure:

```
1: define VIOLATION-X(b)
2:     let $\bar{h}'$ be such that $\bar{h} = ff^{-1}\bar{h}'$
```

```
3:  if $\exists$ distinct $b', b'' \in D \setminus \{b\}$ s.t. $f(b, b'), f(b'', b') \in \Delta$ then
4:      for each decomposition $\{\bar{s}_i s_i\}_{i=0}^{n-1} \bar{s}_n$ of $\bar{h}'$ do
5:          if DB-PATH($\{\bar{s}_i s_i\}_{i=0}^{n-1} \bar{s}_n, b'', b$) then
6:              return TRUE
7:  return FALSE
```

We claim that in any structure $\mathfrak{A}$ interpreting Δ, there is a violating $\bar{h}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ whose first three elements are distinct and belong to D if and only if VIOLATION-X(b) returns TRUE for some $b \in D$. Indeed, if VIOLATION-X(b) returns TRUE for some $b \in D$, the existence of such a violating tour is evident. Conversely, suppose that $\bar{a} = a_0, \ldots, a_{\ell-1}$ is a $\bar{h}$-tour in $\mathfrak{A}$ whose first three elements are distinct and belong to D. By Lemma 4.24, we may assume that $\mathfrak{A}$ has no cycles of length $\leq \ell$. Then, by Lemma 6.25, $\bar{h}$ can be written as $ff^{-1}\bar{h}'$, where $\bar{h}'$ is a path inside the database (i.e. $a_2, \ldots, a_{\ell-1}$), decomposed as $\bar{t}_0 t_0 \cdots \bar{t}_{k-1} t_{k-1} \bar{t}_k$, for some $k \leq \ell$. In that case, it is clear that DB-PATH($\{\bar{t}_i t_i\}_{i=0}^{k-1} \bar{t}_k, a_2, a_0$) will return TRUE; consequently, VIOLATION-X(a_0) will also return TRUE.

We are now ready to define a notion of 'appropriateness' of Δ for the dependency PFD[$\bar{f}f, \bar{g}$], as promised.

Definition 6.26. A complete database Δ is PFD[$\bar{f}f, \bar{g}$]-*appropriate*, for a given path-functional dependency PFD[$\bar{f}f, \bar{g}$], if there exists no element b in its active domain D such that VIOLATION-II-VII(b), VIOLATION-III-VIII(b), VIOLATION-IV(b), VIOLATION-V(b), VIOLATION-VI-IX(b), or VIOLATION-X(b) returns TRUE.

Intuitively, a complete database Δ is PFD[$\bar{f}f, \bar{g}$]-appropriate if any model $\mathfrak{A}$ extending it cannot contain any tours violating the path-functional dependency PFD[$\bar{f}f, \bar{g}$] like the ones in Figure 6.9 (ii) – (x). As already discussed, whether Δ is PFD[$\bar{f}f, \bar{g}$]-appropriate depends only upon Δ (since it is complete), and is checked using the above procedures.

Lemma 6.27. If Δ is a complete database and PFD[$\bar{f}f, \bar{g}$] is a path-functional dependency, it can be checked in exponential time (in the size of $\bar{f}, \bar{g}$ and Δ) whether Δ is PFD[$\bar{f}f, \bar{g}$]-appropriate.

Proof. Notice that DB-PATH and DB-TOUR terminate in polynomial time in respect of their input size. Moreover, it is easily seen that in every procedure mentioned in Definition 6.26, each loop is executed at most an exponential number of times in the size of the sequences $\bar{f}$ and $\bar{g}$. Since each procedure is executed for all elements $b \in D$ in the worst case, the whole task requires exponential time in the size of $\bar{f}$, $\bar{g}$ and Δ. □

The only remaining case is the one corresponding to Lemma 6.17 and Figure 6.9 (i). Such a violating tour, however, cannot be detected starting inside Δ, since it does not ever enter Δ and its start can be anywhere in an arbitrarily large model. We will see in Theorem 6.29 that it is related to the (finite) satisfiability of a particular set of formulas, so we postpone dealing with it until then.

6.5 Main result

For the rest of this section, we fix a path-functional dependency $\wp = \mathrm{PFD}[\bar{f}f, \bar{g}]$, the related formulas $\omega_1^*\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(x)$ and $\omega_2^*\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(y)$ as in Section 6.4, and the related sets $F_\wp$, $I_{\wp,\omega_1^*}$ and $I_{\wp,\omega_2^*}$ as discussed in Section 6.3. Finally, to reduce clutter we let $\boldsymbol{F} = \bigwedge F$ and $\boldsymbol{I} = \bigwedge(I_{\wp,\omega_1^*} \cup I_{\wp,\omega_2^*})$.

Before stating and proving our main theorem, we need a simple lemma, related to violating tours as in Lemma 6.17 and Figure 6.9 (i). Let $\bar{h} = ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$, fix ℓ to be the length of $\bar{h}$, and let

$$\chi(x) = \bigvee \left\{\omega_1^*\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(x) \mid \bar{h} = ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3\right\}.$$

Lemma 6.28. Let Δ^* be a completion for Δ. Then, $\wp$ is violated in any model of $\Delta^*, \exists x\, \chi(x), \boldsymbol{F}$. Conversely, suppose $\mathfrak{A}$ is a structure such that $\mathfrak{A} \models \Delta^*, \boldsymbol{F}$ and $\mathfrak{A}$ has no observable cycles of length $\leq \ell$. If, in addition, $\mathfrak{A}$ contains an $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a}$ whose three initial elements are distinct and $\bar{a} \cap D = \emptyset$ (as in Figure 6.9 (i)), then $\mathfrak{A} \models \exists x\, \chi(x)$.

Proof. Let $\mathfrak{A}$ be a model of $\Delta^*, \exists x\, \chi(x), \boldsymbol{F}$. Since $\mathfrak{A} \models \exists x\, \chi(x) \wedge \boldsymbol{F}$, it follows that $\mathfrak{A} \models \exists x\, \omega_1^*\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(x)$, for some decomposition $ff^{-1}\bar{h}_1 h_1 \bar{h}_2 h_2 \bar{h}_3$ of $\bar{h}$. By Lemma 6.1, $\mathfrak{A} \models \exists x\, \omega_1\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(x)$; thus, there exists an $a \in A$ such that $\mathfrak{A} \models \omega_1\langle \bar{h}_1 h_1, \bar{h}_2 h_2, \bar{h}_3\rangle(a)$. But then it is evident (recall the semantics of the fan predicates in Lemma 6.15), that a is the start of an $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour in $\mathfrak{A}$, whose first elements are distinct (and it is also tree-shaped). This is (by definition) a violation of $\wp$—recall the discussion at the end of Section 6.1.

The converse is an easy consequence of Lemma 6.17. □

Theorem 6.29. $\mathrm{FinSat}(\mathcal{GC}^2\mathcal{DP}^2)$ is in $\mathsf{EXPTIME}$.

Proof. Let $\Delta, \psi : \wp$ (where $\wp = \mathrm{PFD}[\bar{f}f, \bar{g}]$) be given. Convert ψ into a new formula φ in normal form (as in Lemma 4.12) and search for a completion Δ^* of Δ (this can be done in exponential time). Recall from Section 6.1 that we only check for critical violations of $\wp$. Thus, to check for all violations, the following should be repeated for each dependency $\wp_{1\,..\,i} = \mathrm{PFD}[\bar{f}f_{1\,..\,i}, \bar{g}]$ $(0 < i \leq \ell)$, where $\bar{f}f_{1\,..\,i}$ is a prefix of length i of $\bar{f}f$.

First, check whether Δ^* is $\wp$-appropriate. By Lemma 6.27, this can be done in exponential time in the size of $\bar{f}, \bar{g}$ and Δ. If Δ^* is not $\wp$-appropriate, fail. Otherwise, check whether $\Delta^*, \varphi, \neg\exists x\, \chi(x), \boldsymbol{F}, \boldsymbol{I}$ is satisfiable. If the latter is satisfiable succeed, otherwise fail. Indeed, by Lemma 6.28, $\wp$ is violated in all models $\mathfrak{A}$ such that $\mathfrak{A} \models \exists x\, \chi(x)$. Thus, if we are ever going to satisfy Δ^*, φ under $\wp$, it has to be in a structure $\mathfrak{A}$ such that $\mathfrak{A} \models \neg\exists x\, \chi(x)$. Now, suppose that $\mathfrak{A}$ is a model of $\Delta^*, \varphi, \neg\exists x\, \chi(x), \boldsymbol{F}, \boldsymbol{I}$. By Lemma 4.24 we may assume that no cycles of length $\leq \ell$ exist in $\mathfrak{A}$. By Lemma 6.28, $\mathfrak{A}$ cannot contain any $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tours of the form illustrated in Figure 6.9 (i) (corresponding to Lemma 6.17). Further, given that Δ^* is $\wp$-appropriate, and by the semantics of the fan and isth predicates in Lemmas 6.15 and 6.16, $\mathfrak{A}$ cannot contain any $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tours of the form illustrated in Figure 6.9 (ii) – (x) (corresponding to Lemmas 6.19 – 6.25). But, then, we have covered all possible cases for violations of $\wp$. (The only remaining case would be an $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ with $a_0, a_2 \in D$ and $a_1 \notin D$. This, however, can never occur in $\mathfrak{A}$ since it is assumed to have no cycles of length $\leq \ell$, while a_0, a_1, a_2, a_0 would constitute a cycle of length 3—recall Remark 5.3.) Conversely, it is easily seen that if Δ^* is not $\wp$-appropriate, or if it is $\wp$-appropriate but $\Delta^*, \varphi, \neg\exists x\, \chi(x), \boldsymbol{F}, \boldsymbol{I}$ is unsatisfiable, then there exists no model of Δ^*, φ under $\wp$ (or at all). The $\mathsf{EXPTIME}$ upper bound, then, follows from Theorem 5.18. $\square$

The hardness for FinSat($\mathcal{GC}^2\mathcal{DP}^2$) follows from the fact that $\mathcal{GC}^2$ is $\mathsf{EXPTIME}$-hard. Thus, FinSat($\mathcal{GC}^2\mathcal{DP}^2$) is $\mathsf{EXPTIME}$-complete. The approach for ‘general’ satisfiability is the same as Sat($\mathcal{GC}^2\mathcal{D}$), thus we also have that Sat($\mathcal{GC}^2\mathcal{DP}^2$) is $\mathsf{EXPTIME}$-complete.

