# 2. Constraint Databases, Queries, and Query Languages

Jan Van den Bussche

## 2.1 Introduction

We formally define the constraint database model, the concept of query in this model, and the basic constraint query languages provided by the relational calculus, the relational algebra, and DATALOG. We show how a computationally complete constraint query language can be obtained by augmenting the constraint relational calculus with basic programming language features. We look into some basic model-theoretic issues concerning the constraint relational calculus, in particular the equivalence problem. The notion of o-minimal structure turns out to be a useful abstraction to discuss these issues in some generality. We will see that equivalence of relational calculus queries on constraint databases is typically undecidable, but that it is decidable in the special case of conjunctive queries on constraint databases.

## 2.2 Logic

We start by recalling the needed basic concepts from logic.

**Definition 2.2.1 (Vocabulary).** *A vocabulary, or signature, $\Omega$, consists of three sets: a set $\mathcal{F}$ of function symbols, a set $\mathcal{P}$ of predicate symbols, and a set $\mathcal{C}$ of constant symbols, together with an arity function that associates a natural number to each element of $\mathcal{F}$ and $\mathcal{P}$.*

For example, $\Omega = (+, \cdot, 0, 1, <)$ is a vocabulary with two function symbols of arity two ($+$ and $\cdot$), one predicate symbol of arity two $<$, and two constant symbols (0 and 1).

**Definition 2.2.2 (Structure).** *Given a set $\mathcal{U}$ and a signature $\Omega$, an $\Omega$-structure $\mathcal{M}$ on $\mathcal{U}$ is defined by assigning to each $f \in \mathcal{F}$ of arity $n$ a function $f^{\mathcal{M}} : \mathcal{U}^n \to \mathcal{U}$, to each $p \in \mathcal{P}$ of arity $n$ an $n$-ary predicate on $\mathcal{U}$, that is, a set $P^{\mathcal{M}} \subseteq \mathcal{U}^n$, and to each $c \in \mathcal{C}$ an element $c^{\mathcal{M}} \in \mathcal{U}$.*

For example, for $\Omega = (+, \cdot, 0, 1, <)$ and the set $\mathbb{R}$ of real numbers, we define the real field $\mathbf{R}$ by making $+^{\mathbf{R}}$, $\cdot^{\mathbf{R}}$, $<^{\mathbf{R}}$ the usual addition, multiplication, and the ordering on $\mathbb{R}$, and by interpreting $0^{\mathbf{R}}$ and $1^{\mathbf{R}}$ as $0, 1 \in \mathbb{R}$. Note that we shall almost always omit the superscript, since the interpretation of the signature symbols is typically understood from the context.

We now recall, for the sake of completeness, the definition of first-order logic over a signature $\Omega$. Assume a countably infinite set of variables $\mathcal{V}$. First, *terms* are defined inductively as either a variable from $\mathcal{V}$, or a constant from $\mathcal{C}$, or $f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}$ is of arity $n$, and $t_1, \ldots, t_n$ are terms. We write $t(x_1, \ldots, x_k)$ to denote that $t$ is a term that mentions only variables from the list $x_1, \ldots, x_k$ (listed in some agreed order).

Given a structure $\mathcal{M} = \langle \mathcal{U}, \Omega \rangle$, a term $t(x_1, \ldots, x_k)$, and elements $a_1, \ldots, a_k \in \mathcal{U}$, the interpretation of $t$ given $a_1, \ldots, a_k$ in $\mathcal{M}$, denoted by $t^{\mathcal{M}}[a_1, \ldots, a_k]$, or $t^{\mathcal{M}}[\bar{a}]$ for short, is defined in the obvious way:

- If $t$ is a variable $x_i$, then $t^{\mathcal{M}}[\bar{a}]$ is $a_i$;
- If $t$ is a constant symbol $c$, then $t^{\mathcal{M}}[\bar{a}]$ is $c^{\mathcal{M}}$;
- If $t$ is of the form $f(t_1, \ldots, t_n)$, then $t^{\mathcal{M}}[\bar{a}]$ is $f^{\mathcal{M}}(t_1^{\mathcal{M}}[\bar{a}], \ldots, t_n^{\mathcal{M}}[\bar{a}])$.

An *atomic* formula is a formula of the form $t = t'$ or $P(t_1, \ldots, t_n)$ where $t$, $t'$, $t_1, \ldots, t_n$ are terms, and $P \in \mathcal{P}$ is an $n$-ary predicate. General formulas are then built up from atomic formulas by using the boolean connectives ($\wedge$, $\vee$, $\neg$) and the quantifiers ($\forall x$ and $\exists x$). The variables occurring *free* in a formula are defined in the usual manner. Formally, the free variables of:

- an atomic formula, are simple all the variables occurring in it;
- a negative formula $\neg\varphi$, are exactly those of $\varphi$;
- a conjunctive formula $(\varphi \wedge \psi)$, or a disjunctive formula $(\varphi \vee \psi)$, are exactly those that are free in either $\varphi$ or $\psi$;
- a quantified formula $\forall x\, \varphi$ or $\exists x\, \varphi$, are exactly those of $\varphi$, with the exception of $x$ itself.

A *sentence* is a formula without free variables. We write $\varphi(x_1, \ldots, x_k)$ to denote that the free variables of formula $\varphi$ are all from the list $x_1, \ldots, x_k$ (listed in some agreed order).

Given a structure $\mathcal{M} = \langle \mathcal{U}, \Omega \rangle$, a first-order formula $\varphi(x_1, \ldots, x_k)$, and elements $a_1, \ldots, a_k \in \mathcal{U}$, the satisfaction of $\varphi$ by $(a_1, \ldots, a_k)$ in $\mathcal{M}$, denoted by $\mathcal{M} \models \varphi[a_1, \ldots, a_k]$, or $\mathcal{M} \models \varphi[\bar{a}]$ for short, is defined in the usual way:

- $\mathcal{M} \models (t = t')[\bar{a}]$ if $t^{\mathcal{M}}[\bar{a}]$ equals $t'^{\mathcal{M}}[\bar{a}]$.
- $\mathcal{M} \models P(t_1, \ldots, t_n)[\bar{a}]$ if $(t_1^{\mathcal{M}}[\bar{a}], \ldots, t_n^{\mathcal{M}}[\bar{a}]) \in P^{\mathcal{M}}$.
- $\mathcal{M} \models (\neg\varphi)[\bar{a}]$ if $\mathcal{M} \models \varphi[\bar{a}]$ does not hold.
- $\mathcal{M} \models (\varphi \wedge \psi)[\bar{a}]$ if both $\mathcal{M} \models \varphi[\bar{a}]$ and $\mathcal{M} \models \psi[\bar{a}]$.
- $\mathcal{M} \models (\varphi \vee \psi)[\bar{a}]$ if either $\mathcal{M} \models \varphi[\bar{a}]$ or $\mathcal{M} \models \psi[\bar{a}]$.
- $\mathcal{M} \models (\forall x\, \varphi)[\bar{a}]$ if for every element $b \in \mathcal{U}$, we have $\mathcal{M} \models \varphi[\bar{a}, b]$, i.e., formula $\varphi(x_1, \ldots, x_k, x)$ is satisfied by $(a_1, \ldots, a_k, b)$ in $\mathcal{M}$.
- $\mathcal{M} \models (\exists x\, \varphi)[\bar{a}]$ if for some element $b \in \mathcal{U}$, we have $\mathcal{M} \models \varphi[\bar{a}, b]$.

Note that in this definition we are being very formal, using square brackets $\varphi[\bar{a}]$ to denote interpretations for free variables. However, in the sequel, we will also use a more intuitive notation with round brackets $\varphi(\bar{a})$.

The set of all first-order formulas over $\Omega$ is denoted by $\mathrm{FO}(\Omega)$. If $\mathcal{M}$ is an $\Omega$-structure, we also write $\mathrm{FO}(\mathcal{M})$ for $\mathrm{FO}(\Omega)$.

### 2.2.1 Quantifier elimination

We now introduce a very important concept in the context of constraint databases:

**Definition 2.2.3.** *Structure $\mathcal{M}$ is said to* admit quantifier elimination *if for every first-order formula $\varphi(\bar{x})$ over $\Omega$ there exists a quantifier-free formula $\psi(\bar{x})$ such that*

$$\forall \bar{x}(\varphi \leftrightarrow \psi)$$

*is valid in $\mathcal{M}$. If such a $\psi$ can be effectively computed, given $\varphi$, we say that the quantifier elimination is* effective.

As we will see in this chapter, it is exactly in the context of structures admitting effective quantifier elimination that we will be able, at least in principle, to implement constraint database systems with first-order logic query languages. Popular structures with this property include the real field $\langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$ and its restrictions that exclude multiplication, or include only order (for these restrictions it does not matter whether we use real or rational numbers).

## 2.3 The Constraint Database Model

The framework of constraint databases with corresponding query languages can be applied to many different classes of constraints. We first need to define constraints formally. Since the generalized relations we met in Chapter 1 are subsets – finite or infinite – of some interpreted domain, such as the real field, constraints are defined to be certain first-order formulas over the domain. For example, in the case of the real field, polynomial inequality constraints are just atomic first-order formulas over the structure $\langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$.

### 2.3.1 Constraints

Here is the formal definition of "constraint" as it is used in the context of constraint databases:

**Definition 2.3.1 (Constraints).** *Let $\Omega$ be a vocabulary. A* constraint *over $\Omega$ is an atomic first-order formula over $\Omega$, or the negation of an atomic formula.*

The representational power of constraints is captured by the following definition.

**Definition 2.3.2 (Definability with Constraints).** *Given a vocabulary $\Omega$ and a structure $\mathcal{M} = \langle \mathcal{U}, \Omega \rangle$, a set $X \subseteq \mathcal{U}^n$ is called* definable on $\mathcal{M}$ with $\Omega$-constraints *if it can be obtained as a finite boolean combination of sets of the form*

$$\{(a_1, \ldots, a_n) \in \mathcal{U}^n \mid \mathcal{M} \models \varphi(a_1, \ldots, a_n)\}$$

*where $\varphi$ is an $\Omega$-constraint. If $\mathcal{M}$ is clear from the context, we will simply say $X$ is* definable.

*Given $\mathcal{M} = \langle \mathcal{U}, \Omega \rangle$ and $\mathcal{M}' = \langle \mathcal{U}, \Omega' \rangle$, we say that $\Omega$- and $\Omega'$-constraints are equivalent over $\mathcal{U}$ if exactly the same sets are definable with $\Omega$ and $\Omega'$-constraints in $\mathcal{M}$ and $\mathcal{M}'$.*

*Example 2.3.1.* The following classes of constraints will often be used in this book.

1. *Polynomial inequality constraints* are constraints over $\Omega = (+, \cdot, 0, 1, <)$. Such constraints correspond to conditions of the form $p > 0$ or $p \leq 0$, where $p$ is a polynomial with integer coefficients. We interpret these constraints over the structures $\mathbf{R} = \langle \mathbb{R}, \Omega \rangle$, $\mathbf{Q} = \langle \mathbb{Q}, \Omega \rangle$, $\mathbf{Z} = \langle \mathbb{Z}, \Omega \rangle$, and the like.
2. *Linear constraints* are constraints over $\Omega = (+, <, 0, 1)$. Such constraints correspond to conditions of the form $p > 0$ or $p \leq 0$, where $p$ is a linear function $a_1 x_1 + \cdots + a_j x_j + a_0$ with integer coefficients. We interpret these constraints over the structures $\mathbf{R}_{\text{lin}} = \langle \mathbb{R}, \Omega \rangle$, $\mathbf{Q}_{\text{lin}} = \langle \mathbb{Q}, \Omega \rangle$, $\mathbf{Z}_{\text{lin}} = \langle \mathbb{Z}, \Omega \rangle$, and the like.
3. *Dense order constraints over the rationals* are constraints over $\Omega = (<, (c)_{c \in \mathbb{Q}})$, interpreted over the structure $\langle \mathbb{Q}, \Omega \rangle$ (that is, rational numbers with order and constants for every $c \in \mathbb{Q}$). Such constraints are conditions of the form $x < y$, $x \geq y$, $x < c$, or $x \geq c$, where $x$ and $y$ are variables, and $c$ is a constant. Similarly, one can define dense order constraints over the reals, or, for that matter, over any set on which a dense order is available.
4. *Equality constraints over an arbitrary infinite domain $\mathcal{U}$* are constraints over the signature $((c)_{c \in \mathcal{U}})$. Such constraints are conditions of the form $x = y$, $x \neq y$, $x = c$, or $x \neq c$, where $x$ and $y$ are variables, and $c$ is a constant.

