Safe Database Queries with External Functions

Hong-Cheu Liu and Jeffrey X. Yu

Department of Computer Science
The Australian National University, Canberra, ACT 0200, Australia
E-mail: {hcliu, yu}@cs.anu.edu.au

Abstract

We consider the theory of database queries with functions on the complex value data model. The notion of a syntactic criteria, called “embedded allowed”, for queries which guarantee embedded domain independence, is generalized for this model. We show that all embedded allowed calculus (or fix-point) queries are external function-domain independent and continuous. We also develop an algorithm for translating embedded allowed queries into equivalent algebraic expressions as a basis for evaluating safe queries in all calculus-based query classes.

In the second part of this paper, we compare the expressive power of various complex value query languages extended with functions and their certain restricted versions. If we assume polynomial time (space) complexity of external functions and type usage by the database, such as density, then strongly embedded allowed inflationary (partial) fix-point queries express precisely \textsc{Qptime} (\textsc{Qpspace}). Finally, we show the relationship between properties such as embedded domain independence, finiteness and embedded allowed in various calculus-based query languages.

1 Introduction

The theory of database queries has grown into a rich technical area, which includes the expressiveness and complexity of query languages, the domain independence and safety of queries, query translation and optimization, dependencies and database design. Some attempts to extend the theory of (complex value) database queries to augment with arithmetic operators and external functions have already been made [8, 17, 19, 12]. None of these attempts, however, address all the aspects mentioned above. In particular, the aspects of a higher order calculus queries with fix-point and the issue of efficient query evaluation have received less attention. In this paper, we undertake a more comprehensive study of the theory of complex value database queries that are extended with external functions.

Support for both complex values and user-defined functions is important in DBMSs. In response to these requirements, SQL3 generalizes the relational model into an object model offering abstract data types and therefore allows users to define data types which suit their applications [4]. Tables can contain collections of objects (complex values) as well as tuples. DBMSs like Postgres and Informix allow users to register user-defined external functions, written in a programming language, into the system and to use them as built-in functions.

However, most query languages such as SQL, QUEL and O2SQL are calculus-based languages. It is well known that some calculus queries cannot be answered sensibly. In the database field, to answer sensibly means that the values of any correct answer lie within the active domain of the query or the input [18]. The database research community has developed notions of “domain independence” and “safety” to capture intuitive properties related to this finitude.

If the answers of relational calculus queries are possibly infinite sets or even undefined, it will be difficult to provide a well-defined procedure for translating user queries into equivalent relational algebra queries which can be evaluated by the underlying relational database systems\footnote{In the past, Van Gelder and Topor in [20] identified such problem in SQL and QUEL.}. The following are examples of unreasonable query phenomenon.

1. \{x \mid \neg \text{Movies (Cries and Whispers, Bergman, x)}\}
2. Given two relations \(R(w, y)\) (\(w\) requires \(y\)) and \(S(x, y)\) (\(x\) supplies \(y\)). The question, “Which suppliers supply all parts required by project IDEAS?” is expressed by \(\{x \mid \forall y [\neg R(I D E A S, y) \lor S(x, y)]\}\). \(3.\) \(\{x \mid \exists y, z [P(x) \land \neg Q(y) \land x + y = z]\}\). The set of correct answers for each of the above queries depends on the domains of the variables.

It is highly desirable to check whether a formula satisfies safety property when we need to support any complex values and any user-defined functions in a query language. However, domain-independence is undecidable even for the flat relational data model without functions.
been several attempts to identify such decidable subclasses of the domain independent formulas in the relational model [2, 7, 8, 11, 16, 18].

The allowed formula proposed by Topor [18] is one of large decidable subclasses of domain independent formulas. The notion of allowed is used to infer from syntactic properties of a calculus formula that a given variable ranges over a bounded set of possible values. Escobar-Molano, Hull and Jacobs [8] introduce the notion of embedded domain independence and generalize the allowed criteria to incorporate scalar functions. A key element in the notion of embedded-allowed developed in their generalization, involves inferring additional bounding information by using function terms. This is captured using finiteness dependencies, which are analogous to functional dependencies and carry information about how system-defined or user-defined functions can restrict the possible range of variables.

Suciu [17] proposes a notion of domain independence, called external-function domain independence, for queries with external functions. This notion generalizes those of generic and domain independent queries on databases without functions. The detailed definitions of the above notions will be reviewed in Section 3.

Benedikt and Libkin [5] introduce a general schema for transferring results about query safety to the finitely-representable setting. They give syntactic characterizations of the queries on finitely representable databases that preserve certain geometric properties.

In this paper, we focus on safe queries expressed in a higher order logic with external functions applied to finite databases rather than to the constraint databases, translation of such queries into equivalent algebraic expressions, and expressive power and complexity of query languages extended with functions. The main contributions of this paper are as follows.

- We generalize the notion of “allowed” to complex value databases and show that all embedded-allowed complex value calculus (or fix-point) queries are external function-domain independent and continuous.

- We develop an algorithm for translating embedded allowed queries into equivalent algebraic expressions as a basis for evaluating safe queries in all calculus-based query classes.

- We compare the expressive power of different calculus-based languages and their embedded allowed versions.

- We show that under the assumptions such as polynomial time (space) complexity of external functions and on type usage by the database, such as density, strongly embedded allowed inflationary (partial) fix-point queries express precisely \textit{QTIME (QSPACE)}.

Finally, we discuss the relationship between properties such as domain independence, finiteness and embedded allowedness in various calculus-based query languages.

2 Motivation and Background

We first briefly review some well known concepts of the complex value data model and the concepts of query on complex value databases. We assume familiarity with the basic notions of relational database theory.

We generally assume that there is only one sort of domain element rather than many (e.g. integer, float, string) since the nature of the elements is irrelevant to this paper. We let \textit{dom} denote a countably infinite set of uninterpreted constants. We also assume a countably infinite set \textit{Rel} of relation names. A database schema is a nonempty finite set \textit{R} of relation names. The set of instances over \textit{R} is denoted \textit{inst}(\textit{R}). We focus on a fixed finite set \textit{F} of functions which are associated with signatures.

