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
BioInformatics Centre &
Real World Computing Partnership Novel Function
Institute of Systems Science Laboratory
Singapore 119597.
Email: limsoon@iss.nus.sg.

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, 5], the second is the use of constraint models, using mathematical constraints to finitely represent infinite information [8].

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 [12]. Others do not have this restriction but are defined only for specific theories. For example, the semantics assigned to C-CALC [7] makes sense only when using the dense-order theory. For others, as $\mathcal{L}yri\mathcal{C}$ [3], 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 $\mathcal{NRC}$ [14] to deal with possibly infinite relations, finitely representable by using the real polynomial constraint theory, and it 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 [5] and on monads [13]. $\mathcal{NRC}$ 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. Moreover, even if fr$\mathcal{NRC}$ has been defined for the theory of real polynomial constraints, other theories can be easily modeled in the same framework.

One of the main result 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 [14]. Thus, when input and output relations represent constraint flat relations, fr$\mathcal{NRC}$ expressions can be mapped in first-order logic extended with polynomial constraints. Note that, even if this result is a consequence of properties of $\mathcal{NRC}$ and constraint query languages, nobody has proved this result before. Moreover, giving a constructive proof, we prove that fr$\mathcal{NRC}$ is effectively computable. In particular, it is equivalent to the constraint relational calculus extended with real polynomial constraints, modulo encoding/decoding of input/output. 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 $\mathcal{L}yri\mathcal{C}$ [3]), clearly shows which are the main issues arising in the compilation of nested constraint query languages to flat constraint query languages.

2 Finitely Representable Nested Relational Calculus

Types In the traditional constraint query setting, a relation can be 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 [6, 8, 10]. We extend this paradigm to sets that can be nested to an arbitrary depth. To this purpose, we choose the theory of real polynomial inequality constraints. In particular, we allow such 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 $\{s\}$ are sets of finite cardinality whose elements are objects of type s. The type $\{_{fr}s\}$ are 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 [6, 8, 10]. 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 [5, 14]. 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.

$\mathcal{NRC}$. $\mathcal{NRC}$ is equivalent to the usual nested relational algebra [1, 5]. 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 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 would be $f(o_1) \cup \cdots \cup f(o_n)$. For example, $\bigcup\{\{(x, x)\} \mid x \in \{1, 2\}\}$ evaluates to $\{(1, 1), (2, 2)\}$. The operations

[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

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

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

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

Rules 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

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.

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

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

each element of e_2 and then returns 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 [4, 13], for writing fr$\mathcal{NRC}$ queries. Recall from [4, 5, 14] 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 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 a little bit of pattern matching, which can be removed in a straightforward manner. 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 [14]. 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. □

We now prove that fr$\mathcal{NRC}$ has the conservative extension property, just like $\mathcal{NRC}$ [14]. As in [14], a set of strongly normalizing rewriting rules that reduces set height is given. Then we show that the induced normal forms have height no more than that of their free variables (i.e., their input variables).

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

Table 1: Rewrite rules

Table 1 shows the rewrite rules that we want to use. Those for $\mathcal{NRC}$ are taken from [14]. 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 rewrite 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 [14], we can prove the following result.

Proposition 3.2 *If $e_1 \leadsto e_2$, then $e_1 = e_2$*[3]*. Moreover, the rewrite system presented in Table 1 is guaranteed to stop no matter in what order these rules are applied (it 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 an expression of fr$\mathcal{NRC}$ having free variables $x_1 : s_1$, ..., $x_n : s_n$ such that e is a normal form with respect to the above rewrite 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.* □

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 [11]. 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.* □

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

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 [6, 8, 10]. As a consequence, we can make use of well-known results [2, 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 [9] 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 [9] 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 [9], S_m has distinct representatives iff m is even. It follows that fr$\mathcal{NRC}$ cannot test for distinct representatives. $\square$

4 Effective Computability and Complexity

Recall that expressions in fr$\mathcal{NRC}$ can iterate over infinite sets. An important question that arises 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 $p_s : s- > s'$ 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 then 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.