7 | Query Answering for $\mathcal{GC}^2\mathcal{DP}^2$

In this chapter we concern ourselves with (finite) query answering for $\mathcal{GC}^2\mathcal{DP}^2$. We first establish the *combined* complexity of (finite) query answering for the above fragment. We then turn our attention to the data complexity of this problem.

7.1 Combined complexity

The main idea is to reduce our problem to (finite) satisfiability for $\mathcal{GC}^2\mathcal{DP}^2$—$\Delta, \varphi \models \psi$ if and only if $\Delta, \varphi, \neg\psi$ is *un*satisfiable (this remains true when path-functional dependencies are present). The obvious difficulty with this approach is that ψ will in general contain more than two variables. It is shown in [PH09] that, by employing Lemma 4.24, one can replace $\neg\psi$ with an equivalent $\mathcal{GC}^2$-sentence. We now provide more details about this process, basically repeating [PH09], and then utilize the techniques introduced to obtain a 2-EXPTIME algorithm for (finite) query answering for $\mathcal{GC}^2\mathcal{DP}^2$.

Recall from Section 2.1 that a (boolean) conjunctive query is a sentence of the form

$$\exists x_1 \ldots \exists x_k \, \theta(x_1, \ldots, x_k), \tag{7.1}$$

where θ is a conjunction of binary predicates over a relational signature τ. Thus, the negation of a conjunctive query like the above, which is what we will be working with, is a sentence of the form

$$\forall x_1 \ldots \forall x_k \, \eta(x_1, \ldots, x_k), \tag{7.2}$$

where η is a *negative clause*, i.e. of the form $\neg p_1(y_1, z_1) \vee \ldots \vee \neg p_n(y_n, z_n)$, where each p_i $(1 \leq i \leq n)$ is some predicate in τ and $y_i, z_i \in \{x_1, \ldots, x_k\}$.

Our next goal is to identify the ways in which a sentence like (7.2) can enforce the existence of cycles (in a structure that satisfies it).

Definition 7.1. Let η be a clause, let T be the set of terms (variables or constants) occuring in η, and let

$$\begin{aligned}E = \{\{t_1, t_2\} \in T \mid t_1 \neq t_2 \text{ and either } t_1, t_2 \text{ both}\\ \text{occur in some literal of } \eta \text{ or are both constants}\}.\end{aligned}$$

Denote by G_η the (undirected) graph (T, E). (G is the empty graph when $\eta = \bot$.) We say that η is *cyclic* if G_η contains a cycle, at least one of whose vertices is a variable; otherwise we say that η is *acyclic*.

Definition 7.2. Let K be a set of constants. A *c-formula* (with respect to K) is a sentence of the form

$$\forall \bar{x} \Big(\Big(\bigwedge_{\substack{x \in \bar{x} \\ c \in K}} x \neq c \ \wedge \bigwedge_{\substack{x, x \in \bar{x} \\ x, x' \text{ distinct}}} x \neq x' \Big) \to \eta \Big), \tag{7.3}$$

where η is a cyclic negative clause.

It is clear that if K is a set of constants, v is a c-formula, and $\mathfrak{A}$ is a structure interpreting the constants in K, then $\mathfrak{A} \not\models v$ implies that $\mathfrak{A}$ contains a cycle of length at most $\|v\|$. This is used in the proof of Lemma 7.3 below.

Fix some set of constants K and a tuple of variables $\bar{x} = x_1, \ldots x_k$. Let Ξ be the set of all functions $\xi : \bar{x} \to \bar{x} \cup K$. For each $\xi \in \Xi$, denote by $\bar{x}_\xi$ the (possibly empty) tuple of variables $\xi(x_1), \ldots, \xi(x_k)$, with all constants and duplicates removed. Further, for any formula θ, denote by θ_ξ the result of simultaneously substituting the terms $\xi(x_1), \ldots, \xi(x_k)$ for all free occurrences of the respective variables $x_1, \ldots, x_k$ in θ. Then, $\forall \bar{x} \theta$ is logically equivalent to the sentence

$$\bigwedge_{\xi \in \Xi} \forall \bar{x}_\xi \Big(\Big(\bigwedge_{\substack{x \in \bar{x} \\ c \in K}} x \neq c \ \wedge \bigwedge_{\substack{x, x' \in \bar{x} \\ x, x' \text{ distinct}}} x \neq x' \Big) \to \theta_\xi \Big) \tag{7.4}$$

and, in addition, if Ξ_1, Ξ_2 are disjoint sets such that $\Xi_1 \cup \Xi_2 = \Xi$, $\forall \bar{x} \theta$ is logically equivalent to the sentence

$$\bigwedge_{\xi \in \Xi_1} \forall \bar{x}_\xi \Big(\Big(\bigwedge_{\substack{x \in \bar{x} \\ c \in K}} x \neq c \ \wedge \bigwedge_{\substack{x, x' \in \bar{x} \\ x, x' \text{ distinct}}} x \neq x' \Big) \to \theta_\xi \Big) \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \theta_\xi. \tag{7.5}$$

Indeed, denoting by φ_1 the formula (7.4) and by φ_2 the formula (7.5), it is evident that $\models \forall x\theta \to \varphi_2$, $\models \varphi_2 \to \varphi_1$, and $\models \varphi_1 \to \forall x\theta$.

Thus, we can rewrite a sentence of the form $\forall\bar{x}\eta$, where η is a negative clause, (possibly) containing constants from a (fixed) set K, as a conjunction of c-formulas and other 'regular' formulas

$$\bigwedge_{\xi\in\Xi_1} \forall\bar{x}_\xi \Big(\Big(\bigwedge_{\substack{x\in\bar{x}\\ c\in K}} x \neq c \;\wedge \bigwedge_{\substack{x,x'\in\bar{x}\\ x,x' \text{ distinct}}} x \neq x'\Big) \to \eta_\xi\Big) \wedge \bigwedge_{\xi\in\Xi_2} \forall\bar{x}_\xi\eta_\xi, \tag{7.6}$$

where Ξ is the set of functions from $\bar{x}$ to $\bar{x} \cup K$, and

$$\begin{aligned} \Xi_1 &= \{\xi \in \Xi \mid \eta_\xi \text{ is cyclic}\}, \\ \Xi_2 &= \{\xi \in \Xi \mid \eta_\xi \text{ is acyclic}\}. \end{aligned}$$

Now, when considering the satisfiability of a sentence like (7.6), the following lemma allows one to discard the c-formulas.

Lemma 7.3. Let φ be a $\mathcal{GC}^2$-formula, Δ a database, and Υ a finite set of c-formulas. Then, $\Delta \cup \{\varphi\} \cup \Upsilon$ is (finitely) satisfiable if and only if $\Delta \cup \{\varphi\}$ is (finitely) satisfiable. Further, the equisatisfiability of $\Delta \cup \{\varphi\} \cup \Upsilon$ and $\Delta \cup \{\varphi\}$ holds in the presence of path-functional dependencies.

Proof. By Lemma 4.12, we may assume that φ is in normal form. The only-if direction is trivial. For the other direction, suppose that $\mathfrak{A}$ is a (finite) model of $\Delta \cup \{\varphi\}$ and let $K \subseteq A$ be the set of elements interpreting individual constants from Δ or Υ. Pick $\Omega > \max\{\|v\| : v \in \Upsilon\}$. By Lemma 4.24, let $\mathfrak{B}$ be a model of $\Delta \cup \{\varphi\}$, such that (i) $K \subseteq B$, (ii) $\mathfrak{A}|_K = \mathfrak{B}|_K$, and (iii) $\mathfrak{B}$ contains no (observable) cycles of length less than Ω. ($\mathfrak{B}$ is finite if $\mathfrak{A}$ is.) It follows that $\mathfrak{B} \models \Upsilon$, otherwise $\mathfrak{B}$ would contain a cycle of length $\leq \Omega$.

We claim that the above result holds even in the presence of path-functional dependencies. Indeed, let $\wp = \mathrm{PFD}[\bar{f}f, \bar{g}]$ be a path-functional dependency. The only-if direction is, again, trivial. For the other direction, suppose that $\Delta \cup \{\varphi\}$ is satisfiable under $\wp$; thus, let $\mathfrak{A}$ be a (finite) model of $\Delta \cup \{\varphi\}$ that contains no $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tours. Now, pick $\Omega > \max\{k, \ell\}$, where $k = \max\{\|v\| : v \in \Upsilon\}$ and ℓ is the length of the sequence $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$; obtain, by Lemma 4.24 a model $\mathfrak{B}$ of $\Delta \cup \{\varphi\}$ as above. Clearly, $\mathfrak{B}$ does not contain any $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tours because $\mathfrak{A}$ does not, and, since $\Omega > \max\{k, \ell\}$, $\mathfrak{B} \models \Upsilon$. $\square$

Finally, sentences of the form $\forall \bar{x}_\xi \eta_\xi$, where η_ξ is an acyclic negative clause, can by replaced by logically equivalent $\mathcal{GC}^2$-sentences (Lemma 7.6).

