Finitely Representable Nested Relations

E. Bertino B. Catania
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico 39/41
20135 Milano, Italy
Email: {bertino,catania}@dsi.unimi.it

Limsoon Wong
Kent Ridge Digital Laboratories
Heng Mui Keng Terrace
Singapore 119613
Email: limsoon@krdl.org.sg

Abstract

Advanced temporal and spatial applications require both the representation of complex objects and the ability to finitely represent infinite relations. Representing such data requires combining the constraint relational model (allowing finite representation of infinite information) and either the nested relational or the object-oriented model (allowing representation of complex objects). In this paper, we extend the nested relational calculus to deal with finitely representable relations. The aim of the language we propose, called fr$\mathcal{NRC}$, is to provide the right formal foundations to analyze nested constraint query languages, overcoming most limitations of already existing languages. As an example of the theoretical foundations of fr$\mathcal{NRC}$, we show that it is effectively computable and has NC data complexity. Moreover, fr$\mathcal{NRC}$ queries are independent of the depth of set nesting in data generated by intermediate computations.

Keywords: Databases, nested relational calculus, constraints.

1 Introduction

The need for sophisticated functionalities has lead to the evolution of database theory, requiring the definition of appropriate data models. In this respect, at least two important research directions have been devised: the first is the definition of complex object models [1, 7], the second is the definition of constraint models, using mathematical constraints to finitely represent infinite information [13].

Several approaches have been proposed to model complex data by using *finitely representable relations*. By finitely representable nested relations we mean relations that are nested and such that the used sets can be either finite, as in the traditional nested relational model, or infinite but finitely representable, as in the constraint relational model. Most of the proposed languages model sets up to a given height of nesting [18]. Others do not have this restriction but are defined only for specific theories [9]. For others, as $\mathcal{L}yri\mathcal{C}$ [5], the definition of a formal basis, supporting the definition and the analysis of relevant language properties, has been left to future work.

The aim of this paper is the definition of a model and a query language for finitely representable nested relations, overcoming some limitations of the previous proposals. Our language is obtained by extending $\mathcal{NRC}$ [20] to deal with possibly infinite relations, finitely representable by using a decidable logical theory admitting variable elimination (this is a necessary and sufficient condition to the definition of constraint query languages [10]), and is called fr$\mathcal{NRC}$. $\mathcal{NRC}$ is similar to the well-known comprehension mechanism in functional programming and its formulation is based on structural recursion [7] and on monads [19]. $\mathcal{NRC}$ has been proved equivalent to most nested relational languages presented before. The choice of this language is motivated by the fact that the formal semantics assigned to $\mathcal{NRC}$ and the structural recursion on which it is based allow us to prove several results about fr$\mathcal{NRC}$ in a simple way. To simplify the discussion, in this paper, fr$\mathcal{NRC}$ is defined by considering the real polynomial constraint theory (REAL). However, starting from results presented in [10], it is simple to show that any other theory admitting quantifier elimination can be easily modeled using the same framework.

One of the main results about this language is that fr$\mathcal{NRC}$, as $\mathcal{NRC}$, has the *conservative extension property*. This means that, when input and output are restricted to deal with a specific degree of nesting, any higher degree of nesting generated by the computation is useless [20]. Thus, when input and output relations represent constraint flat relations, fr$\mathcal{NRC}$ expressions can be mapped into first-order logic extended with the considered logical theory. Conservative extension is a very important property for optimization purposes, since it guarantees that complex computations generating higher degree of nesting can be automatically simplified, generating new expressions equivalent to the original ones but more efficient. Giving a constructive proof, we also prove that fr$\mathcal{NRC}$ is effectively computable. For the sake of simplicity, the proof is provided for REAL but can be easily extended to deal with any other theory admitting quantifier elimination. The same proof shows that the language has NC data complexity. The proposed proof, that can be applied to other languages as well (for example to $\mathcal{L}yri\mathcal{C}$ [5]), clearly shows which are the main issues arising in the compilation of nested constraint query languages onto flat constraint query languages.

2 Finitely Representable Nested Relational Calculus

Types In the traditional constraint query setting [13], a relation is an infinite set of tuples taking values from a given domain, as long as the set is finitely representable by a finite number of constraints, expressed using a decidable logical theory [10, 13, 16]. We extend this paradigm to sets that can be nested to an arbitrary depth. To this purpose, since we assume to deal with REAL, we allow infinite sets of tuples of reals to appear at any depth in a nested relation.[1] However, we do not allow a nested set to have an infinite number of such infinite sets as its elements, to guarantee effective computability and low data complexity. To be precise, the types that we want to consider are: $s ::= \mathbb{R} \mid s_1 \times \cdots \times s_n \mid \{s\} \mid \{_{fr}\mathbb{R} \times \cdots \times \mathbb{R}\}$. The type $\mathbb{R}$ contains all the real numbers. The type $s_1 \times \cdots \times s_n$ contains n-ary tuples whose components have types s_1, ..., s_n respectively. The type $\{s\}$ represents sets of finite cardinality whose elements are objects of type s. The type $\{_{fr}s\}$ represents sets of (possibly) infinite cardinality whose elements are objects of type s, where s is a type of the form $\mathbb{R} \times \cdots \times \mathbb{R}$. We also require each set in $\{_{fr}s\}$ to be finitely representable in the sense of [10, 13, 16]. For convenience, we also introduce a 'type' $\mathbb{B}$ to stand for Booleans. However, for economy, we use the number 0 to stand for *false* and the number 1 to stand for *true*.

