TYPED QUERY LANGUAGES FOR DATABASES CONTAINING QUERIES[†]

FRANK NEVEN[1], JAN VAN DEN BUSSCHE[1], DIRK VAN GUCHT[2], and GOTTFRIED VOSSEN[3]

[1]Limburgs Universitair Centrum, Universitaire Campus, B-3590 Diepenbeek, Belgium
[2]Indiana University, Computer Science Department, Lindley Hall 401B, Bloomington, IN 47408, USA
[3]Institut für Informatik, Universität Münster, Steinfurter Straße 109, D-48161 Münster, Germany

Abstract — This paper introduces and studies the *relational meta algebra*, a statically typed extension of the relational algebra to allow for meta programming in databases. In this meta algebra one can manipulate database relations involving not only stored data values (as in classical relational databases) but also stored relational algebra expressions. Topics discussed include modeling of advanced database applications involving "procedural data"; desirability as well as limitations of a strict typing discipline in this context; equivalence with a first-order calculus; and global expressive power and non-redundancy of the proposed formalism. ©1999 Elsevier Science Ltd. All rights reserved

1. INTRODUCTION

Various advanced database systems, such as active and object-oriented systems, as well as the data dictionaries of standard relational database systems, provide the functionality of "stored procedures". The potential functionality of such systems was already envisaged by Stonebraker and his collaborators in the '80s [21, 22]. However, little work has been done on formal models providing logical foundations for such systems. Indeed, current systems approaches treat stored procedures simply as string values. Only the special case of "schema querying" has received a significant amount of attention (e.g., [8, 14]).

The purpose of the present paper is to contribute towards these needed logical foundations, by proposing and studying an extension of the relational algebra to allow for meta programming. The proposed *relational meta algebra*, denoted by $\mathcal{MA}$, extends the relational algebra with four new operators for computing with relations in which not only ordinary data values, but also relational algebra expressions can be stored. The first new operator is **extract**, used to extract subexpressions from stored expressions. The second is **rewrite**, used to rewrite subexpressions according to certain patterns (as is familiar from algebraic query optimization). The third is **wrap**, used to convert data values to relational algebra expressions.

The fourth and most important new operator of $\mathcal{MA}$ is **eval**, used to dynamically evaluate stored expressions. A fundamental property one wants to achieve is *type safety* of **eval**, in the sense that this dynamic evaluation never results in a run-time error. To guarantee type safety, the operators **extract** and **rewrite** are carefully calibrated so that they preserve syntactical correctness and so that the type of the expressions resulting from their manipulations is determined statically.

The type system we put on $\mathcal{MA}$ is an adaptation of the simple two-level type system discussed by Sheard and Hook in the context of Meta-ML [20]. We type ordinary relations by their width, relational algebra expressions by the type of their result relations, and relations containing relational algebra expressions by typing the columns as containing either ordinary data values or expressions of a designated type. Expressions of $\mathcal{MA}$, finally, are again typed by the type of their result relations (which may contain expressions).

[†]Recommended by Nicole Bidoit

The contents of this paper are summarized as follows. We begin by recalling the necessary definitions concerning relational databases and relational algebra, and introduce our extension of the relational database model to allow for stored relational algebra expressions in relations (Section 2). Then we introduce the operators of $\mathcal{MA}$ (Section 3) and give examples of interesting queries definable in $\mathcal{MA}$ (Section 4). After that, we investigate the expressive power of our formalism (Section 5). Specifically, we establish the following results:

1. We present a many-sorted first-order calculus whose "safe" fragment is equivalent to $\mathcal{MA}$, thus extending Codd's classical theorem on the equivalence of relational algebra and calculus [9][†]. This result is a generalization of Ross' [18], who worked in a model allowing only relation names, not general algebra expressions, to be stored in relations.

