Electronic Notes in Theoretical Computer Science 86 No. 3 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume86.html 37 pages

A Safe Relational Calculus for Functional Logic Deductive Databases [1]

Jesús M. Almendros-Jiménez [2] Antonio Becerra-Terón [3]

Dpto. de Lenguajes y Computación. Universidad de Almería.
Carretera de Sacramento s/n, La Cañada de San Urbano. 04120-Almería. Spain

Abstract

In this paper, we present an extended relational calculus for expressing queries in functional-logic deductive databases. This calculus is based on first-order logic and handles relation predicates, equalities and inequalities over partially defined terms, and approximation equations. For the calculus formulas, we have studied syntactic conditions in order to ensure the domain independence property. Finally, we have studied its equivalence w.r.t. the original query language, which is based on equality and inequality constraints.

Key words: Logic Programming, Functional-Logic Programming, Deductive Databases.

1 Introduction

Functional logic programming is a paradigm which integrates *functions* into *logic programming*, widely investigated during the last years. In fact, many languages, such as *CURRY* [12], *BABEL* [21], and *TOY* [19], among others, have been developed around this research area [11]. On the other hand, it is known that *database technology* is involved in most software applications. For this reason, *programming languages* should include database features in order to cover with 'real world' applications. Therefore, the integration of database technology into functional logic programming may be interesting, in order to increase its application field.

Relational calculus [9] is a formalism for querying *relational databases* [8]. It is the basis of high-level database query languages like SQL, and its simplicity has been one of the keys for the wide adoption from database technology.

[1] This work has been partially supported by the Spanish project of the Ministry of Science and Technology "INDALOG" TIC2002-03968.

[2] Email:jalmen@ual.es
[3] Email:abecerra@ual.es

Relational calculus is based on the use of a fragment of the *first-order logic*. Logic formulas in the relational calculus contain *logic predicates*, which represent *relations*, and use *equality* relations in order to compare *attribute values*. *Free variables* in logic formulas work as *search variables*. The simplest relational calculus handles *conjunctions*, does not support *negation*, and formulas are *existentially quantified*. It allows the handling of tuples belonging to the *cross product* and *join* of two or more input relations. However, *disjunctions*, *universal quantifications* and *negation* can be included in order to handle the *union* of two relations, the *complement* of a relation (i.e. tuples which do not belong to a relation), and the *difference* of two relations (i.e. tuples which belong to a relation but do not belong to the other one).

On the other hand, functional logic programming is a declarative paradigm which uses *equality constraints* as base formalism for querying programs. *Query solving* is based on *equality constraint solving*.

In order to integrate functional logic programming and databases, we propose: (1) to adapt functional logic programs to databases, by considering a suitable *data model* and a *data definition language*; (2) to *consider an extended relational calculus* as query language, which handles the proposed data model; and finally, (3) to provide *semantic foundations* to the new query language.

With respect to (1), the underlying data model of functional logic programming is *complex* from a database point of view [1,7,13,23]. Firstly, types can be defined by using *recursively defined datatypes*, as *lists* and *trees*. Therefore, the attribute values can be *multi-valued*; that is, more than one value (for instance, a set of values enclosed in a list) for a given attribute corresponds to each set of key attributes.

In addition, we have adopted *non-deterministic semantics* from functional-logic programming, investigated in the framework *CRWL* [10]. Under non-deterministic semantics, values can be *grouped into sets*, representing the set of values of the output of a non-deterministic function. Therefore, the data model is complex in a double sense, allowing the handling of complex values built from recursively defined datatypes, and complex values grouped into sets.

Moreover, functional logic programming is able to handle *partial and possibly infinite data*. Therefore, in our setting, an attribute can be partially defined or, even, include possibly infinite information. The first case can be interpreted as follows: the database can include unknown information o partially defined information [17]; and the second one indicates that the database can store *infinite information*, allowing infinite database instances (i.e. *infinite attribute values* or *infinite sets of tuples*). The infinite information can be handled by means of *partial approximations*.

Moreover, we have adopted the handling of *negation* from functional logic programming, studied in the framework *CRWLF* [20]. As a consequence, the data model, here proposed, also handles non-existent information, and partially non-existent information.

Finally, we propose a *data definition language* which, basically, consists on *database schema definitions*, *database instance definitions* and *(lazy) function definitions*. A *database schema definition* includes *relation names*, and a set of *attributes* for each relation. For a given database schema, the *database instances* define *key* and *non-key attribute values*, by means of *(constructor-based) conditional rewriting rules* [10,20], where conditions handle equality and inequality constraints. In addition, we can define a set of functions. These functions will be used by queries in order to handle recursively defined datatypes, also named *interpreted functions* in a database setting. As a consequence, "pure" functional-logic programs can be considered as a particular case of our programs.

With respect to (2), typically the query language of functional logic languages is based on the solving of conjunctions of (in)equality constraints, which are defined w.r.t. some (in)equality relations over terms [10,20].

Our relational calculus will handle *conjunctions* of *atomic formulas*, which are *relation predicates*, *(in)equality relations* over terms, and *approximation equations* in order to handle interpreted functions. Logic formulas are either existentially or universally quantified, depending on whether they include negation or not.

However, it is known in database theory that a suitable query language must ensure the property of *domain independence* [2]. A query is domain independent, whenever the query satisfies, properly, two conditions: *(a) the query output over a finite relation is also a finite relation; and (b) the output relation only depends on the input relations.* In general, it is undecidable, and therefore syntactic conditions have to be developed in such a way that, only the so-called *safe queries* (satisfying these conditions) ensure the property of domain independence. For instance, [1] and [22] propose syntactic conditions, which allow the building of safe formulas in a relational calculus with complex values and linear constraints, respectively. In this line, we have developed syntactic conditions over our query language, which allow the building of the so-called *safe formulas* satisfying the property of domain independence.

Extended relational calculi have been studied as alternative query languages for *deductive databases* [1,18], and *constraint databases* [6,14,15,16,22]. Our extended relational calculus is in the line of [1], in which deductive databases handle complex values in the form of *set* and *tuple* constructors. In our case, we generalize the mentioned calculus for handling *complex values built from (arbitrary) recursively defined datatypes.*

In addition, our calculus is similar to the calculi for constraint databases, in the sense of allowing the handling of *infinite databases*. However, in the framework of constraint databases, infinite databases model *infinite objects* by means of *(linear) equations* and *inequations*, and *intervals* which are handled in a symbolic way. Here, infinite databases are handled by means of *laziness* and partial approximations. Moreover, we handle constraints which consist on equality and inequality relations over complex values.

Finally, and w.r.t. (3), we will show that our relational calculus is *equivalent* to a query language based on *(in)equality constraints*, similar to existent functional logic languages.

Furthermore, we have developed theoretical foundations for the database instances, by defining a *partial order* which represents an *approximation ordering over database instances*, and a suitable *fix point operator* which computes the *least database instance* (w.r.t. the approximation ordering) satisfying a set of conditional rewriting rules.

Finally, remark that this work goes towards the design of a functional logic deductive language for which an operational semantics [3,5], and a relational algebra [4] have already been studied.

The organization of this paper is as follows. Section 2 describes the data model; section 3 presents the extended safe relational calculus; section 4 defines a safe functional-logic query language and states the equivalence of both query languages; section 5 establishes the domain independence property; and finally, section 6 defines the least database satisfying a set of conditional rules.

2 The Data Model

Our data model consists on complex values and partial information, which can be handled in a data definition language based on conditional constructor-based rewriting rules.

2.1 Complex Values

In our framework, we consider two main kinds of partial information: undefined information (ni), represented by $\perp$, which means *information unknown, although it may exist*, and nonexistent information (ne), represented by F, which means that *the information does not exist*.

Now, let's suppose a complex value, storing information about job salary and salary bonus, by means of a data constructor (like a *record*) `s&b(Salary, Bonus)`. Then, we can additionally consider the following kinds of partial information:

s&b(3000, 100)	totally defined information, *expressing that a person's salary is* 3000 €, *and his(her) salary bonus is* 100 €
s&b($\perp$, 100)	partially undefined information (pni), *expressing that a person's salary bonus is known, that is* 100 €, *but not his(her) salary*
s&b(3000, F)	partially nonexistent information (pne), *expressing that a person's salary is* 3000 €, *but (s)he has no salary bonus*

Over these kinds of information, the following (in)equality relations can be defined as follows:

(1) = (*syntactic equality*), expressing that *two values are syntactically equal*;

for instance, the relation $\mathtt{s\&b}(3000, \perp) = \mathtt{s\&b}(3000, \perp)$ is satisfied.

(2) $\downarrow$ (*strong equality*), expressing that *two values are equal and totally defined*; for instance, the relation $\mathtt{s\&b}(3000, 25) \downarrow \mathtt{s\&b}(3000, 25)$ holds, and the relations $\mathtt{s\&b}(3000, \perp) \downarrow \mathtt{s\&b}(3000, 25)$ and $\mathtt{s\&b}(3000, \mathsf{F}) \downarrow \mathtt{s\&b}(3000, 25)$ do not hold.

(3) $\uparrow$ (*strong inequality*), where *two values are (strongly) different, if they are different in their defined information*; for instance, the relation $\mathtt{s\&b}(3000, \perp) \uparrow \mathtt{s\&b}(2000, 25)$ is satisfied, whereas the relation $\mathtt{s\&b}(3000, \mathsf{F}) \uparrow \mathtt{s\&b}(3000, 25)$ does not hold.

In addition, we will consider their negations, that is, $\neq$, $\not\downarrow$ and $\not\uparrow$, which represent a *syntactic inequality*, *(weak) inequality* and *(weak) equality* relation, respectively. Next, we will formally define the above equality and inequality relations.

Assuming *constructor symbols* $c, d, \ldots$ $DC = \cup_n DC^n$ each one with an associated arity, and the symbols $\perp$, F as special cases with arity 0 (not included in DC), and a set $\mathcal{V}$ of variables $X, Y, \ldots$, we can build the set of *c-terms with* $\perp$ *and* F, denoted by $CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V})$. C-terms are complex values including variables which implicitly are universally quantified. We denote by $cterms(t)$ the set of (sub)terms of t. In addition, we can use *substitutions* $Subst_{DC,\perp,\mathsf{F}} = \{\theta \mid \theta : \mathcal{V} \rightarrow CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V})\}$, in the usual way, where the domain of a substitution θ, denoted by $Dom(\theta)$, is defined as usual. id denotes the identity. The above (in)equality relations can be formally defined as follows.

Definition 2.1 [Relations over Complex Values [20]] Given c-terms t, t':

(1) $t = t' \Leftrightarrow_{def} t$ and t' are syntactically equal;

(2) $t \downarrow t' \Leftrightarrow_{def} t = t'$ and $t \in CTerm_{DC}(\mathcal{V})$;

(3) $t \uparrow t' \Leftrightarrow_{def}$ they have a DC-clash, where t and t' have a DC-clash whether they have different constructor symbols of DC at the same position.

In addition, their negations can be defined as follows:

(1') $t \neq t' \Leftrightarrow_{def} t$ and t' have a $DC \cup \{\mathsf{F}\}$-clash;

(2') $t \not\downarrow t' \Leftrightarrow_{def} t$ or t' contains F as subterm, or they have a DC-clash;

(3') $\not\uparrow$ is defined as the least symmetric relation over $CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V})$ satisfying: $X \not\uparrow X$ for all $X \in \mathcal{V}$, $\mathsf{F} \not\uparrow t$ for all t, and if $t_1 \not\uparrow t'_1, ..., t_n \not\uparrow t'_n$, then $c(t_1, ..., t_n) \not\uparrow c(t'_1, ..., t'_n)$ for $c \in DC^n$.

Given that complex values can be partially defined, a *partial ordering* $\leq$ can be considered. This ordering is defined as the least one satisfying: $\perp \leq t$, $X \leq X$, and $c(t_1, ..., t_n) \leq c(t'_1, ..., t'_n)$ if $t_i \leq t'_i$ for all $i \in \{1, ..., n\}$ and $c \in DC^n$. The intended meaning of $t \leq t'$ is that t is *less defined or has less information* than t'. In particular, $\perp$ is the *bottom element*, given that $\perp$ represents $\mathsf{undefined\ information\ (ni)}$, that is, information more refinable can exist. In addition, F is *maximal* under $\leq$ (F satisfies the relations $\perp \leq \mathsf{F}$ and $\mathsf{F} \leq \mathsf{F}$), representing nonexistent information (ne), that is, no further refinable information can be obtained, given that it does not exist.

Now, we can consider sets of (partial) c-terms $\mathcal{SET}(CTerm_{DC,\perp,\mathsf{F}}\ (\mathcal{V}))$ which, in our framework, will be used for representing multi-valued attributes and the output from non-deterministic functions. We denote by $cterms(\mathcal{CV})$ the set of (sub)terms of the c-terms of $\mathcal{CV} \in \mathcal{SET}(CTerm_{DC,\perp,\mathsf{F}}$.

Given that these sets can be infinite and c-terms can be also infinite, we need to define a partial order over sets representing an *approximation ordering* over (possibly infinite) sets of c-terms. The approximation ordering is defined as follows: $\mathcal{CV}_1 \sqsubseteq \mathcal{CV}_2$, where $\mathcal{CV}_1, \mathcal{CV}_2 \in \mathcal{SET}(CTerm_{DC,\perp,\mathsf{F}}\ (\mathcal{V}))$, iff for all $t_1 \in \mathcal{CV}_1$ there exists $t_2 \in \mathcal{CV}_2$ such that $t_1 \leq t_2$, and for all $t_2 \in \mathcal{CV}_2$ there exists $t_1 \in \mathcal{CV}_1$ such that $t_1 \leq t_2$. The defined order is such that $\mathcal{CV}_1\psi \sqsubseteq \mathcal{CV}_2\psi$ if $\mathcal{CV}_1 \sqsubseteq \mathcal{CV}_2$ for every substitution ψ. Finally, we can define over sets of c-terms the following *equality* and *inequality* relations.

Definition 2.2 [Relations over Sets of Complex Values] Given $\mathcal{CV}_1$ and $\mathcal{CV}_2 \in \mathcal{SET}(CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V}))$:

(1) $\mathcal{CV}_1 \bowtie \mathcal{CV}_2$ holds, whenever at least one finite value in $\mathcal{CV}_1$ and $\mathcal{CV}_2$ is *strongly equal*; and

(2) $\mathcal{CV}_1 <> \mathcal{CV}_2$ holds, whenever at least one value in $\mathcal{CV}_1$ and $\mathcal{CV}_2$ is *strongly different*;

and their negations:

(1') $\mathcal{CV}_1 \not\bowtie \mathcal{CV}_2$ holds, whenever all values in $\mathcal{CV}_1$ and $\mathcal{CV}_2$ are *weakly different*; and

(2') $\mathcal{CV}_1 \not<\!\!> \mathcal{CV}_2$ holds, whenever all values in $\mathcal{CV}_1$ and $\mathcal{CV}_2$ are *weakly equal*.

2.2 Data Definition Language

We propose a *data definition language* which, basically, consists on *database schema definitions*, *database instance definitions* and *(lazy) function definitions*.

A database schema definition includes *relation names*, and a set of *attributes* for each relation. For a given database schema, the *database instances* define *key* and *non-key attribute values*, by means of *(constructor-based) conditional rewriting rules*, where conditions handle equality and inequality constraints. In addition, we can define a set of functions. These functions will be used by queries in order to handle recursively defined datatypes, also named *interpreted functions* in a database setting.

Definition 2.3 [Database Schemas] Assuming a Milner's style polymorphic type system, a *database schema* S is a finite set of *relation schemas* $R_1, \ldots, R_p$ in the form of $R_j(\underline{A_1} : T_1, \ldots, \underline{A_k} : T_k, A_{k+1} : T_{k+1}, \ldots, A_n : T_n)$, $1 \leq j \leq p$, wherein the relation names are a pairwise disjoint set, and the relation schemas $R_1, \ldots, R_p$ include a pairwise disjoint set of typed *attributes*[4] $(A_1 : T_1, \ldots, A_n : T_n)$.

In the relation schema R, $A_1, \ldots, A_k$ represent *key attributes* and A_{k+1}, $\ldots$, A_n are *non-key attributes*, denoted by the sets $Key(R)$ and $NonKey(R)$, respectively. Key values are supposed to identify each tuple of the relation. Finally, we denote by $nAtt(R) = n$ and $nKey(R) = k$, the number of attributes and key attributes defined in R, respectively.

Definition 2.4 [Databases] A *database* D is a triple (S, DC, IF), where S is a *database schema*, $DC = \cup_{n\geq 0} DC^n$ is a set of *constructor symbols*, and $IF = \cup_{n\geq 0} IF^n$ represents a set of *interpreted function symbols*.

We denote the set of *defined schema symbols* (i.e. relation and non-key attribute symbols) by $DSS(D)$, and the set of *defined symbols* by $DS(D)$ (i.e. $DSS(D)$ together with IF). As an example of database, we can consider the following one:

$$S \left\{ \begin{array}{l} \mathtt{person\_job}(\underline{\mathtt{name}} : \mathtt{people}, \mathtt{age} : \mathtt{nat}, \mathtt{address} : \mathtt{dir}, \mathtt{job\_id} : \mathtt{job}, \mathtt{boss} : \mathtt{people}) \\ \mathtt{job\_information}(\underline{\mathtt{job\_name}} : \mathtt{job}, \mathtt{salary} : \mathtt{nat}, \mathtt{bonus} : \mathtt{nat}) \\ \mathtt{person\_boss\_job}(\underline{\mathtt{name}} : \mathtt{people}, \mathtt{boss\_age} : \mathtt{cbossage}, \mathtt{job\_bonus} : \mathtt{cjobbonus}) \\ \mathtt{peter\_workers}(\underline{\mathtt{name}} : \mathtt{people}, \mathtt{work} : \mathtt{job}) \end{array} \right.$$

$$DC \left\{ \begin{array}{l} \mathtt{john} : \mathtt{people},\ \mathtt{mary} : \mathtt{people},\ \mathtt{peter} : \mathtt{people} \\ \mathtt{lecturer} : \mathtt{job},\ \mathtt{associate} : \mathtt{job},\ \mathtt{professor} : \mathtt{job} \\ \mathtt{add} : \mathtt{string} \times \mathtt{nat} \rightarrow \mathtt{dir} \\ \mathtt{b\&a} : \mathtt{people} \times \mathtt{nat} \rightarrow \mathtt{cbossage} \\ \mathtt{j\&b} : \mathtt{job} \times \mathtt{nat} \rightarrow \mathtt{cjobbonus} \end{array} \right.$$

$$IF \left\{ \mathtt{retention\_for\_tax} : \mathtt{nat} \rightarrow \mathtt{nat} \right.$$

where S includes the schemas `person_job` (storing information about people and their jobs) and `job_information` (storing generic information about jobs), and the *"views"* `person_boss_job`, and `peter_workers`, which will take key values from the set of key values defined for `person_job`.