Expressions To express queries over our finitely representable nested relations, we extend the nested relational calculus $\mathcal{NRC}$ defined in [7, 20]. We call the extended calculus fr$\mathcal{NRC}$, standing for *finitely representable* $\mathcal{NRC}$. The syntax and typing rules of fr$\mathcal{NRC}$ are presented in Figure 1. We often omit the type superscripts as they can be inferred. An expression e having free variables $\vec{x}$ is interpreted as a function $f(\vec{x}) = e$, which given input $\vec{O}$ produces $e[\vec{O}/\vec{x}]$ as its output. An expression e with no free variable can be regarded as a constant function $f(\vec{x}) = e$ that returns e on all input $\vec{x}$. In the following, we present the language incrementally.

NRC. $\mathcal{NRC}$ is equivalent to the usual nested relational algebra [1, 7]. The semantics of the $\mathcal{NRC}$ rules is as follows. Variables x^s are available for each type s. Every real number c is available. The operations for

[1] Even if in this paper we consider real numbers, other domains can be easily considered.

$\mathcal{NRC}$ rules

$$\frac{}{x^s : s} \qquad \frac{}{c : \mathbb{R}}$$

$$\frac{e : s_1 \times \cdots \times s_n}{\pi_i \ e : s_i} \qquad \frac{e_1 : s_1 \quad \cdots \quad e_n : s_n}{(e_1, \ldots, e_n) : s_1 \times \cdots \times s_n}$$

$$\frac{}{\{\}^s : \{s\}} \qquad \frac{e : s}{\{e\} : \{s\}} \qquad \frac{e_1 : \{s\} \quad e_2 : \{s\}}{e_1 \cup e_2 : \{s\}} \qquad \frac{e_1 : \{t\} \quad e_2 : \{s\}}{\bigcup\{e_1 \mid x^s \in e_2\} : \{t\}}$$

$$\frac{e_1 : \mathbb{R} \quad e_2 : \mathbb{R}}{e_1 = e_2 : \mathbb{B}} \qquad \frac{e_1 : \mathbb{B} \quad e_2 : s \quad e_3 : s}{\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 : s} \qquad \frac{e : \{\mathbb{R}\}}{\mathit{empty}\ e : \mathbb{B}}$$

Rules for finitely representable sets and constraints

$$\frac{}{\{_{fr}\}^s : \{_{fr}s\}} \qquad \frac{e : s}{\{_{fr}e\} : \{_{fr}s\}} \qquad \frac{e_1 : \{_{fr}s\} \quad e_2 : \{_{fr}s\}}{e_1 \cup_{fr} e_2 : \{_{fr}s\}} \qquad \frac{e_1 : \{_{fr}s_1\} \quad e_2 : \{_{fr}s_2\}}{\bigcup\{_{fr}e_1 \mid x^{s_2} \in_{fr} e_2\} : \{_{fr}s_1\}}$$

$$\frac{e_1 : \mathbb{R} \quad e_2 : \mathbb{R}}{e_1 + e_2 : \mathbb{R}} \qquad \frac{e_1 : \mathbb{R} \quad e_2 : \mathbb{R}}{e_1 - e_2 : \mathbb{R}} \qquad \frac{e_1 : \mathbb{R} \quad e_2 : \mathbb{R}}{e_1 \cdot e_2 : \mathbb{R}} \qquad \frac{e_1 : \mathbb{R} \quad e_2 : \mathbb{R}}{e_1 \div e_2 : \mathbb{R}}$$

$$\frac{}{R : \{_{fr}\mathbb{R}\}} \qquad \frac{e : \{_{fr}\mathbb{R}\}}{\mathit{empty}_{fr}\ e : \mathbb{B}}$$

Rule for integrating sets and finitely representable sets

$$\frac{e_1 : \{_{fr}s_1\} \quad e_2 : \{s_2\}}{\bigcup\{_{fr}e_1 \mid x^{s_2} \in e_2\} : \{_{fr}s_1\}}$$

Figure 1: fr$\mathcal{NRC}$ syntax and typing rules

tuples are standard. Namely, $(e_1, \ldots, e_n)$ forms an n-tuple whose i component is e_i and $\pi_i\ e$ returns the i component of the n-tuple e. $\{\}$ forms the empty set. $\{e\}$ forms the singleton set containing e. $e_1 \cup e_2$ unions the two sets e_1 and e_2. $\bigcup\{e_1 \mid x \in e_2\}$ maps the function $f(x) = e_1$ over all elements in e_2 and then returns their union; thus if e_2 is the set $\{o_1, \ldots, o_n\}$, the result of this operation is $f(o_1) \cup \cdots \cup f(o_n)$. For example, $\bigcup\{\{(x,x)\} \mid x \in \{1,2\}\}$ evaluates to $\{(1,1),(2,2)\}$. The operations for Booleans are also quite typical, with the understanding that *true* is represented by 1 and *false* is represented by 0. $e_1 = e_2$ returns *true* if e_1 and e_2 have the same value and returns *false* otherwise. *empty* e returns *true* if e is an empty set and returns *false* otherwise. Finally, *if* e_1 *then* e_2 *else* e_3 evaluates to e_2 if e_1 is *true* and evaluates to e_3 if e_1 is *false*; it is undefined otherwise.