The complex value data model allows the application of two basic constructors, \textit{tuple} and \textit{set} constructors recursively. The abstract syntax of sorts (types) of this data model is given by

\[ \tau = \text{dom} \langle B_1 : \tau, \ldots, B_k : \tau \rangle \{ \tau \} \]

where \(k \geq 0\) and \(B_1, \ldots, B_k\) are distinct attributes [2].

\textbf{Example} Consider the sort of a complex value relation \(R\): \(\{ < A : \text{dom}, B : \text{dom}, C : \{ < A : \text{dom}, E : \{ \text{dom} \} \rangle \} \rangle\). A value of this sort is \(\{ < A : a, B : b, C : \{ < A : d, E : \{ \} \rangle, < A : e, E : \{ e, f, \} \rangle \} \rangle\).

We define \textit{dom}(\tau, D), for some type \(\tau\) and a given subset \(D\) of \textit{dom} to be: (1) \(\text{dom}(\text{dom}, D) = D\), (2) If \(\tau\) is a type, \(\{ \tau \}\) is a set type with domain: \(\text{dom}(\{ \tau \}, D) = P_{\text{fin}}(\text{dom}(\tau, D))\), where \(P_{\text{fin}}(X)\) denotes the set of finite subsets of \(X\), and (3) If \(\tau_1, \ldots, \tau_r\) are types, \(\langle \tau_1, \ldots, \tau_n \rangle >\) is a tuple type with domain: \(\text{dom}(\langle \tau_1, \ldots, \tau_n \rangle, D) = \{ < a_1, \ldots, a_n > | a_i \in \text{dom}(\tau_i, D) \} \).

An interpretation \(I = (\text{d, f, I})\), where \(d \subseteq \text{dom}, F = (f_1, \ldots, f_i), f_i : \text{dom}(\tau_i, D) \rightarrow \text{dom}(\tau_i, D)\) are functions; \(I = (R_1, \ldots, R_k)\) is a database instance, \(R_1, \ldots, R_k\) are its relations. A database \(DB\) is given by interpretation of each relation \(R_i\) as a finite relation over \(d\) and augmented with a number of external functions \(f_1, \ldots, f_i\).

The active domain of a database instance \(I\), denoted \textit{adom}(\textbf{I}), is the set of all constants occurring in \(I\). This is defined analogously for formulas \(\phi\) and queries \(q\). In addition, we use \textit{adom}(\textit{q}, \textit{I}) as an abbreviation for \textit{adom}(\textit{q}) \cup \textit{adom}(\textit{I}).

A database query can be viewed as a partial function \(q\) mapping any database \(DB\) with interpretation \(d, f, I)\)
to \( q(DB) \in \text{dom}\{ \tau, d \} \), \( \tau \) is some type for query result.

Three fundamental properties usually imposed upon queries are **genericity**, **domain independence** and **computability**. **Genericity** means that a query mapping relations to relations is invariant under isomorphisms. A query \( q \) (with no embedded functions) is **domain independent** if \( q(DB) = q(DB') \) for any two databases \( DB = (d, I), DB' = (d', I) \) with the same instance \( I \). Finally, a query \( q \) is **computable** if there exists a Turing Machine \( M \) such that for each database \( DB = (d, I) \) (a) if \( q(DB) \) is defined, \( M \) halts on input an encoding of \( I \) with output an encoding of \( q(DB) \), or (b) if \( q(DB) \) is undefined, then \( M \) does not terminate on input an encoding of \( I \).

If query \( q \) is domain independent, then the value of \( q \) on \( DB \) can be determined by evaluating \( q \) using the finite set \( \text{dom}(q, I) \) as the underlying domain. When (interpreted or external) functions are incorporated into the query, it no longer guarantees genericity and domain independence. So we need to reconsider what type of queries is "safe", that is a query which can be answered sensibly. Some attempts to extend the definition of domain independence for the complex value data model extended with external functions have already been made [1, 17]. None of these attempts, however, address the aspects of calculus queries with fix-points and the relationship between safety, extended domain independence, and finiteness. The first goal of this paper is to investigate the notion of domain independence of complex value calculus queries with fix-points.

In modern intelligent database systems, we expect query systems to be able to handle a wider range of calculus formulas correctly and efficiently. Accordingly, they will require more general query translators and optimizers. Our second goal is to generalize the usual approach to implementing the calculus, namely, to develop an algorithm for translating em-allowed formulas into equivalent algebra queries.

As in the case of the relational data model, query languages have been developed so far from three paradigms, namely: algebraic, logic, deductive. The practical integration of external functions in query languages is well developed and generally well understood, but the expressive power and complexity of queries have received less attention. In Section 5, we compare the expressive power of the languages \( \text{CALC}^v, \text{CALC}^v + \mu^+, \text{CALC}^v + \mu, \text{Datalog}^v, \text{COL} \) and certain restricted versions. The languages are reviewed in the Appendix. Our third goal in this paper is to investigate the expressive power and complexity of query language in the presence of external functions.

### 3 Safe Queries

Before presenting our main results, we need to review two notions of domain independence. The notion of "embedded domain independence" was proposed to generalize “domain independence” to incorporate functions [8]. The fundamental idea behind this notion is that, for any query \( q \), there is a bound on the number of times functions (and their inverses) can be applied [11]. The answer to \( q \) on an input instance \( I \) depends on the closure of \( \text{dom}(q, I) \). We review this notion for complex objects as follows.

Given a database instance \( I \) and a query \( q \), let \( C_q \) be a set of constants that appear in \( q \). Following [1], we define \( \text{term}^n(DB) \), for some database \( DB \) with interpretation \((d, F, I)\) by:

\[
\text{term}^0(DB) \overset{\text{def}}{=} \text{atom}(I, C_q)
\]

\[
\text{term}^{n+1}(DB) \overset{\text{def}}{=} \text{term}^n(DB) \cup \{ \text{atom}(f_i(x)) \mid f_i \in F; x \in \text{dom}(\tau_i, \text{term}^n(DB)), i = 1, \ldots, l \}
\]

where \( \text{atom}(I, C_q) \) are all values in domain \( d \) mentioned in the instance \( I \) and \( C_q \). Two databases \( DB_1 = (d_1, F_1, I) \) and \( DB_2 = (d_2, F_2, I) \) agree on \( \text{atom}(I, C_q) \) at level \( n \) if (1) \( \text{term}^{n+1}(DB_1) = \text{term}^{n+1}(DB_2) \) and (2) \( \forall x \in \text{dom}(\tau_i, \text{term}^n(DB)) \), \( f_i \in F_1, f'_i \in F_2, f_i(x) = f'_i(x) \), i.e., \( f_i \) and \( f'_i \) agree on any input whose atomic values are in \( \text{term}^n(DB) \).