2. We illustrate an interesting limitation on the expressive power of $\mathcal{MA}$, due to its inherently typed nature: there are computationally extremely simple queries, well-typed at the input and output sides, which are nevertheless not definable in $\mathcal{MA}$, intuitively because their computation requires untyped intermediate results (which cannot be represented by a meta algebra computation). The equivalence with the calculus allows an elegant model-theoretic proof of this observation.

3. We show that $\mathcal{MA}$ is a conservative extension of the relational algebra, in the sense that as far as queries over ordinary relations (not containing stored expressions) are concerned, $\mathcal{MA}$ is no more expressive than the relational algebra[‡].

4. We give a rigorous proof of the intuitively clear fact that eval is a primitive operator: it cannot be simulated using the other operators. This stands in contrast to the situation in a complete programming language such as Lisp [1], where eval is clearly definable in Lisp without eval and thus not primitive. Also the other operators of $\mathcal{MA}$ are shown to be primitive.

The present paper is a follow-up on an earlier paper by three of us [25]. There, we studied the expressive power of evaluating stored relational algebra programs in a completely untyped setting. Relational algebra programs were encoded in data relations, and the standard operators of the relational algebra were used to manipulate these "program relations". This approach resulted in a powerful, but difficult to use, query language called the *reflective relational algebra (RA)*. Our main result was that by adding eval to the relational algebra much more queries on classical relational databases become definable. This stands in contrast to the conservative extension property of $\mathcal{MA}$ with respect to the standard relational algebra we prove here. In fact, our motivation for the work reported in this paper was the desire to understand the situation where typing and type safety are mandatory, and to design a formalism that is more programmer-friendly than $\mathcal{RA}$.

Two obvious directions for further research left open by our work are *(i)* to experiment with how our model for typed meta database programming can be applied in practice; and *(ii)* to better understand the precise expressive power of the relational meta algebra. Concerning *(i)*, it could be interesting to try to integrate our model into the SQL3 or OQL context. Concerning *(ii)*, a concrete open problem is whether or not the query "give all expressions of maximal length stored in relation R" is expressible in the relational meta algebra.

2. RELATIONS, EXPRESSIONS, AND META RELATIONS

2.1. Relational Databases and Relational Algebra

Assume a sufficiently large supply of *relation names* is given, where each relation name has an associated *arity* (a natural number). To denote that relation name R has arity n we write $R:n$. A *database schema* is a finite set of relation names.

[†]Generalizations of Codd's theorem to extensions of the relation model have always been a popular research topic (e.g., [13, 15, 2, 11, 5]).

[‡]Analogous conservative extension properties are known for complex object databases [17, 26, 23] and spatial databases [16].

Assume further a universe $\mathbf{V}$ of data values is given. A *relation of arity* n is a finite subset of $\mathbf{V}^n$. An *instance* of a database schema $\mathcal{S}$ is a mapping $\mathcal{I}$ on $\mathcal{S}$ which assigns to each relation name $R : n \in \mathcal{S}$ a relation $\mathcal{I}(R)$ of arity n.

Fix a schema $\mathcal{S}$. We denote the set of relational algebra expressions over $\mathcal{S}$ by $\mathcal{A}$. Each expression has an arity; as for relation names, to denote that expression e has arity n we write $e : n$. Formally:

- For each $v \in \mathbf{V}$, $\{(v)\} : 1$ is in $\mathcal{A}$.

- Each $S : n \in \mathcal{S}$ is in $\mathcal{A}$.

- If $e_1 : n$ and $e_2 : n$ are in $\mathcal{A}$, then so are $(e_1 \cup e_2) : n$ and $(e_1 - e_2) : n$.

- If $e_1 : n_1$ and $e_2 : n_2$ are in $\mathcal{A}$, then so is $(e_1 \times e_2) : n_1 + n_2$.