Finitely representable relations and constraints. We add constructs analogous to the finite set constructs of $\mathcal{NRC}$ to manipulate finitely representable sets and constructs for arithmetics to express real polynomial constraints.[2] The semantics of the first four rules is analogous to those of finite sets, except that each operation does not return a set but a finitely representable set. The four arithmetic operations have the usual interpretation. $\mathit{empty}_{fr}\ e$ tests if the finitely representable set e of reals is empty. Finally, the symbol R denotes the infinite (but finitely representable) set of all real numbers. It is the presence of this symbol R that allows to express unbounded quantification. For example, given a polynomial $f(x)$, we can express its set of roots easily: $\bigcup\{_{fr}\ \mathit{if}\ f(x) = 0\ \mathit{then}\ \{_{fr}x\}\ \mathit{else}\ \{_{fr}\} \mid x \in_{fr} R\}$. Similarly, we can express the usual linear order on the reals, because the formula $\exists z.(z \neq 0) \wedge (y - x = z^2)$, which holds iff $x < y$, is expressible as $\mathit{not}(\mathit{empty}_{fr}(\bigcup\{_{fr}\ \mathit{if}\ \mathit{not}(z=0)\ \mathit{then}\ \mathit{if}\ y - x = z \cdot z\ \mathit{then}\ \{_{fr}\ z\}\ \mathit{else}\ \{_{fr}\}\ \mathit{else}\ \{_{fr}\} \mid z \in_{fr} R\}))$, with *not* implemented in the obvious way.

[2]Note that different sets of rules can be inserted to represent different logical theories (for example, dense-order).

Integrating sets and finitely representable sets. The constructs described above let us manipulate finite sets and finitely representable sets independently. In order for these two kinds of sets to interact, we need one more construct; see Figure 1. This construct let us convert a finite set of real tuples into a finitely representable one. The semantics of $\bigcup\{_{fr} e_1 \mid x \in e_2\}$ is to apply the function $f(x) = e_1$ to each element of e_2 and then return their union as a finitely representable set. That is, if e_2 is the set $\{o_1, \ldots, o_n\}$, then it produces the finitely representable set $f(o_1) \cup_{fr} \cdots \cup_{fr} f(o_n)$. For example, the conversion of a finite set e of real tuples to a finitely representable one can be expressed as $\bigcup\{_{fr}\{_{fr} x\} \mid x \in e\}$.

Before we study fr$\mathcal{NRC}$ properties, let us briefly introduce a nice shorthand, based on the comprehension notation [6, 19], for writing fr$\mathcal{NRC}$ queries. Recall from [6, 7, 20] that the comprehension $\{e \mid A_1, \ldots, A_n\}$, where each A_i either has the form $x_i \in e_i$ or is an expression e_i of type $\mathbb{B}$, has a direct correspondent in $\mathcal{NRC}$ that is given by recursively applying the following equations:

- $\{e \mid x_i \in e_i, \ldots\} = \bigcup\{\{e \mid \ldots\} \mid x_i \in e_i\}$
- $\{e \mid e_i, \ldots\} = \mathit{if}\ e_i\ \mathit{then}\ \{e \mid \ldots\}\ \mathit{else}\ \{\}$

The comprehension notation is very user-friendly. For example, it allows us to write $\{(x, y) \mid x \in e_1,\ y \in e_2\}$ for the Cartesian product of e_1 and e_2 instead of the clumsier $\bigcup\{\bigcup\{\{(x, y)\} \mid y \in e_2\} \mid x \in e_1\}$.

The comprehension notation can be extended naturally to all fr$\mathcal{NRC}$ expressions. We can interpret the comprehension $\{_{fr} e \mid A_1, \ldots, A_n\}$, where each A_i either has the form $x_i \in e_i$ or has the form $x_i \in_{fr} e_i$ or is an expression e_i of type $\mathbb{B}$, as an expression of fr$\mathcal{NRC}$ by recursively applying the following equations:

- $\{_{fr} e \mid x_i \in e_i, \ldots\} = \bigcup\{_{fr}\{_{fr} e \mid \ldots\} \mid x_i \in e_i\}$
- $\{_{fr} e \mid x_i \in_{fr} e_i, \ldots\} = \bigcup\{_{fr}\{_{fr} e \mid \ldots\} \mid x_i \in_{fr} e_i\}$
- $\{_{fr} e \mid e_i, \ldots\} = \mathit{if}\ e_i\ \mathit{then}\ \{_{fr} e \mid \ldots\}\ \mathit{else}\ \{_{fr}\}$

For example, the query to find the roots of $f(x)$ becomes $\{_{fr} x \mid x \in_{fr} R,\ f(x) = 0\}$. Similarly, the query to test if $x < y$ becomes $\mathit{not}(\mathit{empty}_{fr}(\{_{fr} z \mid z \in_{fr} R,\ \mathit{not}(z = 0),\ y - x = z \cdot z\}))$.