A calculus query \( q \) is **embedded domain independent** at level \( n \) if, for all interpretations \( S_1 = (d_1, F_1, I) \) and \( S_2 = (d_2, F_2, I) \) which agree on \( \text{atom}(I, C_q) \) to level \( n \), \( q \) yields the same output on \( S_1 \) and \( S_2 \). \( q \) is **embedded domain independent** if for some \( n \) it is embedded domain independent at level \( n \).

Next we will review the notion of external-function domain independent queries proposed by Suciu [17]. Let \( DB_1, DB_2 \) be two databases with interpretations \((d_1, F_1, I_1), (d_2, F_2, I_2)\) respectively. A **morphism** \( \xi : DB_1 \rightarrow DB_2 \) is a partial injective function \( \xi : d_1 \rightarrow d_2 \), which can be lifted from the base type to partial functions at any type \( \tau, \xi : \text{dom}(\tau, D) \rightarrow \text{dom}(\tau, D') \), such that (1) for every \( i, \xi(R_i) \) is defined and \( \xi(R_i) = R'_i \), where \( R_i \in I_1 \), \( R'_i \in I_2 \) and (2) for any \( x \in \text{dom}(\tau, d_1) \), if \( f_i(\xi(x)) \) is defined then so is \( \xi(f_i(x)) \) and \( f'_i(\xi(x)) = \xi(f_i(x)) \), where \( f_i \in F_1, f'_i \in F_2 \). Let us write \( e_1 \sqsubseteq e_2 \), whenever expression \( e_1 \) is undefined or \( e_1 = e_2 \), \( q \) is **external-function domain independent (ef-domain independent)** iff for every morphism \( \xi : DB_1 \rightarrow DB_2, q(DB_2) \subseteq q(DB_1) \).

As domain independence is an undecidable problem, it is desirable to find a simple syntactic condition on queries that implies domain independence. The formulas that are thereby restricted are called **safe**. Several decidable syntactic criteria for safe queries have been developed in the literature. The notion of safe calculus queries developed in [8], called **embedded allowed (em-allowed)**, is based on inferring from syntactic properties of a calculus formula that a variable range, for all practical purposes, over a bounded set of possible values. In this paper, we extend the definition of ‘em-allowed’ to complex value model and then
show that all queries in em-allowed \( \text{CALC} \) (or \( \text{CALC} + \mu^+ \)) are \(ef\)-domain independent.

A key element in the notion of em-allowed formulas is the definition of the bounding function \( bd \) which associates finiteness dependencies (FinDs) to formulas. First we introduce FinD over basic type and complex value type variables.

We write \( \phi(x_1, \ldots, x_n) \) to indicate that \( x_1, \ldots, x_n \) is a set of variables occurring freely in the formula \( \phi \). If \( \sigma \) is a valuation over free(\( \phi \)), then the notion of the interpretation \( (d, F, I) \) satisfying \( \phi \) under \( \sigma \), denoted by \( I =_{(d, F)} \phi[\sigma] \), is defined in the usual manner.

Intuitively, a formula \( \phi \) satisfies the FinD \( \{x_1, x_2\} \rightarrow \{y_1, y_2\} \) if for each database \( DB \), \( x_i \) range over \( \text{dom}(\tau_i, \text{term}^k(DB)) \), \( i = 1, 2 \), then \( \phi \) will be true only on assignments which map \( y_i \) into \( \text{dom}(\tau_i, \text{term}^{k+l}(DB)) \), for some \( l \).

**Definition** Let \( \phi \) be a formula and \( X \rightarrow Y \) a FinD over variable set \( X \). A formula \( \phi \) satisfies the finiteness dependency \( X \rightarrow Y \), denoted \( \phi =_{d} X \rightarrow Y \), if for each database \( DB \) and each \( k \geq 0 \) there is some \( \sigma \) such that \( \forall \forall y \in Y \), \( \sigma(y) \in \text{dom}(\tau_i, \text{term}^{k+l}(DB)) \) whenever \( \sigma \) is a variable assignment for \( X \) satisfying \( \sigma(x_i) \in \text{dom}(\tau_i, \text{term}^k(DB)) \), \( \forall x_i \in X \) and \( I =_{(d, F)} \phi[\sigma] \).

**Example** Let the sort of relation \( R \) be \( \{t : \text{dom}(x) > \} \). Letting
\[
\phi \equiv R(t, x) \land t \in x \land \neg Q(t) \land f(t) = y
\]
it can be shown that \( \phi =_{0} \rightarrow tx, \phi =_{0} t \rightarrow x, \phi =_{0} t \rightarrow y, \phi =_{0} t \rightarrow xy \). \( \square \)

It can be shown that FinDs satisfy the following inference rules. (1) \( XY \rightarrow X \), (2) \( XW \rightarrow YU \), if \( X \rightarrow Y \) and \( U \subseteq W \), and (3) \( X \rightarrow Z \), if \( X \rightarrow Y \) and \( Y \rightarrow Z \). If \( \Gamma \) is a set of FinDs and \( Y \rightarrow Y \) a FinD then \( \Gamma \vdash Y \rightarrow Y \) if \( Y \rightarrow Y \) can be proved from \( \Gamma \) using the inference rules given above. If \( \Gamma \) is a set of FinDs over a variable set \( V \) then the _closure_ of \( \Gamma \) over \( V \) is
\[
\Gamma^*_V = \{ X \rightarrow Y \mid XY \subseteq V \text{ and } \Gamma \vdash X \rightarrow Y \}
\]
For a formula \( \phi \), \( \Gamma^*_\phi \) is a shorthand for \( \Gamma^*_{\text{free}(\phi)} \).

We present the overall definition of \( bd \), and then define the notion of embedded allowed.