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.* □

The above "compilation procedure" showed 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 our spatial databases 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 [8]. Consider the diagram introduced in Proposition 4.2. As f' is expressed in first-order logic extended with polynomial constraints, it follows from [8] 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

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

References

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

[2] 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.

[3] 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.

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

[5] 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.

[6] S. Grumbach and J. Su. Finitely representable databases. In *Proc. of 13th ACM Symposium on Principles of Database Systems*, pages 289–300, May 1994.

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

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

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

[10] 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.

[11] 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.

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

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

[14] 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.

A Some Relevant Proofs

Proposition 5.2 *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. We note that the left and right squares commute by definitions of p_s, q_s, and $\sim$. We now construct $(f)'$ by induction on the structure of the fr$\mathcal{NRC}$ expression that defines f and we argue that the middle square and thus the entire diagram commutes. To simplify notations, we omit the subscript s from p_s and q_s in the proof below. We also omit the argument for the more obvious cases.

- Case $f(\vec{x}) = x_i$. Set $(f)'(O) = \{_{fr} x_i \mid (x_1, \ldots, x_n) \in_{fr} O\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. Then $(f)'(O) \sim p(x_i)$. Since $q \circ p = id$, we have $q((f)'(O)) = x_i$. So the middle square commutes.

- Case $f(\vec{x}) = c$. Set $(f)'(O) = \{_{fr} c\}$

- Case $f(\vec{x}) = \pi_i e$. Let $g(\vec{x}) = e$. Set $(f)'(O) = \{_{fr} x_i \mid (x_1, \ldots, x_n) \in_{fr} (g)'(O)\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g)'(O)) = e$. Thus $(g)'(O) \sim p(e)$. By definition, $p(\pi_i e) = \{_{fr} x_i \mid (x_1, \ldots, x_n) \in_{fr} p(e)\}$. Then $(f)'(O) \sim p(\pi_i e)$. Hence $q((f)'(O)) = \pi_i e$. So the middle square commutes.

- Case $f(\vec{x}) = (e_1, \ldots, e_n)$. Let $g_i(\vec{x}) = e_i$. Set $(f)'(O) = \{_{fr} (x_1, \ldots, x_n) \mid x_1 \in_{fr} (g_1)'(O)$, ..., $x_n \in_{fr} (g_n)'(O)\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_i)'(O)) = e_i$. Thus $(g_i)'(O) \sim p(e_i)$. By definition, $p((e_1, \ldots, e_n)) = \{_{fr} (x_1, \ldots, x_n) \mid x_1 \in_{fr} p(e_1), \ldots, x_n \in_{fr} p(e_n)\}$. Then $(f)'(O) \sim p((e_1, \ldots, e_n))$. Hence $q((f)'(O)) = (e_1, \ldots, e_n)$. So the middle square commutes.

- Case $f(\vec{x}) = \{\}$. Set $(f)'(O) = \{_{fr} (0, 0, \vec{0})\}$.

- Case $f(\vec{x}) = \{e\}$. Let $g(\vec{x}) = e$. Set $(f)'(O) = \{_{fr} (1, 1, x) \mid x \in_{fr} (g)'(O)\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g)'(O)) = e$. Thus $(g)'(O) \sim p(e)$. By definition, $p(\{e\}) = \{_{fr} (1, 1, x) \mid x \in_{fr} p(e)\}$. Then $(f)'(O) \sim p(\{e\})$. Hence $q((f)'(O)) = \{e\}$. So the middle square commutes.

- Case $f(\vec{x}) = \mathit{empty}_{fr} e$. Let $g(\vec{x}) = e$. Set $(f)'(O) = \{_{fr} 1 \mid (0, \vec{0}) \in_{fr} (g)'(O)\} \cup_{fr} \{_{fr} 0 \mid (1, x) \in_{fr} (g)'(O)\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g)'(O)) = e$. Thus $(g)'(O) \sim p(e)$. Thus e is empty iff $(g)'(O) = \{_{fr} (0, \vec{0})\}$. Thus $q((f)'(O)) = \mathit{empty}_{fr} e$. So the middle square commutes.