- If $e : n$ is in $\mathcal{A}$, then so are

  - $\sigma_{i=j}(e) : n$, where $i, j \in \{1, \ldots, n\}$; and
  - $\pi_{i_1, \ldots, i_p}(e) : p$, where $i_1, \ldots, i_p \in \{1, \ldots, n\}$.

Given an instance $\mathcal{I}$ of $\mathcal{S}$, an $\mathcal{A}$-expression $e : n$ over $\mathcal{S}$ evaluates to a relation of arity n, which we denote by $[\![e]\!]^{\mathcal{I}}$, in the well-known manner:

- $[\![\{(v)\}]\!]^{\mathcal{I}}$ is the constant one-column one-tuple relation $\{(v)\}$.

- $[\![R]\!]^{\mathcal{I}} := \mathcal{I}(R)$.

- $[\![e_1 \cup e_2]\!]^{\mathcal{I}} := [\![e_1]\!]^{\mathcal{I}} \cup [\![e_2]\!]^{\mathcal{I}}$.

- $[\![e_1 - e_2]\!]^{\mathcal{I}} := [\![e_1]\!]^{\mathcal{I}} - [\![e_2]\!]^{\mathcal{I}}$.

- $[\![e_1 \times e_2]\!]^{\mathcal{I}} := \{(x_1, \ldots, x_{n_1}, y_1, \ldots, y_{n_2}) \mid (x_1, \ldots, x_{n_1}) \in [\![e_1]\!]^{\mathcal{I}}, (y_1, \ldots, y_{n_2}) \in [\![e_2]\!]^{\mathcal{I}}\}$.

- $[\![\sigma_{i=j}(e)]\!]^{\mathcal{I}} := \{t \in [\![e]\!]^{\mathcal{I}} \mid t(i) = t(j)\}$.

- $[\![\pi_{i_1, \ldots, i_p}(e)]\!]^{\mathcal{I}} := \{(t(i_1), \ldots, t(i_p)) \mid t \in [\![e]\!]^{\mathcal{I}}\}$.

For selection and projection, the notation "$t(i)$" stands for the value of tuple t in column i.

Example 1 Suppose $\mathcal{S} = \{S : 2, T : 2\}$. Consider the $\mathcal{A}$-expression $e : 2 = \pi_{1,4}\sigma_{2=3}(S \times T)$ over $\mathcal{S}$. For any instance $\mathcal{I}$ of $\mathcal{S}$, which assigns concrete binary relations $\mathcal{I}(S)$ and $\mathcal{I}(T)$ to S and T, the binary relation $[\![e]\!]^{\mathcal{I}}$ equals the composition of $\mathcal{I}(S)$ and $\mathcal{I}(T)$.

Value selection can be expressed by combining the other operators: for example, $\sigma_{1=\text{`John'}}(S)$ can be expressed as $\pi_{1,2}\sigma_{1=3}(S \times \{(\text{`John'})\})$. $\qquad\square$

Note that in a projection $\pi_{i_1, \ldots, i_p}$, p is allowed to be 0, in which case we obtain a nullary relation. A nullary relation can contain the empty tuple, or it can be empty; these two cases are usually taken to represent the Boolean values true and false. Hence, nullary expressions can be used to express Boolean queries.

2.2. Extending the Model

We want to extend the basic relational database model to allow not only data values, but also relational algebra expressions to be stored in relations. Thereto, the simple type system based on arities has to be extended first:

Definition 1 A *type* is a tuple $\tau = [\tau_1, \ldots, \tau_n]$, where each τ_i is either the symbol 0, or is of the form $\langle m \rangle$, where m is a natural number. In the first case, we say that i is a *data column of* τ; in the second case, we say that i is an *expression column of* τ.

We can now define typed tuples, and relations, containing expressions as follows:

Definition 2 Let S be a schema, and let $\tau = [\tau_1, \ldots, \tau_n]$ be a type. A *tuple of type τ over S* is a tuple $(x_1, \ldots, x_n)$, such that for each $i = 1, \ldots, n$:

- if τ_i is 0 then x_i is a data value (i.e., an element of $\mathbf{V}$).