The first view includes, for each person, the pairs in the form of records constituted by: (a) his/her boss and boss' age, by using the complex c-term `b&a(people, nat)`; and (b) his/her job and job salary bonus, by using the complex c-term `j&b(job, nat)`. The second view includes workers whose boss is `peter`. The set DC includes constructor symbols for the types `people`, `job`, `dir`, `cbossage` and `cjobbonus`, and IF defines the interpreted function symbol `retention_for_tax`, which computes the free tax salary. In addition, we can consider database schemas involving *(possibly) infinite databases* such as shown as follows:

[4] We can suppose attributes qualified with the relation name when the names coincide.

$$
\begin{array}{ll}
S & \left\{\begin{array}{l}
\mathtt{2Dpoint}(\underline{\mathtt{coord}} : \mathtt{cpoint},\ \mathtt{color} : \mathtt{nat})\\
\mathtt{2Dline}(\underline{\mathtt{origin}} : \mathtt{cpoint},\ \underline{\mathtt{dir}} : \mathtt{orientation}, \mathtt{next} : \mathtt{cpoint}, \mathtt{points} : \mathtt{cpoint},\\
\mathtt{list\_of\_points} : \mathtt{list}(\mathtt{cpoint}))
\end{array}\right.\\
DC & \left\{\begin{array}{l}
\mathtt{north} : \mathtt{orientation}, \mathtt{south} : \mathtt{orientation}, \mathtt{east} : \mathtt{orientation}, \mathtt{west} : \mathtt{orientation}, \ldots\\
[\,] : \mathtt{list\ A},\quad [\,|\,] : \mathtt{A} \times \mathtt{list\ A} \rightarrow \mathtt{list\ A}\\
\mathtt{p} : \mathtt{nat} \times \mathtt{nat} \rightarrow \mathtt{cpoint}
\end{array}\right.\\
IF & \left\{\ \mathtt{select} : (\mathtt{list\ A}) \rightarrow \mathtt{A}\right.
\end{array}
$$

wherein the schemas `2Dpoint` and `2Dline` are defined for representing bidimensional points and lines, respectively. `2Dpoint` includes the point coordinates (`coord`) and `color`. Lines represented by `2Dline` are defined by using a starting point (`origin`) and direction (`dir`). Furthermore, `next` indicates the next point to be drawn in the line, `points` stores the *(infinite) set* of points of this line, and `list_of_points` the *(infinite) list* of points of the line. Here, we can see the double use of complex values: (1) a set (which can be implicitly assumed), and (2) a list.

Definition 2.5 [Schema Instances] A *schema instance* $\mathcal{S}$ of a database schema S is a set of *relation instances* $\mathcal{R}_1, \ldots \mathcal{R}_p$, where each relation instance $\mathcal{R}_j$, $1 \leq j \leq p$, is a (possibly infinite) set of tuples of the form $(V_1, \ldots, V_n)$ for the relation $R_j \in S$, with $n = nAtt(R_j)$ and $V_i \in \mathcal{SET}(CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V}))$. In particular, each V_l ($l \leq nKey(R_j)$) satisfies $V_l \in CTerm_{DC,\mathsf{F}}(\mathcal{V})$.

The last condition forces the key attribute values to be one-valued and without including $\perp$. However, non-key attributes can be multivalued with an infinite set of values and infinite values. Attribute values can be non-ground (i.e. including variables), wherein the variables are implicitly universally quantified.

Definition 2.6 [Database Instances] A *database instance* $\mathcal{D}$ of a database $D = (S, DC, IF)$ is a triple $(\mathcal{S}, \mathcal{DC}, \mathcal{IF})$, where $\mathcal{S}$ is a schema instance, $\mathcal{DC} = CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V})$, and $\mathcal{IF}$ is a set of *function interpretations* $f^{\mathcal{D}}, g^{\mathcal{D}}, \ldots$ satisfying $f^{\mathcal{D}} : CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V})^n \rightarrow \mathcal{SET}(CTerm_{DC,\perp,\mathsf{F}}\ (\mathcal{V}))$ is monotone, that is, $f^{\mathcal{D}}(t_1, \ldots, t_n) \sqsubseteq f^{\mathcal{D}}(t'_1, \ldots, t'_n)$ if $t_i \leq t'_i$, $1 \leq i \leq n$, for each $f \in IF^n$.

Functions are monotone w.r.t. the approximation ordering defined over c-terms and sets of c-terms. Deterministic functions define an unitary set; otherwise they represent non-deterministic functions.

Next, we will show an example of schema instance for the database schemas `person_job`, `job_information`, and the database views `person_boss_job` and `peter_workers`:

$$
\mathtt{person\_job}\ \left\{\begin{array}{l}
(\mathtt{john}, \{\perp\}, \{\mathtt{add}('\mathtt{6th\ Avenue}', 5)\}, \{\mathtt{lecturer}\}, \{\mathtt{mary}, \mathtt{peter}\})\\
(\mathtt{mary}, \{\perp\}, \{\mathtt{add}('\mathtt{7th\ Avenue}', 2)\}, \{\mathtt{associate}\}, \{\mathtt{peter}\})\\
(\mathtt{peter}, \{\perp\}, \{\mathtt{add}('\mathtt{5th\ Avenue}', 5)\}, \{\mathtt{professor}\}, \{\mathsf{F}\})
\end{array}\right.
$$

job_information	(lecturer, {1200}, {F}) (associate, {2000}, {F}) (professor, {3200}, {1500})
person_boss_job	(john, {b&a(mary, ⊥), b&a(peter, ⊥)}, {j&b(lecturer, F)}) (mary, {b&a(peter, ⊥)}, {j&b(associate, F)}) (peter, {b&a(F, ⊥)}, {j&b(professor, 1500)})
peter_workers	(john, {lecturer}) (mary, {associate})

With respect to the modeling of (possibly) infinite databases, we can consider the following instance of the relation schema `2Dline`, including approximation values to infinite values in the attributes:

2Dpoint	(p(0, 0), {1}), (p(0, 1), {2}), (p(1, 0), {F}), ...
2Dline	(p(0, 0), north, {p(0, 1)}, {p(0, 1), p(0, 2), ⊥}, {[p(0, 0), p(0, 1), p(0, 2)|⊥]}), ... (p(1, 1), east, {p(2, 1)}, {p(2, 1), p(3, 1), ⊥}, {[p(1, 1), p(2, 1), p(3, 1)|⊥]}), ...

Instances (key and non-key attribute values, and interpreted functions) are defined by means of *constructor-based conditional rewriting rules.*

Definition 2.7 [Conditional Rewriting Rules] A *constructor-based conditional rewriting* rule RW for a symbol $H \in DS(D)$ has the form

$$H\ t_1 \ldots t_n := r \Leftarrow C$$

representing that *r is the value of $H\ t_1 \ldots t_n$, whenever the condition C is satisfied.* In this kind of rule:

(i) $(t_1, \ldots, t_n)$ is a linear tuple (i.e. each variable in it occurs only once) with $t_i \in CTerm_{DC}(\mathcal{V})$;

(ii) $r \in Term_D(\mathcal{V})$;

(iii) C is a set of constraints of the form $e \bowtie e'$, $e <> e'$, $e \not\bowtie e'$, $e \not<> e'$, where $e, e' \in Term_D(\mathcal{V})$; and

(iv) *extra variables* are not allowed, i.e. $var(r) \cup var(C) \subseteq var(t_1, \ldots, t_n)$.

$Term_D(\mathcal{V})$ represents the set of *terms* or *expressions* built from a database D (i.e. built from DC, $DS(D)$ and variables of $\mathcal{V}$). We denote by $cterms(e)$ the set of (sub)terms of e. Each term or expression e represents a set, in such a way that, the set of constraints allows comparing sets, accordingly to the semantics of the relations defined over sets of complex values: $\bowtie, <>, \not\bowtie, \not<>$ (see definition 2.2). For instance, the above mentioned instances can be defined by the following rules:

person_job	person_job john := ok.	person_job mary := ok.
	person_job peter := ok.	
	address john := add('6th Avenue', 5).	address mary := add('7th Avenue', 2).
	address peter := add('5th Avenue', 5).	
	job_id john := lecturer.	job_id mary := associate.
	job_id peter := professor.	
	boss john := mary.	boss john := peter.
	boss mary := peter.	
job_information	job_information lecturer := ok.	job_information associate := ok.
	job_information professor := ok.	
	salary lecturer := retention_for_tax 1500.	
	salary associate := retention_for_tax 2500.	
	salary professor := retention_for_tax 4000.	
	bonus professor := 1500.	
person_boss_job	person_boss_job Name := ok ⇐ person_job Name ⋈ ok.	
	boss_age Name := b&a(boss Name, address (boss Name)).	
	job_bonus Name := j&b(job_id (Name), bonus (job_id (Name))).	
peter_workers	peter_workers Name := ok ⇐ person_job Name ⋈ ok, boss Name ⋈ peter.	
	work Name := job_id Name.	
retention_for_tax	retention_for_tax Fullsalary := Fullsalary − (0.2 ∗ Fullsalary).	

The rules $R\, t_1\, \ldots\, t_k := r \Leftarrow C$, where r is a term of type typeok, allow the setting of $t_1, \ldots, t_k$ as key values of the relation R. typeok consists of a unique special value ok (ok is a shorthand of *object key*). The rules $A\ t_1\ \ldots\ t_k := r \Leftarrow C$, where $A \in NonKey(R)$, set r as the value of the non-key attribute A for the tuple of R with key values $t_1, \ldots, t_k$, whenever the set of constraints C holds. In these kinds of rules, $t_1, \ldots, t_k, r$ can be non-ground values, and thus the key and non-key attribute values are so too. Rules for the non-key attributes $A\ t_1\ \ldots\ t_k := r \Leftarrow C$ are implicitly constrained to the form $A\ t_1\ \ldots\ t_k := r \Leftarrow R\ t_1\ \ldots\ t_k \bowtie \texttt{ok},\ C$, in order to guarantee that $t_1, \ldots, t_k$ are key values defined in a tuple of R.

As can be seen in the rules, undefined information (ni) is interpreted, whenever *there are no rules* for a given attribute. In addition, whenever the attribute is defined by rules, it is assumed that the attribute will include nonexistent information (ne) for the keys for which either the attribute is not defined or the constraints of the rule are not satisfied. This behavior fits with the failure of reduction of conditional rewriting rules proposed in [20]. Once $\perp$ and F are introduced as special cases of attribute values, the view person_boss_job will include partially undefined (pni) and partially nonexistent (pne) information. In addition and due to the form of the rules which define the key attribute values of person_boss_job and peter_workers, we can consider both as views

Table 1
Examples of (Functional-Logic) Queries

Query	Description	Answer
	Handling of Multi-valued Attributes	
boss X $\bowtie$ peter.	*who has* peter *as boss?*	Y/john Y/mary
address (boss X) $\bowtie$ Y, job_id X $\not\bowtie$ lecturer.	*To obtain non-*lecturer *people and their bosses' addresses*	X/mary, Y/add('5th Avenue', 5)
	Handling of Partial Information	
job_bonus X $\not\Diamond$ j&b(associate, Y).	*To obtain people whose all jobs are equal to* associate, *and their salary bonuses, although they do not exist*	X/mary, Y/F
	Handling of Infinite Databases	
select (list_of_points p(0, 0) Z) $\bowtie$ p(0, 2).	*To obtain the orientation of the line from* p(0, 0) *to* p(0, 2)	Z/north

defined from person_job.

Now, we can consider (functional-logic) *queries*, which are similar to the condition of a conditional rewriting rule. Its formal definition will be presented in section 4. For instance, table 1 shows some examples, with their corresponding meanings and expected answers.

3 Extended Relational Calculus

Next, we present the *extension of the relational calculus*, by showing its syntax, safety conditions, and, finally, its semantics.

3.1 Syntax and Safety Conditions

Let's start with the syntax of the extended relational calculus.

Definition 3.1 [Atomic Formulas] Given a database $D = (S, DC, IF)$, the *atomic formulas* are expressions of the form:

(i) $R(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$, where R is a schema of S, the variables $x_i's$ are pairwise distinct, $k = nKey(R)$, and $n = nAtt(R)$

(ii) $x = t$, where $x \in \mathcal{V}$ and $t \in CTerm_{DC}(\mathcal{V})$

(iii) $t \Downarrow t'$ or $t \Uparrow t'$, where $t, t' \in CTerm_{DC}(\mathcal{V})$

(iv) $e \lhd x$, where $e \in Term_{DC,IF}(\mathcal{V})$ [5], and $x \in \mathcal{V}$

In the above definition, (i) represents *relation predicates*, (ii) *syntactic equality*, (iii) *(strong) equality and inequality equations*, which have the same meaning as the corresponding relations (see section 2.1, definition 2.1). Finally, (iv) is an *approximation equation*, representing approximation values obtained from interpreted functions.

Definition 3.2 [Calculus Formulas] A *calculus formula* φ against a database instance $\mathcal{D}$ has the form $\{x_1, \ldots, x_n \mid \phi\}$, such that ϕ is a conjunction of the form $\phi_1 \wedge \ldots \wedge \phi_n$ where each ϕ_i has the form ψ or $\neg\psi$, and each ψ is an *existentially quantified conjunction of atomic formulas*. Variables x_i's are the *free variables* of ϕ, denoted by $free(\phi)$. Finally, variables x_i's occurring in all atomic formulas $R(\bar{x})$ are distinct, and the same happens to variables x's occurring in approximation equations $e \lhd x$.

Formulas can be built from $\forall, \rightarrow, \vee, \leftrightarrow$ whenever they are logically equivalent to the defined calculus formulas. For instance, the (functional-logic) query $\mathcal{Q}_\mathsf{s} \equiv$ `retention_for_tax X` $\bowtie$ `salary` (`job_id peter`) w.r.t the database schemas `person_job` and `job_information`, requests `peter`'s full salary, and obtains as answer `X/4000` €. This query can be written in the proposed relational calculus as follows:

$$\begin{array}{l} \varphi_\mathsf{s} \equiv \{\mathtt{x} \mid (\exists \mathtt{y}_1.\exists \mathtt{y}_2.\exists \mathtt{y}_3.\exists \mathtt{y}_4.\exists \mathtt{y}_5.\ \mathtt{person\_job}(\mathtt{y}_1,\mathtt{y}_2,\mathtt{y}_3,\mathtt{y}_4,\mathtt{y}_5)\ \wedge \mathtt{y}_1 = \mathtt{peter} \wedge \\ \qquad \exists \mathtt{z}_1.\exists \mathtt{z}_2.\exists \mathtt{z}_3.\ \mathtt{job\_information}(\mathtt{z}_1,\mathtt{z}_2,\mathtt{z}_3)\ \wedge\ \mathtt{z}_1 = \mathtt{y}_4\ \wedge\ \exists \mathtt{u}. \\ \qquad \mathtt{retention\_for\_tax}\ \mathtt{x} \lhd \mathtt{u}\ \wedge\ \mathtt{z}_2 \Downarrow \mathtt{u})\} \end{array}$$

In this case, φ_s expresses the following meaning: to obtain the full salary, that is, $\mathtt{retention\_for\_tax}\ \mathtt{x} \lhd \mathtt{u}$ and $\exists \mathtt{z}_1.\exists \mathtt{z}_2.\exists \mathtt{z}_3.\mathtt{job\_information}(\mathtt{z}_1,\mathtt{z}_2,\mathtt{z}_3) \wedge \mathtt{z}_2 \Downarrow \mathtt{u}$, for `peter`, that is, $\exists \mathtt{y}_1. \ldots \exists \mathtt{y}_5.\ \mathtt{person\_job}(\mathtt{y}_1, \ldots, \mathtt{y}_5) \wedge \mathtt{y}_1 = \mathtt{peter} \wedge \mathtt{z}_1 = \mathtt{y}_4$.

In database theory, it is known that any query language must ensure the property of *domain independence* [2]. A query is *domain independent*, whenever the query satisfies, properly, two conditions: *(a) the query output over a finite relation is also a finite relation; and (b) the output relation only depends on the input relations.* In general, it is undecidable, and therefore syntactic conditions have to be developed in such a way that, only the so-called *safe queries* (satisfying these conditions) ensure the property of domain independence. For example, in [2], the variables occurring in calculus formulas must be *range restricted*. In our case, we generalize the notion of *range restricted* to c-terms. In addition, we require *safety conditions over atomic formulas*, and *conditions over bounded variables.*

Now, given a calculus formula φ against a database D, we define the following sets of variables:

[5] Terms which do not include schema symbols.

(i) *Key variables.*
$formula\_key(\varphi) = \{x_i \mid there\ \ exists\ R(x_1, \ldots, x_i, \ldots, x_n)\ occurring\ in\ \varphi\ and\ 1 \leq i \leq nKey(R)\}$;

(ii) *Non-key variables.*
$formula\_nonkey(\varphi) = \{x_j \mid there\ exists\ \ R(x_1, \ldots, x_j, \ldots, x_n)\ occurring\ in\ \varphi\ and\ nKey(R) + 1 \leq j \leq n\}$; and

(iii) *Approximation variables.*
$approx(\varphi) = \{x \mid there\ exists\ e \lhd x\ occurring\ in\ \varphi\}$.

Definition 3.3 [Safe Atomic Formulas] An atomic formula is *safe* in φ in the following cases:

(i) $R(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$ is safe, if the variables $x_1, \ldots, x_n$ are bound in φ, and for each x_i, $i \leq nKey(R)$, there exists one equation $x_i = t_i$ in φ;

(ii) $x = t$ is safe, if the variables occurring in t are distinct from the variables of $formula\_key(\varphi)$, and $x \in formula\_key(\varphi)$;

(iii) $t \Downarrow t'$ *and* $t \Uparrow t'$ are safe, if the variables occurring in t and t' are distinct from the variables of $formula\_key(\varphi)$;

(iv) $e \lhd x$ is safe, if the variables occurring in e are distinct from the variables of $formula\_key(\varphi)$, and x is bound in φ.

Definition 3.4 [Range Restricted C-Terms of Calculus Formulas] A c-term is *range restricted* in a calculus formula φ if either:

(i) it occurs in $formula\_key(\varphi) \cup formula\_nonkey(\varphi)$, or

(ii) there exists one equation $e \diamondsuit_c e'$ ($\diamondsuit_c \equiv =, \Uparrow, \Downarrow$, or $\lhd$) in φ, such that it belongs to $cterms(e)$ (resp. $cterms(e')$) and every c-term of e' (resp. e) is range restricted in φ.

Range restricted c-terms are variables occurring in the scope of a relation predicate or c-terms compared (by means of syntactic, strong (in)equalities, and approximation equations) with variables in the scope of a relation predicate. Therefore, all of them take values from the schema instance.

Definition 3.5 [Safe Formulas] A calculus formula φ against a database D is *safe*, if:

(i) all c-terms and atomic formulas occurring in φ are range restricted and safe, respectively and,

(ii) the only bounded variables are variables of $formula\_key(\varphi) \cup formula\_non key(\varphi) \cup approx(\varphi)$.

For instance, the previous $\varphi_{\mathtt{s}}$ is *safe*, given that the c-term `peter` is *range restricted* (by means of $\mathtt{y_1} = \mathtt{peter}$), and the variables $\mathtt{u}, \mathtt{x}$ are also *range restricted* (by means of $\mathtt{retention\_for\_tax\ x} \lhd \mathtt{u}$ and $\mathtt{z_2} \Downarrow \mathtt{u}$). Once we have defined the conditions over the built formulas, we guarantee that they represent "queries" against a database. Negation can be used in combination

Table 2
Examples of Calculus Formulas

Query	Calculus Formula
boss X $\bowtie$ peter.	$\{x \mid (\exists y_1.\exists y_2.\exists y_3.\exists y_4.\exists y_5.\ \mathtt{person\_job}(y_1, y_2, y_3, y_4, y_5) \wedge y_1 = x \wedge y_5 \Downarrow \mathtt{peter})\}$
address (boss X) $\bowtie$ Y, job_id X $\not\bowtie$ lecturer.	$\{x, y \mid (\exists y_1.\exists y_2.\exists y_3.\exists y_4.\exists y_5.\ \mathtt{person\_job}(y_1, y_2, y_3, y_4, y_5) \wedge y_1 = x \wedge \exists z_1.\exists z_2.\exists z_3.\exists z_4.\exists z_5.\mathtt{person\_job}(z_1, z_2, z_3, z_4, z_5) \wedge z_1 = y_5 \wedge z_3 \Downarrow y) \wedge(\forall v_4.((\exists v_1.\exists v_2.\exists v_3.\exists v_5.\ \mathtt{person\_job}(v_1, v_2, v_3, v_4, v_5) \wedge v_1 = x) \rightarrow \neg v_4 \Downarrow \mathtt{lecturer}))\}$
job_bonus X $\not\lhd\!\rhd$ j&b(associate, Y).	$\{x, y \mid (\forall y_3.(\exists y_1.\exists y_2.\ \mathtt{person\_boss\_job}(y_1, y_2, y_3) \wedge y_1 = x) \rightarrow \neg y_3 \Uparrow \mathtt{j\&b}(\mathtt{associate}, y))\}$
select (list_of_points p(0,0) Z) $\bowtie$ p(0,2).	$\{z \mid (\exists y_1.\exists y_2.\exists y_3.\exists y_4.\exists y_5.\ \mathtt{2Dline}(y_1, y_2, y_3, y_4, y_5) \wedge y_1 = \mathtt{p}(0,0) \wedge y_2 = z \ \wedge\ \exists u.\mathtt{select}\ y_5 \lhd u \ \wedge\ u \Downarrow \mathtt{p}(0,2))\}$

with strong (in)equality relations; for instance, the calculus formula

$$\varphi_0 \equiv \neg\exists x_1.x_2.x_3.x_4.x_5.\mathtt{person\_job}(x_1, \ldots, x_5) \wedge x_1 = \mathtt{mary} \wedge x_5 \Downarrow y$$

requests people who are not a mary's boss. In this case, y is restricted to take values from the attribute boss of the relation person_job. Therefore, the obtained answers are $\{y/\mathtt{mary}\}$ and $\{y/\mathsf{F}\}$. Table 2 shows *(safe) calculus formulas* built from the *queries* presented in table 1.