  The following variations of the emptiness test are used in subsequent cases:

  $$\mathit{myempty}(X) = \mathit{empty}_{fr} \{_{fr} 0 \mid 0 \in_{fr} (\mathit{empty}_{fr})'(X)\}$$

$$myempty'(X) = empty_{fr}\{_{fr}1 \mid (1,x) \in_{fr} X\}$$

- Case $f(\vec{x}) = e_1 \cup e_2$. Let $g_i(\vec{x}) = e_i$. Set $(f)'(O) = \textit{if } myempty((g_1)'(O)) \textit{ then } (\textit{if } myempty((g_2)'(O)) \textit{ then } \{_{fr}\vec{0}\} \textit{ else } (g_2)'(O)) \textit{ else } (\textit{if } myempty((g_2)'(O)) \textit{ then } (g_1)'(O) \textit{ else } (g_1)'(O) \cup_{fr} A(O)$, where $A(O) = \{_{fr}(1, i+k+1, x) \mid (1,i,x) \in_{fr} (g_2)'(O),\ k = \max\{_{fr}j \mid (1,j,y) \in_{fr} (g_1)'(O)\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_i)'(O)) = e_i$. There are four subcases. The subcases where either e_1 or e_2 is empty are trivial. So suppose e_1 and e_2 are both not empty. First note that $A(O) \sim (g_2)'(O)$ and thus $q(A(O)) = e_2$. Next observe that for any $(1,i,x)$ in $(g_1)'(O)$ and $(1,j,y)$ in $A(O)$, it is the case that $i < j$. Thus for any h so that one of the set $X = \{_{fr}(1,i,x) \mid (1,i,x) \in_{fr} (f)'(O),\ i = h\}$, $Y = \{_{fr}(1,i,x) \mid (1,i,x) \in_{fr} (g_1)'(O), i = h\}$, $Z = \{_{fr}(1,i,x) \mid (1,i,x) \in_{fr} A(O),\ i = h\}$ is not empty, it is the case that $X = Y$ or $X = Z$. Consequently, for any o in $(f)'(O)$, we have o in $e_1 \cup e_2$ and vice versa. That is, $q((f)'(O)) = e_1 \cup e_2$. So the middle square commutes.

- Case $f(\vec{x}) = \bigcup\{e_1 \mid y \in e_2\}$. Let $g_1(\vec{x}, y) = e_1$ and $g_2(\vec{x}) = e_2$. Set $(f)'(O) = \textit{if } myempty((g_2)'(O)) \textit{ then } \{_{fr}(0,0,\vec{0})\} \textit{ else if } myempty'(A(O)) \textit{ then } \{_{fr}(0,0,\vec{0})\} \textit{ else } A(O)$, where $A(O) = \bigcup\{_{fr} B(O,i) \mid (1,i,y) \in_{fr} (g_2)'(O)\}$, where $B(O,i) = \{_{fr}(1, k*i+h+1, w) \mid (1,h,w) \in_{fr} C(O,i),\ k = \max\{_{fr}h \mid (1,j,u) \in_{fr} (g_2)'(O),\ j < i,\ (1,h,v) \in_{fr} C(O,j)\}\}$, where $C(O,i) = (g_1)'(\{_{fr}(z,u) \mid z \in_{fr} O,\ (1,j,u) \in_{fr} (g_2)'(O),\ i = j\})$.

  This is the most complex case. Suppose $q(O) = p(\vec{x})$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_2)'(O)) = e_2$. Thus $(g_2)'(O) \sim p(e_2)$. Now there are two subcases. For the first subcase, suppose e_2 is empty. Then $myempty((g_2)'(O))$ is true. Then $q((f)'(O)) = \{_{fr}\} = f(\vec{x})$. So the middle square commutes in this subcase.