Definition 7.4. Let η be a clause. We call η *splittable* if, by re-ordering its literals, it can be written as $\eta_1 \vee \eta_2$, with η_1 and η_2 non-empty, and $\mathrm{Vars}(\eta_1) \cap \mathrm{Vars}(\eta_2) = \emptyset$; otherwise η is *unsplittable*.

Lemma 7.5. Let η be a non-ground clause. If η is unsplittable and acyclic, then it contains at most one individual constant.

Proof. Suppose, for contradiction, that η contains two distinct constants c and d; thus, $\{c, d\}$ is an edge of the graph G_η. Because η is non-ground, c and d cannot appear in the same literal of η. Hence, there are two distinct variables, say x and y, such that x appears in the same literal with c and y appears in the same literal with d. Because η is unsplittable, there has to be a path $\bar{p}$ in G_η connecting x and y. Then, G_η contains the cycle $c, \bar{p}, d, c$, of length ≥ 3, which is absurd. $\square$

Lemma 7.6. Let $\eta(x, \bar{x})$ be a negative clause with no individual constants, involving exactly the variables $x, \bar{x}$. Suppose further that $\eta(x, \bar{x})$ is non-empty, unsplittable, and acyclic. Then, there exists a $\mathcal{GC}^2$-formula $\omega(x)$ such that $\forall \bar{x} \eta(x, \bar{x})$ and $\omega(x)$ are logically equivalent.

Proof. Simple induction on the number of variables in η, taking advantage of the fact that G_η is acyclic (i.e. a tree). See [PH09] for more details. $\square$

Before we state our main result for this section, we need the following simple lemma, which allows us to remove the constants appearing in a formula at the expense of adding some ground literals.

Lemma 7.7. Let $\varphi(x)$ be a formula containing at most one free variable x, Θ a set of sentences, and K a set of individual constants. Let p be a predicate not occurring in φ or Θ. Then, the sets

$$\Theta \cup \{\varphi(c) \mid c \in K\} \text{ and } \Theta \cup \{p(c) \mid c \in K\} \cup \{\forall x(\varphi(x) \vee \neg p(x))\}$$

are satisfiable over the same domains.

Proof. Let ψ be the sentence $\forall x(\varphi(x) \vee \neg p(x))$. It is clear that $\{p(c), \psi\} \models \varphi(c)$, for all $c \in K$. Now, suppose $\mathfrak{A} \models \bigwedge \Theta \cup \{\varphi(c) \mid c \in K\}$; then we can expand $\mathfrak{A}$ to a structure $\mathfrak{A}'$ that interprets p, by setting $p^{\mathfrak{A}'} = \{c^{\mathfrak{A}} \mid c \in K\}$. $\square$

The next theorem closely follows [PH09, Theorem 4] (our inputs, and, as a result, the time bounds are different).

Theorem 7.8. DFinQAns($\mathcal{GC}^2\mathcal{DP}^2$) and DQAns($\mathcal{GC}^2\mathcal{DP}^2$) are in 2-EXPTIME.

Proof. We give a proof for DFinQAns($\mathcal{GC}^2\mathcal{DP}^2$); the proof for DQAns($\mathcal{GC}^2\mathcal{DP}^2$) is analogous. Let Δ, a database, φ, a $\mathcal{GC}^2$-sentence, ψ, a boolean conjunctive query, and PFD$[\bar{f}, \bar{g}]$, a path-functional dependency, be given. We wish to determine whether $\Delta, \varphi \models_{\text{fin}} \psi$ under PFD$[\bar{f}, \bar{g}]$, which amounts to determining whether

$$\bigwedge \Delta \wedge \varphi \wedge \neg\psi, \tag{7.7}$$

is finitely *un*satisfiable under PFD$[\bar{f}, \bar{g}]$. Thus, we provide a way to decide finite satisfiability for (7.7) under PFD$[\bar{f}, \bar{g}]$. For the rest of the proof we assume that Δ is complete, for, if not, we can search for a completion of Δ in exponential time with respect to the size of Δ.

Since ψ is a (positive) conjunctive query, $\neg\psi$ is of the form $\forall\bar{x}\eta$, where η is a negative clause. Let K be the set of constants appearing in Δ and ψ; let $n = |\Delta|$ be the cardinality of Δ, let m be the arity of $\bar{x}$, and let Ξ be the set of all functions $\xi : \bar{x} \to \bar{x} \cup K$. Thus, $|\Xi| \leq (n+m)^m$; that is, $|\Xi|$ is bounded above by an exponential function in the size of our input.

As discussed earlier, we can rewrite $\forall\bar{x}\eta$ as in (7.6), thus (7.7) is logically equivalent to

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge_{\xi \in \Xi_1} \forall \bar{x}_\xi \Big(\Big(\bigwedge_{\substack{x \in \bar{x} \\ c \in K}} x \neq c \;\wedge \bigwedge_{\substack{x, x' \in \bar{x} \\ x, x' \text{ distinct}}} x \neq x' \Big) \to \eta_\xi \Big) \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta_\xi, \tag{7.8}$$

where

$$\begin{aligned} \Xi_1 &= \{\xi \in \Xi \mid \eta_\xi \text{ is cyclic}\}, \\ \Xi_2 &= \{\xi \in \Xi \mid \eta_\xi \text{ is acyclic}\}. \end{aligned}$$

For simplicity, write the latter formula as

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta_\xi, \tag{7.9}$$

where Υ is finite set of c-formulas with respect to D.

Notice the previous two formulas (7.8) and (7.9) are of size $O(|\Xi|)$, and, thus, exponential in the size of our input. In addition, they can be computed in time bounded by a polynomial function of $|\Xi|$.

Formula (7.9) can further be simplified as

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta^*_\xi, \tag{7.10}$$

where each η^*_ξ ($\xi \in \Xi_2$) is $\top$ if any ground literal of η_ξ appears in Δ; otherwise η^*_ξ is the result of deleting from η_ξ all ground literals whose negation appears in Δ. If no literals remain, η^*_ξ is taken to be $\bot$. Now, because Δ is assumed to be complete, it contains every ground literal or its negation, over the relevant signature. Thus, each resulting formula η^*_ξ will be non-ground. In addition, we write each $\forall \bar{x} \eta^*_\xi$ in (7.10) as

$$\forall \bar{x}_\xi (\eta^0_\xi \vee \ldots \vee \eta^m_\xi),$$

for some $m \geq 0$ (depending on ξ), where each η^i_ξ ($0 \leq i \leq m$) is unsplittable, and (non-deterministically) choose one such η^i_ξ. Note that m is at most linear in $\|\forall \bar{x} \eta^*_\xi\|$, thus all possible choices (for each $\forall \bar{x} \eta^*_\xi$) can be tried in double exponential time with respect to our input size. We are, then, left we with a formula

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta^{i(\xi)}_\xi, \tag{7.11}$$

where each $\eta^{i(\xi)}_\xi$ is acyclic, unsplittable, and non-ground; hence, by Lemma 7.5, each $\eta^{i(\xi)}_\xi$ contains at most one constant—without loss of generality we may assume it contains exactly one constant $c^{i(\xi)}$.

It is clear that (7.10), and thus (7.9), is finitely satisfiable only if one of the formulas (7.11) is finitely satisfiable. To determine whether a formula of the form (7.11) is finitely satisfiable, we will employ Theorem 6.29; to use it, however, we need to remove all constants that do not appear in Δ. To this end, let $\theta^{i(\xi)}_\xi$ be the result of replacing all occurrences of $c^{i(\xi)}$ in each $\eta^{i(\xi)}_\xi$ ($\xi \in \Xi_2$) with x (where x does not occur in $\eta^{i(\xi)}_\xi$), and p_ξ be a new predicate for each $\xi \in \Xi_2$, subject to the restriction that if $\eta^{i(\xi)}_\xi = \eta^{i(\xi')}_{\xi'}$ ($\xi, \xi' \in \Xi_2$) then $p_\xi = p_{\xi'}$. Thus, because the constants replaced are over the signature of ψ, the number of all the predicates p_ξ is bounded above by a polynomial function in the size of our input. Then, by

Lemma 7.7, (7.11) is satisfiable over the same domains as

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall x \bar{x}_\xi (\theta_\xi^{i(\xi)} \vee \neg p_\xi(x)). \tag{7.12}$$

Note that (7.12) can be computed in linear time with respect to the size of (7.11), thus in exponential time with respect to the size of our input. By Lemma 7.6, for each $\forall x \bar{x}_\xi (\theta_\xi^{i(\xi)} \vee \neg p_\xi(x))$, there is a logically equivalent $\mathcal{GC}^2$-formula $\forall x\, \omega_\xi(x)$. Let ω be the conjunction of all these $\forall x\, \omega_\xi(x)$ ($\xi \in \Xi_2$). Then, (7.12) is logically equivalent to

$$\bigwedge \Delta \wedge (\varphi \wedge \omega) \wedge \bigwedge \Upsilon. \tag{7.13}$$

Now, by Lemma 7.3, (7.13) is finitely satisfiable if and only if

$$\bigwedge \Delta \wedge (\varphi \wedge \omega) \tag{7.14}$$

is finitely satisfiable. To determine whether (7.14) is finitely satisfiable under PFD$[\bar{f}, \bar{g}]$, we employ Theorem 6.29. Thus, the result is determined in exponential time with respect to the size of (7.14); and, because the size of (7.14) is exponential with respect to the size of our input, the result is determined in double exponential time with respect to our input. Recall from the above that this procedure is in fact performed for each possible formula (7.11), from which (7.14) is derived, but this still leaves us in 2-EXPTIME. □

Since $\mathcal{GC}^2$ properly contains the description logic $\mathcal{ALCQI}$, the result due to Lutz [Lut08] establishes in combination with Theorem 7.8 that DQAns($\mathcal{GC}^2\mathcal{DP}^2$) is 2-EXPTIME-complete. (Recall Section 4.3.) We remark that Lutz's result is with respect to infinite models. However, we believe that a careful analysis of his proof (which we were unable to do due to lack of time) will reveal that it also holds for finite models. If this turns out to be true, we immediately obtain that DFinQAns($\mathcal{GC}^2\mathcal{DP}^2$) is also 2-EXPTIME-complete.