One may ask why we did not consider polynomial or linear constraints with *rational* coefficients. The reason is that they are not more expressive than those with integer coefficients. Indeed, let $p$ be a polynomial in which all coefficients are rational numbers $a_1/b_1, \ldots, a_m/b_m$, where the $a_i$s and $b_i$s are integers. Let $p' = \prod_{i=1}^{m} b_i \cdot p$. Then $p'$ is as a polynomial with integer coefficients, and $p > 0$ iff $p' > 0$. In Chapter 9, however, linear constraints with *algebraic* coefficients will also be discussed, and these are indeed more expressive than the standard linear constraints defined above.

### 2.3.2 Constraint databases

We are now ready to define what a constraint database exactly is, and which information it represents.

**Definition 2.3.3 (Constraint Database Model).** *Fix a vocabulary $\Omega$.*

1. *A* constraint $k$-tuple*, in variables $x_1, \ldots, x_k$, over $\Omega$, is a finite conjunction $\varphi_1 \wedge \cdots \wedge \varphi_N$, where each $\varphi_i$, for $1 \leq i \leq N$, is an $\Omega$-constraint. Furthermore, the variables in each $\varphi_i$ are all among $x_1, \ldots, x_k$.*
2. *A* constraint relation of arity $k$*, over $\Omega$, is a finite set $r = \{\psi_1, \ldots, \psi_M\}$, where each $\psi_i$, for $1 \leq i \leq M$, is a constraint $k$-tuple in the same variables $x_1, \ldots, x_k$.*
3. *The* formula corresponding to *a constraint relation $r$ is the disjunction $\psi_1 \vee \cdots \vee \psi_M$. We denote this formula by $\varphi_r$; note that it is quantifier-free.*
4. *A* constraint database *is a finite collection of constraint relations.*

In database theory, a $k$-ary relation $r$ is assumed to be a *finite* set of $k$-tuples (or points in a $k$-dimensional space). We will use the term *unrestricted relation* for arbitrary finite or infinite sets of points in a $k$-dimensional space. It is possible to develop query languages using such unrestricted relations. In order to be able to do something useful with them, however, we need a finite representation that we can manipulate. This is exactly what the constraint tuples provide.

**Definition 2.3.4 (Interpretation of Constraint Relations).** *Let $\Omega$ be a vocabulary, and $\mathcal{M} = \langle \mathcal{U}, \Omega \rangle$ an $\Omega$-structure. Let $r$ be a constraint relation of arity $k$ over $\Omega$, and let $\varphi_r(x_1, \ldots, x_k)$ be the formula corresponding to $r$. Then $r$ represents the unrestricted $k$-ary relation which consists of all points $(a_1, \ldots, a_k)$ such that $\varphi_r(a_1, \ldots, a_k)$ is true in $\mathcal{M}$. That is, it represents the set*

$$[\![r]\!]_{\mathcal{M}} n \quad = \quad \{(a_1, \ldots, a_k) \in \mathcal{U}^k \mid \mathcal{M} \models \varphi_r(a_1, \ldots, a_k)\} .$$

Observe that interpretations of constraint relations over $\mathcal{M}$ are precisely the sets we called *definable on $\mathcal{M}$ with $\Omega$-constraints* (cf. Definition 2.3.2). Indeed, constraint relations are just boolean combinations of constraints, presented in disjunctive normal form. It will be convenient to carry this terminology further as follows:

**Definition 2.3.5 (Definable Databases).** *Any finite collection of unrestricted relations is called an* unrestricted database*. An unrestricted database is called* definable *if all its relations are definable in the sense of Definition 2.3.2. A constraint database $D$* represents *a definable database $D'$ if $D$ consists precisely of one representation for each unrestricted relation in $D'$.*

We will now see three fundamental examples of constraint relations.

*Example 2.3.2 (Representing the Relational Model).* Let a relation $r$ consist of the tuples $(1, 2)$ and $(3, 4)$. These tuples are equivalent to the constraint 2-tuples $x = 1 \wedge y = 2$ and $x = 3 \wedge y = 4$. Therefore, $r$ corresponds to the set $\{x = 1 \wedge y = 2, x = 3 \wedge y = 4\}$ and to the formula $\varphi_r \equiv (x = 1 \wedge y = 2) \vee (x = 3 \wedge y = 4)$.

In general, every finite $k$-ary relation $r$ on a set $\mathcal{U}$ with $m$ tuples $(a_1^i, \ldots, a_k^i)$, $i = 1, \ldots, m$, can be represented as a constraint relation using just equality constraints. That is, if $r'$ is the constraint relation given by the formula

$$\varphi_{r'}(x_1, \ldots, x_k) \quad = \quad \bigvee_{i=1}^{m} ((x_1 = a_1^i) \wedge \cdots \wedge (x_k = a_k^i)) \; ,$$

then

$$[\![r']\!]_{\mathcal{M}} n \quad = \quad r \; ,$$

where $\mathcal{M} = \langle \mathcal{U}, (c)_{c \in \mathcal{U}} \rangle$.

*Example 2.3.3.* Let us now illustrate the framework using linear constraints. Let the constraint relation $r$ consist of the two constraint tuples

$$(y = 2 \cdot x \wedge \neg(x = y)) \quad \text{and} \quad (1 < x + y) \; .$$

Corresponding to this $r$ is the DNF formula

$$\varphi_r(x, y) = (y = 2 \cdot x \wedge \neg(x = y)) \vee (1 < x + y) \; .$$

This formula $\varphi_r$ describes an *infinite* set of points in 2-dimensional space: namely the half plane $x + y > 1$ and the line $y = 2 \cdot x$ without the point $x = y = 0$.

*Example 2.3.4.* Similarly, the constraint relation consisting of the two polynomial inequality constraint tuples

$$(x^2 + y^2 = 1) \quad \text{and} \quad ((x - 1)^2 + y^2 = 1 \wedge 2x \geq 1)$$

describes a circle partly overlapping another circle, as illustrated in Figure 2.3.1. Of course, $x^2$ is an abbreviation for $x \cdot x$.

In the case of polynomial inequality or linear constraints over the reals, constraint relations correspond to concepts well known in real algebraic geometry.

**Definition 2.3.6 (Real Semi-Algebraic and Semi-Linear Sets).** *A set $X \subseteq \mathbb{R}^n$ is called* semi-algebraic *if it is definable with polynomial inequality constraints over the real field $\mathbf{R}$. A set $X \subseteq \mathbb{R}^n$ is called* semi-linear *if it is definable with linear constraints over $\mathbf{R}_{\text{lin}}$.*

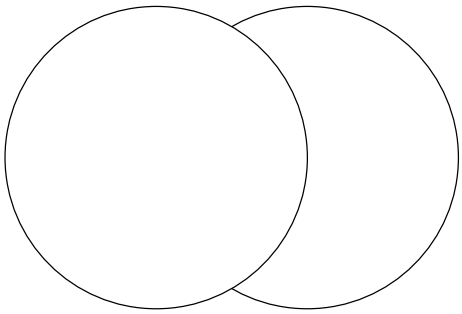


**Fig. 2.3.1.** Figure in the real plane represented by the real polynomial inequality constraints relation of Example 2.3.4.

Thus, the constraint relations definable with polynomial inequality constraints over the reals are precisely the semi-algebraic sets, and the constraint relations definable with linear constraints over the reals are precisely the semi-linear sets. Or, put in yet another way: semi-algebraic sets are Boolean combinations of sets defined by conditions of the form $p(x_1, \ldots, x_n) > 0$, with $p$ a polynomial in the variables $x_1$, ..., $x_n$ with integer coefficients; semi-linear sets are Boolean combinations of sets defined by conditions of the form $a_1x_1 + \cdots + a_nx_n > b$, where $a_1$, ..., $a_k$, and $b$ are integers.

### 2.3.3 Testing equality of constraint relations

The representation of a definable relation by a constraint relation is, of course, not unique. To give a simple example using equality constraints, the singleton $\{(c, c)\}$ is represented by the constraint tuple $x = c \wedge y = x$ as well as by the constraint tuple $x = c \wedge y = c$.

In general, two constraint relations $r$ and $r'$, interpreted over a structure $\mathcal{M}$, represent the same unrestricted relation iff the formula $\forall \bar{x}(\varphi_r(\bar{x}) \leftrightarrow \varphi_{r'}(\bar{x}))$ is true of $\mathcal{M}$. Hence:

**Proposition 2.3.7.** *Equivalence of constraint relations is effectively decidable if (and only if) the universal theory of $\mathcal{M}$ is effectively decidable.*

By the *universal theory* of $\mathcal{M}$, we mean the set of all first-order sentences in prenex normal form that are true of $\mathcal{M}$, whose quantifier prefix consists of universal quantifiers only.

All structures mentioned in Example 2.3.1 have a decidable universal theory. As a matter of fact, for these structures, the *complete theory*, i.e., the set of *all* first-order sentences true in the structure, is effectively decidable. This follows from the following two properties of these structures:

1. they admit effective quantifier elimination (cf. Definition 2.2.3), so that, in particular, every sentence can be converted to a boolean combination of atomic sentences;

2. the truth of every atomic sentence in the structure is effectively decidable.

The second property (about atomic sentences) is quite obvious, in the sense that structures without this property would not be practical to work with. The first property (about effective quantifier elimination) is by no means obvious. Still, for all structures mentioned in Example 2.3.1, it is satisfied. We will see in the next section that effective quantifier elimination is crucial for the whole constraint database approach to go through.

*Example 2.3.5.* For the real field $\mathbf{R}$, the effective decidability of atomic sentences merely says that one can effectively add, multiply, and compare natural numbers. Indeed, the only constant symbols in $\mathbf{R}$ are 0 and 1, so the only thing an atomic sentence over $\mathbf{R}$ does is compare two natural numbers built from 0 and 1 using addition and multiplication.

That effective quantifier elimination is possible in the real field is essentially Tarski's famous decision method for the theory of the reals. A well known illustration of quantifier elimination over the reals is provided by high-school mathematics: the formula

$$\exists x(a \cdot x^2 + b \cdot x + c = 0)$$

is equivalent to the quantifier-free formula

$$b^2 - 4 \cdot a \cdot c \geq 0 \; .$$

## 2.4 Queries on Constraint Databases

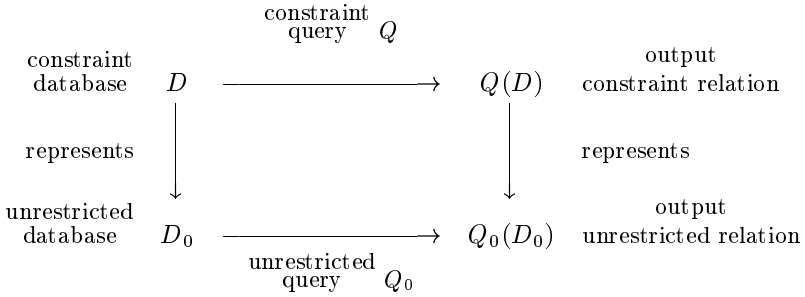
In this section, we combine constraints with the fundamental notion of database query. We formally define queries to constraint databases, and introduce the relational calculus (augmented with constraints) as a basic constraint query language. We also discuss the equivalence with the relational algebra.