  For the second subcase, we assume that e_2 is not empty. Then $myempty((g_2)'(O))$ is false. By the hypothesis on g_2, we know that for each i such that the set $y_i = \{_{fr}u \mid (1,j,u) \in_{fr} (g_2)'(O), i = j\}$ is not empty, we have $q(y_i)$ is an element o_i of e_2. Moreover, there is one such i for each element of e_2. Then by hypothesis on g_1, we have $q(C(O,i)) = g_1(\vec{x}, o_i)$ for each such i. It is also obvious that $q(B(O,i)) = g_1(\vec{x}, o_i)$ for each such i, provided $g_1(\vec{x}, o_i)$ is not empty. Note that if $g_1(\vec{x}, o_i)$ is empty, then $B(O,i)$ is also empty, as opposed to being a singleton zero tuple.

  However, $B(O,i)$ has an advantage over $C(O,i)$ because the numbers it uses to identify the elements of $g_1(\vec{x}, o_i)$ are distinct from those of $B(O,j)$ whenever $i \neq j$. To see this, suppose $k = \max\{_{fr}h \mid (1,j,u) \in_{fr} (g_2)'(O),\ j < i,\ (1,h,v) \in_{fr} C(O,j)\}$. Then k is the maximum identifier that is used to identify elements in $g_1(\vec{x}, o_j)$, for $j < i$. This k exists because $g_1(\vec{x}, o_j)$ is finite for each o_j in e_2. Then $k*i+1$ is greater than the cardinality of the union of $g_1(\vec{x}, o_j)$ for $j < i$.

  We now have two subsubcases. For the first subsubcase, suppose $g_1(\vec{x}, o_i)$ is empty for each o_i in e_2. Then $B(O,i)$ is empty for all such o_i. Then $A(O)$ is also empty. Then $myempty'(A(O))$ is true. Then $q((f)'(O)) = \{_{fr}\} = f(\vec{x})$. So the middle square commutes in this subsubcase.

  For the second subsubcase, we assume that there are o_1, ..., o_n in e_2 such that $g_1(\vec{x}, o_i)$ is not empty and $f(\vec{x}) = g_1(\vec{x}, o_1) \cup_{fr} \cdots \cup_{fr} g_1(\vec{x}, o_n)$. Then $(f)'(O) = A(O) = B(O, o_1) \cup_{fr} \cdots \cup_{fr} B(O, o_n)$. Then $q((f)'(O)) = f(\vec{x})$. This finishes the final subsubcase.

- Case $f(\vec{x}) = (e_1 = e_2)$. Let $g_i(\vec{x}) = e_i$. Set $(f)'(O) = \mathit{if}\ \mathit{empty}_{fr}\{_{fr}1 \mid x \in_{fr} (g_1)'(O), y \in_{fr} (g_2)'(O), x = y\}\ \mathit{then}\ \{_{fr}\mathit{false}\}\ \mathit{else}\ \{_{fr}\mathit{true}\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_i)'(O)) = e_i$. So $(g_i)'(O) \sim p(e_i)$. Since $e_i : \mathbb{R}$, we have $(g_i)'(O) = \{_{fr}e_i\}$. Then it is obvious that $q((f)'(O)) = (e_1 = e_2)$. So the middle square commutes.

- Case $f(\vec{x}) = \mathit{empty}\ e$. Let $g(\vec{x}) = e$. Set $(f)'(O) = \{_{fr}\mathit{empty}_{fr}\{_{fr}1 \mid (1, i, x) \in_{fr} (g)'(O)\}\}$.

- Case $f(\vec{x}) = \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3$. Let $g_i(\vec{x}) = e_i$. Set $(f)'(O) = \mathit{if}\ \mathit{empty}_{fr}\{_{fr}1 \mid 0 \in_{fr} (g_1)'(O)\}\ \mathit{then}\ (g_2)'(O)\ \mathit{else}\ (g_3)'(O)$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_1)'(O)) = e_1$. Thus $(g_1)'(O) \sim p(e_1)$. Since $e_1 : \mathbb{B}$, we have $(g_2)'(O) = \{_{fr}1\}$ if e_1 is true and $(g_2)'(O) = \{_{fr}0\}$ if e_1 is false. Then $\mathit{empty}_{fr}\{_{fr}1 \mid 0 \in_{fr} (g_1)'(O)\}$ is true iff e_1 is true. Then it follows by hypothesis on e_2 and e_3 that $q((f)'(O)) = \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3$. So the middle square commutes.