7.2 Data complexity

We now turn our attention to the data complexity of DFinQAns($\mathcal{GC}^2\mathcal{DP}^2$) and DQAns($\mathcal{GC}^2\mathcal{DP}^2$). That is, we study the computational complexity of those problems when the background theory and the path-functional dependency (or dependencies) are fixed; only the database is allowed to vary. As in the previous section, we will reduce (finite) query answering to (finite) satisfiability. Thus, we first need to establish the data complexity of (finite) satisfiability for $\mathcal{GC}^2\mathcal{DP}^2$.

Theorem 7.9. Let φ be a fixed $\mathcal{GC}^2$-sentence and $\mathrm{PFD}[\bar{f}f, \bar{g}]$, abbreviated $\wp$, a fixed path-functional dependency. Then, the problems $\mathrm{DFinSat}_{\varphi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ and $\mathrm{DSat}_{\varphi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ are both in NP.

Proof. Let Δ, a database, be given; let D be the active domain of Δ. By Lemma 4.12, we may assume that φ is in normal form. We provide a procedure for deciding the finite satisfiability of $\Delta, \varphi : \mathrm{PFD}[\bar{f}f, \bar{g}]$ that can be executed in time bounded by a linear function of $|\Delta|$. Recall from Section 6.1 that we only check for critical violations of $\mathrm{PFD}[\bar{f}f, \bar{g}]$; thus, to check for all violations, the following should be repeated for each dependency $\mathrm{PFD}[\bar{f}f_{1..i}, \bar{g}]$ $(0 < i \leq \ell)$, where $\bar{f}f_{1..i}$ is a prefix of length i of $\bar{f}f$. Let φ' be the sentence $\varphi \wedge \neg\exists x\chi(x) \wedge \boldsymbol{F} \wedge \boldsymbol{I}$, where $\boldsymbol{F}$, $\boldsymbol{I}$, and $\chi(x)$ are as in Section 6.5. The procedure is as follows:

1. Guess a completion Δ^* of Δ.

2. Check if Δ^* is $\mathrm{PFD}[\bar{f}f, \bar{g}]$-appropriate. If not, fail.

3. Recalling Section 5.3, guess, for each constant $c \in D$, the values of the vectors $\gamma^c_{\pi,\mathtt{s}}$, for each $\mathtt{s} \in \bigcup\{\mathrm{flip}(\mathtt{s}') : \mathtt{s}' \in \Lambda_c\}$, and $\delta^c_{\pi,\mathtt{t}}$, for each $\mathtt{t} \in \bigcup\{\mathrm{flip}(\mathtt{t}') : \mathtt{t}' \in M_c\}$, for each $c \in D$.

4. Write down the system $\mathcal{E}$ with respect to φ' and Δ^* as in Section 5.3—the vectors from Step 3 are used in $\mathcal{E}_4$—and guess a small enough $(\leq \boldsymbol{C})$ solution of $\mathcal{E}$ in the positive integers. If a solution does not exist fail.

5. Succeed.

The procedure for the problem $\mathrm{DSat}_{\varphi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ is exactly the same as the above, except in Step 4, where we seek solutions in the set $\mathbb{N} \cup \{\aleph_0\}$, as described at the end of Chapter 5.

We now consider the running time of the above procedure. Step 1 can be executed in non-deterministic polynomial time with respect to $|\Delta|$. Step 2 can be executed in linear time with respect to $|\Delta|$. Step 3 can be executed in linear time with respect to $|\Delta|$—for each $c \in D$, the required vectors $\gamma^c_{\pi,\mathtt{s}}$ and $\delta^c_{\pi,\mathtt{t}}$ can be guessed in constant time. For Step 4, observe that the size of $\mathcal{E}$ is linear with respect to $|\Delta|$. In particular (referring to Section 5.3) the size of the systems $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ is bounded above by a constant; the size of $\mathcal{E}_4$ is linear with respect to $|\Delta|$. Thus, Step 4 can be performed in non-deterministic polynomial time with respect to $|\Delta|$. Consequently, the above procedure can be executed in non-deterministic polynomial time with respect to Δ. Notice that executing the procedure for each dependency $\text{PFD}[\bar{f}f_{1..i}, \bar{g}]$, as described above, still leaves it in NP, since the size of $\text{PFD}[\bar{f}f, \bar{g}]$ is bounded above by a constant. The procedure corresponding to $\text{DSat}_{\varphi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ is also in NP, by a similar argument.

Finally, we show that the above procedure succeeds if and only if $\Delta \cup \{\varphi\}$ is finitely satisfiable under $\text{PFD}[\bar{f}f, \bar{g}]$. Indeed, if Step 2 succeeds, it is clear that no violating tour of the forms illustrated in Figure 6.9 (ii) – (x), corresponding to Lemmas 6.19 – 6.25 can occur in any model of Δ^*, φ. Further, if Step 4 succeeds, there exists by Lemma 5.16 a finite model $\mathfrak{A}$ of Δ^*, φ. In addition, $\mathfrak{A} \models \neg\exists x\, \chi(x) \wedge \boldsymbol{F} \wedge \boldsymbol{I}$, thus, by Lemma 6.28, $\mathfrak{A}$ cannot contain any violating tours of the form illustrated in Figure 6.9 (i), corresponding to Lemma 6.17. (By Lemma 4.24 we may assume that $\mathfrak{A}$ contains no cycles of length $\leq \|ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}\|$.) Then, $\mathfrak{A}$ is a model of Δ, φ in which no violations of $\text{PFD}[\bar{f}f, \bar{g}]$ occur. The converse implication is immediate.

The argument for 'general' satisfiability is analogous. $\square$

The following lemma shows that we cannot do better than the bound in Theorem 7.9 (unless $\mathsf{P} = \mathsf{NP}$). In fact, this is true even without path-functional dependencies (i.e. for $\mathcal{GC}^2\mathcal{D}$).

Theorem 7.10. There exists a $\mathcal{GC}^2$-sentence φ for which $\text{DFinSat}_\varphi(\mathcal{GC}^2\mathcal{D})$ and $\text{DSat}_\varphi(\mathcal{GC}^2\mathcal{D})$ are NP-hard.

Proof. Reduction from `3COLOURING`. Let φ be the conjunction of the following sentences, over the signature $\langle c_1(\cdot), c_2(\cdot), c_3(\cdot), e(\cdot,\cdot)\rangle$ ($\oplus$ is exclusive or):

(i) $\forall x(c_1(x) \oplus c_2(x) \oplus c_3(x))$;

(ii) $\forall x \forall y(e(x,y) \leftrightarrow e(y,x))$;

(iii) $\neg\exists x \exists y(e(x,y) \wedge \bigvee_{1 \leq i \leq 3} (c_i(x) \wedge c_i(y)))$.

The unary predicate $c_1(x)$ is to be read as 'x has color 1'; likewise for $c_2(x)$ and $c_3(x)$. The binary predicate $e(x,y)$ is to be read as 'there is a directed edge from

x to y'. Thus, in any structure satisfying φ, (i) ensures that every element has colour 1, 2, or 3; (ii) ensures that e is symmetric; and (iii) ensures that no two adjacent elements are coloured with the same colour.

Now, given any undirected graph $G = (V, E)$, we construct the database

$$\Delta_G = \{e(v_1, v_2) \mid (v_1, v_2) \in E\}.$$

It is easily checked that $\Delta_G \cup \{\varphi\}$ is (finitely) satisfiable if and only if G is 3-colourable. $\square$

For the data complexity of (finite) query answering for $\mathcal{GC}^2\mathcal{DP}^2$ we proceed as in the proof of Theorem 7.8, utilizing Theorem 7.9. The next theorem is essentially the same as [PH09, Theorem 4].

Theorem 7.11. Let φ be a fixed $\mathcal{GC}^2$-sentence, ψ be a fixed boolean conjunctive query, and PFD[$\bar{f}f, \bar{g}$], abbreviated $\wp$, be a fixed path-functional dependency. Then, the problems $\mathrm{DQAns}_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ and $\mathrm{DFinQAns}_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ are both in coNP.

Proof. The proof is completely analogous with the proof of Theorem 7.8, but the sizes of the formulas that we write are different. Again, we give a proof for $\mathrm{DFinQAns}_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$; the proof for $\mathrm{DQAns}_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ is similar.

Let Δ, a database, be given; let $n = |\Delta|$, and let D be the active domain of Δ. We wish to determine whether $\Delta, \varphi \models_{\text{fin}} \psi$ under PFD[$\bar{f}, \bar{g}$], which amounts to checking the *un*satisfiability of

$$\bigwedge \Delta \wedge \varphi \wedge \neg\psi \tag{7.15}$$

under PFD[$\bar{f}f, \bar{g}$]. For the rest of the proof we assume that Δ is complete, for, if not, a completion of Δ can be guessed in non-deterministic polynomial time with respect to $|\Delta|$. As in the proof of Theorem 7.8, we rewrite (7.15) into

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge_{\xi \in \Xi_1} \forall \bar{x}_\xi \Big(\Big(\bigwedge_{\substack{x \in \bar{x} \\ c \in K}} x \neq c \;\wedge \bigwedge_{\substack{x, x' \in \bar{x} \\ x \neq x' \text{ distinct}}} x \neq x' \Big) \to \eta_\xi \Big) \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta_\xi, \tag{7.16}$$

where K is the set of constants in Δ and ψ, Ξ is the set of all functions from $\bar{x}$ to $\bar{x} \cup K$, and

$$\Xi_1 \;=\; \{\xi \in \Xi \mid \eta_\xi \text{ is cyclic}\},$$

$$\Xi_2 = \{\xi \in \Xi \mid \eta_\xi \text{ is acyclic}\}.$$

Let m be the arity of $\bar{x}$; m is constant. Notice that $|\Xi| \leq (n+m)^m$. Thus, $|\Xi|$ is bounded above by a function polynomial in n and, hence, $|\Delta|$. In addition, formula (7.16) can be computed in polynomial time with respect to $|\Xi|$. For simplicity, we write the latter formula as

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta_\xi, \tag{7.17}$$

where Υ is finite set of c-formulas with respect to D.