In addition to comprehension, we also find it convenient to use pattern matching. For example, we write $\{(x, z) \mid (x, y) \in e_1,\ (y', z) \in e_2,\ y = y'\}$ for relational composition instead of the more official $\{(\pi_1\ xy, \pi_2\ yz) \mid xy \in e_1,\ yz \in e_2,\ \pi_2\ xy = \pi_1\ yz\}$.

We should also remark that while fr$\mathcal{NRC}$ provides only equality test on $\mathbb{R}$ and emptiness tests on $\{\mathbb{R}\}$ and $\{_{fr}\mathbb{R}\}$, these operations can be lifted to every type s using fr$\mathcal{NRC}$ as the ambient language; see [20]. Similarly, commonly used operations such as set membership, set subset tests, set difference, and set intersection are expressible at all types in fr$\mathcal{NRC}$.

3 Conservative Extension Property

Given a type s, the height of s is defined as the depth of nesting of set brackets $\{\cdot\}$ and $\{_{fr}\cdot\}$ in s. Given an expression e of fr$\mathcal{NRC}$, the height of e is defined as the maximum height of all the types that appear in e's typing derivation. For example, $\{(x, y) \mid x \in e_1,\ y \in e_2\}$ has height 1 if both e_1 and e_2 have height 1. On the other hand, $\{(x, \{_{fr} z \mid z \in_{fr} R, z < x\}) \mid x \in e\}$ have height 2 if e has height 1.

Definition 3.1 A language $\mathcal{L}$ is said to have the **conservative extension property** if every function $f : s_1 \to s_2$ that is expressible in $\mathcal{L}$ can be expressed using an expression of height no more than the maximum between the heights of s_1 and s_2. $\square$

$\pi_i(e_1, \ldots, e_n) \rightsquigarrow e_i$
$\mathit{if}\ \mathit{true}\ \mathit{then}\ e_1\ \mathit{else}\ e_2 \rightsquigarrow e_1$
$\mathit{if}\ \mathit{false}\ \mathit{then}\ e_1\ \mathit{else}\ e_2 \rightsquigarrow e_2$
$\{\} \cup e \rightsquigarrow e$
$e \cup \{\} \rightsquigarrow e$
$\mathit{empty}(e_1 \cup \cdots \cup e_n) \rightsquigarrow \mathit{false}$, if some e_i has the form $\{e\}$
$\mathit{empty}(e_1 \cup \cdots \cup e_n) \rightsquigarrow \mathit{true}$, if every e_i has the form $\{\}$
$\mathit{empty}_{fr}(e_1 \cup_{fr} \cdots \cup_{fr} e_n) \rightsquigarrow \mathit{false}$, if some e_i has the form $\{_{fr} e\}$
$\mathit{empty}_{fr}(e_1 \cup_{fr} \cdots \cup_{fr} e_n) \rightsquigarrow \mathit{true}$, if every e_i has the form $\{_{fr}\}$
$\bigcup\{e \mid x \in \{\}\} \rightsquigarrow \{\}$
$\bigcup\{e_1 \mid x \in \{e_2\}\} \rightsquigarrow e_1[e_2/x]$
$\bigcup\{e_1 \mid x \in e_2 \cup e_3\} \rightsquigarrow \bigcup\{e_1 \mid x \in e_2\} \ \cup\ \bigcup\{e_1 \mid x \in e_3\}$
$\bigcup\{e_1 \mid x \in \bigcup\{e_2 \mid y \in e_3\} \rightsquigarrow \bigcup\{\bigcup\{e_1 \mid x \in e_2\} \mid y \in e_3\}$
$\bigcup\{e_1 \mid x \in \mathit{if}\ e_2\ \mathit{then}\ e_3\ \mathit{else}\ e_4\} \rightsquigarrow \mathit{if}\ e_2\ \mathit{then}\ \bigcup\{e_1 \mid x \in e_3\}\ \mathit{else}\ \bigcup\{e_1 \mid x \in e_4\}$
$\bigcup\{_{fr} e \mid x \in_{fr} \{_{fr}\}\} \rightsquigarrow \{_{fr}\}$
$\bigcup\{_{fr} e_1 \mid x \in_{fr} \{_{fr} e_2\}\} \rightsquigarrow e_1[e_2/x]$
$\bigcup\{_{fr} e_1 \mid x \in_{fr} e_2 \cup_{fr} e_3\} \rightsquigarrow \bigcup\{_{fr} e_1 \mid x \in_{fr} e_2\} \cup_{fr} \bigcup\{_{fr} e_1 \mid x \in_{fr} e_3\}$
$\bigcup\{_{fr} e_1 \mid x \in_{fr} \bigcup\{_{fr} e_2 \mid y \in_{fr} e_3\}\} \rightsquigarrow \bigcup\{_{fr} \bigcup\{_{fr} e_1 \mid x \in_{fr} e_2\} \mid y \in_{fr} e_3\}$
$\bigcup\{_{fr} e_1 \mid x \in_{fr} \mathit{if}\ e_2\ \mathit{then}\ e_3\ \mathit{else}\ e_4\ \} \rightsquigarrow \mathit{if}\ e_2\ \mathit{then}\ \bigcup\{_{fr} e_1 \mid x \in_{fr} e_3\}\ \mathit{else}\ \bigcup\{_{fr} e_1 \mid x \in_{fr} e_4\}$
$\bigcup\{_{fr} e \mid x \in \{\}\} \rightsquigarrow \{\}$
$\bigcup\{_{fr} e_1 \mid x \in \{e_2\}\} \rightsquigarrow e_1[e_2/x]$
$\bigcup\{_{fr} e_1 \mid x \in e_1 \cup e_2\} \rightsquigarrow \bigcup\{_{fr} e_1 \mid x \in e_2\} \cup \bigcup\{_{fr} e_1 \mid x \in e_3\}$
$\bigcup\{_{fr} e_1 \mid x \in \bigcup\{e_2 \mid y \in e_3\}\} \rightsquigarrow \bigcup\{_{fr} \bigcup\{_{fr} e_1 \mid x \in e_2\} \mid y \in e_3\}$
$\bigcup\{_{fr} e_1 \mid x \in_{fr} \bigcup\{_{fr} e_2 \mid y \in e_3\}\} \rightsquigarrow \bigcup\{_{fr} \bigcup\{_{fr} e_1 \mid x \in_{fr} e_2\} \mid y \in e_3\}$
$\bigcup\{_{fr}\ e_1 \mid x \in \mathit{if}\ e_2\ \mathit{then}\ e_3\ \mathit{else}\ e_4\} \rightsquigarrow \mathit{if}\ e_1\ \mathit{then}\ \bigcup\{_{fr} e_1 \mid x \in e_3\}\ \mathit{else}\ \bigcup\{_{fr} e_1 \mid x \in e_4\}$