3.2 Semantics of Relational Calculus

Now, we define the semantics of the relational calculus. With this aim, we need to define the following notions.

Definition 3.6 [Denotation of Terms] The *denoted values* of a term $e \in Term_{DC,IF}(\mathcal{V})$ in an instance $\mathcal{D}$ of a database $D = (S, DC, IF)$ w.r.t. a substitution θ, represented by $\llbracket e \rrbracket^{\mathcal{D}}\theta$, are defined as follows:

(i) $\llbracket X \rrbracket^{\mathcal{D}}\theta =_{def} \{X\theta\}$, for $X \in \mathcal{V}$;
(ii) $\llbracket c \rrbracket^{\mathcal{D}}\theta =_{def} \{c\}$, for $c \in DC^0$;
(iii) $\llbracket c(e_1, \ldots, e_n) \rrbracket^{\mathcal{D}}\theta =_{def} c(\llbracket e_1 \rrbracket^{\mathcal{D}}\theta, \ldots, \llbracket e_n \rrbracket^{\mathcal{D}}\theta)$ [6], for all $c \in DC^n$, $n > 0$;
(iv) $\llbracket f\ e_1\ \ldots\ e_n \rrbracket^{\mathcal{D}}\theta =_{def} f^{\mathcal{D}}\ \llbracket e_1 \rrbracket^{\mathcal{D}}\theta\ \ldots\ \llbracket e_n \rrbracket^{\mathcal{D}}\theta$, for all $f \in IF^n$.

The denoted values for a term or expression represent the set of values which defines a non-deterministic (resp. deterministic) interpreted function.

Definition 3.7 [Active Domain of Terms] The *active domain* of a term $e \in Term_{DC,IF}(\mathcal{V})$ in a calculus formula φ w.r.t an instance $\mathcal{D}$ of database $D = (S, DC, IF)$, denoted by $adom(e, \mathcal{D})$, is defined as follows:

[6] To simplify denotation, we write $\{c(t_1, \ldots, t_n) \mid t_i \in S_i\}$ as $c(S_1, \ldots, S_n)$ and $\{f(t_1, \ldots, t_n) \mid t_i \in S_i\}$ as $f(S_1, \ldots, S_n)$ where $S_i's$ are certain sets.

(i) $adom(x, \mathcal{D}) =_{def} \bigcup_{\psi \in Subst_{DC,\perp,\mathsf{F}}, (V_1,\ldots,V_i,\ldots,V_n) \in \mathcal{R}} V_i\psi$, if there exists an atomic formula $R(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n)$ in φ;

(ii) $adom(x, \mathcal{D}) =_{def} adom(e, \mathcal{D})$, if $e \lhd x$ occurs in φ;

(iii) $adom(x, \mathcal{D}) =_{def} \{\perp\}$, otherwise;

(iv) $adom(c, \mathcal{D}) =_{def} \{\perp\}$, if $c \in DC^0$;

(v) $adom(c(e_1, \ldots, e_n), \mathcal{D}) =_{def} c(adom(e_1, \mathcal{D}), \ldots, adom(e_n, \mathcal{D}))$, if $c \in DC^n$, $n > 0$;

(vi) $adom(f\ e_1 \ldots e_n, \mathcal{D}) =_{def} f^{\mathcal{D}} adom(e_1, \mathcal{D}) \ldots adom(e_n, \mathcal{D})$, if $f \in IF^n$.

The active domain of variables representing key and non-key attributes includes the complete set of values defined in the schema instance for the corresponding attribute. In the case of approximation variables, the active domain contains the complete set of values of the interpreted function. For example, the active domain of $\mathtt{x_5}$ in the atomic formula $\mathtt{person\_job}(\mathtt{x_1}, \ldots, \mathtt{x_5})$ is $\{\mathtt{mary}, \mathtt{peter}, \mathsf{F}\}$, corresponding to the set of values included in the database instance for the attribute boss. In other words, the active domain is used in order to restrict the set of answers which defines a calculus formula w.r.t the database instance. For instance, the previous formula φ_0 restricts the variable y to be valued in the active domain of $\mathtt{x_5}$, that is, $\{\mathtt{peter}, \mathtt{mary}, \mathsf{F}\}$, and therefore, obtaining as answers $\{\mathtt{y/mary}\}$ and $\{\mathtt{y}/\mathsf{F}\}$. Remark that the isolated equation $\neg\mathtt{x_5} \Downarrow \mathtt{y}$ is satisfied for $\{\mathtt{x_5/peter}, \mathtt{y/lecturer}\}$ w.r.t. $\not\downarrow$. However the value lecturer is not in the active domain of $\mathtt{x_5}$.

Finally, note that we have to instantiate the schema instance, whenever it includes variables in order to obtain the complete set of values represented by an attribute (see case (i) of the above definition).

Definition 3.8 [Satisfiability] Given a calculus formula $\{\bar{x} \mid \phi\}$, the *satisfiability* of ϕ in a database instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ under a substitution θ, such that $dom(\theta) \subseteq free(\phi)$, (in symbols $(\mathcal{D}, \theta) \models_C \phi$) is defined as follows:

(i) $(\mathcal{D}, \theta) \models_C R(x_1, \ldots, x_n)$, if there exists $(V_1, \ldots, V_n) \in \mathcal{R}$ ($\mathcal{R} \in \mathcal{S}$), such that $x_i\theta \in V_i\psi$ for every $1 \leq i \leq n$ and $V_j\psi \in CTerm_{DC,\mathsf{F}}$ for every $1 \leq j \leq k$, where $\psi \in Subst_{DC,\perp,\mathsf{F}}$;

(ii) $(\mathcal{D}, \theta) \models_C x = t$, if $x\theta = t\theta$, and $t\theta \in adom(x, \mathcal{D}) \cup \{t\}$;

(iii) $(\mathcal{D}, \theta) \models_C t \Downarrow t'$, if $t\theta \downarrow t'\theta$, and $t\theta, t'\theta \in adom(t, \mathcal{D}) \cup adom(t', \mathcal{D})$;

(iv) $(\mathcal{D}, \theta) \models_C t \Uparrow t'$, if $t\theta \uparrow t'\theta$, and $t\theta, t'\theta \in adom(t, \mathcal{D}) \cup adom(t', \mathcal{D})$;

(v) $(\mathcal{D}, \theta) \models_C e \lhd x$, if $x\theta \in [\![e]\!]^{\mathcal{D}}\theta$, and $x\theta \in adom(e, \mathcal{D})$;

(vi) $(\mathcal{D}, \theta) \models_C \phi_1 \wedge \phi_2$, if $\mathcal{D}$ satisfies ϕ_1 and ϕ_2 under θ;

(vii) $(\mathcal{D}, \theta) \models_C \exists x.\phi$, if there exists v, such that $\mathcal{D}$ satisfies ϕ under $\theta \cdot \{x/v\}$;

(viii) $(\mathcal{D}, \theta) \models_C \neg\phi$, if $(\mathcal{D}, \theta) \not\models_C \phi$, where:

(a) $(\mathcal{D}, \theta) \not\models_C R(x_1, \ldots, x_n)$, if for all $(V_1, \ldots, V_k, \ldots, V_n) \in \mathcal{R}$ ($\mathcal{R} \in \mathcal{S}$) and $\psi \in Subst_{DC,\perp,\mathsf{F}}$, then $x_i\theta \neq V_i\psi$ for some i such that $1 \leq$

$i \le k$, but there exist tuples $(W_1, \ldots, V_i, \ldots, W_k, \ldots, W_n) \in \mathcal{R}$ and $\psi_i \in Subst_{DC,\perp,\mathsf{F}}$ such that $x_i\theta \in V_i\psi_i$, $(1 \le i \le k)$ and $V_j\psi_j \in CTerm_{DC,\mathsf{F}}$, $(1 \le j \le k)$,

(b) $(\mathcal{D}, \theta) \not\models_C x = t$, if $x\theta \neq t\theta$, and $t\theta \in adom(x, \mathcal{D}) \cup \{t\}$;

(c) $(\mathcal{D}, \theta) \not\models_C t \Downarrow t'$, if $t\theta \not\downarrow t'\theta$, and $t\theta, t'\theta \in adom(t, \mathcal{D}) \cup adom(t', \mathcal{D})$;

(d) $(\mathcal{D}, \theta) \not\models_C t \Uparrow t'$, if $t\theta \not\uparrow t'\theta$, and $t\theta, t'\theta \in adom(t, \mathcal{D}) \cup adom(t', \mathcal{D})$;

(e) $(\mathcal{D}, \theta) \not\models_C e \lhd x$, if $x\theta \notin \llbracket e \rrbracket^{\mathcal{D}}\theta$, and $x\theta \in adom(e, \mathcal{D})$;

(f) $(\mathcal{D}, \theta) \not\models_C \phi_1 \wedge \phi_2$, if $(\mathcal{D}, \theta) \models_C \phi_1$ or $(\mathcal{D}, \theta) \models_C \phi_2$;

(g) $(\mathcal{D}, \theta) \not\models_C \exists x.\phi$, if for all v, then $(\mathcal{D}, \theta \cdot \{x/v\}) \not\models_C \phi$;

(h) $(\mathcal{D}, \theta) \not\models_C \neg\phi$, if $(\mathcal{D}, \theta) \models_C \phi$.

With regard to the use of both denotation and active domain in the notion of satisfiability, in the previous formula φ_0, and w.r.t. the formula $\neg \mathtt{x_5} \Downarrow \mathtt{y}$, we have that $\mathtt{adom}(\mathtt{x_5}, \mathcal{D}) = \{\mathtt{peter}, \mathtt{mary}, \mathsf{F}\}$ and $\mathtt{adom}(\mathtt{y}, \mathcal{D}) = \{\perp\}$. Moreover, $\theta_1 = \{\mathtt{y/mary},\ \mathtt{x_5/peter}\}$ and $\theta_2 = \{\mathtt{y}/\mathsf{F}, \mathtt{x_5/peter}\}$ satisfies that $\mathtt{y}\theta_1, \mathtt{y}\theta_2 \in \mathtt{adom}(\mathtt{x_5}, \mathcal{D}) \cup \mathtt{adom}(\mathtt{y}, \mathcal{D})$; therefore, $\mathtt{x_5}\theta_1 \not\downarrow \mathtt{y}\theta_1$ and $\mathtt{x_5}\theta_2 \not\downarrow \mathtt{y}\theta_2$ are satisfied. However, no more values for the variable y can be used for satisfying of $\neg \mathtt{x_5} \Downarrow \mathtt{y}$. Therefore, we take into account the domain of the variables (in general, the active domain of the c-terms) in order to satisfy the calculus formulas. It ensures the domain independence property as we will see later.

With respect to the negation, we have to explicitly define the meaning of the negated formulas, due to, for instance, $\neq$, $\not\downarrow$ and $\not\uparrow$ are not the "logical" negation of the corresponding relations $=$, $\downarrow$ and $\uparrow$. For instance, neither $\perp \downarrow \mathtt{0}$, nor $\perp \not\downarrow \mathtt{0}$ are satisfied. The same happens to atomic formulas of the form $R(x_1, \ldots, x_n)$, which are satisfied for tuples of $\mathcal{R}$, and they are not satisfied for combinations of such tuples.

Finally, given a calculus formula $\varphi \equiv \{x_1, \ldots, x_n \mid \phi\}$, we define the *set of answers* of φ w.r.t. an instance $\mathcal{D}$, denoted by $Ans(\mathcal{D}, \varphi)$, as follows: $Ans(\mathcal{D}, \{x_1, \ldots, x_n \mid \phi\}) = \{(x_1\theta, \ldots, x_n\theta) \mid \theta \in Subst_{DC,\perp,\mathsf{F}}$ *and* $(\mathcal{D}, \theta) \models_C \phi\}$.

4 Safe Functional Logic Queries

In this section, we will define safety conditions over functional-logic queries in order to propose a query language for functional logic deductive databases which: (a) on one hand, it ensures the domain independence property; and (b) on the other hand, it is equivalent to the proposed relational calculus. With this aim, we need the following definitions.

Definition 4.1 [Query Keys] The set of *query keys* of a key attribute $A_i \in Key(R)$ $(R \in S)$ occurring in a term $e \in Term_D(\mathcal{V})$, denoted by $query\_key(e, A_i)$, is defined as follows:

$$query\_key(e, A_i) =_{def} \{t_i \in CTerm_{DC,\mathsf{F}}(\mathcal{V}) \mid H\ e_1 \ldots t_i \ldots e_k \textit{ occurs in } e \textit{ and } H \in \{R\} \cup NonKey(R)\}$$

Now, the set of query keys in a query $\mathcal{Q}$ is defined as follows:

$$query\_key(\mathcal{Q}) =_{def} \cup_{A_i \in Key(R)} query\_key(\mathcal{Q}, A_i) \text{ where}$$

$$query\_key(\mathcal{Q}, A_i) =_{def} \cup_{e \Diamond_q e' \in \mathcal{Q}} (query\_key(e, A_i) \cup query\_key(e', A_i))$$

with $\Diamond_q \equiv \bowtie$, $<>$, $\not\bowtie$, or $\not<>$.

Definition 4.2 [Range Restricted C-Terms of Queries] A c-term t is *range restricted* in $\mathcal{Q}$, if either:

(a) t belongs to $\cup_{s \in query\_key(\mathcal{Q})} cterms(s)$, or

(b) there exists a constraint $e \Diamond_q e'$, such that t belongs to $cterms(e)$ (resp. $cterms(e')$) and every c-term occurring in e' (resp. e) is *range restricted.*

In the above case (a), we will say that t is a subterm of a query key.

Definition 4.3 [Safe Queries] A query $\mathcal{Q}$ is *safe* if all c-terms occurring in $\mathcal{Q}$ are range restricted.

For instance, let's consider the following query: $\mathcal{Q}_s \equiv$ `retention_for_tax X` $\bowtie$ `salary(job_id peter)`, corresponding to previously mentioned calculus formula φ_s. $\mathcal{Q}_s$ is *safe*, given that the constant `peter` is a *query key* (and thus range restricted) and therefore the variable `X` is also *range restricted.* Analogously to calculus, we need to define the denoted values and the active domain of a database term (which includes relation names and non-key attributes) in a functional-logic query.

Definition 4.4 [Denotation of Database Terms] Given a term $e \in Term_D(\mathcal{V})$ *the denotation of* e in an instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ of database $D = (S, DC, IF)$ under a substitution θ, is defined as follows:

(i) $[\![R\ e_1\ \ldots\ e_k\ ]\!]^{\mathcal{D}}\theta =_{def} \{\mathtt{ok}\}$, if there exists a tuple $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_n) \in \mathcal{R}$, and $\psi \in Subst_{DC,\perp,\mathsf{F}}$, such that $([\![e_1]\!]^{\mathcal{D}}\theta, \ \ldots\ , [\![e_k]\!]^{\mathcal{D}}\theta) = (V_1\psi, \ldots, V_k\psi)$ and $V_i\psi \in CTerm_{DC,\mathsf{F}}$, $1 \leq i \leq k$, where $\mathcal{R} \in \mathcal{S}$ and $k = nKey(R)$;

(ii) $[\![R\ e_1\ \ldots\ e_k\ ]\!]^{\mathcal{D}}\theta =_{def} \{\mathsf{F}\}$, if for all tuple $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_n) \in \mathcal{R}$, and $\psi \in Subst_{DC,\perp,\mathsf{F}}$, then $[\![e_i]\!]^{\mathcal{D}}\theta \neq V_i\psi$ for some i, $1 \leq i \leq k$, but there exist tuples $(W_1, \ldots, V_i, \ldots, W_k, \ldots, W_n) \in \mathcal{R}$ and $\psi_i \in Subst_{DC,\perp,\mathsf{F}}$ such that $[\![e_i]\!]^{\mathcal{D}}\theta = V_i\psi_i$ and $V_i\psi \in CTerm_{DC,\mathsf{F}}$, $1 \leq i \leq k$, where $\mathcal{R} \in \mathcal{S}$ and $k = nKey(R)$;

(iii) $[\![R\ e_1\ \ldots\ e_k\ ]\!]^{\mathcal{D}}\theta =_{def} \{\mathsf{F}\}$, if $\theta = id$ and for all tuple $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_n) \in \mathcal{R}$, and $\psi \in Subst_{DC,\perp,\mathsf{F}}$, then $[\![e_i]\!]^{\mathcal{D}}\theta \neq V_i\psi$ for some i, $1 \leq i \leq k$;

(iv) $[\![R\ e_1\ \ldots\ e_k\ ]\!]^{\mathcal{D}}\theta =_{def} \{\perp\}$ otherwise, for all $R \in S$;

(v) $[\![A_i\ e_1\ \ldots\ e_k\ ]\!]^{\mathcal{D}}\theta =_{def} V_i\psi$, if there exists a tuple $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_i, \ldots, V_n) \in \mathcal{R}$, and $\psi \in Subst_{DC,\perp,\mathsf{F}}$, such that $([\![e_1]\!]^{\mathcal{D}}\theta, \ \ldots\ , [\![e_k]\!]^{\mathcal{D}}\theta) = (V_1\psi, \ldots, V_k\psi)$ and $V_j\psi \in CTerm_{DC,\mathsf{F}}$, $1 \leq j \leq k$, where $\mathcal{R} \in \mathcal{S}$, and $i > nKey(R) = k$;

(vi) $[\![A_i\ e_1\ \ldots\ e_k\ ]\!]^{\mathcal{D}}\theta =_{def} \{\mathsf{F}\}$, if $[\![R\ e_1\ \ldots\ e_k\ ]\!]^{\mathcal{D}}\theta = \{\mathsf{F}\}$;

(vii) $\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}}\theta =_{def} \{\bot\}$ otherwise, for all $A_i \in NonKey(R)$;

(viii) $\llbracket X \rrbracket^{\mathcal{D}}\theta =_{def} \{X\theta\}$, for all $X \in \mathcal{V}$;

(ix) $\llbracket c \rrbracket^{\mathcal{D}}\theta =_{def} \{c\}$, for all $c \in DC^0$;

(x) $\llbracket c(e_1, \ldots, e_n) \rrbracket^{\mathcal{D}}\theta =_{def} c(\llbracket e_1 \rrbracket^{\mathcal{D}}\theta, \ldots, \llbracket e_n \rrbracket^{\mathcal{D}}\theta)$, for all $c \in DC^n$;

(xi) $\llbracket f\ e_1\ \ldots\ e_n \rrbracket^{\mathcal{D}}\theta =_{def} f^{\mathcal{D}}\ \llbracket e_1 \rrbracket^{\mathcal{D}}\theta\ \ldots\ \llbracket e_n \rrbracket^{\mathcal{D}}\theta$, for all $f \in IF^n$.