- Case $f(\vec{x}) = \{_{fr}\}$. Set $(f)'(O) = \{_{fr}\vec{0}\}$.

- Case $f(\vec{x}) = \{_{fr}e\}$. Let $g(\vec{x}) = e$. Set $(f)'(O) = \{_{fr}(1, x) \mid x \in_{fr} (g)'(O)\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g)'(O)) = e$. Thus $(g)'(O) \sim p(e)$. Since $e : \mathbb{R} \times \cdots \times \mathbb{R}$, we know that $(g)'(O) = \{_{fr}e\}$. Thus $q((f)'(O)) = \{_{fr}e\}$. So the middle square commutes.

- Case $f(\vec{x}) = e_1 \cup_{fr} e_2$. Let $g_i(\vec{x}) = e_i$. Set $(f)'(O) = \mathit{if}\ \mathit{myempty}((g_1)'(O))\ \mathit{then}\ (\mathit{if}\ \mathit{myempty}((g_2)'(O))\ \mathit{then}\ \{_{fr}\vec{0}\}\ \mathit{else}\ (g_2)'(O))\ \mathit{else}\ (\mathit{if}\ \mathit{myempty}((g_2)'(O))\ \mathit{then}\ (g_1)'(O)\ \mathit{else}\ (g_1)'(O) \cup_{fr} (g_2)'(O)$.

- Case $f(\vec{x}) = \bigcup\{_{fr}e_1 \mid y \in_{fr} e_2\}$. Let $g_1(\vec{x}, y) = e_1$ and $g_2(\vec{x}) = e_2$.
  Set $(f)'(O) = \mathit{if}\ \mathit{myempty}\ ((g_2)'(O))\ \mathit{then}\ \{_{fr}\vec{0}\}\ \mathit{else}\ \mathit{if}\ \mathit{myempty}'(A(O))\ \mathit{then}\ \{_{fr}(0, 0, \vec{0})\}\mathit{else}\ A(O)$, where $A(O) = \bigcup\{_{fr}\ \mathit{if}\ \mathit{myempty}\ (B(O, y))\ \mathit{then}\ \{_{fr}\}\ \mathit{else}\ B(O, y) \mid (1, y) \in_{fr} (g_2)'(O)\}$, where $B(O, y) = (g_1)'(\{_{fr}(z, y) \mid z \in_{fr} O\})$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_2)'(O)) = e_2$. Thus $(g_2')(O) \sim p(e_2)$. Now there are two subcases. For the first subcase, suppose e_2 is empty. Then $\mathit{myempty}((g_2)'(O))$ is true. Then $q((f)'(O)) = \{_{fr}\} = \bigcup\{_{fr}e_1 \mid y \in_{fr} e_2\}$. So the middle square commutes in this subcase.

  For the second subcase, we assume that e_2 is not empty. Then $\mathit{myempty}((g_2)'(O))$ is false and $(g_2)'(O) = \{_{fr}(1, x) \mid x \in_{fr} e_2\}$ is forced. It is clear that $q(\{_{fr}(z, y) \mid z \in_{fr} O\}) = (\vec{x}, y)$ for each y in e_2. By hypothesis, $q(B(O, y)) = g_1(\vec{x}, y)$ for each y in e_2. We now have two subsubcases. For the first subsubcase, suppose $g_1(\vec{x}, y)$ is empty for each y in e_2. Then $\mathit{myempty}(B(O, y))$ is true for each y in e_2. Then $\mathit{myempty}'(A(O))$ is true. Then $q((f)'(O)) = \{_{fr}\} = f(\vec{x})$. So the middle square commutes in this subsubcase.

  For the second subsubcase, we assume that there are y_1, ..., y_n in e_2 such that $g_1(\vec{x}, y_i)$ is not empty and $f(\vec{x}) = g_1(\vec{x}, y_1) \cup_{fr} \cdots \cup_{fr} g_1(\vec{x}, y_n)$. Then $(f)'(O) = A(O) = B(O, y_1) \cup_{fr} \cdots \cup_{fr} B(O, y_n)$. Then $q((f)'(O)) = f(\vec{x})$. This finishes the final subsubcase. So the middle square commutes.