**Definition** Given a formula \( \phi \), \( bd(\phi) \) returns the set of FinDs, as in Figure 1.

In Figure 1, the formulas 1 to 11 and their associated functions \( bd \) were presented in [8]. We add formulas 12 to 18 for the complex value model. The operator \( \circ \) is defined as follows: Given the sets \( \Gamma_1, \ldots, \Gamma_n \), of FinDs, \( \Gamma_1 \circ \ldots \circ \Gamma_n = \{X_1 \ldots X_n \rightarrow Y \mid X_i \rightarrow Y \in \Gamma_i \text{ for } i \in [1, \ldots, n]\} \) [8]

We now present our generalized notion of em-allowed for complex value model.

**Definition** A formula \( \phi \) is embedded allowed (em-allowed) if:
(a) \( bd(\phi) =_{0} \rightarrow \text{free}(\phi) \);
(b) for each sub-formula \( \exists x \psi \) of \( \phi \), \( bd(\psi) =_{0} \rightarrow \text{free}(\exists x \psi) \rightarrow (\exists x \psi) \);
(c) for each sub-formula \( \forall x \psi \) of \( \phi \), \( bd(\neg \psi) =_{0} \rightarrow \text{free}(\forall x \psi) \rightarrow (\forall x \psi) \);
(d) for each sub-formula \( x \subseteq \{y \mid \phi(y)\} \), \( bd(\phi) =_{0} \rightarrow \text{free}(\phi) \rightarrow y \).

**Example** Consider the formula \( \phi = p(x,y) \land \exists u = f(x,y) \land \neg q(u,t) \land t = f(u) \). We have \( \theta \rightarrow xy, xy \rightarrow u \) and \( u \rightarrow t \). We can get \( \theta \rightarrow xyu \) which satisfies the condition (a). \( u \rightarrow t \) satisfies the condition (b). So \( \phi \) is em-allowed. \( \square \)

In [12], Liu and Yu have proved that every em-allowed formula in \( \text{CALC}^v \) is \(ef\)-domain independent and embedded domain independent. If all external functions are computable, it is easy to get the following result.

**Proposition 1** Every em-allowed formula defines an \(ef\)-domain independent, computable query.

The notion of \(ef\)-domain independence is used only in conjunction with query languages without iterations, as shown in [17], and fails when extended to languages with fix-points. The notion of \(ef\)-domain independence is more appropriate for queries with external functions than the notion of \(ef\)-domain independence. For this reason we investigate the aspects of calculus queries with fix-points.

Now we review the concept of continuous as follows. A query \( q \) is _continuous_ if for any database \( DB = (d, F, I) \) for which \( q(DB) \) is defined, there is some finite approximation \( DB_0 = (d_0, F_0, I) \) (i.e., \( d_0 \) is finite and \( DB_0 \subseteq DB \)) such that \( q(DB_0) = q(DB) \). [17]
<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$bd(\phi)$</th>
</tr>
</thead>
</table>
| $R(\tau_1, \ldots, \tau_n)$ | $\{0 \rightarrow X\}^{*}, \phi$  
where $X$ is set of variables that are members of $\{\tau_1, \ldots, \tau_n\}$ |
| $\neg R(\bar{\tau})$ | $\theta^{*}, \phi$  
for $\phi$ not of the form $R(\bar{\tau})$ |
| $\neg \xi$ | $bd(\text{pushnot}(\neg \xi))$  
if $\tau$ is not a variable, or $\tau$ is a variable occurring in one of $\tau_1, \ldots, \tau_n$ |
| $f(\tau_1, \ldots, \tau_n) = \tau$ | $\theta^{*}, \phi$  
if $\tau$ is a variable not occurring in any of $\tau_1, \ldots, \tau_n$,  
where $X$ is set of variables occurring in $\tau_1, \ldots, \tau_n$ |
| $x = y$ | $\{x \rightarrow y, y \rightarrow x\}^{*}, \phi$  
if $x$ is a variable not occurring in $\phi$,  
$X$ is set of variables occurring in $\phi$. |
| $\tau_1 \neq \tau_2$ | $\theta^{*}, \phi$ |
| $\xi_1 \land \ldots \land \xi_n$ | $(bd(\xi_1) \cup \ldots \cup bd(\xi_n))^{*}, \phi$  
for each $\phi$ that has a $\cup$ operator |
| $\xi_1 \lor \ldots \lor \xi_n$ | $(bd(\xi_1) \otimes \ldots \otimes bd(\xi_n))^{*}, \phi$  
for each $\phi$ that has a $\otimes$ operator |
| $\exists \bar{x} \xi_1$ | $(bd(\xi_1) \cdot \text{all FinD in which some variable in } \bar{x} \text{ occurs})^{*}, \phi$  
if $\tau$ is an $\exists$ operator |
| $\forall \bar{x} \xi_1$ | $(bd(\xi_1) \cdot \text{all FinD in which some variable in } \bar{x} \text{ occurs})^{*}, \phi$  
if $\tau$ is an $\forall$ operator |
| $x \subseteq \{y \mid \phi(y)\}$ | $\{X \rightarrow x\}$  
if $x$ is a variable not occurring in $\phi$,  
$X$ is set of variables occurring in $\phi$. |
| $\tau \in \pi$ | $\{X \rightarrow \pi\}^{*}, \phi$  
if $\tau$ is a variable not occurring in $\pi$,  
where $X$ is set of variables occurring in $\pi$ |
| $\tau \in \pi$ | $\theta^{*}, \phi$  
if $\tau$ is a variable not occurring in $\pi$,  
where $X$ is set of variables occurring in $\pi$ |
| $\tau \subseteq \pi$ | $\{X \rightarrow \pi\}^{*}, \phi$  
if $\tau$ is a variable not occurring in $\pi$,  
where $X$ is set of variables occurring in $\pi$ |
| $\tau \subseteq \pi$ | $\theta^{*}, \phi$  
if $\tau$ is a variable not occurring in $\pi$,  
where $X$ is set of variables occurring in $\pi$ |
| $x.i$ | $\text{if } x \text{ is of type } <\tau_1, \ldots, \tau_n>$,  
then for each $i$, $1 \leq i \leq n$, $x \rightarrow x.i$ |
| $x$ | $\text{if } x \text{ is of type } <\tau_1, \ldots, \tau_n>$, $\theta \rightarrow x.i$ for each $i$  
then $\theta \rightarrow x$ |

**Figure 1.** The function $bd$. 

A complex value algebra, denoted $\text{ALG}^{cv}$, is a functional language based on a small set of operations. An important subset of $\text{ALG}^{cv}$, denoted $\text{ALG}^{cv}(\Sigma)$, is formed from the core operators of $\text{ALG}^{cv}$ by removing the powerset operator and adding the nest operator [6]. Let $\Sigma = \{f_1, \ldots, f_l\}$ be a signature. $\text{ALG}^{cv}(\Sigma)$ is the nested relation algebra over $\Sigma$. In [17], Suciu proposed that all queries in $\text{ALG}^{cv}(\Sigma)$ are $ef$-domain independent and continuous. We expect all em-allowed queries expressed in $\text{CALC}^{cv+\mu^+}$ to be also $ef$-domain independent and continuous. We prove this as follows.