Formula (7.17) is further be simplified as

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta^*_\xi, \tag{7.18}$$

where each η^*_ξ ($\xi \in \Xi_2$) is $\top$ if any ground literal of η_ξ appears in Δ; otherwise η^*_ξ is the result of deleting from η_ξ all ground literals whose negation appears in Δ. If no literals remain, η^*_ξ is taken to be $\bot$. Now, because Δ is assumed to be complete, it contains every ground literal or its negation, over the relevant signature. Thus, each resulting formula η^*_ξ will be non-ground. In addition, we write each $\forall \bar{x} \eta^*_\xi$ in (7.10) as

$$\forall \bar{x}_\xi (\eta^0_\xi \vee \ldots \vee \eta^m_\xi),$$

for some $m \geq 0$ (depending on ξ), where each η^i_ξ ($0 \leq i \leq m$) is unsplittable, and (non-deterministically) choose one such η^i_ξ. Because each such m is constant, this non-deterministic step can be complete in constant time with respect to $|\Xi_2|$, and, hence, in polynomial time with respect to $|\Delta|$. We are left we with

$$\bigwedge \Delta \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall \bar{x}_\xi \eta^{i(\xi)}_\xi, \tag{7.19}$$

where each $\eta^{i(\xi)}_\xi$ is acyclic, unsplittable, and non-ground; hence, by Lemma 7.5, each $\eta^{i(\xi)}_\xi$ contains at most one constant—without loss of generality we may assume it contains exactly one constant $c^{i(\xi)}$.

It is clear that (7.18), and thus (7.17), is finitely satisfiable only if one of the formulas (7.11) is finitely satisfiable. To remove all constants that appear outside

Δ, let $\theta_\xi^{i(\xi)}$ be the result of replacing all occurrences of $c^{i(\xi)}$ in each $\eta_\xi^{i(\xi)}$ ($\xi \in \Xi_2$) with x (where x does not occur in $\eta_\xi^{i(\xi)}$), and p_ξ be a new predicate for each $\xi \in \Xi_2$, subject to the restriction that if $\eta_\xi^{i(\xi)} = \eta_{\xi'}^{i(\xi')}$ ($\xi, \xi' \in \Xi_2$) then $p_\xi = p_{\xi'}$. Thus, because the constants replaced are over the signature of ψ, which is fixed, the number of all the predicates p_ξ is bounded above by a constant. Then, by Lemma 7.7, (7.11) is satisfiable over the same domains as

$$\bigwedge \cup \wedge \varphi \wedge \bigwedge \Upsilon \wedge \bigwedge_{\xi \in \Xi_2} \forall x \bar{x}_\xi (\theta_\xi^{i(\xi)} \vee \neg p_\xi(x)). \tag{7.20}$$

Note that (7.20) can be computed in linear time with respect to the size of (7.19), thus in polynomial time with respect to the size of our input. By Lemma 7.6, for each $\forall x \bar{x}_\xi (\theta_\xi^{i(\xi)} \vee \neg p_\xi(x))$, there is a logically equivalent $\mathcal{GC}^2$-formula $\forall x\, \omega_\xi(x)$. Let ω be the conjunction of all these $\forall x\, \omega_\xi(x)$ ($\xi \in \Xi_2$)—note that ω is independent of Δ. Then, (7.20) is logically equivalent to

$$\bigwedge \Delta \wedge (\varphi \wedge \omega) \wedge \bigwedge \Upsilon. \tag{7.21}$$

Now, by Lemma 7.3, (7.21) is finitely satisfiable if and only if

$$\bigwedge \Delta \wedge (\varphi \wedge \omega) \tag{7.22}$$

is finitely satisfiable. To determine whether (7.22) is finitely satisfiable under PFD$[\bar{f}, \bar{g}]$, we employ Theorem 7.9. Thus, we obtain the promised bound—recall that we are interested in checking *un*satisfiability. □

The lower bound for DQAns$_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ and DFinQAns$_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ is established using Theorem 7.10. Indeed, notice that, even without path-functional dependencies, query answering is at least as hard as *un*satisfiability: if p is a predicate not occurring in Δ or φ, then $\Delta \cup \varphi \models \exists x p(x)$ if and only if $\Delta \cup \{\varphi\}$ is unsatisfiable. (This is independent of models being finite or infinite.) But, it follows from Theorem 7.10 that (finite) *un*satisfiability for $\mathcal{GC}^2\mathcal{D}$—hence for $\mathcal{GC}^2\mathcal{DP}^2$ too—is coNP-hard. Thus, the problems DQAns$_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ and DFinQAns$_{\varphi,\psi,\wp}(\mathcal{GC}^2\mathcal{DP}^2)$ are coNP-complete.

8 Conclusion

In this thesis we studied the interplay of (binary) path-functional dependencies with the two-variable guarded fragment with counting and a database. Our work elucidates the interaction among the above domains and is, thus, pertinent to both database and computational logic research. Our contributions are the following:

- In Chapter 5 we modified the proof in [PH07] so as to accommodate the presence of a database. [PH07] establishes the complexity of (finite) satisfiability for the two-variable guarded fragment with counting. The proof is by reduction to integer linear programming. We were unable to find a way to use this result directly, so we adapted the proof by adding inequalities that take a given database into account and showed that these inequalities have the desired effect. The challenging part in this approach was to write the minimum number of inequalities required to obtain our result: all the obvious ways to incorporate the database lead to a higher complexity bound. As a result, one needs to be careful about what information from the database is ‘recorded’ by means of the inequalities. This, of course, makes the proof of correctness rather technical.

- In Chapter 6 we showed how (finite) satisfiability under (binary) path-functional dependencies for the two-variable guarded fragment with counting and a database can be reduced to ‘plain’ (finite) satisfiability for this fragment. This was done by identifying violations of a given path-functional dependency with certain kinds of tours in graphs, and then showing how to enumerate and detect all possible shapes of such tours. These shapes are rather complex, but it turns out they can be decomposed into simple tree-shaped components. All these tree-shaped components and various more complex configurations containing them can be succinctly described by guarded two-variable formulas with counting quantifiers. We remark that Lemma 4.24 is crucial for this whole approach.

- In Chapter 7 we established the complexity of (finite) query answering for the two-variable guarded fragment with counting and a database under (binary) path-functional dependencies. We adopted the approach in [PH09] where, again using Lemma 4.24, conjunctive queries are rewritten and replaced by equivalent guarded two-variable formulas with counting quantifiers, and (finite) query answering is reduced to (finite) satisfiability. (This is for the upper bound; the lower bound was obtained using a result due to Lutz [Lut08].) The proofs of Theorems 7.8 and 7.11 closely followed the corresponding proofs in [PH09], but, of course, taking into account the different time bounds (mostly for the first result) and the possible issues arising from the presence of path-functional dependencies. In Theorem 7.9 we established the upper bound for the data complexity of (finite) satisfiability under path-functional dependencies, and in Theorem 7.10 we established the lower bound for the same problem. (These results are used in 7.11, since, as mentioned above, query answering is reduced to satisfiability.)

In summary, we established the complexity of (finite) satisfiability for the guarded two-variable fragment with counting and a database, we introduced new techniques to systematically eliminate (binary) path-functional dependencies so that (finite) satisfiability in their presence can be handled with existing methods, and established the complexity of (finite) query answering—both combined and data complexity—for the above fragment (with path-functional dependencies).

Future work

Multiple dependencies

The obvious generalization of this research is to consider the fragment $\mathcal{GC}^2\mathcal{DP}^n$ (for any positive integer $n > 2$) allowing n-ary path-functional dependencies, i.e. expressions of the form

$$\Delta, \psi : \mathrm{PFD}[\bar{f}_1, \ldots, \bar{f}_n],$$

where Δ is a database, ψ is a $\mathcal{GC}^2$-sentence and $\bar{f}_1, \ldots, \bar{f}_n$ are sequences of functional predicates. The semantics is similar to the binary case: we say that Δ, ψ

is *satisfiable under* the dependency $\mathrm{PFD}[\bar{f}_1, \ldots, \bar{f}_n]$ if there exists a structure $\mathfrak{A}$ such that $\mathfrak{A} \models \Delta, \psi$ and, for all $a, b \in A$,

$$\bigwedge_{1 \leq i \leq n} \bar{f}_i^{\mathfrak{A}}(a) \bowtie \bar{f}_i^{\mathfrak{A}}(b) \implies a = b.$$

(Recall that $\bar{h}^{\mathfrak{A}}(a)$ denotes the $\bar{h}$-path in $\mathfrak{A}$ starting at a; $\bowtie$ denotes path convergence. All such paths are assumed to be total.) In that case, we write $\mathfrak{A} \models \Delta, \psi : \mathrm{PFD}[\bar{f}_1, \ldots, \bar{f}_n]$. We speak of *finite* satisfiability when in the above definition we impose the extra restriction that $\mathfrak{A}$ be finite. If, for a given structure $\mathfrak{A}$, there are two *distinct* elements $a, b \in A$ such that $\bar{f}_i^{\mathfrak{A}}(a) \bowtie \bar{f}_i^{\mathfrak{A}}(b)$, for each i $(1 \leq i \leq n)$, we say that the path-functional dependency $\mathrm{PFD}[\bar{f}, \bar{g}]$ is *violated* in $\mathfrak{A}$. Each such pair of elements a, b is called a *violating pair* for $\mathrm{PFD}[\bar{f}, \bar{g}]$.