Definition 4.5 [Active Domain of Database Terms] Given a database instance $\mathcal{D}$, the *active domain* of $e \in Term_D(\mathcal{V})$ w.r.t $\mathcal{D}$ and a query $\mathcal{Q}$, denoted by $adom(e, \mathcal{D})$, is defined as follows:

(i) $adom(t, \mathcal{D}) =_{def} \{\ t \mid t \in cterms(V_i\psi), \psi \in Subst_{DC,\bot,\mathsf{F}}, (V_1, \ldots, V_i, \ldots, V_n) \in \mathcal{R}\}$, if $t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_i)$, $A_i \in Key(R)$; and $\{\bot\}$ otherwise, for all $t \in CTerm_{\bot,\mathsf{F}}(\mathcal{V})$;

(ii) $adom(c, \mathcal{D}) = \{\bot\}$ if $c \in DC^0$;

(iii) $adom(c(e_1, \ldots, e_n), \mathcal{D}) =_{def} c(adom(e_1, \mathcal{D}), \ldots, adom(e_n, \mathcal{D}))$, if $c(e_1, \ldots, e_n)$ is not a c-term, for all $c \in DC^n, n > 0$;

(iv) $adom(f\ e_1 \ldots e_n, \mathcal{D}) =_{def} f^{\mathcal{D}} adom(e_1, \mathcal{D}) \ldots adom(e_n, \mathcal{D})$, for all $f \in IF^n$;

(v) $adom(R\ e_1 \ldots e_k, \mathcal{D}) =_{def} \{\mathtt{ok}, \mathsf{F}, \bot\}$, for all $R \in S$;

(vi) $adom(A_i\ e_1 \ldots e_k, \mathcal{D}) =_{def} \bigcup_{\psi \in Subst_{DC,\bot,\mathsf{F}}, (V_1,\ldots,V_i,\ldots,V_n) \in \mathcal{R}} V_i\psi$, for all $A_i \in NonKey(R)$.

Both sets are also used for defining the set of query answers.

Definition 4.6 [Query Answers] Given a database instance $\mathcal{D}$, θ is an *answer* of $\mathcal{Q}$ w.r.t. $\mathcal{D}$ (in symbols $(\mathcal{D}, \theta) \models_Q \mathcal{Q}$) in the following cases:

(i) $(\mathcal{D}, \theta) \models_Q e \bowtie e'$, if there exist $t \in \llbracket e \rrbracket^{\mathcal{D}}\theta$ and $t' \in \llbracket e' \rrbracket^{\mathcal{D}}\theta$, such that $t \downarrow t'$, and $t, t' \in adom(e, \mathcal{D}) \cup adom(e', \mathcal{D})$;

(ii) $(\mathcal{D}, \theta) \models_Q e <> e'$, if there exist $t \in \llbracket e \rrbracket^{\mathcal{D}}\theta$ and $t' \in \llbracket e' \rrbracket^{\mathcal{D}}\theta$, such that $t \uparrow t'$, and $t, t' \in adom(e, \mathcal{D}) \cup adom(e', \mathcal{D})$;

(iii) $(\mathcal{D}, \theta) \models_Q e \not\bowtie e'$ if for all $t \in \llbracket e \rrbracket^{\mathcal{D}}\theta$ and $t' \in \llbracket e' \rrbracket^{\mathcal{D}}\theta$, then $t \not\downarrow t'$, and $t, t' \in adom(e, \mathcal{D}) \cup adom(e', \mathcal{D})$;

(iv) $(\mathcal{D}, \theta) \models_Q e \not<> e'$, if for all $t \in \llbracket e \rrbracket^{\mathcal{D}}\theta$ and $t' \in \llbracket e' \rrbracket^{\mathcal{D}}\theta$, then $t \not\uparrow t'$, and $t, t' \in adom(e, \mathcal{D}) \cup adom(e', \mathcal{D})$.

Now, the *set of answers* of a safe query $\mathcal{Q}$ w.r.t. an instance $\mathcal{D}$, denoted by $Ans(\mathcal{D}, \mathcal{Q})$, is defined as follows: $Ans(\mathcal{D}, \mathcal{Q}) =_{def} \{(X_1\theta, \ldots, X_n\theta) \mid Dom(\theta) \subseteq var(\mathcal{Q}) = \{X_1, \ldots, X_n\}, (\mathcal{D}, \theta) \models_Q \mathcal{Q}\}$.

4.1 Calculus and Functional Logic Queries Equivalence

Now, we can state the equivalence of both query languages.

Table 3
Transformation Rules

$$\textbf{(1)}\quad \frac{\phi \wedge \exists \bar{z}.\psi \oplus e \bowtie e', \mathcal{Q}}{\phi \wedge \exists \bar{z}.\exists x.\exists y.\psi \wedge e \lhd x \;\wedge\; e' \lhd y \;\wedge\; x \Downarrow y \oplus \mathcal{Q}}$$

$$\textbf{(2)}\quad \frac{\phi \wedge \neg\exists \bar{z}.\psi \oplus e \not\bowtie e', \mathcal{Q}}{\phi \wedge \neg\exists \bar{z}.\exists x.\exists y.\psi \wedge e \lhd x \;\wedge\; e' \lhd y \;\wedge\; x \Downarrow y \oplus \mathcal{Q}}$$

$$\textbf{(3)}\quad \frac{\phi \wedge \exists \bar{z}.\psi \oplus e <> e', \mathcal{Q}}{\phi \wedge \exists \bar{z}.\exists x.\exists y.\psi \wedge e \lhd x \;\wedge\; e' \lhd y \;\wedge\; x \Uparrow y \oplus \mathcal{Q}}$$

$$\textbf{(4)}\quad \frac{\phi \wedge \neg\exists \bar{z}.\psi \oplus e \not<> e', \mathcal{Q}}{\phi \wedge \neg\exists \bar{z}.\exists x.\exists y.\psi \wedge e \lhd x \;\wedge\; e' \lhd y \;\wedge\; x \Uparrow y \oplus \mathcal{Q}}$$

$$\textbf{(5)}\quad \frac{\phi \wedge (\neg)\exists \bar{z}.\psi \wedge R\ e_1 \ldots e_k \lhd x \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{z}.\exists y_1.\ldots.\exists y_n.\psi \wedge R(y_1, \ldots, y_k, \ldots, y_n) \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_k \lhd y_k[x|ok] \oplus \mathcal{Q}}$$
$\%\ \mathtt{R} \in \mathtt{S}$

$$\textbf{(6)}\quad \frac{\phi \wedge (\neg)\exists \bar{z}.\psi \wedge A_i\ e_1 \ldots e_k \lhd x \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{z}.\exists y_1.\ldots.\exists y_n.\psi \wedge R(y_1, \ldots, y_k, \ldots, y_i, \ldots, y_n) \;\wedge\; e_1 \lhd y_1 \wedge \ldots \wedge e_k \lhd y_k \wedge y_i \lhd x \oplus \mathcal{Q}}$$
$\%\ \mathtt{A_i} \in \mathtt{NonKey(R)}$

$$\textbf{(7)}\quad \frac{\phi \wedge (\neg)\exists \bar{z}.\psi \wedge f\ e_1 \ldots e_n \lhd x \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{z}.\exists y_1 \ldots y_n.\psi \wedge f\ y_1 \ldots y_n \lhd x \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_n \lhd y_n \oplus \mathcal{Q}}$$
$\%\ \mathtt{f\ e_1 \ldots e_n} \notin \mathtt{Term_{DC,IF}}(\mathcal{V})$

$$\textbf{(8)}\quad \frac{\phi \wedge (\neg)\exists \bar{z}.\psi \wedge c(e_1, \ldots, e_n) \lhd x \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{z}.\exists y_1 \ldots y_n.\psi \wedge c(y_1, \ldots, y_n) \lhd x \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_n \lhd y_n \oplus \mathcal{Q}}$$
$\%\ \mathtt{c(e_1 \ldots e_n)} \notin \mathtt{Term_{DC,IF}}(\mathcal{V})$

$$\textbf{(9)}\quad \frac{\phi \wedge (\neg)\exists \bar{z}.\psi \wedge t \lhd x \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{z}.\psi \wedge x = t \oplus \mathcal{Q}}$$
$\%\ \mathtt{x} \in \mathtt{formula\_key}(\phi \wedge (\neg)\exists \bar{z}.\psi \wedge \mathtt{t} \lhd \mathtt{x})$

$$\textbf{(10)}\quad \frac{\phi \wedge (\neg)\exists \bar{z}.\exists x.\psi \wedge t \lhd x \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{z}.\psi[x|t] \oplus \mathcal{Q}}$$
$\%\ \mathtt{x} \notin \mathtt{formula\_key}(\phi \wedge (\neg)\exists \bar{z}.\exists x.\psi \wedge \mathtt{t} \lhd \mathtt{x})$

Theorem 4.7 (Queries and Calculus Formulas Equivalence) *Let $\mathcal{D}$ be an instance, then:*

(i) *given a safe query $\mathcal{Q}$ against $\mathcal{D}$, there exists a safe calculus formula $\varphi_{\mathcal{Q}}$ such that $Ans(\mathcal{D}, \mathcal{Q}) = Ans(\mathcal{D}, \varphi_{\mathcal{Q}})$*

(ii) *given a safe calculus formula φ against $\mathcal{D}$, there exists a safe query $\mathcal{Q}_\varphi$ such that $Ans(\mathcal{D}, \varphi) = Ans(\mathcal{D}, \mathcal{Q}_\varphi)$*

Proof. *The idea is to transform a* safe query *into a* safe calculus formula *and viceversa, applying the set of transformation rules of table 3. In order to transform a safe query $\mathcal{Q}$ into a safe calculus formula $\varphi_{\mathcal{Q}}$, we have to apply the transformation rules in top-down, starting from $\mathcal{Q}$. Analogously, in order to transform a safe calculus formula φ into a safe query $\mathcal{Q}_\varphi$, we have to apply the transformation rules in bottom-up, starting from φ. Now, given*

$$\frac{\phi \oplus \ \mathcal{Q}}{\phi^* \oplus \mathcal{Q}^*}$$

and a database instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$, we have to prove:

(a) *there exists a substitution η, such that $\bar{x}\eta \in Ans(\mathcal{D}, \phi) \cap Ans(\mathcal{D}, \mathcal{Q})$ where $\bar{x} = free(\phi) \cup var(\mathcal{Q})$ iff there exists a substitution η^*, such that $\bar{x}\eta^* \in Ans(\mathcal{D}, \phi^*) \cap Ans(\mathcal{D}, \mathcal{Q}^*)$ where $\bar{x} = free(\phi^*) \cup var(\mathcal{Q}^*)$ and $\eta = \eta^*|_{free(\phi)\cup var(\mathcal{Q})}$.*
Here, $\bar{x}\eta$ denotes a tuple $(x_1\eta, \ldots, x_n\eta)$ and we write $\bar{x}\eta \in Ans(\mathcal{D}, \varphi) \cap Ans(\mathcal{D}, \mathcal{Q})$ whenever $(\mathcal{D}, \eta) \models_C \varphi$ and $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$; finally, $\eta^*|_{free(\phi)\cup var(\mathcal{Q})}$ expresses the substitution restricted to the variables of $\mathcal{Q}$ and the free variables of ϕ.

(b) *ϕ is a safe calculus formula and $\mathcal{Q}$ is a safe query iff ϕ^* is a safe calculus formula and $\mathcal{Q}^*$ is a safe query where, here, the safety condition is:*
- *the c-terms of the queries are range restricted by definition 3.4 and by definition 4.2;*
- *the c-terms of the calculus formulas are range restricted by definition 3.4 and definition 4.2;*
- *the equations $e_1 \diamondsuit_q e_2 \in \mathcal{Q}$ do not contain variables from $formula\_key$ (ϕ);*
- *the safety condition of atomic formulas (definition 3.3) is replaced by: "$R(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$ is safe, if the variables $x_1, \ldots, x_n$ are bound in φ, and for each x_i, $i \leq nKey(R)$, there exists one equation $e_i \lhd x_i$ or $x_i = t_i$ occurring in φ";*

Note that this safety definition is more general. However, whether $\phi = \emptyset$ or $\mathcal{Q} = \emptyset$, then the safety condition coincides with the original definitions (see definitions 4.2 and 3.4, respectively).

Here, we prove the main cases of (a) and (b).

$$\textbf{(1)}\ \frac{\phi \wedge \exists \bar{\mathsf{z}}.\ \psi \oplus \mathsf{e}_1 \bowtie \mathsf{e}_2,\ \ \mathcal{Q}}{\phi \wedge \exists \bar{\mathsf{z}}.\exists \mathsf{x}.\exists \mathsf{y}.\ \psi \wedge \mathsf{e}_1 \lhd \mathsf{x}\ \ \wedge\ \ \mathsf{e}_2 \lhd \mathsf{y}\ \ \wedge\ \ \mathsf{x} \Downarrow \mathsf{y} \oplus\ \mathcal{Q}}$$

(a) *Given a substitution η such that $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge \exists \bar{z}.\psi) \cap Ans(\mathcal{D}, \{e_1 \bowtie e_2,\ \mathcal{Q}\})$, then $(\mathcal{D}, \eta) \models_C \phi \wedge \exists \bar{z}.\psi$, $(\mathcal{D}, \eta) \models_Q e_1 \bowtie e_2$ and $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$. In particular, $(\mathcal{D}, \eta) \models_Q e_1 \bowtie e_2$ iff there exists $t_1 \in [\![e_1]\!]^{\mathcal{D}}\eta$ and $t_2 \in [\![e_2]\!]^{\mathcal{D}}\eta$ such that $t_1 \downarrow t_2$ and $t_1, t_2 \in adom(e_1, \mathcal{D}) \cup adom(e_2, \mathcal{D})$. Now, let η^* be a substitution such that $\eta^* = \eta \cdot \{x|t_1,\ y|t_2\}$, then $x\eta^* \in [\![e_1]\!]^{\mathcal{D}}\eta^*$ and $y\eta^* \in [\![e_2]\!]^{\mathcal{D}}\eta^*$ and therefore iff $(\mathcal{D}, \eta^*) \models_C e_1 \lhd x \wedge e_2 \lhd y$. In addition, by definition (3.7), $adom(x, \mathcal{D}) = adom(e_1, \mathcal{D})$ and $adom(y, \mathcal{D}) = adom(e_2, \mathcal{D})$ and given that $x\eta^* \downarrow y\eta^*$, then $x\eta^*, y\eta^* \in adom(x, \mathcal{D}) \cup adom(y, \mathcal{D})$ and thus iff $(\mathcal{D}, \eta^*) \models_C x \Downarrow y$. Therefore $(\mathcal{D}, \eta^*) \models_C (e_1 \lhd x \wedge e_2 \lhd y \wedge x \Downarrow y)$ and, finally, $(\mathcal{D}, \eta) \models_C \phi$, $(\mathcal{D}, \eta) \models_C (\exists \bar{z}.\exists x.\exists y.\ \psi \wedge e_1 \lhd x \wedge e_2 \lhd y \wedge x \Downarrow y)$ and $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$ so that, iff $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge (\exists \bar{z}.\exists x.\exists y.\ \psi \wedge e_1 \lhd x \wedge$*

$e_2 \lhd y \wedge x \Downarrow y)) \cap Ans(\mathcal{D}, \mathcal{Q})$.

(b) *Suppose that* $\phi \wedge \exists \bar{z}.\psi$, *and* $e_1 \bowtie e_2$, $\mathcal{Q}$ *are safe, that is,*

- *the equations and atomic formulas of* ϕ *and* ψ *are safe*
- *the c-terms of* ϕ *and* $\mathcal{Q}$ *are range restricted*
- *the c-terms of* e_1 *and* e_2 *are range restricted*

then applying **(1)***:*

- *the equations and atomic formulas of* ϕ *and* ψ *are safe*
- *those range restricted c-terms in* ϕ, ψ *and* $\mathcal{Q}$ *by means of* $e_1 \bowtie e_2$, *are now range restricted by means of* $e_1 \lhd x$, $e_2 \lhd y$, $x \Downarrow y$
- *the formula* $\exists \bar{z}.\exists x.\exists y.\ \psi \wedge e_1 \lhd x \wedge e_2 \lhd y \wedge x \Downarrow y$ *is safe, given that, by hypothesis, the c-terms of* e_1 *and* e_2 *are range restricted and, therefore, the variables* x *and* y *are range restricted. In addition, the equations* $e_1 \lhd x$, $e_2 \lhd y$, $x \Downarrow y$ *are safe, given that* e_1 *and* e_2 *do not contain, by hypothesis, key variables and the variables* x *and* y *are variables distinct from key variables due to the renaming of quantified variables.*

$$\textbf{(6)}\quad \frac{\phi \wedge (\neg)\exists \bar{\mathsf{z}}.\psi \wedge \mathsf{A_i}\ \mathsf{e_1} \ldots \mathsf{e_k} \lhd \mathsf{x} \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{\mathsf{z}}.\exists \mathsf{y_1}.\ldots.\exists \mathsf{y_n}.\psi \wedge \mathsf{R}(\mathsf{y_1},\ldots,\mathsf{y_k},\ldots,\mathsf{y_i},\ldots,\mathsf{y_n}) \ \wedge\ \mathsf{e_1} \lhd \mathsf{y_1} \wedge \ldots \wedge \mathsf{e_k} \lhd \mathsf{y_k} \wedge \mathsf{y_i} \lhd \mathsf{x} \oplus \mathcal{Q}}$$

$\%\ \mathtt{A_i} \in \mathtt{NonKey(R)}$

(a) *Given a substitution* η, *such that* $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge \exists \bar{z}.\ \psi \wedge A_i\ e_1 \ldots e_k \lhd x) \cap Ans(\mathcal{D}, \mathcal{Q})$, *then* $(\mathcal{D}, \eta) \models_C \phi$, $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi \wedge A_i\ e_1 \ldots e_k \lhd x$ *and* $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$. *Now,* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi \wedge A_i\ e_1 \ldots e_k \lhd x$ *iff there exists a substitution* η' *such that* $(\mathcal{D}, \eta') \models_C A_i\ e_1 \ldots e_k \lhd x$. *Therefore iff* $x\eta' \in [\![A_i\ e_1 \ldots e_k]\!]^{\mathcal{D}}\eta'$ *that is,* $v_i = x\eta' \in V_i\eta_V$ *for a given substitution* η_V, *whenever* $([\![e_1]\!]^{\mathcal{D}}\eta', \ldots, [\![e_k]\!]^{\mathcal{D}}\eta') = (V_1\eta_V, \ldots, V_k\eta_V)$ *and there exists a tuple* $(V_1, \ldots, V_k, \ldots, V_i, \ldots, V_n) \in \mathcal{R}$. *Now, let* η^* *be a substitution, such that* $\eta^* = \eta' \cdot \{y_1|v_1, \ldots, y_n|v_n\}$ *and* $v_1 \in V_1\eta_V, \ldots, v_n \in V_n\eta_V$; *therefore, iff* $(\mathcal{D}, \eta^*) \models_C R(y_1, \ldots, y_n)$ *and given that* $y_1\eta^* \in [\![e_1]\!]^{\mathcal{D}}\eta^* \ \ldots\ y_k\eta^* \in [\![e_k]\!]^{\mathcal{D}}\eta^*$ *then iff* $(\mathcal{D}, \eta^*) \models_C e_i \lhd y_i$. *Finally, given that* $y_i\eta^* = x\eta$ *then iff* $(\mathcal{D}, \eta^*) \models_C y_i \lhd x$ *and we can prove* $(\mathcal{D}, \eta^*) \models_C R(y_1, \ldots, y_k, \ldots, y_i, \ldots, y_n) \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_k \lhd y_k \wedge y_i \lhd x$. *Finally,* $(\mathcal{D}, \eta) \models_C \phi$, $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\exists y_1.\ldots\exists y_n.\ \psi \wedge R(y_1, \ldots, y_k, \ldots, y_i, \ldots, y_n) \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_k \lhd y_k \wedge y_i \lhd x$ *and* $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$, *and therefore iff* $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge (\exists \bar{z}.\exists y_1 \ldots \exists y_n.\ \psi \wedge R(y_1, \ldots, y_k, \ldots, y_i, \ldots, y_n) \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_k \lhd y_k \wedge y_i \lhd x)) \cap Ans(\mathcal{D}, \mathcal{Q})$ *where* $\eta = \eta^*|_{var(\mathcal{Q}) \cup free(\phi)}$.