Table 1: Rewriting rules

We now prove that fr$\mathcal{NRC}$ has the conservative extension property, just like $\mathcal{NRC}$ [20]. To this purpose, as in [20], we first provide some rewriting rules, reducing set height. Then, we show that the normal forms induced by such rules have height no more than that of their free variables (i.e., their input variables).

Table 1 shows the rewriting rules that we want to use. Those for $\mathcal{NRC}$ are taken from [20]. As usual, we assume that bound variables are renamed to avoid capture and that $e_1[e_2/x]$ denotes the expression obtained by replacing all free occurrences of x in e_1 by e_2.

It is readily verified that the proposed rewriting rules are sound. That is, expressions obtained from e_1 by rewriting are semantically equivalent to e_1. Furthermore, using a straightforward adaptation of the termination measure given in [20], we can prove that the rewriting system presented in Table 1 is guaranteed to stop no matter in what order these rules are applied. We therefore say that the system is *strongly normalizing.*

Proposition 3.2 *If $e_1 \rightsquigarrow e_2$, then $e_1 = e_2$.*[3] *Moreover, the rewriting system presented in Table 1 is strongly normalizing.* □

The following result follows from the application of a simple induction on the structure of expressions.

Proposition 3.3 *Let $e : s$ be a fr$\mathcal{NRC}$ expression, having free variables $x_1 : s_1$, ..., $x_n : s_n$ such that e is a normal form with respect to the above rewriting system. Then the height of e is at most the maximum of the heights of s, s_1, ..., s_n.* □

Combining Propositions 3.2 and 3.3, we conclude the following.

Theorem 3.4 *fr$\mathcal{NRC}$ has the conservative extension property.* □

[3] The symbol $=$ denotes semantic equivalence.

Paredaens and Van Gucht gave a translation for mapping nested relational algebra expressions having flat relations as input to an equivalent expression in first-order logic with bounded quantification [17]. This translation can be easily adapted to provide a translation for mapping fr$\mathcal{NRC}$ expressions of height 1 to first-order logic with polynomial constraints. Next result follows from this and Theorem 3.4.

Corollary 3.5 *If $f : s_1 \to s_2$ is a function expressible in fr$\mathcal{NRC}$ and s_1 and s_2 have height 1, then f is expressible in first-order logic with polynomial constraints.* □

Thus all functions $f : s_1 \to s_2$ in fr$\mathcal{NRC}$, with s_1 and s_2 of height 1, are effectively computable by compiling into constraint query languages such as those proposed in [10, 13, 16]. As a consequence, we can make use of well-known results [3, etc.] on constraint query languages to analyze the expressiveness of fr$\mathcal{NRC}$ with respect to such functions. For example, an immediate consequence is that fr$\mathcal{NRC}$ cannot express parity test, connectivity test, and transitive closure.

We can also use the above "compilation procedure" to study the expressive power of fr$\mathcal{NRC}$ on functions whose types have heights exceeding 1. We borrow an example from [15] for illustration. A set of sets $O = \{O_1, \ldots, O_n\} : \{\{\mathbb{R}\}\}$ is said to have a family of distinct representatives iff it is possible to pick an element x_i from each O_i such that $x_i \neq x_j$ whenever $i \neq j$. It is known from [15] that $\mathcal{NRC}$ cannot test if a set has distinct representatives. We show it cannot be expressed in fr$\mathcal{NRC}$ either.