### 2.4.1 Constraint queries

So far we have defined a constraint database simply as a finite collection of constraint relations. When querying a database, however, it is useful to give these relations a name. So we define a *database schema* as a finite nonempty set $SC$ of relation names, each with a given arity. A *constraint database with schema $SC$* then becomes a mapping, associating to each relation name $R \in SC$ a constraint relation of the arity given for $R$. An *unrestricted database with schema $SC$* is defined analogously.

We are now ready to discuss the formal definition of a query on constraint databases. In standard relational databases, a query is a mapping, associating to each database an answer relation. In the present constraint setting, however, there is a complication. Conceptually, constraint databases

are representations of unrestricted databases (cf. Definition 2.3.5). So we can think of a query in two ways: on the conceptual level, as a mapping on unrestricted (but definable) databases; or on the level of the representations, as a mapping on constraint databases. Clearly, a query on the representation level makes only sense if it corresponds to a query on the conceptual level, as in the following commuting diagram:




The above considerations lead to the following definitions. We start with the notion of query on the conceptual level:

**Definition 2.4.1 (Unrestricted Query).** *Let $\mathcal{M}$ be a structure over the vocabulary $\Omega$. Let $SC$ be a schema, and let $k$ be a natural number. A* $k$-ary unrestricted query with schema $SC$ over $\mathcal{M}$ *is a function $Q$, mapping unrestricted databases $D$ with schema $SC$ to $k$-ary relations $Q(D)$ on the universe $U$ of $\mathcal{M}$. This mapping $Q$ can be partial (i.e., $Q(D)$ may be undefined for some $D$'s).*

Since we are only interested in unrestricted relations that are definable, we want the following property:

**Definition 2.4.2 (Closure).** *An unrestricted query $Q$ as in Definition 2.4.1 is called* closed *if, for each $D$ on which $Q$ is defined, and which is definable using $\mathcal{M}$-constraints, the relation $Q(D)$ is also definable using $\mathcal{M}$-constraints.*

We next turn to queries on the representation level:

**Definition 2.4.3 (Constraint Query).** *Let $\Omega$ be a vocabulary. Let $SC$ be a schema, and let $k$ be a natural number. A* $k$-ary constraint query with schema $SC$ over $\Omega$ *is a function $Q$, mapping constraint databases $D$ with schema $SC$ over $\Omega$, to $k$-ary constraint relations $Q(D)$ over $\Omega$. This mapping $Q$ can be partial.*

We do not want a constraint query to map two different constraint representations of a same unrestricted database to constraint relations representing different unrestricted relations. So we want at least the following property:

**Definition 2.4.4 (Consistency).** *A constraint query $Q$ as defined in Definition 2.4.3 is called* consistent *if there exists an unrestricted query $Q_0$ such*

*that for any constraint database $D$ and any unrestricted database $D_0$, if $D$ represents $D_0$, then $Q(D)$ is defined if and only if $Q_0(D_0)$ is defined, and in that case, $Q(D)$ represents $Q_0(D_0)$. We say in this case that $Q$* represents *$Q_0$.*

Note that the unrestricted query represented by a consistent constraint query is uniquely defined, and that it automatically satisfies the closure condition of Definition 2.4.2.

*Example 2.4.1.* Let us illustrate the above definitions in the concrete context of polynomial inequality constraints over the real field $\mathbf{R}$. Fix $SC$ to consist of a single relation name $T$ of arity 3. So a definable database consists of a single semi-algebraic set in $\mathbb{R}^3$.

Define the following 2-ary constraint query $Q$: given a constraint database $D$ with $D(T) = \{\psi_1, \ldots, \psi_M\}$, consider the formula

$$\chi := \exists x_3(\psi_1 \vee \cdots \vee \psi_M) \ .$$

Since $\mathbf{R}$ admits effective quantifier elimination, we can compute from $\chi$ a quantifier-free formula $\varphi$, in the variables $x_1$ and $x_2$, equivalent to $\chi$ over $\mathbf{R}$. We then transform $\chi$ into disjunctive normal form:

$$\chi \equiv \gamma_1 \vee \cdots \vee \gamma_\ell \ .$$

Then define $Q(D) := \{\gamma_1, \ldots, \gamma_\ell\}$.

This constraint query $Q$ is consistent; it represents the 2-ary unrestricted query mapping each semi-algebraic set in $\mathbb{R}^3$ to its projection on the $x_1x_2$-plane.

To give an example of a 1-ary unrestricted query that is not closed, consider the one that maps every $D$ to $\mathbb{Z}$. This query is not closed because $\mathbb{Z}$ is not a semi-algebraic set in $\mathbb{R}$.

Finally, to give a (rather contrived) example of a constraint query that is not consistent, consider the one that maps each $D$ to the singleton set consisting of that constraint tuple in $D(T)$ that has the largest sum of all constants appearing in it (if there is no unique such tuple, the constraint query is undefined on $D$). Indeed, consider the constraint databases $D_1$ and $D_2$ with $D_1(T) = \{0 < x_1 \wedge x_1 < 3,\ 1 < x_1 \wedge x_1 < 4\}$ and $D_2(T) = \{0 < x_1 \wedge x_1 < 3,\ 2 < x_1 \wedge x_1 < 4\}$. Both represent exactly the same unrestricted relation, namely the interval $(0, 4)$. However, on $D_1$ the output of the query is $\{1 < x_1 \wedge x_1 < 4\}$, while on $D_2$ it is $\{2 < x_1 \wedge x_1 < 4\}$. These two singleton constraint relations represent different unrestricted relations. Hence, our constraint query cannot possibly represent some unrestricted query, as the latter, being a mapping, cannot have two different results on the same input.

### 2.4.2 The Relational Calculus with Constraints

The relational calculus (first-order logic), with a given a class of constraints, constitutes a very basic constraint query language, which we next introduce.

For more background on the relational calculus in the setting of standard, non-constraint databases, we refer to textbooks on database theory, as well as to the Introduction (Chapter 1).

We fix some schema $SC$ in what follows.

**Definition 2.4.5 (Relational Calculus).** *Let $\Omega$ be a vocabulary. A* relational calculus formula over $\Omega$ *is a first-order logic formula over the expanded vocabulary $(\Omega, SC)$ obtained by expanding $\Omega$ with the relation names (viewed as predicate symbols) of the schema $SC$. This class of formulas is denoted by* $\mathrm{FO}(\Omega, SC)$*, or simply* $\mathrm{FO}(\Omega)$ *if $SC$ is understood from the context.*

The cases $\Omega = (+, \cdot, 0, 1, <)$ and $\Omega = (+, <, 0, 1)$ occur quite often in work on constraint query languages, and a specific notation is in use for them:

**Definition 2.4.6 ($\mathrm{FO} + \textsc{Poly}$ and $\mathrm{FO} + \textsc{Lin}$).** $\mathrm{FO} + \textsc{Poly}$ *will denote the class* $\mathrm{FO}(+, \cdot, 0, 1, <)$*, and* $\mathrm{FO} + \textsc{Lin}$ *the class* $\mathrm{FO}(+, <, 0, 1)$.

*Example 2.4.2.* With $R$ a relation name of arity 2, the following is a formula in $\mathrm{FO} + \textsc{Lin}$

$$R(x_1, x_2) \vee \exists y (R(x_1, y) \wedge R(y, x_2) \wedge (x_1 + x_2 < y) \wedge (x_2 - y < 0)) \, ,$$

and the following is a formula in $\mathrm{FO} + \textsc{Poly}$:

$$R(x_1, x_2) \vee \exists y (R(x_1, y) \wedge R(y, x_2) \wedge (x_1 \cdot x_2 < y) \wedge (x_2 - y^3 < 0)) \, .$$

These are purely syntactical examples; the semantics of relational calculus formulas is defined next.

Given an $\Omega$-structure $\mathcal{M}$, relational calculus formulas over $\Omega$ express total (i.e., everywhere defined) unrestricted queries over $\mathcal{M}$ in the obvious manner:

**Definition 2.4.7 (Unrestricted Relational Calculus Queries).** *Fix an $\Omega$-structure $\mathcal{M} = (\mathcal{U}, \Omega)$. A $\mathrm{FO}(\Omega, SC)$ formula $\varphi(x_1, \ldots, x_k)$ expresses the $k$-ary unrestricted query $Q$ over $\mathcal{M}$ defined as follows. Given an unrestricted database $D$ with schema $SC$ over $\mathcal{M}$, we can expand $\mathcal{M}$ to a structure $\langle \mathcal{M}, D \rangle = \langle \mathcal{U}, \Omega, D \rangle$ over the expanded vocabulary $(\Omega, SC)$ by adding the interpretation $D(T)$, for each relation name $T \in SC$ in $D$, to $\mathcal{M}$. Then*

$$Q(D) := \{(a_1, \ldots, a_k) \in \mathcal{U}^k \mid \langle \mathcal{M}, D \rangle \models \varphi(a_1, \ldots, a_k)\} \, .$$

The above definition raises a most basic question. In which cases do we know that the unrestricted query expressed by a relational calculus formula is closed? And if it is, can we find a corresponding constraint query that is effectively computable? The following proposition answers this question. Although from a purely mathematical point of view it is a perfectly disguised triviality, it is at the same time the most important proposition of this chapter. Indeed, it encapsulates the fundamental mechanism lying behind the use of first-order logic as a constraint query language.

**Proposition 2.4.8.** *If, and only if, $\mathcal{M}$ admits effective quantifier elimination, every relational calculus formula $\varphi$ expresses a consistent, effectively computable, total constraint query, that represents the unrestricted query expressed by $\varphi$.*

*Proof.* Assume $\mathcal{M}$ indeed admits effective quantifier elimination. Given $\varphi$, define the following constraint query $Q_\varphi$. Given a constraint database $D$, replace in $\varphi$ every occurrence of an atomic formula of the form $R(\bar{x})$, with $R \in SC$ a relation name of the schema, by the formula corresponding to $D(R)$ (we might first have to rename some variables used in $\varphi$ to avoid clashes). This yields a first-order formula $\chi$ which is purely over the constraint vocabulary $\Omega$ alone; no relation names of the schema occur in $\chi$. By our assumption, $\chi$ is equivalent, in $\mathcal{M}$, to a quantifier-free formula $\psi$, which we may assume to be in disjunctive normal form $\psi_1 \vee \cdots \vee \psi_\ell$. Then define $Q_\varphi(D) := \{\psi_1, \ldots, \psi_\ell\}$.

Conversely, assume every relation calculus formula expresses a constraint query as stated in the proposition. Then we can effectively eliminate quantifiers in $\mathcal{M}$ as follows. Take a formula of the form $\exists y_1 \psi(y_1, \ldots, y_n)$ where we may assume $\psi$ to be quantifier-free and in disjunctive normal form $\psi_1 \vee \cdots \vee \psi_\ell$. Consider a schema $SC$ having a relation name $R$ of arity $n$, and consider a constraint database $D$ where $D(R) = \{\psi_1, \ldots, \psi_\ell\}$. Finally, consider the relational calculus query $\varphi \equiv \exists y_1 R(y_1, \ldots, y_n)$. By assumption, $\varphi$ expresses an effectively computable, total constraint query $Q$, representing the semantics of $\varphi$. So, applying $Q$ to $D$ yields a constraint relation $r = \{\gamma_1, \ldots, \gamma_p\}$, such that $r$ represents the relation $\{(a_2, \ldots, a_n) \mid \mathcal{M} \models \exists y_1 \psi(y_1, a_2, \ldots, a_n)\}$. In other words, the equivalence

$$\forall y_2 \cdots \forall y_n (\exists y_1 \psi \leftrightarrow (\gamma_1 \vee \cdots \vee \gamma_p))$$

holds in $\mathcal{M}$, so that $(\gamma_1 \vee \cdots \vee \gamma_p)$ is a quantifier-free formula equivalent to $\exists y_1 \psi$ and effectively computable from it, as had to be shown. □

*Example 2.4.3.* Let us illustrate the above using FO + POLY over $\mathbf{R}$. Assume that the schema consists of one binary relation name $R$, and consider the simple FO + POLY formula $\varphi(y) \equiv \exists x R(x, y)$. Let $D$ be the constraint database in which $D(R)$ equals the singleton $\{y = x^2\}$. To evaluate $\varphi$ on $D$, we replace the occurrence of $R$ in $\varphi$ by its defining formula to get

$$\chi(y) \equiv \exists x (y = x^2) \,.$$

This formula is equivalent, in $\mathbf{R}$, to the quantifier-free formula

$$\psi(y) \equiv y = 0 \vee y > 0 \,,$$

which is already in disjunctive normal form. Hence we can define the result of the constraint query $Q_\varphi$ defined by $\varphi$ applied to $D$ as $Q_\varphi(D) := \{y = 0, y > 0\}$.