- if τ_i is $\langle m \rangle$ then x_i is an $\mathcal{A}$-expression over S, of arity m.

A *relation of type τ over S* is a finite set of tuples of type τ over S.

Note that a relation of type $[0, \ldots, 0]$ (n zeros) is an ordinary relation of arity n.

In the kind of systems we intend to model, there will be two kinds of relations. First, we have ordinary relations containing only data values; the schema consisting of the names of these relations is called the *object-level schema*. Second, we have relations containing both data values and $\mathcal{A}$-expressions over the object-level schema; the schema consisting of the names of these relations is then called the *meta-level schema*. Formally:

Definition 3
- A *meta-level schema* is a finite set of relation names, where each relation name has an associated type. To denote that a relation name R has type τ we write $R : \tau$.

- Let $\mathcal{M}$ be a meta-level schema, and let S be a schema disjoint from $\mathcal{M}$ (i.e., having no relation names in common). An *instance of $\mathcal{M}$ over S* is a mapping $\mathcal{J}$ on $\mathcal{M}$ which assigns to each relation name $R : \tau \in \mathcal{M}$ a relation of type τ over S. The pair $(S, \mathcal{M})$ is called a *combined schema*, in which S is referred to as the *object-level schema*.

- Finally, an *instance* of a combined schema $(S, \mathcal{M})$ is simply the union of an instance of S and an instance of $\mathcal{M}$ over S. We refer to such unions as *combined instances*.

Example 2 Let S be the schema of some database which is queried by several users, such as that of a bookstore on the Internet. Queries are represented as $\mathcal{A}$-expressions over S. Suppose we want to monitor the usage made of the database by the users. Then we may want to maintain a meta-level relation *Log* of type $[0, \langle n \rangle]$, containing pairs (u, q), where u is a username and q is a query u has posed. The expression column $\langle n \rangle$ indicates that we focus on queries of some fixed arity n. In this simple example, the object-level schema is S; an instance of S gives the concrete contents of the relations named in S. The meta-level schema $\mathcal{M}$ contains *Log* (and possibly other meta-level relation names); an instance of $\mathcal{M}$ over S gives the concrete contents of the relation *Log* (and possibly of others).   □

3. THE RELATIONAL META ALGEBRA

The relational algebra is a core language for defining queries on ordinary instances. We now want to have a similar formalism for defining queries on combined instances.

First, note that the five operators of the relational algebra can be canonically extended to work on meta-level relations as well as on ordinary, object-level relations. For instance, if $R : [\langle 3 \rangle, \langle 3 \rangle]$ is the name of a relation storing pairs of expressions of arity 3, we can write $\sigma_{1=2}(R)$ to retrieve those pairs from R with identical first and second components. However, the relational algebra operators do not recognize stored expressions as such; they are treated as abstract data values.

Hence, the five relational algebra operators are a good start, but additional operators are needed. We propose four new operators: **extract**, to extract subexpressions out of stored expressions; **rewrite**, to rewrite (subexpressions of) stored expressions; **wrap**, to convert data values into expressions; and **eval**, to dynamically evaluate stored expressions. So, **extract**, **rewrite** and **wrap** work syntactically on stored expressions, while **eval** works semantically. Adding these four operators to the relational algebra yields what we believe is the functionality one should expect from a core meta query language.

3.1. Syntax

We now formally define the expressions of the *relational meta algebra*. Each expression has a type, derived from that of its subexpressions; to denote that expression e has type τ we write $e : \tau$.