- Case $f(\vec{x}) = e_1 \oplus e_2$, where $\oplus$ is either $+$, $-$, $\cdot$, or $\div$. Let $g_i(\vec{x}) = e_i$. Set $(f)'(O) = \{_{fr} x \oplus y \mid x \in_{fr} (g_1)'(O),\ y \in_{fr} (g_2)'(O)\}$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_i)'(O)) = e_i$. Thus $(g_i)'(O) \sim p(e_i)$. Since $e_i : \mathbb{R}$, we must have $(g_i)'(O) = \{_{fr} e_i\}$. Then $q((f)'(O)) = q(\{_{fr} e_1 \oplus e_2\}) = e_1 \oplus e_2$. So the middle square commutes.

- Case $f(\vec{x}) = R$. Set $(f)'(O) = \{_{fr}(1, x) \mid x \in_{fr} R\}$.

- Case $f(\vec{x}) = \bigcup\{_{fr} e_1 \mid y \in e_2\}$. Let $g_1(\vec{x}, y) = e_1$ and $g_2(\vec{x}) = e_2$. Set $(f)'(O) = \mathit{if}\ \mathit{myempty}((g_2)'(O))\ \mathit{then}\ \{_{fr}(0, \vec{0})\}\ \mathit{else}\ \mathit{if}\ \mathit{myempty}'(A(O))\ \mathit{then}\ \{_{fr}(0, \vec{0})\}\ \mathit{else}\ A(O)$, where $A(O) = \bigcup\{_{fr}\ \mathit{if}\ \mathit{myempty}(B(O, i))\ \mathit{then}\ \{_{fr}\}\ \mathit{else}\ B(O, i) \mid (1, i, y) \in_{fr} (g_2)'(O)\}$, where $B(O, i) = (g_1)'(\{_{fr}(z, u) \mid z \in_{fr} O,\ (1, j, u) \in_{fr} (g_2)'(O),\ i = j\})$.

  For this case, suppose $q(O) = \vec{x}$. Thus $O \sim p(\vec{x})$. By hypothesis, $q((g_2)'(O)) = e_2$. Thus $(g_2)'(O) \sim p(e_2)$. We have two subcases. The first is when e_2 is empty. Then $(g_2)'(O)$ is a singleton zero tuple. Then $\mathit{myempty}((g_2)'\ (O))$ is true. Then $(f)'(O) = \{_{fr}(0, \vec{0})\}$. Thus $q((f)'(O)) = \{_{fr}\} \ = f(\vec{x})$. So the middle square commutes in this subcase.

  For the second subcase, we assume that e_2 is not empty. Then $\mathit{myempty}((g_2)'(O))$ is false. By the hypothesis on g_2, we know that for each i such that the set $y_i = \{_{fr} u \mid (1, j, u) \in_{fr} (g_2)'(O), i = j\}$ is not empty, we have $q(y_i)$ is an element o_i of e_2. Moreover, there is one such i for each element of e_2. Then by hypothesis on g_1, we have $q(B(O, i)) = g_1(\vec{x}, o_i)$ for each such i. We now have two subsubcases. For the first subsubcase, suppose $g_1(\vec{x}, o_i)$ is empty for each such i. Then $\mathit{myempty}(B(O, i))$ is true for each such i. Then $\mathit{myempty}'(A(O))$ is true. Then $q((f)'(O)) = \{_{fr}\} \ = f(\vec{x})$. So the middle square commutes in this subsubcase.

  For the second subsubcase, we assume that there are o_1, ..., o_n in e_2 such that $g_1(\vec{x}, o_i)$ is not empty and $f(\vec{x}) = g_1(\vec{x}, o_1) \cup_{fr} \cdots \cup_{fr} g_1(\vec{x}, o_n)$. Then $(f)'(O) = A(O) = B(O, o_1) \cup_{fr} \cdots \cup_{fr} B(O, o_n)$. Then $q((f)'(O)) = f(\vec{x})$. This finishes the final subsubcase. So the middle square commutes. $\square$