Note that this example would fail if we would omit the $<$ predicate, i.e., if we work in the constraint structure $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$. In this context the

unrestricted query expressed by $\varphi$ would *not* be closed. Not surprisingly, the structure $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$ does not admit quantifier elimination either.

### 2.4.3 Computational Feasibility

The proof of Proposition 2.4.8 describes a crude but effective evaluation mechanism for constraint relational calculus queries $\varphi$. Ignoring the conversion to DNF at the end, computing the result of $\varphi$ on a constraint database $D$ is reduced to simply eliminating the quantifiers from a formula obtained by "plugging" the contents of $D$ in the appropriate slots in $\varphi$. Denote this formula by $\text{plug}(D, \varphi)$. We can now make the following two simple but important observations:

1. The size of $\text{plug}(D, \varphi)$ is linear in the size of $D$.
2. Because each constraint relation of $D$ is quantifier-free, the *number of quantifiers occurring* in $\text{plug}(D, \varphi)$ is the same as in $\varphi$, and in particular, is *independent of* $D$.

Why is this important? For most structures, quantifier elimination is usually computationally expensive. Typically used algorithms require exponential, or even double exponential, time in the worst case. However, a finer analysis often reveals that this exponential behavior can be confined to an exponentiality in *the number of quantifiers to be eliminated only*. More precisely, quantifier elimination procedures can often be made to run, on an input formula $\chi$ of size $n$, in time $p(n) \cdot e(q)$, where $p$ is a polynomial in $n$, and $e$ is an exponential, or even double exponential, function in $q$, where $q$ is the number of quantifiers occurring in $\chi$.

Let us now apply this to the situation where $\chi$ is $\text{plug}(D, \varphi)$. From the previous two observations, we get that $n$ is proportional to the size of $D$, and that $q$, and hence also $e(q)$, is a constant independent of $D$. In other words, *evaluation of a constraint query $\varphi$ can be done in polynomial time.*

The above considerations actually apply to all the structures mentioned in Example 2.3.1. So, we have the following definition and proposition:

**Definition 2.4.9 (Data Complexity).** *Let $Q$ be a total constraint query, and let $f$ be a function on the natural numbers. Then we say that the* data complexity of $Q$ is $O(f)$ in time *if there is an algorithm that computes, given as input a constraint database $D$ of size $n$, the output $Q(D)$ in $O(f(n))$ time.*

*We can similarly define data complexity in space.*

**Proposition 2.4.10.** *Over any structure mentioned in Example 2.3.1, every constraint relational calculus query has polynomial-time data complexity.* □

We point out that sharper complexity bounds can be derived for various specific structures; these will be discussed in Chapters 4 and 7.

### 2.4.4 The Relational Algebra with Constraints

The classical equivalence between the relational calculus and the relational algebra in standard relational databases, carries over very easily to constraint databases. In this section, we show how. Our presentation will be rather brief, because, by and large, the connection between algebra and calculus in the constraint model does not involve any other fundamental insights beyond those already involved in the connection between algebra and calculus in standard relational databases.

As always, we fix a vocabulary $\Omega$, a structure $\mathcal{M}$ over $\Omega$, and a database schema $SC$. The *relational algebra expressions (RAEs)*, and their *arities*, are inductively defined as follows. Let $\mathcal{U}$ denote the domain of $\mathcal{M}$.

- $\mathcal{U}$ is a RAE of arity 1.
- Each relation name $R \in SC$ is a RAE. Its arity is given by $SC$.
- If $e_1$ and $e_2$ are RAEs of arities $k_1$ and $k_2$ respectively, then the *cartesian product* $(e_1 \times e_2)$ is a RAE of arity $k_1 + k_2$, and, *provided that* $k_1 = k_2$, the *union* $(e_1 \cup e_2)$ and the *difference* $(e_1 - e_2)$ are RAEs of arity $k_1$.
- If $e$ is a RAE of arity $k$, and $i_1, \ldots, i_p \in \{1, \ldots, k\}$, then the *projection* $\pi_{i_1,\ldots,i_p}(e)$ is a RAE of arity $p$.
- Finally, if $e$ is a RAE of arity $k$, and $\theta$ is a quantifier-free formula over $\Omega$ on the variables $x_1, \ldots, x_k$, then the *selection* $\sigma_\theta(e)$ is a RAE of arity $k$.

A RAE $e$ of arity $k$ defines a $k$-ary unrestricted query $Q_e$ and a total computable $k$-ary constraint query $Q'_e$ in the following straightforward manner. To be able to deal effectively with projection, we have to assume (as in the relational calculus, cf. Proposition 2.4.8) that $\mathcal{M}$ admits effective quantifier elimination. Let $D$ be an unrestricted database with schema $SC$, and let $D'$ be a constraint database with schema $SC$.

- If $e$ is $\mathcal{U}$, then $Q_e(D) := \mathcal{U}$, and $Q'_e(D') := \{x_1 = x_1\}$.
- If $e$ is $R \in SC$, then $Q_e(D) := D(R)$, and $Q'_e(D') := D'(R)$.
- If $e$ is $(e_1 \times e_2)$, then $Q_e(D) := Q_{e_1}(D) \times Q_{e_2}(D)$. Rename the variables $x_1, \ldots, x_{k_2}$ used in $Q'_{e_2}(D')$ to $x_{k_1+1}, \ldots, x_{k_1+k_2}$, respectively. Then
$$Q'_e(D') := \{\psi \wedge \chi \mid \psi \in Q'_{e_1}(D'),\ \chi \in Q'_{e_2}(D')\}$$
- If $e$ is $(e_1 \cup e_2)$, then $Q_e(D) := Q_{e_1}(D) \cup Q_{e_2}(D)$, and $Q'_e(D') := Q'_{e_1}(D') \cup Q'_{e_2}(D')$.
- If $e$ is $(e_1 - e_2)$, then $Q_e(D) := Q_{e_1}(D) - Q_{e_2}(D)$. Consider the formula
$$\gamma \equiv \Big(\bigvee_{\psi \in Q'_{e_1}(D')} \psi\Big) \wedge \neg\Big(\bigvee_{\psi \in Q'_{e_2}(D')} \psi\Big)\ .$$
Put $\gamma$ in DNF $\gamma_1 \vee \cdots \vee \gamma_\ell$. Then $Q'_e(D') := \{\gamma_1, \ldots, \gamma_\ell\}$.

– If $e$ is $\pi_{i_1,\ldots,i_p}(e')$, with $e'$ of arity $k$, then $Q_e(D) := \{(a_{i_1},\ldots,a_{i_p}) \mid (a_1,\ldots,a_k) \in Q_{e'}(D)\}$. Suppose the formula corresponding to $Q'_{e'}(D')$, after renaming the variables $x_1$, ..., $x_k$ to $y_1$, ..., $y_k$, respectively, is $\psi(y_1,\ldots,y_k)$. Consider the formula

$$\chi(x_1,\ldots,x_p) \equiv \exists y_1 \cdots \exists y_n \psi \wedge \bigwedge_{j=1}^{p} x_j = y_{i_j} \ .$$

Compute a quantifier-free formula $\gamma$, equivalent to $\chi$, in DNF $\gamma_1 \vee \cdots \vee \gamma_\ell$. Then $Q'_e(D') := \{\gamma_1,\ldots,\gamma_\ell\}$.

– Finally, if $e$ is $\sigma_\theta(e')$, then $Q_e(D) := \{\bar{a} \mid \bar{a} \in Q_{e'}(D) \text{ and } \mathcal{M} \models \theta(\bar{a})\}$. Suppose the formula corresponding to $Q'_{e'}(D')$ is $\psi$. Put $\psi \wedge \theta$ in DNF, yielding $\gamma_1 \vee \cdots \vee \gamma_\ell$. Then $Q'_e(D') := \{\gamma_1,\ldots,\gamma_\ell\}$.

Now the following property is straightforwardly verified:

**Proposition 2.4.11.** *For every RAE $e$, the constraint query $Q'_e$ represents the unrestricted query $Q_e$. In particular, $Q'_e$ is consistent and $Q_e$ is closed.* □

A relational calculus formula $\varphi$ and a RAE $e$ are called *equivalent* if they define the same unrestricted query. The reader familiar with the classical equivalence of relational algebra and calculus in standard relational databases, will encounter no problems in verifying the analogue for constraint databases:

**Theorem 2.4.12.** *Let $\Omega$ be a vocabulary and $SC$ a database schema. Every relational calculus formula over $(\Omega, SC)$ can be effectively converted into an equivalent RAE over $(\Omega, SC)$, and vice versa.* □

At this point it must be stressed that the evaluation mechanism we described for constraint relational algebra queries only served to show that, in principle, the relational algebra serves as an effective constraint query language. In concrete implementations, which will typically depend on the particulars of the constraint structure $\mathcal{M}$, other, more efficient strategies will be used for evaluating relational algebra constraint queries. We refer to Parts III and IV for concrete discussions of such evaluation strategies.

## 2.5 Computationally Complete Constraint Query Languages

In this section we show that by extending the relational calculus with the standard programming features of assignment statements, sequential composition, and while loops, we obtain a constraint query language that is computationally complete in the context of all constraint structures that "embed the natural numbers."

Programs in the language $\mathrm{FO}(\Omega, SC) + \textsc{While}$ (or simply $\mathrm{FO} + \textsc{While}$ if the constraint vocabulary $\Omega$ and the database schema $SC$ are understood) are sequences of *assignment statements* and *while-loops*. We assume a sufficient supply of *relation variables*, each with an appropriate arity, which can be thought of as new relation names not present in the given schema $SC$.

An assignment statement is an expression of the form

$$X := \{\bar{x} \mid \varphi(\bar{x})\}\ ,$$

where $X$ is a relation variable of arity equal to the length of the vector $\bar{x}$ of variables, and $\varphi$ is an $\mathrm{FO}(\Omega, SC')$ formula, where $SC'$ is the expansion of $SC$ with the relation variables introduced in previously occurring assignment statements.

A while-loop is an expression of the form

$$\textsc{While}\ \varphi\ \textsc{do}\ P\ \textsc{od}$$

where $\varphi$ is as above, but without free variables, and $P$ is in turn an $\mathrm{FO} + \textsc{While}$ program (called the *body* of the loop).

The semantics of a program applied to an input constraint database $D$ is the obvious one of operational, step by step execution. So, the effect of an assignment statement is to evaluate the relational calculus query on the right-hand side on the constraint database consisting of $D$ augmented with the values of the previously assigned-to relation variables, and to assign the result of this evaluation to the relation variable on the left-hand side; the effect of a while-loop is to execute the body as long as $\varphi$ evaluates to true.