**Theorem 1** All em-allowed queries in $\text{CALC}^{cv+\mu^+}$ are ef-domain independent and continuous.

**Proof.** If $\phi \in \text{CALC}^{cv}$ and is em-allowed then $\phi$ is $ef$-domain independent and continuous. This is demonstrated in the course of translating em-allowed formulas in $\text{CALC}^{cv}$ into equivalent algebra queries in $\text{ALG}^{cv}(\Sigma)$ (See the next section). All queries in the nested relational algebra $\text{ALG}^{cv}(\Sigma) + \mu^+$ are $ef$-domain independent and continuous [17]. $\text{ALG}^{cv}(\Sigma)$ is equivalent to $\text{ALG}^{cv}(\Sigma)$ without powerset with external functions. By definition, it is easy to check that the powerset query is $ef$-domain independent and continuous. $\text{CALC}^{cv+\mu^+}$ is equivalent to $\text{CALC}^{cv}$ [2]. Therefore, every em-allowed queries in $\text{CALC}^{cv+\mu^+}$ is $ef$-domain independent and continuous. □
The transformation rules, $T$, conditions. Note that each transformation preserves the ENF used to transform those sub-formulas that violate ENF.

Step-1: First, replace all sub-formulas of the form $\forall y (x \rightarrow \varphi(y))$ with $x \subseteq \{ y \mid \varphi(y) \}$. Second, replace any remaining sub-formula of the form $\forall \varphi$ by $\neg \exists \neg \varphi$. Finally, rename the quantified variables if necessary.

Step-2: Transform the em-allowed formula $F$ obtained in Step-1 into an equivalent formula $F'$ in Existential Normal Form ($ENF$).

Step-3: Transform the formula $F'$ into an equivalent complex value algebra normal form $\psi$ ($ALG^{cv}NF$).

Step-4: Translate $\psi$ into an equivalent algebra expression $E_\psi$.

Step-1 is accomplished using the transformations mentioned above. We briefly describe from Step-2 to Step-4 below. In Step 2, we consider a calculus formula in terms of its parse tree where the leaves of the tree are atomic formulas. There is a sub-formula which corresponds to each internal node labeled by $\land$, $\lor$, $\neg$, $\exists x$ or a sub-formula $x \subseteq \{ y \mid \varphi(y) \}$.

We introduce the concept of Existential Normal Form. Our notion of ENF is slightly different from that of [8], in that we add item 4.

**Definition** A formula $F$ is in Existential Normal Form ($ENF$) if and only if:

1. It is simplified
2. Each disjunction in the formula satisfies:
   a. The parent of the disjunction, if it has one, is $\land$.
   b. Each operand of the disjunction is a positive formula.
3. The parent, if any, of a conjunction of negative formulas is $\exists$.
4. For each sub-formula $x \subseteq \{ y \mid \varphi(y) \}$ in the formula $F$, $\varphi$ is in $ENF$.

The transformation rules, $T_8 - T_{12}$, stated in [8] can be used to transform those sub-formulas that violate $ENF$ conditions. Note that each transformation preserves the em-allowed property.

**Example** The following formula

$$\varphi = \neg (\neg f(x) = y \land x \in z) \land \neg T(x) \land S(z)$$

can be translated into

$$\varphi' = ((f(x) = y \land x \in z) \lor T(x)) \land S(z)$$

which is in $ENF$.

Step 3: We start by defining the concepts of a maximal sub-formula and complex value algebra normal form, then we give a necessary transformation rule not presented in the relational model.

**Definition** A sub-formula $G$ of a formula $F$ is maximal if either

- $G$ is $F$; or
- $G$ is a positive non-arithmetic operand of the operator $\land$; or
- $G$ is a child of one of the operators $\exists$, $\lor$, $\neg$ or a range $x \subseteq \{ y \mid \varphi(y) \}$.

**Example** Let $F$ be the formula $(p(x) \lor \neg q(x)) \land \neg r(x) \land x < y \land x \subseteq \{ t \mid s(z) \land t \in z \}$. Then the maximal sub-formulas of $F$ are $p(x)$, $q(x)$, $\neg q(x)$, $p(x) \lor \neg q(x)$, $r(x)$, $x \subseteq \{ t \mid s(z) \land t \in z \}$, $s(z)$, $t \in z$, $s(z) \land t \in z$ and $F$ itself. The sub-formula $\neg r(x)$ and $x < y$ are not maximal.

Our aim is to transform a given em-allowed query into an equivalent query. In fact, all of whose maximum sub-queries can be translated. In this step, the function $bd$ is crucial to decide whether a sub-formula is em-allowed or not.

**Definition** An em-allowed formula $F$ is in complex value algebra normal form ($ALG^{cv}NF$) if $F$ is in $ENF$ and every maximum sub-formula of $F$ is em-allowed.

In addition to the rules $T_{13}$ to $T_{16}$ stated in [8], we need the following rule to transform $ENF$ em-allowed formula to complex value algebra normal form.

**Rule** $T_{17}$: $\xi_1 \land \ldots \land \xi_m \land x \subseteq \{ y \mid \varphi(y) \} \rightarrow \xi_1 \land \ldots \land \xi_m \land x \subseteq \{ y \mid \varphi(y) \land \xi_{i_1} \land \ldots \land \xi_{i_k} \}$ where $\varphi(y)$ is not em-allowed, but $\varphi(y) \land \xi_{i_1} \land \ldots \land \xi_{i_k}$ is em-allowed.