As in the binary case, we are interested in deciding (finite) satisfiability under a given path-functional dependency like the above. With argumentation similar to the one for the binary case, we may restrict our attention to *critical* violations of any given n-ary $(n > 2)$ path-functional dependency. It follows, then, from the above definition that a dependency $\wp = \mathrm{PFD}[\bar{f}_1 f_1, \ldots, \bar{f}_n]$ is violated in a structure $\mathfrak{A}$ if there exists (in $\mathfrak{A}$) a configuration of the form illustrated in Figure 8.1, involving a violating pair a, b.

Here, too, we wish to define such violations in terms of the (distinct) elements c, d, and e. Thus, we might be tempted to give the following *false* definition of a violation for $\wp$, in terms of the elements c, d, and e: $\wp$ is violated in $\mathfrak{A}$ if there are three distinct elements c, d, and e (in A) such that, for each j $(2 \leq j \leq n)$, there exists an $f_1 f_1^{-1} \bar{f}_1^{-1} \bar{f}_j \bar{f}_j^{-1} \bar{f}_1$-tour (in $\mathfrak{A}$) whose first three elements are c, d, and e. Of course, this definition does not guarantee that all these putative tours 'pass through' the same element b, after concluding the initial $f_1 f_1^{-1} \bar{f}_1^{-1}$-part of the tour, and then through the same element a, after concluding the $\bar{f}_j \bar{f}_j^{-1}$-part of the tour. Thus, it cannot be used to check dependency violations. For the correct version of the definition we require the following notions.

Definition 8.1. An $\bar{h}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ is $\bar{r}$-*initially-congruent* with an $\bar{h}'$-tour $\bar{a}' = a'_0, \ldots, a'_{\ell-1}$ if $\bar{h} = \bar{r}\,\bar{t}$, $\bar{h}' = \bar{r}\,\bar{t}'$, and $\bar{a}, \bar{a}'$ can be decomposed respectively as $a_0 \cdots a_\eta \cdots a_{\ell-1}$ and $a'_0 \cdots a'_\eta \cdots a'_{\ell-1}$, where $a_0 \cdots a_\eta$ and $a'_0 \cdots a'_\eta$ are the *same* $\bar{r}$-path (i.e. $a_k = a'_k$, for all $0 \leq k \leq \eta$), $a_\eta \cdots a_\ell$ is a $\bar{t}$-path, and $a'_\eta \cdots a'_\ell$ is a $\bar{t}'$-path. We refer to the element a_η $(= a'_\eta)$ as the *initial fork*.

Definition 8.2. An $\bar{h}$-tour $\bar{a} = a_0, \ldots, a_{\ell-1}$ is $\bar{s}$-*finally-congruent* with an $\bar{h}'$-tour

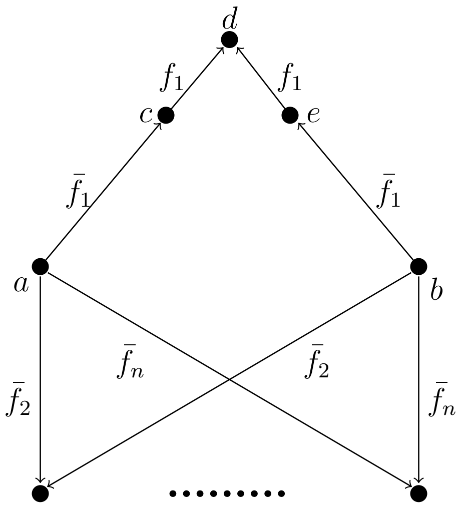


Figure 8.1: A violating pair a, b for PFD$[\bar{f}_1 f_1, \dots, \bar{f}_n]$. The elements c, d, and e are distinct.

$\bar{a}' = a'_0, \dots, a'_{\ell-1}$ if $\bar{h} = \bar{t}\,\bar{s}$, $\bar{h}' = \bar{t}'\,\bar{s}$, and $\bar{a}, \bar{a}'$ can be decomposed respectively as $a_0 \cdots a_\xi \cdots a_{\ell-1}$ and $a'_0 \cdots a'_\xi \cdots a'_{\ell-1}$, where $a_0 \cdots a_\xi$ is a $\bar{t}$-path, $a'_0 \cdots a'_\xi$ is a $\bar{t}'$-path, and $a_\xi \cdots a_\ell$ and $a'_\xi \cdots a'_\ell$ are the *same* $\bar{s}$-path respectively (i.e. $a_k = a'_k$, for all $\xi \leq k \leq \ell$). We refer to the element a_ξ $(= a'_\xi)$ as the *final join.*

Definition 8.3. Two tours, an $\bar{h}$-tour $\bar{a} = a_0, \dots, a_{\ell-1}$ and an $\bar{h}'$-tour $\bar{a}' = a'_0, \dots, a'_{\ell-1}$, are $(\bar{r}, \bar{s})$-*congruent* if $\bar{h} = \bar{r}\,\bar{t}\,\bar{s}$, $\bar{h}' = \bar{r}\,\bar{t}'\,\bar{s}$, and $\bar{a}$ is $\bar{r}$-initially-congruent *and* $\bar{s}$-finally-congruent with $\bar{a}'$ (or vice versa). This definition generalizes in the obvious way for more than two tours: three or more tours are $(\bar{r}, \bar{s})$-congruent if they are pairwise $(\bar{r}, \bar{s})$-congruent.

An illustration of m $(\bar{r}, \bar{s})$-congruent tours $\{\bar{r}\,\bar{t}_i\,\bar{s}\}_{1 \leq i \leq m}$ can be found in Figure 8.2. We are now able to give the correct definition of a violation for the path-functional dependency $\wp$, in terms of the elements c, d, and e: $\wp$ is violated in $\mathfrak{A}$ if there are three distinct elements c, d, and e (in A) such that, there exist $(f_1 f_1^{-1} \bar{f}_1^{-1}, \bar{f}_1)$-congruent $f_1 f_1^{-1} \bar{f}_1^{-1} \bar{f}_j \bar{f}_j^{-1} \bar{f}_1$-tours $(2 \leq j \leq n)$ whose first three elements are c, d, and e. (It helps in general to keep in mind Figure 8.1.)

This line of research should identify how to detect such violations. Query answering for n-ary path-functional dependencies is essentially the same as in Chapter 7.

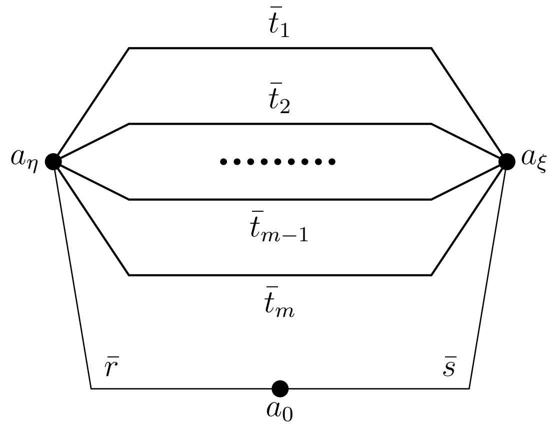


Figure 8.2: Some $(\bar{r}, \bar{s})$-congruent tours starting (and ending) at a_0; a_η is their initial fork and a_ξ their final join.

Another type of path-functional dependencies

Another possible direction of research is to allow expressions of the form

$$\Delta, \psi : \mathrm{PFD}[\bar{f}_1, \ldots, \bar{f}_n; \bar{g}_1, \ldots, \bar{g}_m].$$

We say that Δ, ψ is *satisfiable under* the dependency $\mathrm{PFD}[\bar{f}_1, \ldots, \bar{f}_n; \bar{g}_1, \ldots, \bar{g}_m]$ if there exists a structure $\mathfrak{A}$ such that $\mathfrak{A} \models \Delta, \psi$ and, for all $a, b \in A$,

$$\bigwedge_{1 \le i \le n} \bar{f}_i^{\mathfrak{A}}(a) \bowtie \bar{f}_i^{\mathfrak{A}}(b) \implies \bigwedge_{1 \le i \le m} \bar{g}_i^{\mathfrak{A}}(a) \bowtie \bar{g}_i^{\mathfrak{A}}(b);$$

then, we write $\mathfrak{A} \models \Delta, \psi : \mathrm{PFD}[\bar{f}_1, \ldots, \bar{f}_n]$. We speak of *finite* satisfiability when in the above definition we impose the extra restriction that $\mathfrak{A}$ be finite. In this case, we are dealing with configurations like the one illustrated in Figure 8.3. (We may assume that $n, m > 1$, otherwise the problem becomes trivial.)

In contrast to the previous subsection, here we want to ensure that *if* for each i $(1 \le i \le n)$ there exist two converging $\bar{f}_i$-paths starting at a and b respectively *then* for each j $(1 \le j \le m)$ there exist two converging $\bar{g}_j$-paths starting at a and b respectively. In terms of the (distinct) elements c, d, and e, we want to ensure that *if* there exist $(f_1 f_1^{-1} \bar{f}_1^{-1}, \bar{f}_1)$-congruent $f_1 f_1^{-1} \bar{f}_1^{-1} \bar{f}_i \bar{f}_i^{-1} \bar{f}_1$-tours $(2 \le i \le n)$ whose first three elements are c, d, and e, *then* there exist $(f_1 f_1^{-1} \bar{f}_1^{-1}, \bar{f}_1)$-congruent $f_1 f_1^{-1} \bar{f}_1^{-1} \bar{g}_j \bar{g}_j^{-1} \bar{f}_1$-tours $(1 \le j \le m)$ whose first three elements are c, d, and e. If one has a solution to the problem described in the previous subsection, it should be straigtforward do generalize it for the present problem. For, any such solution will provide a method to detect multiple congruent tours of a given family; then, this method can be used to detect whether the existence of a given family of (congruent) tours implies the existence of another family of (congruent) tours. Query answering, again, is essentially the same as in Chapter 7.