Corollary 3.6 *fr$\mathcal{NRC}$ cannot test if a set of sets has distinct representatives.*
Proof. fr$\mathcal{NRC}$ cannot express parity test. It follows that it cannot test if a chain has an even number of nodes. Let a set $X_m = \{(x_1, x_2), \ldots, (x_{m-1}, x_m)\}$ be given, where $m > 2$. Then we can construct in fr$\mathcal{NRC}$ the set $S_m = \{\{x_1\}, \{x_m\}, \{x_1, x_3\}, \{x_2, x_4\}, \ldots, \{x_{m-2}, x_m\}\}$. According to [15], S_m has distinct representatives iff m is even. It follows that fr$\mathcal{NRC}$ cannot test for distinct representatives. □

4 Effective Computability and Complexity

Recall that expressions in fr$\mathcal{NRC}$ can iterate over infinite sets. An important question is whether every function expressible in fr$\mathcal{NRC}$ is computable. In the previous section, we saw that if a function in fr$\mathcal{NRC}$ has input and output of height 1, then it is computable. In this section, we lift this result to functions of all heights.

Our strategy is as follows. We find a total computable function $p_s : s \to s'$ to encode nested finitely representable sets into flat finitely representable sets. We also find a partial computable decoding function $q_s : s' \to s$ so that $q_s \circ p_s = id$. Finally, we find a translation $(\cdot)'$ that maps $f : s_1 \to s_2$ in fr$\mathcal{NRC}$ to $(f)' : s_1' \to s_2'$ in fr$\mathcal{NRC}$ such that $q_{s_2} \circ (f)' \circ p_{s_1} = f$. Note that $(f)'$ has height 1 and is thus computable.

Before we define p and q, let us first define s', the type to which s is encoded. Notice that s' always has the form $\{_{fr}\mathbb{R} \times \cdots \times \mathbb{R}\}$.

- $\mathbb{R}' = \{_{fr}\mathbb{R}\}$
- $(s_1 \times \cdots \times s_n)' = \{_{fr}t_1 \times \cdots \times t_n\}$, where $s_i' = \{_{fr}t_i\}$.
- $\{_{fr}s\}' = \{_{fr}\mathbb{R} \times s\}$
- $\{s\}' = \{_{fr}\mathbb{R} \times \mathbb{R} \times t\}$, where $s' = \{_{fr}t\}$

The encoding function $p_s : s \to s'$ is defined by induction on s. In what follows, $\vec{0}$ stands for a tuple of zeros $(0, \ldots, 0)$ having the appropriate arity. A finitely representable set is coded by tagging each element by 1 if the set is nonempty and is coded by a tuple of zeros if it is empty. A finite set is coded by tagging each element by 1 and by a unique identifier if the set is nonempty and is coded by a tuple of zeros if it is empty. More precisely:

- $p_{\mathbb{R}}(o) = \{_{fr} o\}$
- $p_{s_1 \times \cdots \times s_n}((o_1, \ldots, o_n)) = \{_{fr}(x_1, \ldots, x_n) \mid x_1 \in_{fr} p_{s_1}(o_1), \ldots, x_n \in_{fr} p_{s_n}(o_n)\}$
- $p_{\{_{fr} s\}}(O) = \{_{fr}(0, \vec{0})\}$, if O is empty. Otherwise, $p_{\{_{fr} s\}}(O) = \{_{fr}(1, x) \mid x \in_{fr} O\}$.
- $p_{\{s\}}(O) = \{_{fr}(0, 0, \vec{0})\}$, if O is empty. Otherwise, $p_{\{s\}}(O) = O_1 \cup_{fr} \cdots \cup_{fr} O_n$, if $O = \{o_1, \ldots, o_n\}$ and $O_i = \{_{fr}(1, i, x) \mid x \in_{fr} p_s(o_i)\}$. Note that we allow the i's above to be any numbers, so long as they are distinct positive integers.

We use $\bigcup\{e_1 \mid x \in_{fr} e_2\}$ to stand for the application of $f(x) = e_1$ to each element of e_2, provided the finitely representable set e_2 has finite number of elements, and return the finite union of the results. Then the comprehension notation $\{e \mid A_1, \ldots, A_n\}$ is extended to allow A_i to be of the form $x_i \in_{fr} e_i$ and the translation equations are augmented to include the equation: $\{e \mid x_i \in_{fr} e_i, \ldots\} = \bigcup\{\{e \mid \ldots\} \mid x_i \in_{fr} e_i\}$.

The decoding function $q_s : s' \to s$, which strips tags and identifiers introduced by p_s, can be defined as follows:

- $q_{\mathbb{R}}(O) = o$, if $O = \{_{fr} o\}$.
- $q_{s_1 \times \cdots \times s_n}(O) = (o_1, \ldots, o_n)$, if $o_i = q_{s_i}(\{_{fr} x_i \mid (x_1, \ldots, x_n) \in_{fr} O\})$.
- $q_{\{_{fr} s\}}(O) = \{_{fr} x \mid (1, x) \in_{fr} O\}$.
- $q_{\{s\}}(O) = \{q_s(\{_{fr} y \mid (1, j, y) \in_{fr} O,\ i = j\}) \mid (1, i, x) \in_{fr} O\}$.