Definition 4 Fix a combined schema $(\mathcal{S}, \mathcal{M})$. The set $\mathcal{MA}$ of *relational meta algebra expressions over* $(\mathcal{S}, \mathcal{M})$ is the smallest set satisfying:

1. For each $v \in \mathbf{V}$, $\{(v)\} : [0]$ is in $\mathcal{MA}$.

2. Each relation name $S : n \in \mathcal{S}$ is in $\mathcal{MA}$, and is of type $[0, \ldots, 0]$ (n zeros).

3. Each relation name $R : \tau \in \mathcal{M}$ is in $\mathcal{MA}$.

4. If $e_1 : \tau$ and $e_2 : \tau$ are in $\mathcal{MA}$, then so are $(e_1 \cup e_2) : \tau$ and $(e_1 - e_2) : \tau$.

5. If $e_1 : \tau$ and $e_2 : \omega$ are in $\mathcal{MA}$ with $\tau = [\tau_1, \ldots, \tau_n]$ and $\omega = [\omega_1, \ldots, \omega_m]$, then so is $(e_1 \times e_2) : [\tau_1, \ldots, \tau_n, \omega_1, \ldots, \omega_m]$.

6. If $e : \tau$ is in $\mathcal{MA}$ with $\tau = [\tau_1, \ldots, \tau_n]$, then so are

   - $\sigma_{i=j}(e) : \tau$, where $i, j \in \{1, \ldots, n\}$ such that $\tau_i = \tau_j$; and

   - $\pi_{i_1, \ldots, i_p}(e) : [\tau_{i_1}, \ldots, \tau_{i_p}]$, where $i_1, \ldots, i_p \in \{1, \ldots, n\}$.

7. If $e : \tau$ is in $\mathcal{MA}$ with $\tau = [\tau_1, \ldots, \tau_n]$ and i is a data column of τ[†], then $\mathtt{wrap}_i(e) : [\tau_1, \ldots, \tau_n, \langle 1 \rangle]$ is in $\mathcal{MA}$.

8. If $e : \tau$ is in $\mathcal{MA}$ with $\tau = [\tau_1, \ldots, \tau_n]$ and i is an expression column of τ, then the following expressions are also in $\mathcal{MA}$:

   - $\mathtt{extract}_{i:m}(e) : [\tau_1, \ldots, \tau_n, \langle m \rangle]$, where m is a natural number;

   - $\mathtt{rewrite\text{-}one}_{i:\alpha \to \beta}(e)$ and $\mathtt{rewrite\text{-}all}_{i:\alpha \to \beta}(e)$, both of type $[\tau_1, \ldots, \tau_n, \tau_i]$, where $\alpha \to \beta$ is a *rewrite rule over* $\mathcal{S}$ *with respect to* τ (to be defined shortly); and

   - $\mathtt{eval}_i(e) : [\tau_1, \ldots, \tau_n, 0, \ldots, 0]$ (ℓ zeros), where ℓ is given by $\tau_i = \langle \ell \rangle$.

3.2. Rewrite Rules

To finish the above definition we need to define the system of rewrite rules on which the $\mathtt{rewrite}$ operators are based. Thereto the classical notion of a term rewrite rule [12] must be adapted to our setting.

Let $\mathcal{S}$ be a schema and let $\tau = [\tau_1, \ldots, \tau_n]$ be a type. Let $C \subseteq \{1, \ldots, n\}$ be the set of expression columns of τ, and for $j \in C$ let ℓ_j be given by $\tau_j = \langle \ell_j \rangle$.

Definition 5 A *rewrite rule over* $\mathcal{S}$ *with respect to* τ is a rule of the form $\alpha \to \beta$, where α and β are $\mathcal{A}$-expressions of the same arity, over the augmented schema $\mathcal{S} \cup \{\Box_j \mid j \in C\}$. Here, each $\Box_j$ is an *expression variable of arity* ℓ_j. We call α and β *patterns with respect to* τ.

An expression variable $\Box_j$ is formally nothing but a specially reserved relation name of arity ℓ_j; intuitively it should be thought of as a placeholder for subexpressions of arity ℓ_j.

[†]Recall Definition 1 for the notions of data and expression column.