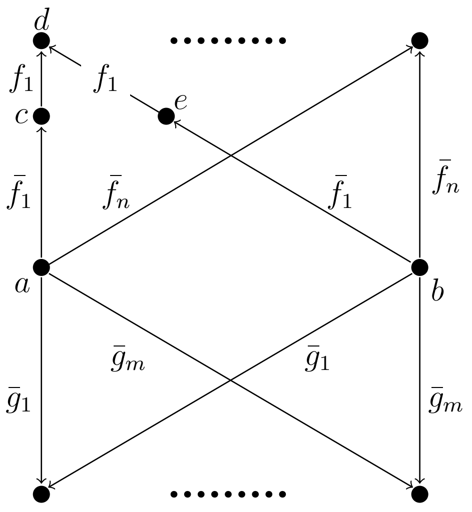


Figure 8.3: Enforcing PFD$[\bar{f}_1 f_1, \ldots, \bar{f}_n; \bar{g}_1, \ldots, \bar{g}_m]$.

Implementation and possible challenges

Almost all the procedures that we have provided in previous chapters use brute force. In particular, brute force is used when completing a given database and to search through all possible decompositions of the $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-sequence for each given path-functional dependency PFD$[\bar{f}f, \bar{g}]$. Since the search space for these operations is exponential, this approach will certainly face many difficulties in practice. Thus, a possible direction of research would be to identify the parts (if any) of the search space for the above operations that can be avoided and, in general, various other optimizations that can improve performance in practice.

For example, suppose that we want to decide if Δ, ψ : PFD$[\bar{f}f, \bar{g}]$ is (finitely) satisfiable. If ψ does not involve any predicates from the sequences $\bar{f}f, \bar{g}$ then no violations of the dependency need to involve elements outside Δ (since the dependency is a universal statement and cannot enforce the existence of new elements). Then, to check if the dependency is violated one only has to check for the existence of $ff^{-1}\bar{f}^{-1}\bar{g}\bar{g}^{-1}\bar{f}$-tours (with three initial distinct elements) in the database.

But, such a check does not require a search through all possible decompositions of the sequence $ff^{-1}\bar{f}^{-1}\bar{g}g^{-1}\bar{f}$, and can thus be completed in polynomial time with respect to the size of Δ and the length of this sequence.

Path-functional dependencies and $\mathcal{GF}$

The techniques in this thesis make essential use of counting—which leads to undecidability in conjunction with the guarded fragment—but, with care, they can be made to work in the two-variable setting—and thus regain decidability. A natural question arises then: how important is counting for the detection of path-functional dependecy violations? That is, do our results still hold if we do not use counting quantifiers and allow instead an arbitrary number of variables, i.e. work in the guarded fragment?

Other types of queries

Another possible research direction is to consider the complexity of query answering with respect to other types of queries. A popular extension of conjunctive queries are *regular path queries*. These allow the specification of queries using regular expressions and are thus more expressive than plain conjunctive queries. For example, one can express queries like 'return all the cities accessible from Manchester through any combination of bus or train services'. Regular path queries have been very popular in the graph database community. They have recently been studied in the context of some description logics [BOS15], so their study in the context of the guarded fragment—with or without path-functional dependencies—will be valuable.

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[ABU79] Alfred V Aho, Catriel Beeri, and Jeffrey D Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems (TODS)*, 4(3):297–314, 1979.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[AK89] Serge Abiteboul and Paris C Kanellakis. *Object identity as a query language primitive*, volume 18. ACM, 1989.

[ANvB98] Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[Arm74] William Ward Armstrong. Dependency structures of data base relationships. In *IFIP congress*, volume 74, pages 580–583. Geneva, Switzerland, 1974.

[ASU79] Alfred V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalences among relational expressions. *SIAM Journal on Computing*, 8(2):218–246, 1979.

[Baa03] F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BBH96] Franz Baader, Martin Buchheit, and Bernhard Hollander. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1):195–213, 1996.

[BDK92] François Bancilhon, Claude Delobel, and Paris Kanellakis. *Building an object-oriented database system: the story of 0 2.* Morgan Kaufmann Publishers Inc., 1992.

[BDRV01] Patrick Blackburn, Maarten De Rijke, and Yde Venema. Modal logic, volume 53 of cambridge tracts in theoretical computer science, 2001.

[BGG01] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem.* Springer, 2001.

[BGO10] Vince Barany, Georg Gottlob, and Martin Otto. Querying the guarded fragment. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 1–10. IEEE, 2010.

[BIS90] David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[Bor85] Alexander Borgida. Features of languages for the development of information systems at the conceptual level. *IEEE Software*, 2(1):63, 1985.

[BOS15] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res.(JAIR)*, 53:315–374, 2015.

[BtCS11] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, pages 356–367, 2011.

[BtCS15] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3):22, 2015.

[BV81] Catriel Beeri and Moshe Y Vardi. The implication problem for data dependencies. In *International Colloquium on Automata, Languages, and Programming*, pages 73–85. Springer, 1981.

[BV84] Catriel Beeri and Moshe Y Vardi. A proof procedure for data dependencies. *Journal of the ACM (JACM)*, 31(4):718–741, 1984.

[BvBW06] P. Blackburn, J.F.A.K. van Benthem, and F. Wolter. *Handbook of Modal Logic*. Studies in Logic and Practical Reasoning. Elsevier Science, 2006.

[Cal96] Diego Calvanese. Unrestricted and finite model reasoning in class-based representation formalisms. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma La Sapienza. In *In Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95), number 1013 in Lecture Notes in Computer Science*. Citeseer, 1996.

[CBB$^+$97] Roderic Geoffrey Galton Cattell, Douglas K Barry, Dirk Bartels, Mark Berler, Jeff Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland, and Drew Wade. *The object database standard: ODMG 2.0*, volume 131. Morgan Kaufmann Publishers San Mateo, 1997.

[CDGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 149–158. ACM, 1998.

[CDGL02] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In *Computational Logic: Logic Programming and Beyond*, pages 41–60. Springer, 2002.

[CFI92] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[CGK08] Andrea Calı, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Proc. of KR*, pages 70–80, 2008.

[Chu36a] Alonzo Church. A note on the entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.

[Chu36b] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[CLR03] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 260–271. ACM, 2003.

[CM77] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.

[Cod70] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cod72] Edgar F Codd. *Relational completeness of data base sublanguages*. IBM Corporation, 1972.

[Cod88] EF Codd. Extending the database relational model to capture more meaning. In *Readings in database systems*, pages 457–475. Morgan Kaufmann Publishers Inc., 1988.

[DGHP13] Marcello D'Agostino, Dov M Gabbay, Reiner Hähnle, and Joachim Posegga. *Handbook of tableau methods*. Springer Science & Business Media, 2013.

[DGL79] Burton Dreben, Warren D Goldfarb, and Harry R Lewis. *The decision problem: Solvable classes of quantificational formulas*. 1979.

[DGL96] Giuseppe De Giacomo and Maurizio Lenzerini. Tbox and abox reasoning in expressive description logics. *KR*, 96(316-327):10, 1996.

[DLW95] Anuj Dawar, Steven Lindell, and Scott Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, 119(2):160–175, 1995.

[DNR08] Alin Deutsch, Alan Nash, and Jeff Remmel. The chase revisited. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 149–158. ACM, 2008.

[dR95] Maarten de Rijke. Modal model theory. In *Annals of Pure and Applied Logic*. Citeseer, 1995.

[EH85] E Allen Emerson and Joseph Y Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of computer and system sciences*, 30(1):1–24, 1985.

[Ehr61] Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49(129-141):13, 1961.

[EJ88] E Allen Emerson and Charanjit S Jutla. The complexity of tree automata and logics of programs. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 328–337. IEEE, 1988.

[Fag77] Ronald Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems (TODS)*, 2(3):262–278, 1977.

[Fag82] Ronald Fagin. Horn clauses and database dependencies. *Journal of the ACM (JACM)*, 29(4):952–985, 1982.

[FKMP05] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[Fra50] Roland Fraïssé. Sur une nouvelle classification des systemes de relations. *COMPTES RENDUS HEBDOMADAIRES DES SEANCES DE L ACADEMIE DES SCIENCES*, 230(11):1022–1024, 1950.

[Fra84] Ronald Fraïsé. Sur quelques classification des systemes de relations. *Université d' Alger, Publications Scientifiques, Série A*, 1:35–182, 1984.

[Für81] Martin Fürer. Alternation and the ackermann case of the decision problem. *L' Enseignement Math*, 27:137–162, 1981.

[Für84] Martin Fürer. The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems). In *Logic and Machines: Decision problems and complexity*, pages 312–319. Springer, 1984.

[G$^+$09] W3C OWL Working Group et al. OWL 2 web ontology language document overview. 2009.

[GBIGKK13] Víctor Gutiérrez-Basulto, Yazmín Ibañez-García, Roman Kontchakov, and Egor V Kostylev. Conjunctive queries with negation over dl-lite: A closer look. In *International Conference on Web Reasoning and Rule Systems*, pages 109–122. Springer, 2013.

[GHLS08] Birte Glimm, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. Conjunctive query answering for the description logic. *Journal of Artificial Intelligence Research*, 31:157–204, 2008.

[Gia95] G. De Giacomo. *Decidability of class-based knowledge representation formalisms.* PhD thesis, Universitá di Roma "La Sapienza", 1995.

[GJ79] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[GK72] Yu Sh Gurevich and IO Koryakov. Remarks on berger's paper on the domino problem. *Siberian Mathematical Journal*, 13(2):319–321, 1972.