It is clear that p_s and q_s are both computable, even though they cannot be expressed in fr$\mathcal{NRC}$. Moreover, using the fact that $p_s(O)$ is never empty, by induction on the structure of s we can show that q_s is inverse of p_s.

Proposition 4.1 $q_s \circ p_s = id$. □

Note that p_s is not deterministic. Let $O_1 : s'$ and $O_2 : s'$. Then we say $O_1 \sim O_2$ if $q_s(O_1) = q_s(O_2)$. That is, O_1 and O_2 are equivalent encodings of an object $O : s$. It is clear that whenever $O_1 \sim O_1'$, ..., and $O_n \sim O_n'$, then $\{_{fr}(x_1, \ldots, x_n) \mid x_1 \in_{fr} O_1, \ldots, x_n \in_{fr} O_n\} \sim \{_{fr}(x_1, \ldots, x_n) \mid x_1 \in_{fr} O_1', \ldots, x_n \in_{fr} O_n'\}$. It is also obvious that whenever $O \sim O'$, then $\{_{fr} x_i \mid (x_1, \ldots, x_n) \in_{fr} O\} \sim \{_{fr} x_i \mid (x_1, \ldots, x_n) \in_{fr} O'\}$. We can now state the following key proposition (see [4] for the proof).

Proposition 4.2 *For every function $f : s_1 \to s_2$ in fr$\mathcal{NRC}$, there is a function $(f)' : s_1' \to s_2'$ such that*

$$\begin{tikzcd}
s_1 \arrow[r, "id"] \arrow[d, "p_{s_1}"'] & s_1 \arrow[r, "f"] & s_2 \arrow[r, "id"] & s_2 \arrow[d, "p_{s_2}"] \\
s_1' \arrow[r, "\sim"'] & s_1' \arrow[u, "q_{s_1}"'] \arrow[r, "(f)'"'] & s_2' \arrow[u, "q_{s_2}"] \arrow[r, "\sim"'] & s_2'
\end{tikzcd}$$

Proof Sketch. Left and right squares commute by definitions of p_s, q_s, and $\sim$. It is then possible to construct $(f)'$ by induction on the structure of the fr$\mathcal{NRC}$ expression that defines f such that the middle square and thus the entire diagram commutes. □

Now let $f : s_1 \to s_2$ be a function in fr$\mathcal{NRC}$, where s_1 and s_2 have arbitrary nesting depths. Proposition 4.2 implies that there is a function $(f)' : s_1' \to s_2'$ in fr$\mathcal{NRC}$ such that $q_{s_2} \circ (f)' \circ p_{s_1} = f$. Since s_1' and s_2' are both of height 1, by Theorem 3.4, we can assume that $(f)'$ has height 1. Then by Corollary 3.5, we conclude that $(f)'$ is effectively computable. Since q_s and p_s are also computable, we have the very desirable result below.

Theorem 4.3 *All functions expressible in fr$\mathcal{NRC}$ are effectively computable.* $\square$

The above "compilation procedure" shows that fr$\mathcal{NRC}$ can be embedded in first-order logic with polynomial constraints, modulo the encodings p_s and q_s (thus, fr$\mathcal{NRC}$ is closed). The converse is also true. For example, a formula $\exists x.\Phi(x)$ can be expressed in fr$\mathcal{NRC}$ as $not(empty_{fr}\{_{fr}1 \mid x \in_{fr} R,\ \Phi(x)\})$. So fr$\mathcal{NRC}$ does not gain us extra expressive or computational power, compared to the usual constraint query languages. However, it gives a more natural data model and a more convenient query language, since it is no longer necessary to model data as a set of flat tables.

Results about data complexity of fr$\mathcal{NRC}$ can be obtained from results presented in Section 4 and from [13]. Consider the diagram introduced in Proposition 4.2. As f' is expressed in first-order logic extended with polynomial constraints, it follows from [13] that its data complexity is in NC. Moreover, it is simple to show that encoding and decoding functions p_s and q_s are also in NC. Thus, we obtain the following result (similar results can be obtained by considering other constraint theory).

Proposition 4.4 *fr$\mathcal{NRC}$ has data complexity in NC.* $\square$

5 Comparison with Related Work

Other approaches have been proposed to model infinite sets in constraint data models. Such approaches can be classified according to the following criteria (see Table 2):

- *Theory and underlying query language.* Different theories have been considered by different languages. The choice of a particular theory and language allows to obtain specific complexity and expressivity results. Often, less general theories are chosen only to guarantee a low data complexity (see for example $\mathcal{L}yri\mathcal{C}$). This is not the case of C-CALC, whose semantics strictly depends on the chosen theory.

- *Maximal set height.* Some proposed languages model sets up to a fixed set height. Among the proposed languages, only C-CALC and fr$\mathcal{NRC}$ in the nested relational framework, and $\mathcal{L}yri\mathcal{C}$, in the object-oriented framework, allow the representation of arbitrary complex data.