Assume that we work over a constraint structure admitting effective quantifier elimination. Since every step in the execution of a program consists of a relational calculus query, which is always consistent, total, and computable, each program expresses a consistent, computable constraint query, which however may be only partially defined because of possibly non-terminating loops.

*Example 2.5.1.* Let us illustrate this language in the context of real polynomial inequality constraints (so we will work with $\mathrm{FO} + \textsc{Poly} + \textsc{While}$ over $\mathbf{R}$). Assume the schema contains a binary relation name $S$. So for any constraint database $D$, $D(S)$ represents a semi-algebraic set in the plane $\mathbb{R}^2$. Consider the following program, in which we use the abbreviation $\overline{(x_1, y_1)(x_2, y_2)}$ for the closed line segment between the two points $(x_1, y_1)$ and $(x_2, y_2)$ (this is easily expressible in $\mathrm{FO} + \textsc{Poly}$).

$X := \{(x_1, y_1, x_2, y_2) \mid S(x_1, y_1) \wedge S(x_2, y_2) \wedge \overline{(x_1, y_1)(x_2, y_2)} \subseteq S\}$;
$Y := \emptyset$;
$\textsc{While}\ Y \neq X\ \textsc{do}$
  $Y := X$;
  $X := X \cup \{(x_1, y_1, x_2, y_2) \mid \exists x_3 \exists y_3 (X(x_1, y_1, x_3, y_3) \wedge X(x_3, y_3, x_2, y_2))\}$
$\textsc{od}$.

In the first statement, the program initializes relation variable $X$ to the set of pairs of points in $S$ for which the closed line segment between them also lies entirely in $S$. Then, in the while-loop, $X$ is transitively closed. If the while-loop terminates, then the final value of $X$ will hold all pairs of points in $S$ that are connected by a piecewise linear curve lying entirely in $S$. For example, on constraint databases $D$ where the semi-algebraic set represented by $D(S)$ is actually semi-*linear* (cf. Definition 2.3.6), the while-loop will always terminate. An example of a database $D$ on which the while-loop will not terminate is given by $D(S) := \{x^2 < y \wedge y < x^2 + 1\}$; this constraint relation represents the (unbounded) region between the standard parabola and its translation by one unit up the $y$-axis. An illustration is given in Figure 2.5.1.

**Fig. 2.5.1.** Region represented by the constraint relation $\{x^2 < y \wedge y < x^2 + 1\}$. The program of Example 2.5.1 will not terminate given this constraint relation as input.

Let $\mathcal{M}$ be a structure with domain $\mathcal{U}$. We say that $\mathcal{M}$ *embeds the natural numbers* if the following conditions are satisfied:

- $\mathcal{U}$ contains the set $\mathbb{N}$ of natural numbers as a subset;
- some binary relation is definable in $\mathcal{M}$ (in the sense of Definition 2.3.2) which is a function, and which is a superset of the binary relation $\{(n, n+1) \mid n \in \mathbb{N}\}$.
- the singleton unary relation $\{(0)\}$ is also definable in $\mathcal{M}$.

Obvious examples of such structures include the real field, the reals or rationals with addition, and the integers with addition.

Naturally, two constraint queries are called *equivalent* if they represent the same unrestricted query. We are going to show:

**Theorem 2.5.1.** *Assume the constraint structure $\mathcal{M}$ admits effective quantifier elimination and embeds the natural numbers. Then for every partial computable constraint query $Q$ there is an equivalent* FO + WHILE *program.*

*Proof.* We begin by observing that, since $\mathcal{M}$ embeds the natural numbers, we can simulate arbitrary counter programs in FO + WHILE. Indeed, a counter variable $i$ with value $n$ can be simulated by a unary relation variable holding the singleton $\{(n)\}$. Simulation of incrementing a counter is possible since the successor relation is definable in FO, and resetting a counter or testing it for zero is possible since the singleton $\{(0)\}$ is definable. As a consequence, since counter machines have full computational power on the natural numbers, FO + WHILE has as well.

To simplify the argument, let us assume that the database schema consists of a single, binary relation name $S$. So, constraint databases are defined by quantifier-free formulas in two variables $x$ and $y$. Now it is possible to encode such formulas, and the terms occurring in them, by natural numbers, in such a way that the encoding of a term or formula comes before the encoding of any other term or formula in which it occurs as a subterm or subformula. (One simple way of doing this is by encoding a term or formula, being a string over some finite alphabet $\Sigma$, by the corresponding natural number the string represents when viewed as a number written in base $|\Sigma|$.)

We can then write a program *Encode* that computes the smallest natural number encoding a formula equivalent to the formula corresponding to the input constraint database. Once we have this number, we can, as observed, perform any desired computation on it. So we perform the computation corresponding to the given constraint query we want to express. Finally, using a program *Decode*, we derive, from the natural number resulting from this computation, the constraint relation it encodes. We thus have computed the result of the constraint query on the given input constraint database.

The program *Encode* runs through the natural numbers $n = 0, 1, 2, \ldots$ (stored in a counter variable *Counter*) one by one, maintaining two relation variables *Term* and *Formula*. If $n$ encodes a term $t$, then all tuples $(n, a, b, c)$ are added to *Term* for which $t$ evaluates to $c$ when variable $x$ is interpreted by $a$ and $y$ by $b$. If $n$ encodes a formula $\psi$, then all tuples $(n, a, b)$ are added to *Formula* for which $\psi(a, b)$ is true. If $n$ neither encodes a term nor a formula, we simply skip to $n+1$. This process is repeated until we encounter the first $n$ such that $\forall x \forall y(\mathit{Formula}(n, x, y) \leftrightarrow S(x, y))$ holds. This $n$ is the desired result of *Encode*. The program *Decode* is entirely similar, except that here we stop the loop when we have reached $n$ equal to the natural number given as input to *Decode*. The desired result of *Decode* is then given by $\{(x, y) \mid \mathit{Formula}(n, x, y)\}$.

It remains to show how these relations *Term* and *Formula* can be maintained. We do a simple case analysis.

- If $n$ encodes a constant symbol $c$, then perform

  $\mathit{Term} := \mathit{Term} \cup \{(n, x, y, c) \mid \mathit{Counter}(n)\}$.

- If $n$ encodes a term of the form $f(t_1, \ldots, t_r)$, then, supposing the encodings of the subterms $t_1, \ldots, t_r$ are stored in counter variables $\mathit{Counter}_1, \ldots,$

$Counter_r$ respectively, perform

$$Term := Term \cup \{(n,x,y,z) \mid Counter(n) \wedge \exists n_1 \cdots \exists n_r (\bigwedge_{i=1}^{r} Counter_i(n_i) \wedge \exists z_1 \cdots \exists z_r (\bigwedge_{i=1}^{r} Term(n_i,x,y,z_i) \wedge z = f(z_1,\ldots,z_r)))\} .$$

– If $n$ encodes an atomic formula of the form $t_1 = t_2$, then perform

$$Formula := Formula \cup \{(n,x,y) \mid Counter(n) \wedge \exists n_1 \exists n_2 (\bigwedge_{i=1}^{2} Counter_i(n_i) \wedge \exists z (\bigwedge_{i=1}^{2} Term(n_i,x,y,z)))\} .$$

– If $n$ encodes an atomic formula of the form $p(t_1,\ldots,t_r)$, then perform

$$Formula := Formula \cup \{(n,x,y) \mid Counter(n) \wedge \exists n_1 \cdots \exists n_r (\bigwedge_{i=1}^{r} Counter_i(n_i) \wedge \exists z_1 \cdots \exists z_r (\bigwedge_{i=1}^{r} Term(n_i,x,y,z_i) \wedge p(z_1,\ldots,z_r)))\} .$$

– If $n$ encodes a formula of the form $\psi_1 \vee \psi_2$, then, supposing the encodings of subformulas $\psi_1$ and $\psi_2$ are stored in counter variables $Counter_1$ and $Counter_2$ respectively, perform

$$Formula := Formula \cup \{(n,x,y) \mid Counter(n) \wedge \exists n_1 \exists n_2 (\bigwedge_{i=1}^{2} Counter_i(n_i) \wedge (\bigvee_{i=1}^{2} Formula(n_i,x,y)))\} .$$

– Finally, if $n$ encodes a formula of the form $\neg\psi_1$, then perform

$$Formula := Formula \cup \{(n,x,y) \mid Counter(n) \wedge \exists n_1 (Counter_1(n_1) \wedge \neg Formula(n_1,x,y))\} .$$

This concludes the proof of the theorem. □

## 2.6 Equivalence and satisfiability in the relational calculus with constraints

A basic problem for any query language is the equivalence of formulas: given two formulas, do they express the same query? For query languages with sufficient expressive power, more often than not, this problem is undecidable.

It is undecidable, for example, for the standard relational calculus on finite relational databases. What about the constraint setting?

Let us begin with a precise definition. For any fixed structure $\mathcal{M}$:

**Definition 2.6.1.** *Two relational calculus formulas over $\mathcal{M}$ are called* equivalent over $\mathcal{M}$ *if they express the same unrestricted query over $\mathcal{M}$.*

As far as effective decidability is concerned, the equivalence problem is equivalent (sic) to the satisfiability problem, defined as follows:

**Definition 2.6.2.** *A relational calculus sentence over $\mathcal{M}$ is called* constraint-satisfiable *if it is satisfied on at least one definable database over $\mathcal{M}$.*

The focus on definable as opposed to arbitrary unrestricted databases, in the above definition, is important, as illustrated by the following example.

*Example 2.6.1.* Consider $\mathrm{FO} + \textsc{Poly}$ over $\mathbf{R}$ and a schema with a unary relation name $S$. The formula

$$\forall x(S(x) \to x \geq 0) \wedge S(0) \wedge \forall x > 0(S(x) \leftrightarrow S(x-1))$$

is true of a unrestricted database $D$ over $\mathbf{R}$ if and only if $D(S)$ equals the set of natural numbers. But this is not a definable (semi-algebraic) subset of $\mathbf{R}$, so although formula $\alpha$ is satisfiable by an unrestricted database, it is *not* constraint-satisfiable.

We next present a rather general undecidability result for constraint-satisfiability. The argument is based on the following property of structures:

**Definition 2.6.3.** *For any infinite structure $\mathcal{M}$, we say that* the relational calculus over $\mathcal{M}$ can express finiteness *if there exists a relational calculus sentence over $\mathcal{M}$ and the singleton schema $\{S\}$ (with $S$ a unary relation name) which is true of any definable database $D$ over $\mathcal{M}$ if and only if $D(S)$ is finite.*

Note that it is not at all obvious that one should be able to express finiteness in the relational calculus. For instance, the classical compactness theorem from first-order model theory implies that any first-order sentence having arbitrary large finite models also has an infinite model, so finiteness of arbitrary unrestricted relations is certainly not expressible in first-order logic. Still, as we will see shortly, if one focuses (as we do in constraint databases) only on relations that are definable using constraints, finiteness can become expressible in the relational calculus over certain constraint structures $\mathcal{M}$.