(b) *Suppose that* ϕ, $(\exists \bar{z}.\ \psi \wedge A_i\ e_1 \ldots e_k \lhd x)$, *and* $\mathcal{Q}$ *are safe; that is,*

- *the equations and atomic formulas of* ϕ *and* ψ *are safe*
- *the c-terms of* ϕ, ψ *and* $\mathcal{Q}$ *are range restricted*
- *the c-terms of* $e_1, \ldots, e_k$ *are range restricted, and the equation* $A_i\ e_1 \ldots$

$e_k \lhd x$ *is safe; that is,* $e_1 \ldots e_k$ *do not contain key variables, and the variable* x *is bounded and range restricted*

then applying **(6)***:*

- *the equations and atomic formulas of* ϕ *and* ψ *are safe by the renaming of quantified variables*
- *the c-terms of* $\mathcal{Q}$*,* ϕ *and* ψ *are range restricted, now, by means of* $R(y_1, \ldots, y_n)$*,* $e_1 \lhd y_1, \ldots, e_k \lhd y_k$*, and* $y_i \lhd x$ *if they were range restricted by means of* $A_i\ e_1 \ldots e_k \lhd x$
- *the formula* $(\exists \bar{z}.\exists y_1 .\ldots \exists y_n.\ \psi \wedge R(y_1, \ldots, y_k, \ldots, y_n) \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_k \lhd y_k \wedge y_i \lhd x)$ *is safe, given that the c-terms of* $e_1, \ldots, e_k$ *and the variables* $y_1, \ldots, y_n, x$ *are range restricted; in addition, the equations* $e_1 \lhd y_1 \wedge \ldots \wedge e_k \lhd y_k \wedge y_i \lhd x$ *are safe, given that the variables* $y_1, \ldots, y_k, y_i$ *are bounded, the variable* x *is bounded by hypothesis,* $e_1, \ldots, e_k$ *do not contain key variables by hypothesis, and the variable* y_i *is not a key variable. Finally, the atomic formula* $R(y_1, \ldots, y_k, \ldots, y_i, \ldots, y_n)$ *contains new variables by the renaming of quantified variables; moreover, for each* y_i*,* $(1 \le j \le k)$*, there exists an equation* $e_i \lhd y_i$*.*

$$\textbf{(7)}\quad \frac{\phi \wedge (\neg)\exists \bar{\mathsf{z}}.\psi \wedge \mathsf{f}\ \mathsf{e_1} \ldots \mathsf{e_n} \lhd \mathsf{x} \oplus \mathcal{Q}}{\phi \wedge (\neg)\exists \bar{\mathsf{z}}.\exists \mathsf{y_1} \ldots \mathsf{y_n}.\psi \wedge \mathsf{f}\ \mathsf{y_1} \ldots \mathsf{y_n} \lhd \mathsf{x} \wedge \mathsf{e_1} \lhd \mathsf{y_1} \wedge \ldots \wedge \mathsf{e_n} \lhd \mathsf{y_n} \oplus \mathcal{Q}}$$

$\%\ \mathtt{f}\ \mathsf{e_1} \ldots \mathsf{e_n} \notin \mathtt{Term}_{\mathtt{DC,IF}}(\mathcal{V})$

(a) *Given a substitution* η*, such that* $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge \exists \bar{z}.\ \psi \wedge f\ e_1 \ldots e_n \lhd x) \cap Ans(\mathcal{D}, \mathcal{Q})$*, then* $(\mathcal{D}, \eta) \models_C \phi$*,* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi \wedge f\ e_1 \ldots e_n \lhd x$ *and* $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$*. Now,* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi \wedge f\ e_1 \ldots e_n \lhd x$ *iff there exists a substitution* η' *such that* $(\mathcal{D}, \eta') \models_C f\ e_1 \ldots e_n \lhd x$*. Therefore* $x\eta' \in [\![f\ e_1 \ldots e_n]\!]^{\mathcal{D}}\eta'$*, that is,* $x\eta' \in f^{\mathcal{D}}\ [\![e_1]\!]^{\mathcal{D}}\eta' \ldots [\![e_n]\!]^{\mathcal{D}}\eta'$*. Now, there exist c-terms* $t_1, \ldots, t_n$*, such that* $t_1 \in [\![e_1]\!]^{\mathcal{D}}\eta' \ldots t_n \in [\![e_n]\!]^{\mathcal{D}}\eta'$ *and therefore iff* $x\eta' \in f^{\mathcal{D}}\ t_1 \ldots t_n$*. Now, let* η^* *be a substitution, such that* $\eta^* = \eta' \cdot \{y_1|t_1, \ldots, y_n|t_n\}$ *then, we have that* $y_1\eta^* \in [\![e_1]\!]^{\mathcal{D}}\eta^* \ldots y_n\eta^* \in [\![e_n]\!]^{\mathcal{D}}\eta^*$*. Finally, given that* $x\eta' \in f^{\mathcal{D}}\ t_1 \ldots t_n$*, then iff* $x\eta^* \in f^{\mathcal{D}}\ [\![y_1]\!]^{\mathcal{D}}\eta^* \ldots [\![y_n]\!]^{\mathcal{D}}\eta^*$*; that is,* $x\eta^* \in [\![f\ y_1 \ldots y_n]\!]^{\mathcal{D}}\eta^*$ *iff* $(\mathcal{D}, \eta) \models_C \phi$*,* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\exists y_1 \ldots \exists y_n.\ \psi \wedge f\ y_1 \ldots y_n \lhd x \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_n \lhd y_n$*, and* $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$*. Therefore iff* $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge (\exists \bar{z}.\exists y_1 \ldots \exists y_n.\ \psi \wedge f\ y_1 \ldots y_n \lhd x \wedge e_1 \lhd y_1 \wedge \ldots \wedge e_n \lhd y_n)) \cap Ans(\mathcal{D}, \mathcal{Q})$ *where* $\eta = \eta^*|_{var(\mathcal{Q}) \cup free(\phi)}$*.*

(b) *Suppose that* ϕ*,* $(\exists \bar{z}.\ \psi \wedge f\ e_1 \ldots e_n \lhd x)$*, and* $\mathcal{Q}$ *are safe; that is,*
- *the equations and atomic formulas of* ϕ *and* ψ *are safe*
- *the c-terms of* ϕ*,* ψ *and* $\mathcal{Q}$ *are range restricted*
- *the c-terms of* $e_1, \ldots, e_n$ *are range restricted, and the equation* $f\ e_1 \ldots e_n \lhd x$ *is safe; that is,* $e_1, \ldots, e_n$ *do not contain key variables, and the variable* x *is bounded and range restricted*

then applying **(7)***:*
- *the equations and atomic formulas of* ϕ *and* ψ *are safe by the renaming*

of quantified variables

- *the c-terms of* $\mathcal{Q}$, ϕ *and* ψ *are range restricted if they were range restricted by means of* $f\ e_1\ \ldots\ e_n \lhd x$
- *the formula* $(\exists \bar{z}.\exists y_1.\ldots.\exists y_n.\ \psi\ \wedge\ f\ y_1 \ldots y_n \lhd x\ \ \wedge\ \ e_1 \lhd y_1 \wedge \ldots \wedge e_n \lhd y_n)$ *is safe given that the c-terms of* $e_1, \ldots, e_n$ *are range restricted and therefore the variables* $y_1, \ldots, y_n$ *are also range restricted; the equations* $f\ y_1 \ldots y_n \lhd x\ \ \wedge\ \ e_1 \lhd y_1 \wedge \ldots \wedge e_n \lhd y_n$ *are safe, given that the variables* $y_1, \ldots, y_n$ *are bounded, the variable* x *is bounded by hypothesis, and* $e_1, \ldots, e_n$*, by hypothesis, do not contain key variables.*

$$\textbf{(9)} \qquad \frac{\phi\ \wedge\ \exists \bar{\mathsf{z}}.\ \psi \wedge \mathsf{t} \lhd \mathsf{x} \oplus \mathcal{Q}}{\phi\ \wedge\ \exists \bar{\mathsf{z}}.\ \psi \wedge \mathsf{x} = \mathsf{t} \oplus \mathcal{Q}}$$

% $\mathtt{x} \in \mathtt{formula\_key}(\phi \wedge \exists \bar{\mathtt{z}}.\psi \wedge \mathtt{t} \lhd \mathtt{x})$ `y t is a c-term`

(a) *Given a substitution* η*, such that* $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge \exists \bar{z}.\ \psi\ \wedge\ t\ \lhd\ x) \cap Ans(\mathcal{D}, \mathcal{Q})$*, then* $(\mathcal{D}, \eta) \models_C \phi$*,* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi\ \wedge\ t\ \lhd\ x$ *and* $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$*. Now,* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi\ \wedge\ t \lhd x$ *iff there exists a substitution* η' *such that* $(\mathcal{D}, \eta') \models_C t \lhd x$*. Therefore,* $x\eta' \in [\![t]\!]^{\mathcal{D}}\eta' = \{t\eta'\}$ *and then* $x\eta' = t\eta'$*. Now, given that* x *is a key variable, then there exists an atomic formula* $R(y_1, \ldots, x, \ldots, y_n)$ *in the calculus formula and a tuple* $(V_1, \ldots, V_{i-1}, V_i, V_{i+1}, \ldots, V_k, \ldots, V_n) \in \mathcal{R}$ *such that* $x\eta' \in V_i\eta_V$ *for a given substitution* η_V*; now, given that* $x \in formula\_key(\phi \wedge (\neg)\exists \bar{z}.\psi \wedge t \lhd x)$ *then* $adom(x, \mathcal{D}) \supseteq V_i\eta_V$ *and* $t\eta' \in V_i\eta_V$*. Therefore iff* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi\ \ \wedge\ \ x = t$ *and thus* $(\mathcal{D}, \eta) \models_C \phi$*,* $(\mathcal{D}, \eta) \models_C \exists \bar{z}.\ \psi\ \ \wedge\ \ x = t$ *and* $(\mathcal{D}, \eta) \models_Q \mathcal{Q}$ *which is true iff* $\bar{x}\eta \in Ans(\mathcal{D}, \phi \wedge (\exists \bar{z}.\ \psi\ \wedge\ x = t)) \cap Ans(\mathcal{D}, \mathcal{Q})$*.*

(b) *Suppose that* ϕ*,* $(\exists \bar{z}.\ \psi\ \wedge\ t \lhd x)$*, and* $\mathcal{Q}$ *are safe; that is,*
- *the equations and atomic formulas of* ϕ *and* ψ *are safe*
- *the c-terms of* ϕ*,* ψ *and* $\mathcal{Q}$ *are range restricted*
- *the c-terms of* t *are range restricted,* x *is a key variable, thus range restricted and, finally, the equation* $t \lhd x$ *is safe; that is,* x *is bounded and* t *does not contain key variables*

then applying **(9)***:*
- *the equations and atomic formulas of* ϕ *and* ψ *are safe by hypothesis*
- *the c-terms of* $\mathcal{Q}$*,* ϕ *and* ψ *are range restricted by means of* $x = t$ *if they were by means of* $t \lhd x$*; the rest of variables by hypothesis, and thus,* $\mathcal{Q}$*,* ϕ *and* ψ *are safe*
- *the formula* $\exists \bar{z}.\ \psi\ \wedge\ x = t$ *is safe given that the c-terms of* t *are range restricted by hypothesis; the equation* $x = t$ *is safe, given that* x *is a key variable and* t *does not contain, by hypothesis, key variables.*

Now, in order to prove the theorem, we prove that:
(i) if $(\emptyset \oplus \mathcal{Q}) \to^n (\varphi_{\mathcal{Q}} \oplus \emptyset)$ *then:*

(a) $\bar{x}\eta \in Ans(\mathcal{D}, \mathcal{Q})$ *iff there exists a substitution* η^* *such that* $\bar{x}\eta^* \in Ans(\mathcal{D}, \varphi_{\mathcal{Q}})$ *where* $\eta^* = \eta|_{var(\mathcal{Q})}$

(b) $\mathcal{Q}$ *is safe w.r.t the definition 4.3 iff* $\varphi_{\mathcal{Q}}$ *is safe w.r.t. the definition 3.5*

(ii) if $(\varphi \oplus \emptyset) \to^n (\emptyset \oplus \mathcal{Q}_\varphi)$ *then:*

(a) $\bar{x}\eta \in Ans(\mathcal{D}, \varphi)$ *iff there exists a substitution* η^* *such that* $\bar{x}\eta^* \in Ans(\mathcal{D}, \mathcal{Q}_\varphi)$ *where* $\eta^* = \eta|_{free(\varphi)}$

(b) φ *is safe w.r.t. the definition 3.5 iff* $\mathcal{Q}_\varphi$ *is safe w.r.t. the definition 4.3*

We prove (i) that is, $(\emptyset \oplus \mathcal{Q}) \to^n (\varphi_{\mathcal{Q}} \oplus \emptyset)$*; analogously, we can prove (ii).*

(a) Let η *be a substitution such that* $\bar{x}\eta \in Ans(\mathcal{D}, \mathcal{Q})$*, then for each transformation step*

$$\frac{\phi \oplus \mathcal{Q}}{\phi^* \oplus \mathcal{Q}^*}$$

there exists a substitution $\eta^* = \eta|_{var(\mathcal{Q}) \cup free(\varphi)}$ *such that* $\bar{x}\eta^* \in Ans(\mathcal{D}, \phi^*) \cap Ans(\mathcal{D}, \mathcal{Q}^*)$*. Therefore, iterating we can conclude the result*

(b) We have that the formula φ *and query* $\mathcal{Q}$ *are safe, iff the formula* φ^* *and the query* $\mathcal{Q}^*$ *are safe. Now, if* $\mathcal{Q}$ *is safe (definition 4.3), we have that is also safe w.r.t. the definition of safety proposed in this theorem. Therefore,* $\varphi_{\mathcal{Q}}$ *is safe and, thus it is safe w.r.t. the definition 3.5*

$\square$

5 Domain Independence

In this section, we will prove the domain independence property over the functional-logic query language, and therefore, by the previously proved equivalence, over the extended relational calculus. Firstly, we need to define some concepts.

A database instance defines a domain which consists on the values of the tuples, c-terms built from these values and data constructors, and finally, the obtained values applying interpreted functions over these values. In particular, we can define the domain of a given attribute, which consists on the set of values of the corresponding attribute in a given database instance.

Definition 5.1 [Domain of an Instance] Given a database instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ of a database $D = (S, DC, IF)$, we define the *domain of* $\mathcal{D}$, denoted by $Dom(\mathcal{D})$, as follows:

$$\begin{aligned} Dom(\mathcal{D}) =_{def} & \{ t \mid (V_1, \ldots, V_n) \in \mathcal{R},\, \eta \in Subst_{DC,\perp,\mathsf{F}},\, t \in cterms(V_i\eta),\, \mathcal{R} \in \mathcal{S} \} \\ & \cup \{ c(t_1, \ldots, t_n) \mid t_i \in Dom(\mathcal{D}),\ c \in DC^n, n > 0 \} \\ & \cup \{ f^{\mathcal{D}}\ t_1 \ldots t_n \mid t_i \in Dom(\mathcal{D}),\ f \in IF^n \} \\ & \cup \{ t_i \mid f^{\mathcal{D}}\ t_1 \ldots t_n = t,\ t \in Dom(\mathcal{D}) \textit{ and } f \in IF^n \} \\ & \cup \{ \mathtt{ok}, \perp, \mathsf{F} \} \end{aligned}$$

Definition 5.2 [Domain of an Attribute] Given a database instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ of a database $D = (S, DC, IF)$, we define the *domain of an attribute* $A_i \in Key(R) \cup NonKey(R), R \in S$, denoted by $Dom(\mathcal{D}, A_i)$, as follows:

$$Dom(\mathcal{D}, A_i) =_{def} \{ t \mid (V_1, \ldots, V_n) \in \mathcal{R},\ \eta \in Subst_{DC,\perp,\mathsf{F}},\, t \in cterms(V_i\eta) \}$$

Remark that in both definitions, tuples can include variables, and thus they can be instantiated by mean of substitutions.

Definition 5.3 [Finite Instances]
An instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ of a database $D = (S, DC, IF)$ is *finite*, if $\mathcal{S}$ and $\mathcal{IF}$ are finite, where:

(i) $\mathcal{S}$ is *finite* iff:
  (a) $\mathcal{S}$ contains a finite set of tuples $(V_1, \ldots, V_k, \ldots, V_n)$, where $k = nKey(R)$ and $R \in S$; and in addition,
  (b) $\mathcal{S}$ is *ground* (and thus $\mathcal{D}$ is *ground*); that is, the values $V_1, \ldots, V_k$ are ground and, finally, $V_{k+1}, \ldots, V_n$ are finite, and their values are ground and finite;

(ii) $\mathcal{IF}$ is finite, if for each function symbol $f \in IF$, then the set $\{t \mid f^{\mathcal{D}}\ s_1 \ldots s_n = t\} \cup \{t_1, \ldots, t_n \mid f^{\mathcal{D}}\ t_1 \ldots t_n = s\}$ is a finite set of finite c-terms for any s_i, $s \in Dom(\mathcal{D})$.

Definition 5.4 [Instance Inclusions]
Given two instances $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ and $\mathcal{D}^* = (\mathcal{S}, \mathcal{DC}^*, \mathcal{IF}^*)$ of two databases $D^* = (S, DC^*, IF^*)$ and $D = (S, DC, IF)$ then we say that $\mathcal{D}$ *is included in* $\mathcal{D}^*$, denoted by $\mathcal{D} \subseteq \mathcal{D}^*$, iff $\mathcal{DC} \subseteq \mathcal{DC}^*$ and $\mathcal{IF} \subseteq \mathcal{IF}^*$ where:

(a) $\mathcal{DC} \subseteq \mathcal{DC}^*$, if $DC \subseteq DC^*$

(b) $\mathcal{IF} \subseteq \mathcal{IF}^*$, if for each function symbol $f \in IF$, then $f^{\mathcal{D}^*}\ s_1 \ldots s_n = f^{\mathcal{D}}\ s_1 \ldots s_n$, and $\{\bar{t} \mid f^{\mathcal{D}^*}\ t_1 \ldots t_n = s\} = \{\bar{t} \mid f^{\mathcal{D}}\ t_1 \ldots t_n = s\}$, for any s_i, $s \in Dom(\mathcal{D})$

Now, we can formally define the property of *domain independence*.

Definition 5.5 [Domain Independence] A calculus formula φ is *domain independent* whenever:

(a) if the instance $\mathcal{D}$ is finite, then $Ans(\mathcal{D}, \varphi)$ is finite; and

(b) given two ground instances $\mathcal{D} \subseteq \mathcal{D}^*$, then $Ans(\mathcal{D}, \varphi) = Ans(\mathcal{D}^*, \varphi)$.

The case (a) establishes that the set of answers is finite, whenever $\mathcal{S}$ and $\mathcal{IF}$ are finite; and (b) states that the output relation (i.e. set of answers) only depends on the input schema instance $\mathcal{S}$, and not on the data constructors (i.e. $\mathcal{DC}$) and interpreted functions (i.e. $\mathcal{IF}$).

In order to prove the property of domain independence, we need some previous results.

Proposition 5.6 *Given a database instance $\mathcal{D}$, a term $e \in Term_D(\mathcal{V})$ and a query $\mathcal{Q}$, then:*

(a) $adom(e, \mathcal{D}) \subseteq Dom(\mathcal{D})$

(b) if for all $t \in CTerm_{DC,\mathsf{F}}(\mathcal{V})$ occurring in e, we have that $t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_i)$ for a given key attribute A_i, then:

$$[\![e]\!]^{\mathcal{D}}\eta \subseteq Dom(\mathcal{D})$$

for every substitution $\eta \in Subst_{DC,\perp,\mathsf{F}}$ such that $t\eta \in Dom(\mathcal{D}, A_i)$ for every $t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_i)$.