3.3. Semantics of the Relational Meta Algebra

In the context of a given combined instance $\mathcal{K}$ of $(\mathcal{S}, \mathcal{M})$, an $\mathcal{MA}$-expression $e : \tau$ over $(\mathcal{S}, \mathcal{M})$ evaluates to a relation $[\![e]\!]^{\mathcal{K}}$ of type τ. We only define $[\![e]\!]^{\mathcal{K}}$ for cases 7 and 8 of Definition 4; the first 6 cases are completely analogous to the semantics of the standard relational algebra.

- $[\![\mathtt{wrap}_i(e)]\!]^{\mathcal{K}} := \{(x_1, \ldots, x_n, (x_i)) \mid (x_1, \ldots, x_n) \in [\![e]\!]^{\mathcal{K}}\}$.

- $[\![\mathtt{extract}_{i:m}(e)]\!]^{\mathcal{K}} := \{(x_1, \ldots, x_n, x) \mid (x_1, \ldots, x_n) \in [\![e]\!]^{\mathcal{K}}$ and x is a subexpression[†] of x_i that is of arity $m\}$.

- $[\![\mathtt{rewrite\text{-}one}_{i:\alpha \to \beta}(e)]\!]^{\mathcal{K}} := \{(x_1, \ldots, x_n, x) \mid (x_1, \ldots, x_n) \in [\![e]\!]^{\mathcal{K}}$ and x is obtained from x_i by replacing *one* occurrence of $f(\alpha)$ as a subexpression in x_i by $f(\beta)\}$. Here f is the mapping on the expression variables occurring in the rewrite rule defined by $f(\square_j) := x_j$.

- $[\![\mathtt{rewrite\text{-}all}_{i:\alpha \to \beta}(e)]\!]^{\mathcal{K}}$ is defined similarly, but now *every* occurrence of $f(\alpha)$ in x_i is replaced by $f(\beta)$.

- $[\![\mathtt{eval}_i(e)]\!]^{\mathcal{K}} := \{(x_1, \ldots, x_n, y_1, \ldots, y_\ell) \mid (x_1, \ldots, x_n) \in [\![e]\!]^{\mathcal{K}}$ and $(y_1, \ldots, y_\ell) \in [\![x_i]\!]^{\mathcal{I}}\}$.

Note that each operator extends each tuple with the result for that tuple, so that the relationship between input and output is preserved.

An $\mathcal{MA}$-expression $e : \tau$ over $(\mathcal{S}, \mathcal{M})$ defines a mapping $[\![e]\!]$ from the set of combined instances of $(\mathcal{S}, \mathcal{M})$ to the set of relations of type τ. Such a mapping is called a *query over* $(\mathcal{S}, \mathcal{M})$ *of type* τ.

We now give some examples to illustrate the definition of the semantics of the basic operators of the meta algebra.

Example 3 We use the combined schema $(\mathcal{S}, \mathcal{M})$, where $\mathcal{S} = \{S : 2, T : 2, U : 2\}$ and $\mathcal{M} = \{R : [0, \langle 4 \rangle, 0, \langle 2 \rangle]\}$. Let $\mathcal{K}$ be a combined instance where $\mathcal{K}(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S

1. $\mathtt{wrap}_1(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S	(a)
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	(b)
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	(c)

2. $\mathtt{extract}_{2:4}(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	$\sigma_{1=4}(S \times T) \cup (S \times S)$
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	$\sigma_{1=4}(S \times T)$
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	$S \times S$
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	$S \times T$
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	$T \times S$

[†]By *subexpression* we mean direct *and* indirect ones. So the subexpressions of $\pi_{1,4}\sigma_{2=3}(R \times S)$ are the expression itself; $\sigma_{2=3}(R \times S)$; $R \times S$; R; and S.