- *Data complexity.* Practical languages are usually required to have PTIME data complexity. Among the proposed languages, $Datalog^{\subset \mathbf{P}(\mathbf{Z})}$ and C-CALC have the higher complexity. The high complexity of $Datalog^{\subset \mathbf{P}(\mathbf{Z})}$ is mainly due to the fact that Datalog is the underlying language, whereas the hyper-exponential complexity of C-CALC is due to the fact that variables may range over sets, thus high-order computation is provided.

From the previous considerations, it follows that fr$\mathcal{NRC}$ overcomes some limitations of the previous proposals to model complex objects in constraint databases. Indeed, no maximum degree of nesting is assumed and different theories can be used to finitely represent relations, ensuring at the same time a low data complexity. Moreover, the formal semantics on which it is based allows one to easily analyze several interesting properties (as conservative extension) of nested constraint relational languages.

6 Concluding remarks

The paper has presented a framework to reason about nested constraint query languages. The framework can be applied to any constraint domain under which the standard first-order constraint query language works [10]. Future work includes the definition of optimization techniques for the proposed language and the

Language	Theory	Underlying query language	Max. set height	Complexity
fr$\mathcal{NRC}$	theories admitting quantifier elimination	$\mathcal{NRC}$	$n \geq 0$	= complexity of $FO(\Phi)$ (FO extended with theory Φ), if $FO(\Phi) \supseteq NC$
$\mathcal{L}yri\mathcal{C}$ [5]	linear polynomial constraints	XSQL [14]	$n \geq 0$	$\subseteq$ PTIME
$Datalog^{\subseteq \mathbf{P}(\mathbf{Z})}$ [8, 18]	set constraints on integer numbers	Datalog	2	$\subseteq$ DEXPTIME
C-CALC [9]	dense-order constraints	relational calculus for complex objects [11]	$n \geq 0$	hyper-exponential
$EGRA(\Phi)$ [2]	theories admitting quantifier elimination	constraint relational algebra [12]	2	= complexity of $FO(\Phi)$ (FO extended with theory Φ), if $FO(\Phi) \supseteq NC$

Table 2: Language comparison

integration of external functions. In particular, an interesting topic is the detection of conditions under which computability can be retained when inserting high-order signatures, defining functions on set types.

References

[1] S. Abiteboul and P. Kanellakis. Query languages for complex object databases. *SIGACT News*, 21(3):9–18, 1990.

[2] A. Belussi, E. Bertino, e B. Catania. An Extended Algebra for Constraint Databases. In *IEEE Transactions on Knowledge and Data Engineering*, 10(5):686-705, 1998. IEEE Computer Society Press, Los Alamitos, California.

[3] M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational expressive power of constraint query languages. In *Proc. of 15th ACM Symposium on Principles of Database Systems*, pages 5–16, June 1996.

[4] E. Bertino, B. Catania, and L. Wong. Finitely representable nested relations. Technical Report, Department of Computer Science, University of Milano, Italy, 1999.

[5] A. Brodsky and Y. Kornatzky. The $\mathcal{L}_{yri}\mathcal{C}$ language: querying constraint objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1995.

[6] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, March 1994.

[7] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, September 1995.

[8] J. Byon and P.Z. Revesz. DISCO: A constraint database system with sets. In *LNCS 1034: Proc. of 1st Int. CONTESSA Database Workshop, Constraint Databases and their Applications*, pages 68–83, 1995.

[9] S. Grumbach and J. Su. Dense-order constraint databases. In *Proc. ACM Symposium on Principles of Database Systems*, pages 66–77, 1995.

[10] S. Grumbach and J. Su. Finitely representable databases. *Journal of Computer and System Science*, 55(2):273-298, 1997.

[11] R. Hull and J. Su. On the expressive power of database queries with intermediate types. *Journal of Computer and System Sciences*, 43(1):219–267, August 1991.

[12] P.C. Kanellakis and D.Q. Goldin. Constraint Programming and Database Query Languages. In *LNCS 789: Proc. of the Int. Symp. on Theoretical Aspects of Computer Software*, pages 96–120, 1994.

[13] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51:25–52, 1995.

[14] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 393–402, 1992.

[15] L. Libkin and L. Wong. On representation and querying incomplete information in databases with multisets. *Information Processing Letters*, 56:209–214, November 1995.

[16] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proceedings of 13th ACM Symposium on Principles of Database Systems*, pages 279–288, May 1994.

[17] J. Paredaens and D. Van Gucht. Converting nested relational algebra expressions into flat algebra expressions. *ACM Transaction on Database Systems*, 17(1):65–93, March 1992.

[18] P.Z. Revesz. Datalog queries of set constraint databases. In *Proc. of the Int. Conf. on Database Systems*, pages 424–438, 1995.

[19] P. Wadler. Comprehending monads. *Mathematical Structures in Computer Science*, 2:461–493, 1992.

[20] L. Wong. Normal forms and conservative extension properties for query languages over collection types. *Journal of Computer and System Sciences*, 52(3):495–505, 1996.