Proof. *The case (a) can be easily proved by analyzing the definitions 3.7 and 5.1. The case (b) can be proved by observing that if $t \in query\_key(\mathcal{Q}, A_i)$ then $[\![t]\!]^{\mathcal{D}}\eta \subseteq Dom(\mathcal{D})$, and therefore, proceeding by induction, it can be proved that $[\![e]\!]^{\mathcal{D}}\eta \subseteq Dom(\mathcal{D})$, whenever for all $t \in CTerm_{DC,\mathsf{F}}(\mathcal{V})$ occurring in e, we have that $t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_i)$* □

Lemma 5.7 (Finiteness) *Given a finite instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ of a database $D = (S, DC, IF)$, a term $e \in Term_D(\mathcal{V})$, and a query $\mathcal{Q}$, then:*

(a) $adom(e, \mathcal{D})$ is finite

(b) if for all $t \in CTerm_{DC,\mathsf{F}}(\mathcal{V})$ occurring in e, we have that $t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_i)$ for a given key attribute A_i, then the set

$$\{\eta \mid Dom(\eta) \subseteq var(e),\ \{\perp\} \neq [\![e]\!]^{\mathcal{D}}\eta,\ t\eta \in Dom(\mathcal{D}, A_i),$$
$$for\ every\ t \in cterms(s)\ with\ s \in query\_key(\mathcal{Q}, A_i)\}$$

is finite

Proof. *By structural induction over e. We analyze the main cases:*

(i) *$e \equiv t$ and $t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_i)$ for a given key attribute $A_i \in R$, $(R \in S)$, then:*

(a)

$$adom(t, \mathcal{D}) =_{def} \{\ t \mid t \in cterms(V_i\psi^*), \psi^* \in Subst_{DC,\perp,\mathsf{F}}, (V_1, \ldots, V_i, \ldots, V_n) \in \mathcal{R}\},$$

and given that $\mathcal{S}$ is finite (i.e. it contains a finite number of tuples and $V_i's$ are ground), then

$\{\ t \mid t \in cterms(V_i\psi^*), \psi^* \in Subst_{DC,\perp,\mathsf{F}}, (V_1, \ldots, V_i, \ldots, V_n) \in \mathcal{R}\}$

is finite, and we can conclude that $adom(e, \mathcal{D})$ *is finite.*

(b) *We have that* $\| e \|^{\mathcal{D}} \eta =_{def} \{t\eta\}$ *and* $t\eta \in Dom(\mathcal{D}, A_i)$*, and given that* $\mathcal{D}$ *is finite, then* $Dom(\mathcal{D}, A_i)$ *is finite by reasoning as previously, and therefore we can conclude that we have a finite set of substitutions* η*.*

(ii) $e \equiv t$ *and* $t \notin cterms(s)$ *for all* $s \in query\_key(\mathcal{Q}, A_i)$*, then:*

(a) $adom(e, \mathcal{D}) =_{def} \{\bot\}$ *is finite.*

(b) *It contradicts that every c-term of e is a subterm of a query key.*

(iii) *if* $e \equiv R\ e_1 \ldots e_k$ $(R \in S)$*, then:*

(a) $adom(R\ e_1 \ldots e_k, \mathcal{D}) =_{def} \{\mathtt{ok}, \mathsf{F}, \bot\}$ *is finite.*

(b) $\| e \|^{\mathcal{D}} \eta =_{def} \{\mathtt{ok}\}$*, if* $(\| e_1 \|^{\mathcal{D}} \eta, \ldots, \| e_k \|^{\mathcal{D}} \eta) = (V_1\eta^*, \ldots, V_k\eta^*)$*, where* $(V_1, \ldots, V_n) \in \mathcal{R}$*. Now, given that* $\mathcal{S}$ *is finite, we have two cases:*

(b.1) $e_i \equiv t_i$*, where* $t_i \in cterms(s_i)$ *with* $s_i \in query\_key(\mathcal{Q}, A_i)$*, then* $t_i\eta \in Dom(\mathcal{D}, A_i)$*; now, we have that* $\| e_i \|^{\mathcal{D}} \eta = \{t_i\eta\}$*. In addition* $t_i\eta$ *should be of* $V_i\eta^*$*, and* $V_i\eta^* \subseteq Dom(\mathcal{D}, A_i)$*, which is finite, and, therefore, we conclude that we have a finite set of substitutions* η

(b.2) *every c-term of* e_i *is a subterm of a query key, then by induction hypothesis we have that* $\{\eta \mid Dom(\eta) \subseteq var(e_i),\ \{\bot\} \neq \| e_i \|^{\mathcal{D}} \eta, t\eta \in Dom(\mathcal{D}, A_i)\ for\ each\ t \in cterms(s)\ with\ s \in query\_key(\mathcal{Q}, A_i)\}$ *is finite; therefore we have a finite set of substitutions* η*.*

(iv) *if* $e \equiv A_i\ e_1 \ldots e_k$ $(A_i \in NonKey(R),\ R \in S)$*, then:*

(a) $adom(A_i\ e_1 \ldots e_k, \mathcal{D}) =_{def} \bigcup_{\{\eta^* \in Subst_{DC,\bot,\mathsf{F}}, (V_1, \ldots, V_k, \ldots, V_i, \ldots, V_n) \in \mathcal{R}\}} V_i\eta^*$

In this case, given that $\mathcal{S}$ *is finite (i.e. contains a finite number of tuples and* V_i*'s are ground), then* $V_i\eta^* = V_i$*, and we can conclude that* $adom(A_i\ e_1 \ldots e_k, \mathcal{D})$ *es finite.*

(b) *Similarly to the previous case.*

(v) *if* $e \equiv c(e_1, \ldots, e_n)$*, then:*

(a) $adom(c(e_1, \ldots, e_n), \mathcal{D}) =_{def} c^{\mathcal{D}}(adom(e_1, \mathcal{D}), \ldots, adom(e_n, \mathcal{D}))$ *where* $c \in DC^n$*; now, by induction hypothesis, we have that* $adom(e_i, \mathcal{D})$ *is finite and, therefore, we can conclude that* $adom(c(e_1, \ldots, e_n), \mathcal{D})$ *is finite.*

(b) *Given that every c-term of* e_i *is a subterm of query key, we can conclude by induction hypothesis that* $\{\eta \mid Dom(\eta) \subseteq var(e),\ \{\bot\} \neq \| e \|^{\mathcal{D}} \eta,\ t\eta \in Dom(\mathcal{D}, A_i)\ for\ every\ t \in cterms(s)\ with\ s \in query\_key(\mathcal{Q}, A_i)\}$ *is finite.*

(vi) *if* $e \equiv f\ e_1\ \ldots\ e_n$ *, then:*

(a) $adom(f\ e_1\ \ldots\ e_n, \mathcal{D}) =_{def} f^{\mathcal{D}}\ adom(e_1, \mathcal{D})\ \ldots\ adom(e_n, \mathcal{D})$ *where* $f \in IF^n$*; now, by induction hypothesis, we have that:* $adom(e_i, \mathcal{D})$ *is finite for each* $1 \leq i \leq n$*. Moreover, given that* $\mathcal{D}$ *is finite, then we have that:* $\{t \mid f^{\mathcal{D}}\ s_1 \ldots s_n = t\}$ *is finite for every* $s_i \in Dom(\mathcal{D})$*. In particular, by proposition 5.6, we have that* $adom(e, \mathcal{D}) \subseteq Dom(\mathcal{D})$*, and thus* $\{t \mid f^{\mathcal{D}}\ adom(e_1, \mathcal{D})\ \ldots\ adom(e_n, \mathcal{D}) = t\}$ *is finite allowing to conclude that* $adom(f\ e_1\ \ldots\ e_n, \mathcal{D})$ *is finite.*

(b) *We have that:* $\| e \|^{\mathcal{D}} \eta =_{def} f^{\mathcal{D}}\ \| e_1 \|^{\mathcal{D}} \eta\ \ldots\ \| e_n \|^{\mathcal{D}} \eta$ *and by proposition*

5.6, $\llbracket e_i \rrbracket^{\mathcal{D}}\eta \subseteq Dom(\mathcal{D})$ *which allows, by induction hypothesis and given that* $\mathcal{D}$ *is finite, reasoning as in the case (a), to conclude that* $\{\eta \,|\, Dom(\eta) \subseteq var(e),\ \{\perp\} \neq \llbracket e \rrbracket^{\mathcal{D}}\eta,\ t\eta \in Dom(\mathcal{D}, A_i),$ *for every* $t \in cterms(s)$ *with* $s \in query\_key(\mathcal{Q}, A_i)\}$ *is finite.*

□

Lemma 5.8 (Denotation and Active Domain w.r.t. Inclusion) *Given two instances* $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ *and* $\mathcal{D}^* = (\mathcal{S}, \mathcal{DC}^*, \mathcal{IF}^*)$ *of two databases* $D = (S, DC, IF)$ *and* $D^* = (S, DC^*, IF^*)$*, such that* $\mathcal{S}$ *is ground and* $\mathcal{D} \subseteq \mathcal{D}^*$*, and a query* $\mathcal{Q}$*, then for each term* $e \in Term_{DC,DS(D)}(\mathcal{V})$*:*

(a) $adom(e, \mathcal{D}) = adom(e, \mathcal{D}^*)$

(b) if for all $t \in CTerm_{DC,\mathsf{F}}(\mathcal{V})$ *occurring in e, such that* $t \in cterms(s)$ *with* $s \in query\_key\ (\mathcal{Q}, A_i)$ *for a given key attribute* A_i*, then* $\llbracket e \rrbracket^{\mathcal{D}}\eta = \llbracket e \rrbracket^{\mathcal{D}^*}\eta$ *for every substitution* η *such that* $t\eta \in Dom(\mathcal{D}, A_i) (= Dom(\mathcal{D}^*, A_i))$ *for every* $t \in cterms(s)$ *with* $s \in query\_key(\mathcal{Q}, A_i)$*.*

Proof. *By structural induction over e. We analyze the main cases:*

(i) $e \equiv t$ *and* $t \in cterms(s)$ *with* $s \in query\_key(\mathcal{Q}, A_i)$*, for a given key attribute* $A_i \in R$ $(R \in S)$*, then:*

(a)

$$adom(t, \mathcal{D}) =_{def} \{\ t \mid t \in cterms(V_i\eta^*), \eta^* \in Subst_{DC,\perp,\mathsf{F}},\ (V_1, \ldots, V_i, \ldots, V_n) \in \mathcal{R}\},$$

and given that $\mathcal{S}$ *is ground and coincides in* $\mathcal{D}$ *and* $\mathcal{D}^*$*, then*

$$adom(t, \mathcal{D}^*) =_{def} \{\ t \mid t \in cterms(V_i\eta^{**}), \eta^{**} \in Subst_{DC,\perp,\mathsf{F}},\ (V_1, \ldots, V_i, \ldots, V_n) \in \mathcal{R}\},$$

where $V_i\eta^{**} = V_i\eta^* = V_i$*, and we can conclude that* $adom(e, \mathcal{D}) = adom(e, \mathcal{D}^*)$*.*

(b) Taking into account that $\llbracket e \rrbracket^{\mathcal{D}}\eta =_{def} \{t\eta\} = \llbracket e \rrbracket^{\mathcal{D}^*}\eta$*, for every* $\eta \in Subst_{DC,\perp,\mathsf{F}}$*.*

(ii) $e \equiv t$ *and* $t \notin cterms(s)$ *for all* $s \in query\_key(\mathcal{Q}, A_i)$ *then:*

(a) $adom(e, \mathcal{D}) =_{def} \{\perp\} = adom(e, \mathcal{D}^*)$*.*

(b) It contradicts that every c-term of e is a subterm of a query key.

(iii) *if* $e \equiv R\ e_1 \ldots e_k$ $(R \in S)$*, then:*

(a) $adom(R\ e_1 \ldots e_k, \mathcal{D}) =_{def} \{\mathtt{ok}, \perp, \mathsf{F}\} = adom(R\ e_1 \ldots e_k, \mathcal{D}^*)$*.*

(b) $\llbracket R\ e_1\ \ldots\ e_k \rrbracket^{\mathcal{D}}\eta =_{def} \{\mathtt{ok}\}$*, if* $(\llbracket e_1 \rrbracket^{\mathcal{D}}\eta,\ \ldots\ , \llbracket e_k \rrbracket^{\mathcal{D}}\eta) = (V_1\eta^*, \ldots, V_k\eta^*)$ *for a given substitution* $\eta^* \in Subst_{DC,\perp,\mathsf{F}}$*, and there exists a tuple* $(V_1, \ldots, V_k,\ V_{k+1}, \ldots, V_n) \in \mathcal{R}$*, where* $\mathcal{R} \in \mathcal{S}$*, and* $k = nKey(R)$*. Now, given that every c-term of e is a subterm of a query key, we have two subcases:*

(b.1) every c-term of $e_1, \ldots, e_k$ *is a subterm of a query key, and, therefore, by induction hypothesis we have that* $\llbracket e_i \rrbracket^{\mathcal{D}}\eta = \llbracket e_i \rrbracket^{\mathcal{D}^*}\eta$

(b.2) $e_j = t_j$ *where* $t_j \in cterms(s_j)$ *and* $s_j \in query\_key(\mathcal{Q}, A_i)$ *for a given attribute* $A_i \in R$ $(R \in S)$*, and we have that* $\llbracket e_j \rrbracket^{\mathcal{D}}\eta = \llbracket e_j \rrbracket^{\mathcal{D}^*}\eta =_{def} \{t_j\eta\}$

Therefore, in both cases, we conclude that:

$$[\![R\ e_1\ \dots\ e_k]\!]^{\mathcal{D}}\eta = [\![R\ e_1\ \dots\ e_k]\!]^{\mathcal{D}^*}\eta$$

(iv) *if* $e \equiv A_i\ e_1 \dots e_k$*, where* $A_i \in NonKey(R)$*, then:*

(a)

$$adom(A_i\ \bar{e}, \mathcal{D}) =_{def} \bigcup_{\{\eta^* \in Subst_{DC,\perp,\mathsf{F}}, (V_1,\dots,V_k,\dots,V_i,\dots,V_n) \in \mathcal{R}\}} V_i\eta^*$$

and

$$adom(A_i\ \bar{e}, \mathcal{D}^*) =_{def} \bigcup_{\{\eta^{**} \in Subst_{DC^*,\perp,\mathsf{F}}, (V_1,\dots,V_k,\dots,V_i,\dots,V_n) \in \mathcal{R}\}} V_i\eta^{**}$$

Now, given that $\mathcal{S}$ *does not change and is ground, we have that:* $V_i = V_i\eta^* = V_i\eta^{**}$ *and, therefore, we conclude:* $adom(A\ e_1\ \dots\ e_k, \mathcal{D}) = adom(A\ e_1\ \dots\ e_k, \mathcal{D}^*)$.

(b) $[\![A\ e_1\ \dots\ e_k]\!]^{\mathcal{D}}\eta =_{def} V_i\eta^*$*, if* $([\![e_1]\!]^{\mathcal{D}}\eta,\ \dots\ ,[\![e_k]\!]^{\mathcal{D}}\eta) = (V_1\eta^*, \dots, V_k\eta^*)$ *for a given substitution* $\eta^* \in Subst_{DC,\perp,\mathsf{F}}$*, and there exists a tuple* $(V_1, \dots, V_k,\ V_{k+1}, \dots, V_i, \dots, V_n) \in \mathcal{R}$*, where* $\mathcal{R} \in \mathcal{S}$*, and* $i > nKey(R)$*. Now, given that every c-term of e is a subterm of a query key, we have two subcases:*

(b.1) *every c-term of* $e_1, \dots, e_k$ *is a subterm of a query key, and thus, by induction hypothesis, we have that* $[\![e_i]\!]^{\mathcal{D}}\eta = [\![e_i]\!]^{\mathcal{D}^*}\eta$

(b.2) $e_j = t_j$ *where* $t_j \in cterms(s), s \in query\_key(\mathcal{Q}, A_i)$ *for a given key attribute* $A_i \in R$ $(R \in S)$*, then we have that* $[\![e_j]\!]^{\mathcal{D}}\eta = [\![e_j]\!]^{\mathcal{D}^*}\eta =_{def} \{t_j\eta\}$

Moreover, given that $\mathcal{S}$ *does not change and is ground, we have that:* $V_i = V_i\eta^* = V_i\eta^{**}$ *where* $[\![A_i\ e_1\ \dots\ e_k]\!]^{\mathcal{D}^*}\eta =_{def} V_i\eta^{**}$ *for a given substitution* $\eta^{**} \in Subst_{DC^*,\perp,\mathsf{F}}$*. Therefore, we conclude in both cases that* $[\![A_i\ e_1\ \dots\ e_k]\!]^{\mathcal{D}}\eta = [\![A_i\ e_1\ \dots\ e_k]\!]^{\mathcal{D}^*}\eta$.

(v) *if* $e \equiv c(e_1, \dots, e_n)$ *where* $c \in DC^n$*, then:*

(a) $adom(c(e_1, \dots, e_n), \mathcal{D}) =_{def} c^{\mathcal{D}}(adom(e_1, \mathcal{D}), \dots, adom(e_n, \mathcal{D}))$

(b) $[\![c(e_1, \dots, e_n)]\!]^{\mathcal{D}}\eta =_{def} c^{\mathcal{D}}([\![e_1]\!]^{\mathcal{D}}\eta, \dots, [\![e_n]\!]^{\mathcal{D}}\eta)$

Now, given that each c-term of e is a subterm of a query key, then each c-term of $e_1, \dots, e_n$ *is a subterm of query key, and thus by induction hypothesis,* $[\![e_i]\!]^{\mathcal{D}}\eta = [\![e_i]\!]^{\mathcal{D}^*}\eta$ *and* $adom(e_i, \mathcal{D}) = adom(e_i, \mathcal{D}^*)$*. Now, given that* $\mathcal{DC}^* \supseteq \mathcal{DC}$ *with* $c \in DC$*, we can conclude that* $adom(c(e_1, \dots, e_n), \mathcal{D}) = adom(c(e_1, \dots, e_n), \mathcal{D}^*)$ *and*
$[\![c(e_1, \dots, e_n)]\!]^{\mathcal{D}}\eta = [\![c(e_1, \dots, e_n)]\!]^{\mathcal{D}^*}\eta$.

(vi) *if* $e \equiv f\ e_1\ \dots\ e_n$ *where* $f \in IF^n$*, then,*

(a) $adom(f\ e_1\ \dots\ e_n, \mathcal{D}) =_{def} f^{\mathcal{D}}\ adom(e_1, \mathcal{D})\ \dots\ adom(e_n, \mathcal{D})$

(b) $[\![f\ e_1\ \dots\ e_n]\!]^{\mathcal{D}}\eta =_{def} f^{\mathcal{D}}\ [\![e_1]\!]^{\mathcal{D}}\eta\ \dots\ [\![e_n]\!]^{\mathcal{D}}\eta$

Now, every c-term of e is a subterm of query key, then every c-term of $e_1, \dots, e_n$ *is a subterm of a query key, and thus, by induction hypothesis and proposition 5.6, then* $[\![e_i]\!]^{\mathcal{D}}\eta = [\![e_i]\!]^{\mathcal{D}^*}\eta \subseteq Dom(\mathcal{D})$ *and* $adom(e_i, \mathcal{D}) = adom(e_i, \mathcal{D}^*) \subseteq Dom(\mathcal{D})$*. Now, given that* $\mathcal{IF}^* \supseteq \mathcal{IF}$ *with* $f \in IF$*, we can conclude that* $adom(f\ e_1\ \dots\ e_n, \mathcal{D}) = adom(f\ e_1\ \dots\ e_n, \mathcal{D}^*)$ *and*
$[\![f\ e_1\ \dots\ e_n]\!]^{\mathcal{D}}\eta = [\![f\ e_1\ \dots\ e_n]\!]^{\mathcal{D}^*}\eta$.