Definition 2.6.3 is relevant to satisfiability because it is linked to *satisfiability in the finite*. A first-order sentence is called satisfiable in the finite if it has a *finite* model. Satisfiability in the finite is especially relevant in the context of standard, finite relational databases. The problem is well known to be undecidable for any vocabulary having at least one predicate symbol of arity $\geq 2$. Now under the assumption of Definition 2.6.3, we can easily reduce constraint-satisfiability to satisfiability in the finite, thus obtaining:

**Proposition 2.6.4.** *If the relational calculus over $\mathcal{M}$ can express finiteness, then constraint-satisfiability (and equivalence) of relational calculus formulas over $\mathcal{M}$ is undecidable for any schema containing at least one relation name of arity $\geq 2$.*

*Proof.* Let $\varphi$ be the sentence expressing finiteness. Let $R$ be a binary relation name. Let $\varphi_1$ be $\varphi$ applied not to $S$, but to the first projection of $R$; formally, $\varphi_1$ is obtained from $\varphi$ by replacing subformulas of the form $S(x)$ by $\exists y R(x, y)$. Similarly, let $\varphi_2$ be $\varphi$ applied to the second projection of $R$. We need $\varphi_1$ and $\varphi_2$ because a binary relation is finite if and only if its two projections are. We now reduce satisfiability in the finite to constraint-satisfiability over $\mathcal{M}$ as follows. A first-order sentence $\psi$ over $\{R\}$ is satisfiable in the finite if and only if the $\mathrm{FO}(\mathcal{M}, \{R\})$ sentence

$$\varphi_1 \wedge \varphi_2 \wedge \psi$$

is constraint-satisfiable. □

A detail which we have swept under the carpet in the above proof is that, in order for the proof to work, it must be possible to define an isomorphic copy of any finite binary relation using $\mathcal{M}$-constraints. We say in this case that $\mathcal{M}$ *accommodates finite relations*. This is usually no problem: if it is not realistic to provide constants for all elements of $\mathcal{M}$, there are usually at least some constants and functions by which we can generate infinitely many elements of $\mathcal{M}$. For example, in $\mathbf{R}$, we can generate all natural numbers using 0, 1, and $+$.

When is Proposition 2.6.4 applicable to a structure? For ordered structures, one well-studied property of structures that guarantees expressibility of finiteness is that of *o-minimality*. A structure is *ordered* if one of its predicates is a total order $<$. Many structures occurring in practice are ordered.

**Definition 2.6.5.** *An ordered structure is said to be* o-minimal *if every definable subset $V$ of the domain $\mathcal{U}$ of the structure can be written as the union of a finite number of singletons and open intervals. By an open interval we mean a set of the form $\{x \mid x < b\}$, $\{x \mid a < x < b\}$, or $\{x \mid a < x\}$, for some $a, b \in \mathcal{U}$.*

Here, "definable subset" can be taken to refer to our usual notion of definability using $\mathcal{M}$-constraints (which corresponds to definability by a quantifier-free formula over $\mathcal{M}$) on condition that $\mathcal{M}$ admits quantifier elimination (as we must assume anyway for the relational calculus to be usable as a constraint query language).[1]

We observe:

[1] For general structures not necessarily admitting quantifier elimination, the concept of o-minimality is still defined in the same way, but then "definable subset" refers to definability by an arbitrary (not necessarily quantifier-free) first-order formula over $\mathcal{M}$, which moreover can use arbitrary domain elements as constants.

**Lemma 2.6.6.** *If $\mathcal{M}$ is o-minimal and densely ordered, then the relational calculus over $\mathcal{M}$ can express finiteness.*

*Proof.* Consider a constraint database $D$ over $\mathcal{M}$ with schema $\{S\}$. Since $\mathcal{M}$ is o-minimal and $D(S)$ is definable, we know that $D(S)$ is a finite union of singletons and open intervals. Since $\mathcal{M}$ is densely ordered, any proper open interval is infinite. Hence, $D(S)$ is finite if and only if it does not contain any such intervals. This is easy first-order expressible as

$$\neg\exists x\exists y(x < y \wedge \forall z(x < z < y \rightarrow S(z))) .$$

□

Lemma 2.6.6 and Proposition 2.6.4 immediately imply:

**Corollary 2.6.7.** *If $\mathcal{M}$ is o-minimal, densely ordered, and accommodates finite relations, then constraint-satisfiability (and equivalence) of relational calculus formulas over $\mathcal{M}$ is undecidable.*

All the ordered structures mentioned in Example 2.3.1 are clearly densely ordered, and they are also o-minimal. Take for example $\mathbf{R}$ (real polynomial constraints). We already noted that $\mathbf{R}$ admits quantifier elimination, so every definable set is definable by a disjunction of conjunctions of constraints of the form $p > 0$ or $p \leq 0$, where $p$ is a polynomial in one real variable. Such a polynomial defines a continuous function with only finitely many zeros. Hence, a constraint $p > 0$ of $p \leq 0$ indeed defines a finite union of singletons and open intervals, as required for o-minimality.

The reader new to constraint databases should not be lulled into believing that every structure admitting quantifier elimination is also o-minimal, as illustrated by the following example:

*Example 2.6.2.* Consider the expansion of the ordered reals $\langle \mathbb{R}, < \rangle$ with a predicate $\mathbb{Q}$ for the rationals. The resulting structure admits quantifier elimination, but is not o-minimal (the set $\mathbb{Q}$ providing a counterexample).

A more practical example is given by the integers with addition, expanded with all congruences modulo $k$ for $k \geq 2$: $\langle \mathbb{Z}, 0, 1, +, <, \equiv_2, \equiv_3, \ldots \rangle$. This structure admits quantifier elimination, but is not o-minimal (the set of even numbers, defined by $\exists y(x = y + y)$, or equivalently, $x \equiv_2 0$, providing a counterexample).

We conclude this section by returning to the compactness theorem from classical first-order model theory, already mentioned after Definition 2.6.3. As suggested there, Lemma 2.6.6 indicates a failure of compactness in the constraint database setting.

Let us be more precise. Fix a schema $SC$. Generalizing Definition 2.6.2, we say that a *set* $\Sigma$ of $\mathrm{FO}(\mathcal{M}, SC)$ sentences is *constraint-satisfiable* if there is a definable database over $\mathcal{M}$ with schema $SC$ in which *every* sentence $\sigma$ of $\Sigma$ is satisfied. We say that *compactness holds over constraint databases over*

*$\mathcal{M}$ with schema $SC$* if for any infinite set $\Sigma$ of $\mathrm{FO}(\mathcal{M}, SC)$ sentences, we have the following: if every finite subset of $\Sigma$ is constraint-satisfiable, then $\Sigma$ itself is constraint-satisfiable.

We now observe:

**Proposition 2.6.8.** *If $\mathcal{M}$ is o-minimal, densely ordered, and accommodates finite relations, compactness does* not *hold over constraint databases over $\mathcal{M}$ with any non-empty schema.*

*Proof.* Let $S$ be a relation name in the schema (without loss of generality, $S$ is assumed to be unary). Lemma 2.6.6 gives us a relational calculus sentence $\omega$ over $\mathcal{M}$ expressing finiteness of $D(S)$ on any definable database $D$ over $\mathcal{M}$. Furthermore, for any natural number $k$, we can write a calculus sentence $\tau_k$ expressing on any unrestricted database $D$ that $D(S)$ contains exactly $k$ different elements. Since we have assumed that $\mathcal{M}$ accommodates finite relations, every $\tau_k$ is constraint-satisfiable. Now consider the set $\Sigma$ consisting of all sentences $\tau_k$ plus the sentence $\omega$. Every finite subset of $\Sigma$ is constraint-satisfiable, but $\Sigma$ itself is not. Hence compactness does not hold. □

The failure of theorems such as compactness serves to indicate that constraint databases and query languages, like standard relational databases and query languages, define an area of computer science which on the one hand relies heavily on logic, but which on the other hand has its own subtle points which need specific attention and which prevent the "blind" application of known facts from logic. This will be illustrated extensively in the chapters dealing with the expressive power of constraint query languages.

## 2.7 Conjunctive queries with constraints

If equivalence of formulas for some query language is undecidable, one may look for interesting sublanguages for which equivalence becomes decidable. In standard relational database theory, for example, the *conjunctive queries* form a useful decidable sublanguage of the standard relational calculus. In the concrete setting of real polynomial constraints, we will now see that the same can be done for conjunctive queries with constraints, namely, containment and equivalence are effectively decidable.

Fix a schema $SC$. As always $\mathcal{M}$ denotes the structure that interprets the constraints.

**Definition 2.7.1.** *A* conjunctive query (with $\mathcal{M}$-constraints) *is a relational calculus formula of the special form*

$$\gamma(\bar{x}) = \exists \bar{y}\ (B \wedge \varphi)\ ,$$

*where*

- *$B$ is a conjunction of atomic formulas over $SC$, using only variables in $\bar{x}$ or $\bar{y}$, and*
- *$\varphi$ is a quantifier-free formula over $\mathcal{M}$ on variables in $\bar{x}$ or $\bar{y}$.*

We call $B$ the *body* of $\gamma$. As defined above, $B$ is a conjunction $B_1 \wedge \cdots \wedge B_n$, but it should not cause confusion that we also use $B$ as if it were a set $\{B_1, \ldots, B_n\}$, whenever convenient. The formula $\varphi$ is called the *constraints* of $\gamma$ (note the plural: $\varphi$ is an arbitrary quantifier-free formula and thus corresponds to a boolean combination of constraints). Clearly, without these constraints (i.e., with only the trivial constraints **true**), we are back to the standard conjunctive queries on standard relational databases.

*Example 2.7.1.* If $SC$ contains a binary relation name $R$, the following are two examples of conjunctive queries with real polynomial constraints:

$$\begin{aligned} \gamma_1(x,z) &= R(x,x) \wedge R(x,z) \wedge (z < 0 \vee x > 0) \\ \gamma_2(x,z) &= \exists y \exists w (R(x,y) \wedge R(y,z) \wedge x > w^2) \end{aligned}$$

The notions of interest for this section are the following:

**Definition 2.7.2.** *For two conjunctive queries $\gamma_1$ and $\gamma_2$ of the same arity, we say that $\gamma_1$ is* contained *in $\gamma_2$ if for every definable database $D$ over $\mathcal{M}$, we have $\gamma_1(D) \subseteq \gamma_2(D)$. If the stronger condition, $\gamma_1(D) \subseteq \gamma_2(D)$ holds for every arbitrary unrestricted database $D$, we say that $\gamma_1$ is* unrestricted contained *in $\gamma_2$.*

Of course we are not really interested in unrestricted containment, since we are not interested in arbitrary unrestricted databases, but only in the definable ones. However, we shall still analyze unrestricted containment first, as this turns out to be very easy. Then, in the setting of real polynomial constraints (so $\mathcal{M}$ is the real field $\mathbf{R}$), we will show that unrestricted containment of conjunctive queries actually coincides with containment on definable databases only! Recall from Example 2.6.1 that they do *not* coincide for general relational calculus formulas; indeed, the negation of the sentence of that example is contained in the constant empty ("false") query on definable databases, but not on unrestricted databases.

*Example 2.7.2.* Consider $\gamma_1$ and $\gamma_2$ from Example 2.7.1. A moment's reflection reveals that $\gamma_1$ is unrestricted contained in $\gamma_2$.

We begin our analysis by defining some helpful terminology. For simplicity of exposition, from now on we assume that the schema $SC$ consists of a single binary relation name $R$. The extension to general schemas is straightforward.