**Example** Consider the following formula

$$\exists u \exists s (S(u) \land R(s) \land x \subseteq \{ y \mid (y + s.A) \in u.C \land (t \in x \land \neg Q(t)) \})$$
We apply the above rule to obtain $\exists y \exists z (S(u) \wedge R(s) \wedge x \subseteq \{ y \mid S(u) \wedge R(s) \wedge y + sA \in u,C \} \wedge (t \in x \wedge \neg Q(t)))$ which is in $ALG^{cv}NF$.

Step-4: Translating an $ALG^{cv}NF$ formula into an equivalent algebra expression can be performed by applying the following method:

- conjunctions are translated into joins or Cartesian products,
- negations into generalized differences (diff) [20],
- existential quantifiers into projections,
- inequalities into selections, and
- equalities and arithmetic operations into append.

Append is a relational operator which is defined in [19]. We apply the above rule to obtain $\exists y \exists z (S(u) \wedge R(s) \wedge x \subseteq \{ y \mid S(u) \wedge R(s) \wedge y + sA \in u,C \} \wedge (t \in x \wedge \neg Q(t)))$.

The equivalent algebra query is obtained using the following program:

\[
\begin{align*}
E_1 &= \text{tup\_create}_C(\text{set\_destroy}(\text{tup\_destroy}(\pi_C(S())))) ; \\
E_2 &= \pi'_{g,C}(\sigma'_{g,C}(S \times E_1)) ; \\
E_3 &= E_2 \times (\pi_3(R)) ; \\
E_4 &= \Phi_{g(2,3)}(E_3) ; \\
E_5 &= \text{nest}^\uparrow_{A,Y,X}(E_4) ; \\
E_6 &= \pi_{A',Y,Z}(E_5) ; \\
E_7 &= \text{replace} < A',X,Z = (\text{rename}_{g\rightarrow r}(X)) \text{ diff Q} > (E_6) ; \\
E_8 &= \pi_X,E_7(E_7) .
\end{align*}
\]

where $g(2,3) =$ column 2 - column 3. Note that $E_2$ is equivalent to unnest$_C(S)$. A detailed description of the complex value algebra including set\_destroy, tup\_create and replace can be found in [2].

### 5 Expressive power

In this section, we compare the expressive power of the languages $CALC^{cv}$, $CALC^{cv} + \mu^+$, $CALC^{cv} + \mu$, $Datatalog^{cv}$, $COL$ and their em-allowed versions. We then focus on bounding complexity of queries by limiting the ranges of variables in evaluating the query. As in [9], we consider using fix-point operators to provide tractable recursion, and limiting the arity and height of higher order types.

The complex value query languages described in the Appendix without external functions are equivalent [10]. This result also hold for the query languages extended with external functions. We state it as follows.

**Theorem 2** The following languages extended with functions are equivalent.

(1) $CALC^{cv}$ (2) $CALC^{cv} + \mu^+$  (3) $CALC^{cv} + \mu$  (4) $Datatalog^{cv}$ (5) $COL$.

**Definition** A rule is em-allowed if each variable that appears in the head also appears in the body and the body is em-allowed. A database program $\mathcal{P}$ is em-allowed if each clause in $\mathcal{P}$ is an em-allowed formula. □

We consider the above languages with em-allowed versions. For each language $L$, we denote by $em-L$ the set of em-allowed queries in $L$.

**Theorem 3** $em-CALC^{cv}$, stratified $em-(Datatalog^{cv})$ and $em-Col$ are equivalent.

Note that $em-CALC^{cv} + \mu^+$ is not equivalent to the above languages since there exist queries in $em-CALC^{cv} + \mu^+$ which are not em-domain independent whereas all queries in the above three languages are em-domain independent.

**Definition** A formula is strongly em-allowed if it is em-allowed and the inclusion predicate does not occur in it. □

For each language $L$, we denote by $sem-L$ the set of strongly em-allowed queries in $L$. For each complexity class C, one can define a corresponding complexity class of queries, denoted by QC. The class of queries QC consists of all queries whose complexity is in C. For example, QTIME consists of all queries whose complexity is in PTIME. Since strongly safe $CALC^{cv}$ has complexity in PTIME [2]. It is strait-forward to get the following result.

**Lemma 1** (1) If all external functions can be evaluated in polynomial time in the size of their arguments, then $sem-CALC^{cv} \subseteq QTIME$; $sem-(CALC^{cv} + \mu^+) \subseteq QTIME$. (2) If all external functions can be evaluated by evaluating polynomial space in the size of their arguments, then $sem-(CALC^{cv} + \mu) \subseteq QSPACE$.

A database is dense with respect to a type $T$ if it makes full use of the type $T$. An $< i,k >$-type is a type whose set height is at most $i$ and whose tuple width is at most $k$. As in [9], we consider density with respect to the set of $< i,k >$ types, rather than individual types. Consider an $< i,k >$-database schema $R$ and let $I \subseteq inst(R)$. Then $I$ is dense with respect to $< i,k >$-types if there exists a polynomial $P$ such that, for each $I \in I$, $\mid dom(< i,k >, atom(I)) \mid \leq P(\mid I \mid)$. See [9] for more details.
If $\mathcal{T}$ is a set of types, $\text{sem}_{\mathcal{T}} - L$ denotes the queries in $L$ for which all sub-formulas that contain variables of a type in $\mathcal{T}$ are strongly em-allowed. \(\text{CALC}_{<i,k>}^{cv}\) denotes the queries using only $i,k$ -types.

**Theorem 4** Let $\mathcal{R}$ be a $< i, k >$ -database schema ($i \geq 1, k > 1$) and $I \subseteq \text{inst}(\mathcal{R})$, be dense with respect to some non-trivial type $T$. Let $\mathcal{T}$ consist of all $< i, k >$ -types other than $T$. (1) If all functions can be evaluated in polynomial time, then $\text{sem}_{\mathcal{T}}(\text{CALC}_{<i,k>}^{cv} + \mu^+) | I = \text{QPTIME} | I$. (2) If all functions can be evaluated by using polynomial space, then $\text{sem}_{\mathcal{T}}(\text{CALC}_{<i,k>}^{cv} + \mu) | I = \text{QSPACE} | I$.