□

Theorem 5.9 (Domain Independence of Safe Queries) *Every safe query is domain independent.*

Proof. *Given an instance $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ of a database $D = (S, DC, IF)$ and a safe query $\mathcal{Q}$, the we can prove:*

(a) If $\mathcal{D}$ is finite, then $Ans(\mathcal{D}, \mathcal{Q})$ is finite

By induction over the number of constraints in $\mathcal{Q}$:

n=1: *We analyze the case $e_1 \bowtie e_2$; now, we can consider the following subcases:*

- *every c-term of e_1 and e_2 is a subterm of a query key. Given a substitution η such that $\bar{x}\eta \in Ans(\mathcal{D}, \mathcal{Q})$ with $\bar{x} = var(e_1) \cup var(e_2)$ then $(\mathcal{D}, \eta) \models_Q e_1 \bowtie e_2$; that is, there exist $t_1 \in [\![e_1]\!]^{\mathcal{D}}\eta$ and $t_2 \in [\![e_2]\!]^{\mathcal{D}}\eta$ such that $t_1 \downarrow t_2$ and $t_1, t_2 \in adom(e_1, \mathcal{D}) \cup adom(e_2, \mathcal{D})$. Now, by (b) of lemma 5.7, we have that $\{\eta \,|\, Dom(\eta) \subseteq var(e_1),\ \{\perp\} \neq [\![e_1]\!]^{\mathcal{D}}\eta,\ t\eta \in Dom(\mathcal{D}, A_i),\ t \in cterms(s)$ with $s \in query\_key\ (\mathcal{Q}, A_i)\}$ and $\{\eta \,|\, Dom(\eta) \subseteq var(e_2),\ \{\perp\} \neq [\![e_2]\!]^{\mathcal{D}}\eta,\ t\eta \in Dom(\mathcal{D}, A_j),\ t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_j)\}$ are finite. Moreover, given that every c-term of e_1 and e_2 is a subterm of a query key, then the previous condition $t\eta \in Dom(\mathcal{D}, A_i)$ holds. Therefore we can conclude that $Ans(\mathcal{D}, \mathcal{Q})$ is finite.*
- *e_1 contains, at least, one non-query key; in this case, given that $\mathcal{Q}$ is a safe query, then every c-term of e_2 is a subterm of a query key; now, given that $\mathcal{D}$ is finite, then by (a) of lemma 5.7 we have that $adom(e_1, \mathcal{D})$ and $adom(e_2, \mathcal{D})$ are finite. Now, given a substitution η such that $\bar{x}\eta \in Ans(\mathcal{D}, \mathcal{Q})$ with $\bar{x} = var(e_1) \cup var(e_2)$ then $(\mathcal{D}, \eta) \models_Q e_1 \bowtie e_2$; that is, there exist $t_1 \in [\![e_1]\!]^{\mathcal{D}}\eta$ and $t_2 \in [\![e_2]\!]^{\mathcal{D}}\eta$ such that $t_1 \downarrow t_2$ and $t_1, t_2 \in adom(e_1, \mathcal{D}) \cup adom(e_2, \mathcal{D})$. Now, by (b) of lemma 5.7, we have that $\{\eta \,|\, Dom(\eta) \subseteq var(e_2),\ \{\perp\} \neq [\![e_2]\!]^{\mathcal{D}}\eta,\ t\eta \in Dom(\mathcal{D}, A_i), t \in cterms(s)$ with $s \in query\_key(\mathcal{Q}, A_i)\}$ is finite; given that $adom(e_1, \mathcal{D}) \cup adom(e_2, \mathcal{D}) \subseteq Dom(\mathcal{D})$ is finite and $\mathcal{D}$ is finite, we have that $\{\eta | Dom(\eta) \subseteq var(e_1),\ \{\perp\} \neq [\![e_1]\!]^{\mathcal{D}}\eta \cap (adom(e_1, \mathcal{D}) \cup adom(e_2, \mathcal{D}))\}$ is also finite, and then we can conclude that $Ans(\mathcal{D}, \mathcal{Q})$ is finite*
- *e_2 contains at least, one non-query key, similarly to the previous case*
- *e_1 and e_2 contain, at least, a non-query key; it contradicts the safety condition*

n>1: *Now, by induction hypothesis, we can reason that if $\mathcal{Q}^* = \mathcal{Q} - \{e_1 \diamondsuit_q e_2\}$, then $Ans(\mathcal{D}, \mathcal{Q}^*)$ is finite. Now, reasoning similarly to previous cases, we have that $Ans(\mathcal{D}, e_1 \diamondsuit_q e_2)$ is finite and given that $Ans(\mathcal{D}, \mathcal{Q}) = Ans(\mathcal{D}, e_1 \diamondsuit_q e_2) \cap Ans(\mathcal{D}, \mathcal{Q}^*)$, we can conclude that $Ans(\mathcal{D}, \mathcal{Q})$ is finite.*

(b) Given two ground instances $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ and $\mathcal{D}^* = (\mathcal{S}, \mathcal{DC}^*, \mathcal{IF}^*)$ of two databases $D = (S, DC, IF)$ and $D^* = (S, DC^*, IF^*)$, such that $\mathcal{D} \subseteq \mathcal{D}^*$, then $Ans(\mathcal{D}, \mathcal{Q}) = Ans(\mathcal{D}^*, \mathcal{Q})$

By induction over the number of constraints in $\mathcal{Q}$:

<u>**n=1:**</u> *We analyze the case* $e_1 \bowtie e_2$*; now we can consider the following subcases:*

- *every c-term of* e_1 *and* e_2 *is a subterm of a query key; then given that* $\mathcal{S}$ *is ground,* $\mathcal{DC}^* \supseteq \mathcal{DC}$ *and* $\mathcal{IF}^* \supseteq \mathcal{IF}$*, then by (a) of lemma 5.8, we have that* $adom(e_1, \mathcal{D}) = adom(e_1, \mathcal{D}^*)$ *and* $adom(e_2, \mathcal{D}) = adom(e_2, \mathcal{D}^*)$*; by (b) of lemma 5.8, we have that* $\llbracket e_1 \rrbracket^{\mathcal{D}}\eta = \llbracket e_1 \rrbracket^{\mathcal{D}^*}\eta$ *and* $\llbracket e_2 \rrbracket^{\mathcal{D}}\eta = \llbracket e_2 \rrbracket^{\mathcal{D}^*}\eta$ *for every substitution* η *such that* $t\eta \in Dom(\mathcal{D}, A_i)$ *and* $t\eta \in Dom(\mathcal{D}, A_j)$*, for every* $t \in cterms(s)$ *with* $s \in query\_key(\mathcal{Q}, A_i)$*,* $t \in cterms(s)$ *with* $s \in query\_key(\mathcal{Q}, A_j)$*. Now, given a substitution* η *such that* $\bar{x}\eta \in Ans(\mathcal{D}^*, \mathcal{Q})$ *where* $\bar{x} = var(e_1) \cup var(e_2)$ *then* $(\mathcal{D}^*, \eta) \models_Q e_1 \bowtie e_2$*; that is, there exist* $t_1 \in \llbracket e_1 \rrbracket^{\mathcal{D}^*}\eta$ *and* $t_2 \in \llbracket e_2 \rrbracket^{\mathcal{D}^*}\eta$ *such that* $t_1 \downarrow t_2$ *and* $t_1, t_2 \in adom(e_1, \mathcal{D}^*) \cup adom(e_2, \mathcal{D}^*)$*. Now, given that* $\llbracket e_1 \rrbracket^{\mathcal{D}}\eta = \llbracket e_1 \rrbracket^{\mathcal{D}^*}\eta$*,* $\llbracket e_2 \rrbracket^{\mathcal{D}}\eta = \llbracket e_2 \rrbracket^{\mathcal{D}^*}\eta$*,* $adom(e_1, \mathcal{D}) = adom(e_1, \mathcal{D}^*)$ *and* $adom(e_2, \mathcal{D}) = adom(e_2, \mathcal{D}^*)$*, we have that there exist* $t_1 \in \llbracket e_1 \rrbracket^{\mathcal{D}}\eta$ *and* $t_2 \in \llbracket e_2 \rrbracket^{\mathcal{D}}\eta$ *such that* $t_1 \downarrow t_2$ *and* $t_1, t_2 \in adom(e_1, \mathcal{D}) \cup adom(e_2, \mathcal{D})$*. Therefore,* $(\mathcal{D}, \eta) \models_Q e_1 \bowtie e_2$ *and we can conclude that* $\bar{x}\eta \in Ans(\mathcal{D}, \mathcal{Q})$*.*
- e_1 *contains, at least, one non-query key; in this case, given that* $\mathcal{Q}$ *is a safe query, the every c-term of* e_2 *is a subterm of a query key; now, given that* $\mathcal{S}$ *is ground,* $\mathcal{DC}^* \supseteq \mathcal{DC}$ *and* $\mathcal{IF}^* \supseteq \mathcal{IF}$*, then by (a) of lemma 5.8, we have that* $adom(e_1, \mathcal{D}) = adom(e_1, \mathcal{D}^*)$ *and* $adom(e_2, \mathcal{D}) = adom(e_2, \mathcal{D}^*)$*; in addition, by (b) of lemma 5.8, we have that* $\llbracket e_2 \rrbracket^{\mathcal{D}}\eta = \llbracket e_2 \rrbracket^{\mathcal{D}^*}\eta$ *for every substitution* η *such that* $t\eta \in Dom(\mathcal{D}, A_i)$*, for every* $t \in cterms(s), s \in query\_key(\mathcal{Q}, A_i)$*. Now, given a substitution* η *such that* $\bar{x}\eta \in Ans(\mathcal{D}^*, \mathcal{Q})$ *where* $\bar{x} = var(e_1) \cup var(e_2)$*, then* $(\mathcal{D}^*, \eta) \models_Q e_1 \bowtie e_2$*; that is, there exist* $t_1 \in \llbracket e_1 \rrbracket^{\mathcal{D}^*}\eta$ *and* $t_2 \in \llbracket e_2 \rrbracket^{\mathcal{D}^*}\eta$ *such that* $t_1 \downarrow t_2$ *and* $t_1, t_2 \in adom(e_1, \mathcal{D}^*) \cup adom(e_2, \mathcal{D}^*)$*. Now, given that* $\llbracket e_2 \rrbracket^{\mathcal{D}}\eta = \llbracket e_2 \rrbracket^{\mathcal{D}^*}\eta$*,* $adom(e_1, \mathcal{D}) = adom(e_1, \mathcal{D}^*)$ *and* $adom(e_2, \mathcal{D}) = adom(e_2, \mathcal{D}^*)$*, then there exist* $t_1 \in \llbracket e_1 \rrbracket^{\mathcal{D}^*}\eta$ *and* $t_2 \in \llbracket e_2 \rrbracket^{\mathcal{D}}\eta$ *such that* $t_1, t_2 \in adom(e_1, \mathcal{D}) \cup adom(e_2, \mathcal{D})$*. Therefore,* $t_1 \in \llbracket e_1 \rrbracket^{\mathcal{D}^*}\eta$ *and* $t_1 \in CTerm_{DC,\mathsf{F}}(\mathcal{V})$ *and, in addition,* $e_1 \in Term_D(\mathcal{V})$*. Now, given that* $\mathcal{DC}^* \supseteq \mathcal{DC}$ *and* $\mathcal{IF}^* \supseteq \mathcal{IF}$ *then* $t_1 \in \llbracket e_1 \rrbracket^{\mathcal{D}}\eta$ *and, therefore,* $(\mathcal{D}, \eta) \models_Q e_1 \bowtie e_2$*, concluding that* $\bar{x}\eta \in Ans(\mathcal{D}, \mathcal{Q})$
- e_2 *contains, at least, a non-query key; similarly to the previous case*
- e_1 *and* e_2 *contain non-query keys; it contradicts the safety condition*

<u>**n>1:**</u> *By the safety condition: there exists, at least, one constraint* $e_1 \diamondsuit_q e_2$*, such that every c-term of* e_1 *(or* e_2*) is a subterm of a query key. Now, by induction hypothesis, we can reason that* $\mathcal{Q}^* = \mathcal{Q} - \{e_1 \diamondsuit_q e_2\}$ *satisfies that* $Ans(\mathcal{D}, \mathcal{Q}^*) = Ans(\mathcal{D}^*, \mathcal{Q}^*)$*. Now, reasoning similarly to the previous cases, we have that* $Ans(\mathcal{D}, e_1 \diamondsuit_q e_2) = Ans(\mathcal{D}^*, e_1 \diamondsuit_q e_2)$ *and, therefore, we can conclude that* $Ans(\mathcal{D}, \mathcal{Q}) = Ans(\mathcal{D}, e_1 \diamondsuit_q e_2) \cap Ans(\mathcal{D}, \mathcal{Q}^*) = Ans(\mathcal{D}^*, e_1 \diamondsuit_q e_2) \cap Ans(\mathcal{D}^*, \mathcal{Q}^*) = Ans(\mathcal{D}^*, \mathcal{Q})$ □

Theorem 5.10 (Domain Independence of Calculus Formulas) *Safe calculus formulas are domain independent.*

Proof. *Consequence of theorem 4.7 and theorem 5.9.* □

6 Least Induced Database

Up to now, we have considered schema definitions, and we have informally shown how instances can be obtained from a set of conditional rewriting rules. However, in this section, we will provide a formal definition, by means of a *fix point operator*, which computes the *least database induced* satisfying a set of rules. The fix point operator can be adopted as operational semantics (by means of a program transformation based on magic-sets, such as the presented one in [5]) for a deductive database language based on functional logic programming.

With this aim, firstly, we define the database instances which satisfy a given set of rules. Secondly, we present an approximation ordering over databases induced from the ordering $\sqsubseteq$ over sets of c-terms. Finally, we propose a fix point operator, showing that the database instance computed by the proposed fix point operator is the least one, which satisfies the set of rules.

Definition 6.1 [Instance Models] A database instance $\mathcal{D}$ *satisfies* a rule $H\ \bar{t}$ $:=\ r \Leftarrow C$, iff

(i) every θ such that $(\mathcal{D}, \theta) \models_Q C$, verifies $[\![H\ \bar{t}\ ]\!]^{\mathcal{D}}\theta \supseteq [\![r]\!]^{\mathcal{D}}\theta$

(ii) every θ such that for some $l_i \in [\![s_i]\!]^{\mathcal{D}}\theta$ $l_i \neq t_i$, $i \in \{1, \ldots, n\}$, then $\mathsf{F} \in [\![H\ \bar{s}\ ]\!]^{\mathcal{D}}\theta$

(iii) every θ such that $(\mathcal{D}, \theta) \not\models_Q C$, verifies $\mathsf{F} \in [\![H\ \bar{t}\ ]\!]^{\mathcal{D}}\theta$

This definition states that the right-hand sides (r) of the rules should be approximations to the values of the left-hand sides $(H(\bar{t}))$. Additionally, $H(\bar{t})$ represents F, whenever neither the terms $\bar{t}$ are syntactically equal to the head of a rule, nor the conditions of a rule are satisfied. A database instance $\mathcal{D}$ *satisfies a set of rules* $RW_1, \ldots, RW_n$, iff $\mathcal{D}$ *satisfies* every RW_i.

Instances can be also *partially ordered* as follows.

Definition 6.2 [Approximation Ordering over Databases] Given a database $D = (S, DC, IF)$ and two instances $\mathcal{D} = (\mathcal{S}, \mathcal{DC}, \mathcal{IF})$ and $\mathcal{D}^* = (\mathcal{S}^*, \mathcal{DC}, \mathcal{IF}^*)$, then $\mathcal{D} \sqsubseteq \mathcal{D}^*$, if:

(i) $V_i \sqsubseteq V_i^*$ for each $k+1 \leq i \leq n$, $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_n) \in \mathcal{R}$ and $(V_1, \ldots, V_k, V_{k+1}^*, \ldots, V_n^*) \in \mathcal{R}^*$, where $\mathcal{R} \in \mathcal{S}$ and $\mathcal{R}^* \in \mathcal{S}^*$, are relation instances of $R \in S$ and $k = nKey(R)$; and

(ii) $f^{\mathcal{D}}(t_1, \ldots, t_n) \sqsubseteq f^{\mathcal{D}^*}(t_1, \ldots, t_n)$ for each $t_1, \ldots, t_n \in \mathcal{DC}$, $f^{\mathcal{D}} \in \mathcal{IF}$ and $f^{\mathcal{D}^*} \in \mathcal{IF}^*$.

In particular, the *bottom database* has an empty set of tuples and each interpreted function is undefined.

In particular, given a set of database instances $\mathcal{DS}$ of a database schema D, we can consider $\mathcal{D}^{\sqcup\mathcal{DS}} = (\mathcal{S}^{\sqcup\mathcal{DS}}, \mathcal{DC}^{\sqcup\mathcal{DS}}, \mathcal{IF}^{\sqcup\mathcal{DS}})$, where $\mathcal{S}^{\sqcup\mathcal{DS}}$ contains relation instances $\mathcal{R}^{\sqcup\mathcal{DS}}$, with tuples

$$(V_1, \ldots, V_k, V_{k+1}^{\sqcup\mathcal{DS}}, \ldots, V_n^{\sqcup\mathcal{DS}}) \text{ where}$$
$$V_i^{\sqcup\mathcal{DS}} = \cup_{\mathcal{R}\in\mathcal{S},\mathcal{S}\in\mathcal{D},\mathcal{D}\in\mathcal{DS},(V_1,\ldots,V_k,V_{k+1},\ldots,V_n)\in\mathcal{R}} V_i$$

for each $k+1 \leq i \leq n$, whenever there exists, at least, a tuple

$$(V_1, \ldots, V_k, \ldots) \in \cup_{\mathcal{R}\in\mathcal{S},\mathcal{S}\in\mathcal{D},\mathcal{D}\in\mathcal{DS}} \mathcal{R}$$

Moreover, $\mathcal{DC}^{\sqcup\mathcal{DS}} = \mathcal{DC}$, and $f^{\sqcup\mathcal{DS}} = \cup_{\mathcal{D}\in\mathcal{DS}} f^{\mathcal{D}}$, for each $f^{\sqcup\mathcal{DS}} \in \mathcal{IF}^{\sqcup\mathcal{DS}}$. With this definition $\mathcal{D}^{\sqcup\mathcal{DS}}$ is the the *least upper bound* of $\mathcal{DS}$ w.r.t. $\sqsubseteq$.