**Definition 2.7.3.** *Let $\gamma(\bar{x})$ be a conjunctive query in the general form given in Definition 2.7.1, and let $D$ be an unrestricted database over $\mathcal{M}$. A* valuation *of $\gamma$ in $D$ is a mapping from all variables in $\gamma$ (quantified or free) to $\mathcal{M}$ such that*

1. *$\varphi$ is true of $\mathcal{M}$ under $f$; and*
2. *$f(B) \subseteq D(R)$, where by $f(B)$ we mean the set $\{(f(u), f(v)) \mid R(u,v) \in B\}$.*

*For any $f$ satisfying the first condition but not necessarily the second, we call $f(B)$ a* $\varphi$-instantiation *of $B$. Note that $\varphi$-instantiations of $B$ can be viewed as finite unrestricted databases.*

The following lemma is trivial but quite helpful:

**Lemma 2.7.4.** *For any conjunctive query $\gamma(\bar{x})$ and unrestricted database $D$:*

$$\gamma(D) = \{f(\bar{x}) \mid f \text{ a valuation of } \gamma \text{ in } D\} .$$

□

We are now ready to show that unrestricted containment of conjunctive queries with constraints is effectively decidable. The proof actually is as straightforward as in the well-known case of conjunctive queries without constraints on standard relational databases. We first have the following easy lemma:

**Lemma 2.7.5.** *For any two conjunctive queries $\gamma_1(\bar{x})$ and $\gamma_2(\bar{x})$, $\gamma_1$ is unrestricted contained in $\gamma_2$ if and only if $\gamma_1$ is contained in $\gamma_2$ on all $\varphi_1$-instantiations of $B_1$, where $B_1$ is the body of $\gamma_1$ and $\varphi_1$ are the constraints of $\gamma_1$.*

*Proof.* The only-if implication is trivial. For the if-implication, let $D$ be an arbitrary unrestricted database, and let $f$ be a valuation of $\gamma_1$ in $D$. We have to show that $f(\bar{x}) \in \gamma_2(D)$. Note that $f(\bar{x})$ trivially is in $\gamma_1(f(B_1))$. Since $f(B_1)$ is a $\varphi_1$-instantiation of $B_1$, it follows that $f(\bar{x})$ is also in $\gamma_2(f(B_1)) \subseteq \gamma_2(D)$ (since $f(B_1) \subseteq D$). □

We now observe:

**Lemma 2.7.6.** *For any two conjunctive queries $\gamma_1(\bar{x})$ and $\gamma_2(\bar{x})$, there exists a sentence $\psi$ over $\mathcal{M}$ such that $\mathcal{M} \models \psi$ if and only if $\gamma_1$ is unrestricted contained in $\gamma_2$.*

*Proof.* Write $\gamma_i(\bar{x})$, for $i = 1, 2$, in the general form

$$\exists \bar{y}_i (B_i \wedge \varphi_i) .$$

By Lemma 2.7.5, it suffices to give a sentence $\psi$ expressing that $\gamma_1$ is contained in $\gamma_2$ on all $\varphi_1$-instantiations of $B_1$. Let $B_2^*$ be the formula obtained from $B_2$ by replacing each atom $R(u,v)$ by the disjunction

$$\bigvee_{R(u',v') \in B_1} (u = u' \wedge v = v') .$$

Then the desired sentence $\psi$ is

$$\forall \bar{x} \forall \bar{y}_1 (\varphi_1 \rightarrow \exists \bar{y}_2 (\varphi_2 \wedge B_2^*)) .$$

□

*Example 2.7.3.* For $\gamma_1$ and $\gamma_2$ of Example 2.7.1, the sentence $\psi$ constructed in the above proof is

$$\begin{aligned}\forall x \forall z ((z < 0 \vee x > 0) \rightarrow \exists y \exists w (x > w^2 \\ \wedge ((x = x \wedge y = x) \vee (x = x \wedge y = z)) \\ \wedge ((y = x \wedge z = x) \vee (y = x \wedge z = z)))) \; .\end{aligned}$$

The last two lines of this sentence are equivalent to $y = x$, so the sentence simplifies to

$$\forall x (x > 0 \rightarrow \exists w (x > w^2))$$

which is indeed a true sentence of $\mathbf{R}$ (take $w = 0$), thus confirming that $\gamma_1$ is unrestricted contained in $\gamma_2$.

Lemma 2.7.6 immediately implies:

**Corollary 2.7.7.** *If the theory of $\mathcal{M}$ is decidable, then unrestricted containment of conjunctive queries with $\mathcal{M}$-constraints is decidable.* □

Now we want to move from unrestricted containment to containment on definable databases only. Thereto, note that since $\varphi_1$-instantiations of $B_1$ are finite, Lemma 2.7.5 implies that unrestricted containment coincides with containment on finite unrestricted databases only. Hence, if the structure $\mathcal{M}$ is such that *every finite unrestricted database over $\mathcal{M}$ is in fact definable over $\mathcal{M}$,* unrestricted containment clearly coincides with containment on definable databases only. The structure $\mathcal{M}$ trivially has this property, for example, if there is an explicit constant for each element of $\mathcal{M}$, as is the case with the structure $\langle \mathbb{Q}, <, (c)_{c \in \mathbb{Q}} \rangle$ (the rationals with order and constants for each rational number).

However, it is not always realistic to assume that we have constants for every element of the structure. A case in point is the real field $\mathbf{R}$, as there are uncountably many real numbers. Nevertheless, we can still show:

**Proposition 2.7.8.** *For conjunctive queries with real polynomial constraints, the notion of unrestricted containment coincides with that of containment on semi-algebraic databases only.*

(Recall from Definition 2.3.6 that "semi-algebraic" is a synonym for "definable with real polynomial constraints.")

*Proof.* Assume $\gamma_1$ is contained in $\gamma_2$ on all semi-algebraic databases. Now suppose, for the sake of arriving at a contradiction, that $\gamma_1$ is not unrestricted contained in $\gamma_2$. By Lemma 2.7.5, this implies that $\gamma_1$ is *not* contained in $\gamma_2$ on some $\varphi_1$-instantiation of $B_1$. Referring to the proof of Lemma 2.7.6, this means that

$$\{(\bar{x}, \bar{y}_1) \mid \varphi_1 \wedge \neg \exists \bar{y}_2 (\varphi_2 \wedge B_2^*)\} \neq \emptyset \; .$$

Observe that the above set is semi-algebraic. Now a known property of non-empty semi-algebraic sets is that they always contain a point of which all the coordinates are algebraic numbers. Such a point represents a $\varphi_1$-instantiation of $B_1$ containing algebraic numbers only. Now, any finite database over $\mathbf{R}$ containing algebraic numbers only is semi-algebraic: any single algebraic number can (by definition) be defined using real polynomial constraints, and any finite database built using numbers can be defined by building conjunctions and disjunctions. Hence, we have found a semi-algebraic $\varphi_1$-instantiation of $B_1$ on which $\gamma_1$ is not contained in $\gamma_2$. But this is in contradiction with the given that $\gamma_1$ is contained in $\gamma_2$ on all semi-algebraic databases. □

Since testing for equivalence can be done by testing for containment in the two directions, and since the theory of the reals is decidable, Proposition 2.7.8 and Corollary 2.7.7 immediately imply the result announced at the beginning of this section:

**Corollary 2.7.9.** *Containment and equivalence of conjunctive queries with real polynomial constraints are effectively decidable.*

To conclude, we offer an example showing that Proposition 2.7.8 cannot be generalized arbitrarily.

*Example 2.7.4.* The simplest example of a structure that satisfies the usual assumptions (quantifier elimination, decidable theory, o-minimal), but over which unrestricted containment does not coincide with containment on definable databases only, is given by the rationals $\langle \mathbb{Q}, < \rangle$ with order but without constants. Take $R$ unary; the only subset of $\mathbb{Q}$ definable with $<$ only is $\mathbb{Q}$ itself, or $\emptyset$. Hence, the conjunctive query

$$\gamma_1(x, y) = R(x) \wedge x < y$$

is contained in

$$\gamma_2(x, y) = R(x) \wedge R(y) \wedge x < y \ .$$

However, $\gamma_1$ is clearly not unrestricted contained in $\gamma_2$; any $D$ where $D(R)$ is a singleton is a counterexample.

## 2.8 DATALOG with constraints

Apart from the relational algebra and calculus, DATALOG is another well-studied query language in standard relational database theory. Just as we did with the relational algebra and calculus, it is natural to adapt DATALOG for use as a constraint query language. (In fact the original inspiration for the development of constraint databases came from constraint logic programming.)

In this chapter, we give only the basic definitions concerning DATALOG as a constraint query language. More detailed analyses of the possibilities and limitations of using DATALOG for various constraint structures are given in other Chapters (4 and 7).

We fix some constraint vocabulary $\Omega$ and some schema $SC$ in what follows.

**Definition 2.8.1 (DATALOG with constraints).** *Let $SC'$ be a schema disjoint from $SC$. A* DATALOG program over $\Omega$ with intentional schema $SC'$ *is a set of rules of the form*

$$\alpha \text{ :- } \beta_1, \ldots, \beta_\ell$$

*where*

- *$\alpha$, called the* head*, is an atomic formula of the form $H(\cdots)$, with $H$ a relation name in $SC'$; and*
- *every $\beta_i$ is an atomic formula over the combined vocabulary $(\Omega, SC, SC')$ .*

Given an $\Omega$-structure $\mathcal{M}$ interpreting the constraints, we also call a DATALOG program over $\Omega$ a DATALOG program over $\mathcal{M}$. We will next define the unrestricted query and the constraint query expressed by a DATALOG program over $\mathcal{M}$ (recall Definitions 2.4.1 and 2.4.3). Unlike the situation in the relational calculus, effective quantifier elimination in $\mathcal{M}$ will no longer be sufficient to guarantee that the unrestricted query expressed by a DATALOG program is closed. Moreover, the constraint query expressed by a DATALOG program will no longer be guaranteed to be total. DATALOG is therefore not as easily usable as a constraint query language as the relational calculus and algebra are.

Let us begin by defining the unrestricted query $Q_P$ expressed by a DATALOG program $P$. Thereto, we first define:

**Definition 2.8.2.** *Let $P$ be a* DATALOG *program over $\mathcal{M}$ with intentional schema $SC'$, and let $D$ be an unrestricted database over $\mathcal{M}$ with schema $SC$. Then $P$ and $D$ determine an operator on unrestricted databases over $\mathcal{M}$ with schema $SC'$, denoted by $T_P^D$, as follows. Let $D'$ be such a database. Let $H$ be a relation name in $SC'$, of arity $k$. Then $T_P^D(D')(H)$ consists of all tuples $(a_1, \ldots, a_k)$ over $\mathcal{M}$ such that there exists a rule*

$$\rho: \quad H(t_1, \ldots, t_k) \text{ :- } \beta_1, \ldots, \beta_\ell$$

*in $P$, and a* valuation $v$ of $\rho$ in $D'$*, such that*

$$(a_1, \ldots, a_k) = (v(t_1), \ldots, v(t_k)) .$$

*Here, a valuation $v$ of $\rho$ in $D'$ is a mapping from the variables occurring in $\rho$ to elements of $\mathcal{M}$ such that for each $i = 1, \ldots, \ell$, $\beta_i$ is true in $\langle \mathcal{M}, D, D' \rangle$ under $v$.*

Note that this operator $T_P^D$ is *monotone*: if $D' \subseteq D''$, then $T_P^D(D') \subseteq T_P^D(D'')$, where for two databases $D_1$ and $D_2$ with schema $SC'$ we write $D_1 \subseteq D_2$ to mean that $D_1(H) \subseteq D_2(H)$ for each $H$ in $SC'$. Since the unrestricted databases over $\mathcal{M}$ with schema $SC'$ form a complete lattice under $\subseteq$, the Tarski fixpoint theorem tells us that $T_P^D$ has a least fixpoint, which we denote by $\textsc{Lfp}(T_P^D)$. We now define:

**Definition 2.8.3.** *Designate from among the relation names in $SC'$ an answer relation $A$, of arity $k$. Then $P$ expresses the $k$-ary unrestricted query over $\mathcal{M}$ that maps each unrestricted database $D$ over $\mathcal{M}$ to* $\textsc{Lfp}(T_P^D)(A)$.

Even when $\mathcal{M}$ admits quantifier elimination, the unrestricted query expressed by a DATALOG program need not be closed, as illustrated by the following example.

*Example 2.8.1.* Consider the following DATALOG program $P$, expressing the transitive closure of binary relation $R$:

$$A(x,y) \text{ :- } R(x,y)$$
$$A(x,y) \text{ :- } R(x,z), A(z,y)\ .$$

Consider the semi-algebraic database $D$ where $D(R)$ is defined by the real polynomial constraint $y = 2 \cdot x$. In the least fixpoint of $T_P^D$, relation $A$ equals the unrestricted relation $\{(x,y) \mid \exists i \in \mathbb{N} : y = 2^i \cdot x\}$. This relation is not semi-algebraic, although $D$ is. Hence, the unrestricted query expressed by $P$ is not closed over real polynomial constraints.