**Proof.** (sketch) The proof is based on that, with order, all queries in $\text{QPTIME} | I$ can be expressed in $\text{sem}_{\mathcal{T}}(\text{CALC}_{<i,k>}^{cv} + \mu^+)$. For each $q$ in $\text{QPTIME}$, there exists a range function which associates to each variable $x$ in $q$. Those range functions take polynomial time. Once those range functions are defined, it can easily be derived that (1) if $x$ is a free variable, $bd(\phi) \models \emptyset \rightarrow x$, and (2) if $x$ is bounded in $\phi$, $bd(\phi) \models \text{free}(\exists x \phi) \rightarrow x$ or $bd(\neg \phi) \models \text{free}(\forall x \phi) \rightarrow x$. i.e., query $q$ is strongly em-allowed. As database makes “full use” of its types, the theorem can be proved without order. So $\text{QPTIME} | I \subseteq \text{sem}_{\mathcal{T}}(\text{CALC}_{<i,k>}^{cv} + \mu^+) | I$.

All sub-formulas which include variables of a type in $\mathcal{T}$ are strongly em-allowed. Those variables can be computed in $\text{PTIME}$ (by Lemma 1). As database is dense w.r.t. $T$. It takes polynomial time to evaluate variables which is of type $T$ [9]. Therefore $\text{sem}_{\mathcal{T}}(\text{CALC}_{<i,k>}^{cv} + \mu^+) | I \subseteq \text{QPTIME} | I$.

\[ \square \]

6 Em-allowed, finiteness and domain independence

We now turn to the relationship between safety, domain independence and finiteness in the various calculus-based query languages we have described.

For $\text{CALC}_{<i,k>}^{cv}$ queries, em-allowed (or safety) implies embedded domain independence, ef-domain independence and finiteness. It is not equivalent to finiteness, as the query $\forall y p(x,y) \land z = f(x)$ is finite but not em-allowed. Clearly, the class of embedded domain independent queries is larger than that of em-allowed queries. For example, $F(y,z) = \exists x [p(x,y) \lor q(y)] \land z = f(y)$ is embedded domain independent but not em-allowed. Although not every domain independent query is em-allowed, the embedded domain independent calculus and em-allowed calculus are equivalent. i.e., for every embedded domain independent formula there exists an equivalent em-allowed formula.

For stratified $\text{Datalog}_{<i,k>}^{cv}$ queries, em-allowed does not imply embedded domain independence [12]. Em-allowed does not imply finiteness as the query $\{ p(0) \leftarrow p(n) \leftarrow p(m) \land \text{succ}(m) = n \}$ is em-allowed but not finite. However, em-allowed does imply a weaker form of finiteness. A database instance $I$ can be regarded as a set of ground atomic formulas for the base predicates of a $\text{Datalog}_{cv}$ program $P$. Let $T^k_p$ be the set of facts about derived predicates in $P$ that can be deduced from $I$ by at most $k$ applications of the rules in $P$. A $\text{Datalog}_{cv}$ program is weakly finite if $T^k_p(I)$ is finite for all $k \geq 0$ and all databases $I$. Then em-allowed implies weak finiteness. However, em-allowed is not equivalent to weakly finite, as the query $\{ p(0) \leftarrow m < 0 \land m > 0 \}$ is weakly finite but not em-allowed.

For fix-point queries, $\text{CALC}_{<i,k>}^{cv} + \mu (\mu^+)$, em-allowed implies finite but it does not imply embedded domain independence (see [17]). By Proposition 1, embedded allowed formulas define ef-domain independent queries.

7 Conclusion

We have presented a theory of database queries for complex value databases that are extended with external functions.

We have extended the binding function proposed in [8] and generalized the notion of “allowed” to incorporate external functions in complex value databases. Significantly, we have shown that all em-allowed complex value calculus (or fix-point) queries are external-function-domain independent and continuous. We expect that the framework developed in this paper could be used to generalize other safety criteria such as “evaluability” in [20].

An algorithm for translating embedded allowed queries into equivalent algebraic expressions has been developed. The algorithm is still open to optimization. We have not considered the issue of query optimization for the resulting complex value algebra expressions, such as query translation with optimization [15] and combining several algebra operators during their execution. There is also the need to address the complexity issue of query evaluation for different query classes.

The expressive power of various complex value query languages and their em-allowed versions have been compared in the paper. As in [10], we have considered using fix-point operators to provide tractable recursion, and limiting the arity and height of higher order types.

In this paper, we have also addressed the relationship between properties such as embedded domain independence, finiteness and em-allowedness in various calculus-based query languages.

**References**


Appendix

We review here several query languages in the presence of external functions with arbitrary inputs and outputs (atomic values, sets, complex values, etc).

A complex value algebra

Let $\Sigma = \{ f_1, \ldots, f_l \}$ be a signature. We defined briefly the complex value algebra over $\Sigma$, $ALG^{cv}(\Sigma)$, following the formalism in [2]. Namely, $ALG^{cv}(\Sigma)$ contains all external functions $f_j$ in $\Sigma$, basic set operations, and projection:

- $\pi_j : T_1 \times \ldots \times T_n \rightarrow T_i$.
- $\set\_create: \{ T \} \rightarrow \{ \{ T \} \}$.
- $\set\_destroy: \{ \{ T \} \} \rightarrow \{ T \}$.
- $\tuple\_create: \{ T_1 \} \times \ldots \times \{ T_n \} \rightarrow \{ \langle A_1 : T_1, \ldots, A_n : T_n \rangle \}$.
- $\tuple\_destroy: \{ \langle T_1 \rangle \} \times \ldots \times \{ \langle T_n \rangle \} \rightarrow \{ \langle T_1 \rangle \}$.

The nested relational algebra $\mathfrak{NRA}(\Sigma)$ [6] is essentially equivalent to $ALG^{cv}(\Sigma)$ without powerset with external functions $f_1, \ldots, f_l$. A detailed description of these operators can be found in [2].

Calculus queries

The atomic calculus formulas are of the following form

$$ R(\tau_1, \ldots, \tau_n), \quad \tau = \tau', \quad \tau \tau' \quad \text{or} \quad \tau \subseteq \tau', $$

where $R \in Rel$ and the $\tau_i$, $\tau$ and $\tau'$ are terms constructed using $\text{dom}$, $\text{var}$, the function symbols in $F$. Formulas are constructed from atomic formulas using the standard connectives and quantifiers: $\land, \lor, \neg, \forall, \exists$. We denote the complex value calculus by $CALC^{cv}$.