[GKV97] Erich Grädel, Phokion G Kolaitis, and Moshe Y Vardi. On the decision problem for two-variable first-order logic. *Bulletin of symbolic logic*, pages 53–69, 1997.

[Gol84] Warren D Goldfarb. The unsolvability of the gödel class with identity. *The Journal of Symbolic Logic*, 49(04):1237–1252, 1984.

[GOR97] Erich Gradel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Logic in Computer Science, 1997. LICS'97. Proceedings., 12th Annual IEEE Symposium on*, pages 306–317. IEEE, 1997.

[Grä90] Erich Grädel. Satisfiability of formulae with one ∀ is decidable in exponential time. *Archive for Mathematical Logic*, 29(4):265–276, 1990.

[Gra98] Erich Gradel. Guarded fragments of first-order logic: A perspective for new description logics? In *In Proc. of 1998 Int. Workshop on Description Logics DL98, Trento, CEUR Electronic Workshop Proceedings*, 1998.

[Grä99] Erich Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, pages 1719–1742, 1999.

[GW99] Erich Gradel and Igor Walukiewicz. Guarded fixed point logic. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 45–54. IEEE, 1999.

[Her95] Bernhard Herwig. Extending partial isomorphisms on finite structures. *combinatorica*, 15(3):365–371, 1995.

[Her10] André Hernich. *Foundations of query answering in relational data exchange.* Logos Verlag Berlin GmbH, 2010.

[Hru92] Ehud Hrushovski. Extending partial isomorphisms of graphs. *combinatorica*, 12(4):411–416, 1992.

[HT00] Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic aboxes. In *AAAI/IAAI*, pages 399–404, 2000.

[IK89] Neil Immerman and Dexter Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83(2):121–139, 1989.

[Imm82] Neil Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25(1):76–98, 1982.

[Imm89] Neil Immerman. Expressibility and parallel complexity. *SIAM Journal on Computing*, 18(3):625–638, 1989.

[Imm91] Neil Immerman. Dspace[n^k]= var[$k + 1$]. In *Structure in Complexity Theory Conference, 1991., Proceedings of the Sixth Annual*, pages 334–340. IEEE, 1991.

[IW94] M. Ito and G. Weddell. Implication problems for functional constraints on databases supporting complex objects. *Journal of Computer and System Sciences*, 49(3):726–768, 1994.

[JK82] David S Johnson and Anthony Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 164–169. ACM, 1982.

[Kaz04] Yevgeny Kazakov. A polynomial translation from the two-variable guarded fragment with number restrictions to the guarded fragment. In *Logics in Artificial Intelligence*, pages 372–384. Springer, 2004.

[Ken79] William Kent. Limitations of record-based information models. *ACM Transactions on Database Systems (TODS)*, 4(1):107–131, 1979.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843, 1995.

[KMT98] Phokion G Kolaitis, David L Martin, and Madhukar N Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 197–204. ACM, 1998.

[KMW62] Andrew S Kahr, Edward F Moore, and Hao Wang. Entscheidungsproblem reduced to the ∀∃∀ case. *Proceedings of the National Academy of Sciences*, 48(3):365–377, 1962.

[Kri59] Saul A Kripke. A completeness theorem in modal logic. *The journal of symbolic logic*, 24(1):1–14, 1959.

[Kri63] Saul A Kripke. Semantical analysis of modal logic I, normal modal propositional calculi. *Mathematical Logic Quarterly*, 9(5-6):67–96, 1963.

[Kri71] S.A. Kripke. *Semantical Considerations on Modal Logic; Naming and Necessity.* Oxford University Press, 1971.

[KTW01] Vitaliy L Khizder, David Toman, and Grant Weddell. On decidability and complexity of description logics with uniqueness constraints. In *International Conference on Database Theory*, pages 54–67. Springer, 2001.

[KV90a] Phokion G Kolaitis and Moshe Y Vardi. 0–1 laws and decision problems for fragments of second-order logic. *Information and Computation*, 87(1):302–338, 1990.

[KV90b] Phokion G Kolaitis and Moshe Y Vardi. On the expressive power of datalog: tools and a case study. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 61–71. ACM, 1990.

[Lad77] Richard E Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM journal on computing*, 6(3):467–480, 1977.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[Lew79] Harry R Lewis. *Unsolvable classes of quantificational formulas.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1979.

[Lew80] Harry R Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21(3):317–353, 1980.

[Lib13] Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

[LST05] Carsten Lutz, Ulrike Sattler, and Lidia Tendera. The complexity of finite model reasoning in description logics. *Information and Computation*, 199(1):132–171, 2005.

[Lut08] Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. In *Automated Reasoning*, pages 179–193. Springer, 2008.

[Mar06] David Marker. *Model theory: an introduction*, volume 217. Springer Science & Business Media, 2006.

[MBW80] John Mylopoulos, Philip A Bernstein, and Harry KT Wong. A language facility for designing database-intensive applications. *ACM Transactions on Database Systems (TODS)*, 5(2):185–207, 1980.

[MHF03] Todd Millstein, Alon Halevy, and Marc Friedman. Query containment for data integration systems. *Journal of Computer and System Sciences*, 66(1):20–39, 2003.

[MMS79] David Maier, Alberto O Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems (TODS)*, 4(4):455–469, 1979.

[Mor75] Michael Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.

[MSY81] David Maier, Yehoshua Sagiv, and Mihalis Yannakakis. On the complexity of testing implications of functional and join dependencies. *Journal of the ACM (JACM)*, 28(4):680–695, 1981.

[Pap03] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[PH05] Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.

[PH07] Ian Pratt-Hartmann. Complexity of the guarded two-variable fragment with counting quantifiers. *J. Log. Comput.*, 17(1):133–155, 2007.

[PH09] Ian Pratt-Hartmann. Data-complexity of the two-variable fragment with counting quantifiers. *Information and Computation*, 207(8):867–888, 2009.

[PST97] Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *Logic in Computer Science, 1997. LICS'97. Proceedings., 12th Annual IEEE Symposium on*, pages 318–327. IEEE, 1997.

[Ris78] Jorma Rissanen. Theory of relations for databasesâĂŤa tutorial survey. In *International Symposium on Mathematical Foundations of Computer Science*, pages 536–551. Springer, 1978.

[Ros06] Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 356–365. ACM, 2006.

[Ros11] Riccardo Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *Journal of Computer and System Sciences*, 77(3):572–594, 2011.

[Sav70] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

[Sch91] Klaus Schild. *A correspondence theory for terminological logics: Preliminary report.* Techn. Univ., 1991.

[Sco62] D. Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.

[SFL83] John Miles Smith, Stephen A Fox, and Terry A Landers. *Adaplex: rationale and reference manual.* Computer Corporation of America, 1983.

[SS75] Hans Albrecht Schmid and J Richard Swenson. On the semantics of the relational data model. In *Proceedings of the 1975 ACM SIGMOD international conference on Management of data*, pages 211–223. ACM, 1975.

[SS80] John Miles Smith and Diane CP Smith. A data base approach to software specification. In *Software Development Tools*, pages 176–204. Springer, 1980.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial intelligence*, 48(1):1–26, 1991.

[StC13] Luc Segoufin and Balder ten Cate. Unary negation. *Logical Methods in Computer Science*, 9(3), 2013.

[Sto74] Larry Joseph Stockmeyer. The complexity of decision problems in automata theory and logic. 1974.

[SY80] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM (JACM)*, 27(4):633–655, 1980.

[tCS11] Balder ten Cate and Luc Segoufin. Unary negation. In *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany*, pages 344–355, 2011.

[Tur36] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 42(2):230–265, 1936.

[TW04] David Toman and Grant E Weddell. Attribute inversion in description logic with path functional dependencies. *Description Logics*, 104:178–187, 2004.

[TW05] David Toman and Grant Weddell. On the interaction between inverse features and path-functional dependencies in description logics. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 603–608, 2005.

[TW08] David Toman and Grant Weddell. On keys and functional dependencies as first-class citizens in description logics. *Journal of Automated Reasoning*, 40(2-3):117–132, 2008.

[Var82] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

[Var95] Moshe Y Vardi. On the complexity of bounded-variable queries. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 266–276. ACM, 1995.

[Var96] Moshe Y Vardi. Why is modal logic so robustly decidable? *Descriptive complexity and finite models*, 31:149–184, 1996.

[VB91] Johan Van Benthem. Language in action. *Journal of philosophical logic*, 20(3):225–263, 1991.

[vBW94] Martin F van Bommel and Grant E. Weddell. Reasoning about equations and functional dependencies on complex objects. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):455–469, 1994.

[VDBVGAG97] Jan Van Den Bussche, Dirk Van Gucht, Marc Andries, and Marc Gyssens. On the completeness of object-creating database transformation languages. *Journal of the ACM (JACM)*, 44(2):272–319, 1997.

[vdM97] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *Journal of Computer and System Sciences*, 54(1):113–135, 1997.

[Vol13] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach.* Springer Science & Business Media, 2013.

[Wan60] Hao Wang. Proving theorems by pattern recognition i. *Communications of the ACM*, 3(4):220–234, 1960.

[Wan65] Hao Wang. Games, logic and computers. In *Computation, Logic, Philosophy*, pages 195–217. Springer, 1965.

[Wed89] G. Weddell. A theory of functional dependencies for object oriented data models. In *Proc. International Conference on Deductive and Object-Oriented Databases*, pages 165–184, 1989.

[Wed92] Grant E Weddell. Reasoning about functional dependencies generalized for semantic data models. *ACM Transactions on Database Systems (TODS)*, 17(1):32–64, 1992.

[Zan76] Carlo Antonio Zaniolo. Analysis and design of relational schemata for database systems. 1976.

[Zan83] Carlo Zaniolo. The database language gem. *ACM Sigmod Record*, 13(4):207–218, 1983.