The situation is not always as bad as in the above example. For example, we will see shortly that for dense order constraints over the rationals, DATALOG programs do express unrestricted queries that are closed.

Let us next define the constraint query expressed by a DATALOG program. Thereto, we first observe that DATALOG rules can be thought of as conjunctive queries. Consider a rule

$$\rho: \quad \alpha \text{ :- } \beta_1, \ldots, \beta_\ell\ ,$$

where without loss of generality we may assume that $\alpha$ is of the form $H(\bar{x})$, where $\bar{x}$ is a tuple of variables (rather than terms in general). We can think of this rule as being the conjunctive query

$$\rho(\bar{x}) = \exists \bar{y}\, (\beta_1 \wedge \cdots \wedge \beta_\ell)\ ,$$

where $\bar{y}$ are the variables occurring in the rule that do not occur in $\bar{x}$. Note that this is a conjunctive query over the combined schema $(SC, SC')$. The *constraint query expressed by a rule* is simply the constraint query expressed by the rule when we think of it as a conjunctive query.

Now given a DATALOG program $P$ with intentional schema $SC'$, and a constraint database $D$, we define a sequence of constraint databases with schema $SC'$, called the *stages of $P$ on $D$*. Stage 0, denoted by $P^0(D)$, is given by

$P^0(D)(H) = \emptyset$ for each $H$ in $SC'$. For $i > 0$, and $H$ in $SC'$, $P^i(D)(H)$ is defined as the union of the results of the constraint queries expressed by all rules having $H$ in their head, applied to the constraint database $(D, P^{i-1}(D))$.

We now define:

**Definition 2.8.4.** *Again designate an answer relation $A$ in $SC'$, of arity $k$. Then $P$ expresses the $k$-ary constraint query over $\mathcal{M}$ that maps each constraint database $D$ over $\mathcal{M}$ to $P^n(D)(A)$, where $n$ is such that $P^n(D) = P^{n+1}(D)$. If such an $n$ does not exist, we say that $P$* does not terminate on $D$*, and in this case the constraint query is not defined on $D$.*

The following example illustrates non-termination:

*Example 2.8.2.* The transitive closure program $P$ from Example 2.8.1 does not terminate on the constraint database $D$ given by $D(R) = \{y = 2 \cdot x\}$. Indeed, for every $i$, relation $A$ in stage $P^i(D)$ contains the constraint tuples $y = 2^j \cdot x$ for $j = 1, \ldots, i$.

Termination is in fact the one and only problem one faces when trying to use DATALOG as a constraint query language. More specifically, we have:

**Proposition 2.8.5.** *Suppose $P$ terminates on every constraint database, i.e., the constraint query $Q_P$ expressed by $P$ is total. Then the unrestricted query expressed by $P$ is closed, and is represented by $Q_P$.*

*Proof.* Assume $P^n(D) = P^{n+1}(D)$, and let $D_0$ be the unrestricted database defined by $D$. It suffices to observe that $P^n(D)$ actually defines $\text{LFP}(T_P^{D_0})$. □

We conclude this section with a positive result for the case of the rationals with order:

**Theorem 2.8.6.** *Let $\mathcal{M}$ be dense order constraints over the rationals, as defined in Example 2.3.1. Then every* DATALOG *program terminates on every constraint database.*

*Proof.* Let $P$ be a program and $D$ a constraint database with dense order constraints over the rationals. Every constraint relation in every stage of $P$ on $D$ consists of constraint tuples that involve only constants occurring in $D$ itself. Indeed, when eliminating quantifiers over the rationals with order, we never need to introduce new constants. Now the crucial point is that for any fixed arity $k$ and any fixed set $C$ of rationals, there are only a finite number of possible constraint $k$-tuples using only constants in $C$. Indeed, such a constraint tuple can only express a conjunction of comparisons among $k$ fixed variables $x_1, \ldots, x_k$, and of comparisons of these variables with constants in $C$. Clearly there are only a finite number of possible such "configurations." (This is discussed in more detail in Section 7.4.) □

As a matter of fact, one can design an evaluation mechanism for DATALOG programs $P$ over the rationals with order that computes a representation of the fixpoint of $P$ on a constraint database $D$ in time polynomial in the size of $D$.

### 2.8.1 Adding negation

DATALOG does not have negation. Indeed, we defined the constraint query expressed by a DATALOG program as an iteration of conjunctive queries, and conjunctive queries are a negationless fragment of the relational calculus. In this brief addendum we indicate how some form of negation can be added to DATALOG. The techniques we will mention here are not at all peculiar to constraint query languages, but are merely borrowed from the literature on deductive databases in the context of the standard relational data model.

*Semipositive* DATALOG. Recall Definition 2.8.1 of the syntax of DATALOG (with constraints). Let us change it slightly by allowing each $\beta_i$ in the body to be also a negated atomic formula, of the form $\neg R(\cdots)$, *on condition that $R$ is a relation name of the input schema $SC$.* This extension of DATALOG is called *semipositive* DATALOG.

Definition 2.8.2, of the operator $T_P^D$, can still be used literally in the case $P$ is semipositive. Because the relation names used in the heads of rules in $P$ must be from the intentional schema $SC'$, while negation in the bodies of rules is only allowed before relation names from the input schema $SC$, which is disjoint from $SC'$, the operator $T_P^D$ is still monotone. Hence, Definition 2.8.3 can still be used literally as well to define the unrestricted query expressed by a semipositive DATALOG program.

*Example 2.8.3.* The following variation of the program from Example 2.8.1 is a semipositive DATALOG program expressing the transitive closure of the *complement* of binary relation $R$:

$$A(x,y) \text{ :- } \neg R(x,y)$$
$$A(x,y) \text{ :- } \neg R(x,z), A(z,y)\ .$$

To define the constraint query expressed by a DATALOG program, in Definition 2.8.4, we iterated conjunctive queries obtained from the rules. We can do something similar for semipositive programs. All we have to do is view the negated atomic formulas $\neg R(\cdots)$ in the bodies of the rules as referring to "negated relations" $R^{neg}$. Formally, we expand the input schema $SC$ with relation names $R^{neg}$ for each $R \in SC$. Given an input constraint database $D$, we expand it correspondingly by letting $D(R^{neg})$ to be a constraint relation defining the complement of the unrestricted relation defined by the original constraint relation $D(R)$. If we now replace each negated atomic formula $\neg R(\cdots)$ in the program by $R^{neg}(\cdots)$, we are back to the case of DATALOG

without negation. In particular, we can now define the stages of a semipositive program $P$ on constraint database $D$, and define the constraint query expressed by $P$ as in Definition 2.8.4 whenever the stages converge on $D$.

The property formulated for DATALOG programs in Proposition 2.8.5, to the effect that the termination of the constraint query corresponds to closure of the unrestricted query expressed by a program, carries over to semipositive programs effortlessly. We thus see that semipositive DATALOG is only a very mild extension of DATALOG.

*Stratified negation.* An easy way to get even more out of semipositive DATALOG is to use a sequence $P_1, \ldots, P_j$ of semipositive programs, such that for each $i < j$, the input schema as well as the intentional schema of $P_i$ are part of the input schema for $P_j$. This guarantees that we can compose: we can apply $P_2$ to the result relations of $P_1$, then $P_3$ to those of $P_2$, and so on. Such a sequence of semipositive programs is called a DATALOG *program with stratified negation*. Program $P_i$ is called the *ith stratum*.

*Example 2.8.4.* The following DATALOG program with stratified negation has two strata, and expresses the transitive closure of the complement of the transitive closure of the complement of relation $R$:

$$\begin{aligned} A(x,y) &\text{ :- } \neg R(x,y) \\ A(x,y) &\text{ :- } \neg R(x,z), A(z,y) \\ B(x,y) &\text{ :- } \neg A(x,y) \\ B(x,y) &\text{ :- } \neg A(x,z), B(z,y)\ . \end{aligned}$$

*Inflationary negation.* A much more drastic way to add negation to DATALOG is to allow negation of *any* atomic formula in the body – so, in particular, to allow negation of intentional relations also (formally, atomic formulas of the form $R(\cdots)$ with $R$ a relation name in $SC'$ rather than $SC$). The $T_D^P$ operator as defined by Definition 2.8.2 is then no longer monotone, however, and its least fixpoint is no longer guaranteed to exist.

A rather crude way out of this semantic problem is to use an *inflationary* semantics. In Definition 2.8.2, we no longer define $T_P^D(D')(H)$ to consist of all tuples obtained from a valuation of some rule in $P$ in $D'$, but rather, we define $T_P^D(D')(H)$ as the *union* of these tuples together with the original relation $D'(H)$. This inflationary operator is thus forced to be monotone.

In order to compute the constraint query expressed by a DATALOG program with arbitrary negation, corresponding with the inflationary semantics, we must make two changes to how the stages $P^i(D)$ are computed, compared to what we said for the case of semipositive programs:

1. In the case of semipositive programs, all we had to do was to expand the input constraint database $D$ with its "negated version" only once, prior to the computation of the stages. Because now also intentional relations are allowed to be negated, we must now do the same after every stage also with the result $P^{i-1}(D)$ of the previous stage.

2. In accordance with the inflationary semantics, we no longer define the value of an intentional constraint relation $H$, at a stage $P^i(D)$, simply as the results of all rules having $H$ in their head, applied to the previous stage $P^{-1}(D)$. Instead, we take the *union* of these results with $P^{i-1}(D)(H)$, i.e., with what we already had for $H$ from the previous stage.

Termination of this computation by stages now again corresponds to closure of the least fixpoint of the inflationary operator $T_D^P$.

Although DATALOG programs with unrestricted, inflationary negation, as a constraint query language, is in general not very practical to work with, some nice positive theoretical results have been obtained for this language. They are reported in Chapters 4 and 7.

## 2.9 Bibliographic Notes

The seminal paper, introducing the idea of constraint databases, and showing that constraint query languages can be computationally feasible for various constraint structures, is by Kanellakis, Kuper, and Revesz [KKR90, KKR95].

There are many good expositions on logic; Enderton [End72] and Ebbinghaus, Flum, and Thomas [EFT94] are two of them. The expositions by Chang and Keisler [CK90] and Hodges [Hod93] on first-order model theory include discussions on quantifier elimination. Extensive background on relational database theory and deductive databases, including DATALOG and negation, is provided by Abiteboul, Hull, and Vianu [AHV95].

Tarski's famous decision procedure for the reals [Tar51] and its ramifications are nicely exposed by van den Dries [van88]. A relatively recent state of the art in algorithms for the theory of the reals is in Renegar's article [Ren92], as well as in the collection by Caviness and Johnson [CJ98]. The best algorithms at the time of this writing are those of Basu [Bas96].

For the reals with addition only (linear constraints), there is a particularly simple (though generally inefficient) algorithm to eliminate quantifiers [HU79], and the (generally intractable) complexity of the associated decision problem has been pinpointed exactly [Ber80]. In the case of dense order (or even simpler, just equality constraints), generally efficient algorithms are known [FG77, KKR95].

The expositions by Bochnak, Coste, and Roy [BCR98] and Benedetti and Risler [BR90] on real algebraic geometry provide extensive discussions of the properties of real semi-algebraic sets. The recent authorative book of van den Dries [vdD98] provides an excellent exposition of the properties of o-minimal structures.

As for credits to some of the more specific results described in this chapter: Theorem 2.5.1 is by Gyssens, Van den Bussche, and Van Gucht [GVdBVG99]. Corollary 2.6.7 and Proposition 2.6.8 are by Grumbach and Su [GS97a].

Corollary 2.7.7 is by Ibarra and Su [IS97], who actually give credit for it to M. Vardi. (The subtle point discussed right after the Corollary was not addressed by Ibarra and Su.)