Example The second component of tuples of relation $R:\{ < A, B > \}$ is replaced by its count number:

$$ \{ x \exists y (R(y)) \land x.A = y.A \land x.B = \text{count}(y.B) \}$$
A query $q$ is an expression $\{x_1, \ldots, x_n \mid \varphi(x_1, \ldots, x_n)\}$. Let the types of $x_1, \ldots, x_n$ be $\Gamma_1, \ldots, \Gamma_n$. The notion of the answer $q(I)$ to the query $q$ on the instance $I$ in the pre-interpretation $(d, F)$ is defined by:

$$q(d, F)(I) = \{(v_1, \ldots, v_n) \mid v_i \text{ is of type } \Gamma_i, I \models_0 F \varphi(v_1, \ldots, v_n)\}$$

**Datalog queries**

Query languages based on the deduction paradigm are extensions of Datalog to incorporate complex values. Those languages are based on the calculus and do not increase the expressive power of the ALG$^C^V$ or CALC$^C^V$. However, certain queries can be expressed in this deduction paradigm more efficiently and with lower complexity than they can be by using the powerset operator in the ALG$^C^V$. We briefly review the concept of Datalog for complex values.

**Definition** A database clause (rule) is an expression of the form

$$p(t) \leftarrow L_1, \ldots, L_n,$$

where the head $p$ is a derived predicate, and each $L_i$ of the body is a literal. A program $P$ is a finite set of rules extended with functions.

We distinguish between the intentional predicates and functions, which appear in heads of rules, and the extensional ones, which appear only in bodies.

**Definition** A Datalog query is a pair $(Q, P)$ where $Q$ is a formula of the form $\leftarrow W$, where $W \in$ CALC$^C^V$, and $P$ is a finite set of rules that includes definitions of predicate symbols which appear in $Q$.

**Fix-point queries**

It has been shown that fix-point operators are redundant in the context of unrestricted higher-order logic. However, a fix-point construct provide a tractable form of recursion, e.g., it can express transitive closure in polynomial space (time) and yield languages which are well-behaved with respect to expressive power. We present inflationary and non-inflationary extensions of the calculus with recursion.

**Partial fix-point operator**

Let $R$ be a database schema, and let $T[\tau_1, \ldots, \tau_m]$ be a typed relation which is not in $R$. Let $\varphi(T)$ be a formula using $T$ and relations in $R$, with $m$ free variables $x_1 : \tau_1, \ldots, x_m : \tau_m$. Then $\mu_T(\varphi(T))$ denotes the relation of type $[\tau_1, \ldots, \tau_m]$ which is the limit, if it exists, of the sequence $\{\Phi_n\}_{n \geq 0}$ defined by

$$\Phi_0 = \emptyset;$$

$$\Phi_n = \{ (x_1, \ldots, x_m) \mid \varphi(\Phi_{n-1}, x_1, \ldots, x_m)\}, \quad n > 0.$$ 

where $\varphi(\Phi_{n-1}, x_1, \ldots, x_m)$ denotes the result of evaluating $\varphi$ on the instance $I$ over $R$ and the instance $\Phi_{n-1}$ over $T$.

The expression $\mu_T(\varphi(T))$ can be used a term or as a relation in more complex formulas like any other relation. For example, if $x$ is a variable of type $[\tau_1, \ldots, \tau_m]$, $x = \mu_T(\varphi(T))$ is a fix-point formula.

The extension of the calculus with $\mu$ is called partial fix-point logic, denoted CALC$^C^V + \mu$. CALC$^C^V + \mu$ formula are built by repeated applications of CALC$^C^V$ operators and the partial fix-point operator, starting from atoms. CALC$^C^V + \mu$ queries over a database schema $R$ are expressions of the form

$$\{ < x_1, \ldots, x_n > \mid \varphi \}$$

where $\varphi$ is a CALC$^C^V + \mu$ formula.

**Inflationary fix-point Operator**

The definition of $\mu_T^i(\varphi(T))$ is identical to that of the partial fix-point operator except that the sequence $\{\Phi_n\}_{n \geq 0}$ is defined as follows:

$$\Phi_0 = \emptyset;$$

$$\Phi_n = \Phi_{n-1} \cup \varphi(\Phi_{n-1}), \quad n > 0.$$ 

Note that the sequence $\{\Phi_n\}_{n \geq 0}$ is increasing: $\Phi_{i-1} \subseteq \Phi_i$ for each $i > 0$, and the sequence converges in all cases.

**Example:** Let $R$ be a relation of sort $\text{dom}$. The power-set of $R$ is computed by $\{ \{ \mu_T^i(\varphi(T))(x) \} \mid T \}$, where $T$ is of sort $\{ \text{dom} \}$ and

$$\varphi(T)(y) \equiv \{ y = 0 \lor \exists x', y'(R(x') \land T(x') \land y = y' \land \{ x' \} \}$$

**COL Language**

The language COL is an extension of stratified Datalog$^C^V$ [3]. A key feature of the COL language is the use of base and derived data functions. Data functions are used to “name” set of objects: $F(t_1, \ldots, t_n) \ni t_0$ is an atom defining $F$. The term $F(t_1, \ldots, t_n)$ is interpreted as the set of all the objects $a$ such that $F(a_1, \ldots, a_n) \ni a$ hold. A program is a set of clauses of the form: $L \leftarrow L_1, \ldots, L_n$, where $L$ is an atom and the $L_i$ are literals. The defined symbol of a clause is either the relation occurring in the head or the function in the left most position (e.g., $F$ in $F(t_1, \ldots, t_n) \ni t_0$). Under some stratification restrictions, the semantics of programs is given by a minimal and justified model that can be computed using a finite sequence of fix-points.

**Example** Consider the following program $P$:

$$F(x) \ni y, y' \leftarrow R(x, y, y')$$

$$S(x, F(x)) \leftarrow R(x, y, y')$$

The predicate $R$ is extensionally defined, whereas the function $F$ and the predicate $S$ are intensionally defined.