Definition 6.3 [Fix Point Operator] Given an instance $\mathcal{A} = (\mathcal{S}^A, \mathcal{DC}^A, \mathcal{IF}^A)$ of a database schema $D = (S, DC, IF)$; we define a *fix point operator* $T_{\mathcal{P}}(\mathcal{A}) = \mathcal{B} = (\mathcal{S}^B, \mathcal{DC}^A, \mathcal{IF}^B)$ as follows:

(i) For each schema $R(A_1, \ldots, A_n)$,$k = nKey(R)$ $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_n) \in \mathcal{R}^B$, $\mathcal{R}^B \in \mathcal{S}^{\mathcal{B}}$, iff

$$\mathtt{ok} \in T_{\mathcal{P}}(\mathcal{A}, R)(V_1, \ldots, V_k)$$
$$\text{and for every } i \geq nKey(R)+1,\ V_i = T_{\mathcal{P}}(\mathcal{A}, A_i)(V_1, \ldots, V_k)$$

(ii) For each $f \in IF$ and $t_1, \ldots, t_n \in CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V})$, $f^{\mathcal{B}} \in \mathcal{IF}^{\mathcal{B}}$ iff

$$f^{\mathcal{B}}(t_1, \ldots, t_n) = T_{\mathcal{P}}(\mathcal{A}, f)(t_1, \ldots, t_n)$$

where given a symbol $H \in DS(D)$ and $s_1, \ldots s_n \in CTerm_{DC,\perp,\mathsf{F}}(\mathcal{V})$, we define:

$$
\begin{aligned}
T_{\mathcal{P}}(\mathcal{D}, H)(s_1, \ldots, s_n) =_{def} \{\ & t \mid \textit{if there exist } H\ \bar{t}\ := r \Leftarrow C \textit{ and } \theta, \\
& \quad \textit{such that } s_i \in [\![t_i]\!]^{\mathcal{D}}\theta,\ (\mathcal{D}, \theta) \models_Q C \textit{ and } t \in [\![r]\!]^{\mathcal{D}}\theta\ \} \\
\cup \{\ & \mathsf{F} \mid \textit{if there exists } H\ \bar{t}\ := r \Leftarrow C, \textit{ such that} \\
& \quad \textit{for some } i \in \{1, \ldots, n\},\ s_i \neq t_i\} \\
\cup \{\ & \mathsf{F} \mid \textit{if there exist } H\ \bar{t}\ := r \Leftarrow C \textit{ and } \theta, \\
& \quad \textit{such that } s_i \in [\![t_i]\!]^{\mathcal{D}}\theta \textit{ and } (\mathcal{D}, \theta) \not\models_Q C\} \\
\cup \{\ & \perp \mid \textit{otherwise}\}
\end{aligned}
$$

Starting from the bottom instance, then the fix point operator computes a chain of database instances $A \sqsubseteq A' \sqsubseteq A'', \ldots$ such that the fix point is the least database instance satisfying a set of conditional rewriting rules. The following theorem will prove this result.

Theorem 6.4 (Least Induced Database)

(i) *The fix point operator $T_{\mathcal{P}}$ has a least fix point $\mathcal{L} = \mathcal{D}^\omega$ where $\mathcal{D}^0$ is the bottom instance and $\mathcal{D}^{k+1} = T_{\mathcal{P}}(\mathcal{D}^k)$*

(ii) *For each safe query $\mathcal{Q}$ and θ: $(\mathcal{L}, \theta) \models_Q \mathcal{Q}$ iff $(\mathcal{D}, \theta) \models_Q \mathcal{Q}$ for each $\mathcal{D}$ satisfying the set of rules.*

Proof.

(i) *Firstly we have to prove that:*

(a) If $\mathcal{D} \sqsubseteq \mathcal{D}'$ *then* $\llbracket e \rrbracket^{\mathcal{D}}\theta \sqsubseteq \llbracket e \rrbracket^{\mathcal{D}'}\theta$ *and* $adom(e, \mathcal{D}) \sqsubseteq adom(e, \mathcal{D}')$

(b) If $\mathcal{DS}$ *is a directed set then* $\llbracket e \rrbracket^{\sqcup\mathcal{DS}}\theta \sqsubseteq \sqcup_{\mathcal{D}\in\mathcal{DS}}\llbracket e \rrbracket^{\mathcal{D}}\theta$ *and* $adom(e, \sqcup\mathcal{DS}) \sqsubseteq \sqcup_{\mathcal{D}\in\mathcal{DS}} adom(e, \mathcal{D})$

We analyze $R\ e_1, \ldots, e_k$ *and* $A_i\ e_1, \ldots, e_k$ *from the cases of the denotation, and for the active domain, it is analogous:*

(1) $e \equiv R\ e_1, \ldots, e_k$*:*

(a) We have the case of $\llbracket R\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}}\theta = \{\mathtt{ok}\}$*, if there exists a tuple* $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_n) \in \mathcal{R}$*, and* $\psi \in Subst_{DC,\perp,\mathsf{F}}$*, such that* $(\llbracket e_1 \rrbracket^{\mathcal{D}}\theta, \ \ldots\ , \llbracket e_k \rrbracket^{\mathcal{D}}\theta) = (V_1\psi, \ldots, V_k\psi)$*; where* $\mathcal{R} \in \mathcal{S}$*,* $k = nKey(R)$*. By definition of* $\sqsubseteq$*, then* $(V_1, \ldots, V_k, V'_{k+1}, \ldots, V'_n) \in \mathcal{R}'$*, where* $\mathcal{R}' \in \mathcal{S}'$*,* $\mathcal{D}' = (\mathcal{S}', \mathcal{DC}', \mathcal{IF}')$*, and by induction hypothesis* $V_i\psi = \llbracket e_i \rrbracket^{\mathcal{D}}\theta \sqsubseteq \llbracket e_i \rrbracket^{\mathcal{D}'}\theta$ *and given that* $V_i\psi \in CTerm_{DC,\mathsf{F}}$ *then* $\llbracket e_i \rrbracket^{\mathcal{D}'}\theta = V_i\psi$*, and therefore* $\llbracket R\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}'}\theta = \{\mathtt{ok}\}$*. Analogously for the cases of* F *and* $\perp$*.*

(b) By definition $\mathcal{S}^{\sqcup\mathcal{D}}$ *contains* $\mathcal{R}^{\sqcup\mathcal{DS}}$*, with tuples* $(V_1, \ldots, V_k, V_{k+1}^{\sqcup\mathcal{DS}}, \ldots, V_n^{\sqcup\mathcal{DS}})$*, where* $V_i^{\sqcup\mathcal{DS}} = \cup_{\mathcal{R}\in\mathcal{S},\mathcal{S}\in\mathcal{D},\mathcal{D}\in\mathcal{DS},(V_1,\ldots,V_k,V_{k+1},\ldots,V_n)\in\mathcal{R}} V_i$ *for each* $k{+}1 \leq i \leq n$*, whenever there exists, at least, a tuple* $(V_1, \ldots, V_k, \ldots) \in \cup_{\mathcal{R}\in\mathcal{S},\mathcal{S}\in\mathcal{D},\mathcal{D}\in\mathcal{DS}}\mathcal{R}$*. By induction hypothesis* $\llbracket e_i \rrbracket^{\sqcup\mathcal{DS}}\theta \sqsubseteq \sqcup_{\mathcal{D}\in\mathcal{DS}}\llbracket e_i \rrbracket^{\mathcal{D}}\theta$*,* $1 \leq i \leq k$*. On the other hand,* $\llbracket R\ e_1\ \ldots\ e_k\ \rrbracket^{\sqcup\mathcal{DS}}\theta = \{\mathtt{ok}\}$ *if there exists* $\llbracket e_i \rrbracket^{\sqcup\mathcal{DS}}\theta = V_i\psi$*. By induction hypothesis there exists* $\mathcal{D}_i \in \mathcal{DS}$ *such that* $V_i\psi \sqsubseteq \llbracket e_i \rrbracket^{\mathcal{D}_i}\theta$*. Given that* $V_i\psi \in CTerm_{DC,\mathsf{F}}$*, then* $\llbracket e_i \rrbracket^{\mathcal{D}_i}\theta = V_i\psi$*. Given that* $\mathcal{DS}$ *is a directed set, then there exists* $\mathcal{D}$*, such that* $\mathcal{D}_i \sqsubseteq \mathcal{D}$ $1 \leq i \leq k$*, and* $\llbracket e_i \rrbracket^{\mathcal{D}}\theta = V_i\psi$*. Therefore* $\mathtt{ok} \in \sqcup_{\mathcal{D}\in\mathcal{DS}}\llbracket R\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}}$*.*

(2) $A_i\ e_1, \ldots, e_k$*:*

(a) We have the case of $\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}}\theta = V_i\psi$*, if there exists a tuple* $(V_1, \ldots, V_k, V_{k+1}, \ldots, V_i, \ldots, V_n) \in \mathcal{R}$*, and* $\psi \in Subst_{DC,\perp,\mathsf{F}}$*, such that* $(\llbracket e_1 \rrbracket^{\mathcal{D}}\theta, \ \ldots\ , \llbracket e_k \rrbracket^{\mathcal{D}}\theta) = (V_1\psi, \ldots, V_k\psi)$*; where* $\mathcal{R} \in \mathcal{S}$*,* $k = nKey(R)$*,* $A_i \in NonKey(R)$*. By definition of* $\sqsubseteq$*, then* $(V_1, \ldots, V_k, V'_{k+1}, \ldots, V'_n) \in \mathcal{R}'$*, where* $V_i \sqsubseteq V'_i$*,* $\mathcal{R}' \in \mathcal{S}'$*,* $\mathcal{D}' = (\mathcal{S}', \mathcal{DC}', \mathcal{IF}')$ *and by induction hypothesis* $V_i\psi = \llbracket e_i \rrbracket^{\mathcal{D}}\theta \sqsubseteq \llbracket e_i \rrbracket^{\mathcal{D}'}\theta$*, and given that* $V_i\psi \in CTerm_{DC,\mathsf{F}}$ *then* $\llbracket e_i \rrbracket^{\mathcal{D}'}\theta = V_i\psi$ *and therefore* $\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}}\theta = V_i\psi \sqsubseteq \llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}'}\theta = V'_i\psi$*. Analogously for the cases of* F *and* $\perp$*.*

(b) By definition, $\mathcal{S}^{\sqcup\mathcal{D}}$ *contains* $\mathcal{R}^{\sqcup\mathcal{DS}}$*, with tuples* $(V_1, \ldots, V_k, V_{k+1}^{\sqcup\mathcal{DS}}, \ldots, V_n^{\sqcup\mathcal{DS}})$*, where* $V_i^{\sqcup\mathcal{DS}} = \cup_{\mathcal{R}\in\mathcal{S},\mathcal{S}\in\mathcal{D},\mathcal{D}\in\mathcal{DS},(V_1,\ldots,V_k,V_{k+1},\ldots,V_n)\in\mathcal{R}} V_i$ *for each* $k{+}1 \leq i \leq n$ *whenever there exists, at least, a tuple* $(V_1, \ldots, V_k, \ldots) \in \cup_{\mathcal{R}\in\mathcal{S},\mathcal{S}\in\mathcal{D},\mathcal{D}\in\mathcal{DS}}\mathcal{R}$*. On the other hand,* $\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\sqcup\mathcal{DS}}\ \theta = V_i^{\sqcup\mathcal{DS}}\psi$ *if there exists* $\llbracket e_i \rrbracket^{\sqcup\mathcal{DS}}\theta = V_i\psi$*. By induction hypothesis there exists* $\mathcal{D}_i \in \mathcal{DS}$ *such that* $V_i\psi \sqsubseteq \llbracket e_i \rrbracket^{\mathcal{D}_i}\theta$*. Given that* $V_i\psi \in CTerm_{DC,\mathsf{F}}$ *then* $\llbracket e_i \rrbracket^{\mathcal{D}_i}\theta = V_i\psi$*. In addition, there exists* $\mathcal{D}_0$ *such that* $\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}_0}\theta = V_i^{\sqcup\mathcal{DS}}\psi$*. Given that* $\mathcal{DS}$ *is a directed set, then there exists* $\mathcal{D}$*, such*

that $\mathcal{D}_i \sqsubseteq \mathcal{D}$, $i = 0, \ldots, k$ *and* $\llbracket e_i \rrbracket^{\mathcal{D}}\theta = V_i\psi$, *and* $\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}}\theta = V_i^{\sqcup\mathcal{DS}}\psi$. *Therefore* $\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\sqcup\mathcal{DS}} \sqsubseteq \sqcup_{\mathcal{D}\in\mathcal{DS}}\llbracket A_i\ e_1\ \ldots\ e_k\ \rrbracket^{\mathcal{D}}$.

In addition, we have to prove that, given a directed set $\mathcal{DS}$: $(\sqcup\mathcal{DS}, \theta) \models_Q \mathcal{Q}$, *then there exists* $\mathcal{D} \in \mathcal{DS}$ *such that* $(\mathcal{D}, \theta) \models_Q \mathcal{Q}$.

It is enough to prove if it holds for each constraint. It is easy generalize the result for a set of constraints. We analyze the case of $e \bowtie e'$:

Suppose $(\sqcup\mathcal{DS}, \theta) \models_Q e \bowtie e'$ *then there exist* $t \in \llbracket e \rrbracket^{\sqcup\mathcal{DS}}\theta$ *and* $t' \in \llbracket e \rrbracket^{\sqcup\mathcal{DS}}\theta$ *such that* $t \downarrow t'$, *and* $t, t' \in adom(e, \sqcup\mathcal{DS}) \cup adom(e', \sqcup\mathcal{DS})$. *By the previous result, there exists* $\mathcal{D}_1$ *such that* $t \in \llbracket e \rrbracket^{\mathcal{D}_1}$, *and there exists* $\mathcal{D}_2$ *such that* $t' \in \llbracket e' \rrbracket^{\mathcal{D}_2}$; *and in addition, there exist* $\mathcal{D}_3$ *and* $\mathcal{D}_4$ *such that* $t, t' \in adom(e, \mathcal{D}_3) \cup adom(e', \mathcal{D}_4)$. *Given that* $\mathcal{DS}$ *is a directed set, then there exists* $\mathcal{D} \in \mathcal{DS}$ *such that* $\mathcal{D}_i \sqsubseteq \mathcal{D}$, *and by the previous result, then* $(\mathcal{D}, \theta) \models e \bowtie e'$.

Finally, we have to prove that $T_{\mathcal{P}}$ *is continuous as is defined.*

- $T_{\mathcal{P}}$ *is monotonic:*
  Given $\mathcal{D}$ *and* $\mathcal{D}'$ *such that* $\mathcal{D} \sqsubseteq \mathcal{D}'$ *then* $\mathcal{D} \models_Q \mathcal{Q}$ *implies* $\mathcal{D}' \models_Q \mathcal{Q}$, *by the previous result. In addition, by the previous result* $\llbracket e \rrbracket^{\mathcal{D}}\eta \sqsubseteq \llbracket e \rrbracket^{\mathcal{D}'}\eta$ *for every* e *and* η. *Therefore* $T_{\mathcal{P}}(\mathcal{D}) \sqsubseteq T_{\mathcal{P}}(\mathcal{D}')$.
- $T_{\mathcal{P}}$ *is continuous:*
  It means that for every directed set $\mathcal{DS}$ *then* $T_{\mathcal{P}}(\sqcup\mathcal{DS}) \sqsubseteq \sqcup\{T_{\mathcal{P}}(\mathcal{D}) | \mathcal{D} \in \mathcal{DS}\}$. *It follows from the previous results given that each rule instance applicable to obtain* $T_{\mathcal{P}}(\sqcup\mathcal{DS}, H)(s_1, \ldots, s_n)$ *is also applicable to obtain* $\sqcup_{\mathcal{D}\in\mathcal{DS}}T_{\mathcal{P}}(\mathcal{D}, H)(s_1, \ldots, s_n)$, *which is equal to* $T_{\mathcal{P}}(\sqcup_{\mathcal{D}\in\mathcal{DS}}\mathcal{D}, H)(s_1, \ldots, s_n)$.

(ii) *It is enough to observe that a database* $\mathcal{D}$ *satisfies a set of rules iff* $T_{\mathcal{P}}(\mathcal{D}) \sqsubseteq \mathcal{D}$. *Therefore* $\mathcal{L}$ *satisfies the set of rules. Now, given* $\mathcal{Q}$ *such that* $(\mathcal{L}, \theta) \models_Q \mathcal{Q}$ *then, by previous results, there exists* $\mathcal{D}^i$ *such that* $(\mathcal{D}^i, \theta) \models_Q \mathcal{Q}$. *Supposing* $\mathcal{D}$ *satisfying the set of rules then* $\mathcal{D}^i \sqsubseteq \mathcal{D}$ *and therefore, by previous results,* $\mathcal{D} \models_Q \mathcal{Q}$.

□

7 Conclusions and Future Work

In this paper, we have studied how to express queries by means of an (extended) relational calculus in a functional logic language integrating databases. We have proved suitable properties for such language, which are summarized in the domain independence property. As future work, we propose two main lines of research: the study of an extension of our relation calculus to be used, also, as data definition language, and the implementation of the language.

References

[1] Abiteboul, S. and C. Beeri, *The Power of Languages for the Manipulation of Complex Values*, The VLDB Journal **4** (1995), pp. 727–794.

[2] Abiteboul, S., R. Hull and V. Vianu, "Foundations of Databases," Addison-Wesley, 1995.

[3] Almendros-Jiménez, J. M. and A. Becerra-Terón, *A Framework for Goal-Directed Bottom-Up Evaluation of Functional Logic Programs*, in: *Proc. of International Symposium on Functional and Logic Programming, FLOPS*, LNCS 2024 (2001), pp. 153–169.

[4] Almendros-Jiménez, J. M. and A. Becerra-Terón, *A Relational Algebra for Functional Logic Deductive Databases*, in: *Procs. of Perspectives of System Informatics, PSI*, LNCS 2890 (2003), pp. 494–508.

[5] Almendros-Jiménez, J. M., A. Becerra-Terón and J. Sánchez-Hernández, *A Computational Model for Funtional Logic Deductive Databases*, in: *Proc. of International Conference on Logic Programming, ICLP*, LNCS 2237 (2001), pp. 331–347.

[6] Benedikt, M. and L. Libkin, "Constraint Databases," Springer, 2000 pp. 109–129.

[7] Buneman, P., S. A. Naqvi, V. Tannen and L. Wong, *Principles of Programming with Complex Objects and Collection Types*, Theoretical Computer Science, TCS **149** (1995), pp. 3–48.

[8] Codd, E. F., *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, CACM **13** (1970), pp. 377–387.

[9] Codd, E. F., *Relational Completeness of Data Base Sublanguages*, in: *R. Rustin (ed.), Database Systems* (1972), pp. 65–98.

[10] González-Moreno, J. C., M. T. Hortalá-González, F. J. López-Fraguas and M. Rodríguez-Artalejo, *An Approach to Declarative Programming Based on a Rewriting Logic*, Journal of Logic Programming, JLP **1** (1999), pp. 47–87.

[11] Hanus, M., *The Integration of Functions into Logic Programming: From Theory to Practice*, Journal of Logic Programming, JLP **19,20** (1994), pp. 583–628.

[12] Hanus, M., *Curry: An Integrated Functional Logic Language, Version 0.8*, Technical report, University of Kiel, Germany (2003).

[13] Hull, R. and J. Su, *Deductive Query Language for Recursively Typed Complex Objects*, Journal of Logic Programming, JLP **35** (1998), pp. 231–261.

[14] Kanellakis, P. and D. Goldin, *Constraint Query Algebras*, Constraints **1** (1996), pp. 45–83.

[15] Kanellakis, P., G. Kuper and P. Revesz, *Constraint Query Languages*, Journal of Computer and System Sciences, JCSS **51** (1995), pp. 26–52.

[16] Kuper, G. M., L. Libkin and J. Paredaens, editors, "Constraint Databases," Springer, 2000.

[17] Libkin, L., *A Semantics-based Approach to Design of Query Languages for Partial Information*, in: *Proc. of Semantics in Databases*, LNCS 1358 (1995), pp. 170–208.

[18] Liu, M., *Deductive Database Languages: Problems and Solutions*, ACM Computing Surveys **31** (1999), pp. 27–62.

[19] López-Fraguas, F. J. and J. Sánchez-Hernández, *$\mathcal{TOY}$: A Multiparadigm Declarative System*, in: *Procs. of Conference on Rewriting Techniques and Applications, RTA*, LNCS 1631 (1999), pp. 244–247.

[20] López-Hernández, F. J. and J. Sánchez-Hernández, *Proving Failure in Functional Logic Programs*, in: *Proc. of the International Conference on Computational Logi, CL*, LNCS 1861 (2000), pp. 179–193.

[21] Moreno-Navarro, J. J. and M. Rodríguez-Artalejo, *Logic Programming with Functions and Predicates: The Language BABEL*, Journal of Logic Programming, JLP **12** (1992), pp. 191–223.

[22] Revesz, P. Z., *Safe Query Languages for Constraint Databases*, ACM Transactions on Database Systems, TODS **23** (1998), pp. 58–99.

[23] Shmueli, O., S. Tsur and C. Zaniolo, *Compilation of Set Terms in the Logic Data Language (LDL).*, Journal of Logic Programming, JLP **12** (1992), pp. 89–119.