3. $\texttt{rewrite-one}_{2:\square_4 \to T}(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S	$\sigma_{1=2}(T) \times \sigma_{1=2}(T)$
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(T \times (T \times S))$
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S)$	f	S	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times T))$

4. $\texttt{rewrite-all}_{2:\square_4 \to T}(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S	$\sigma_{1=2}(T) \times \sigma_{1=2}(T)$
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	$\sigma_{1=4}(S \times T) \cup (S \times S)$
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(T \times (T \times T))$

5. The above results are independent of the values $\mathcal{K}(S)$ and $\mathcal{K}(T)$. This is of course not so for the eval operator, which evaluates expressions over the object-level relations. For example, if $\mathcal{K}(S)$ and $\mathcal{K}(T)$ are the following relations

$$
\begin{array}{c|cc}
S & & \\
\hline
& x & y \\
& x & x
\end{array}
\qquad
\begin{array}{c|cc}
T & & \\
\hline
& u & u \\
& u & x
\end{array}
$$

then $\texttt{eval}_2(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S	x	x	u	u
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	x	y	u	x
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	x	x	u	x
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	x	y	x	y
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	x	y	x	x
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	x	x	x	y
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	x	x	x	x
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	x	y	u	x
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	x	x	u	x

$\square$

3.4. Derived Operators

We next exhibit a variety of derived operators that can be expressed in the meta algebra in a similar way operators like semi-join and division can be expressed in the standard relational algebra. These derived operators illustrate the expressive power of the meta algebra and will turn out useful in various contexts considered below.

We first define the construct operator which constructs new relation algebra expressions from relation algebra expressions stored in relations. This derived operator is very convenient for manipulating relation algebra expressions, as is illustrated in the examples in the next section. Moreover, this operator is used heavily in its most simplest form in the proof of Theorem 4.

At first sight, the pattern matching mechanism of the meta algebra seems rather limited, since $\square$-variables in patterns can only be instantiated by $\mathcal{A}$-expressions occurring as components in tuples in relations. Therefore, we introduce a more liberal notion of patterns where variables can be instantiated by arbitrary $\mathcal{A}$-expressions, and define extract, rewrite-one and rewrite-all operators with such patterns.

Finally, building further on the previous derived operators, we define derived operators selecting tuples matching some pattern. Clearly, the latter operators are useful to check whether a certain $\mathcal{A}$-expression occurs as a subexpression of a component of some tuple; they are used as such in the examples of the next section.

3.4.1. Construct

The first operator constructs new relational algebra expressions from relational algebra expressions stored in relations. Let $R : \tau$ be a relation with $\tau = [\tau_1, \ldots, \tau_n]$, and let $\alpha : m$ be a pattern over S with respect to τ. Then $\mathtt{construct}_\alpha(R)$ is the relation of type $[\tau_1, \ldots, \tau_n, \langle m \rangle]$ consisting of all tuples $(x_1, \ldots, x_n, f_{x_1, \ldots, x_n}(\alpha))$, where $(x_1, \ldots, x_n) \in R$ and f is the mapping on the expression variables occurring in α defined by $f(\square_j) := x_j$.

Example 4 Let R be as in Example 3. Then $\mathtt{construct}_{(\square_2) \cap (\square_4 \times \square_4)}(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S	$(\sigma_{1=2}(S) \times \sigma_{1=2}(T)) \cap (S \times S)$
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	$(\sigma_{1=4}(S \times T) \cup (S \times S)) \cap (U \times U)$
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	$(\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))) \cap (S \times S)$

and $\mathtt{construct}_{\pi_{1,4}\sigma_{2=3}(\square_4 \times \square_4)}(R)$ equals

a	$\sigma_{1=2}(S) \times \sigma_{1=2}(T)$	d	S	$\pi_{1,4}\sigma_{2=3}(S \times S)$
b	$\sigma_{1=4}(S \times T) \cup (S \times S)$	e	U	$\pi_{1,4}\sigma_{2=3}(U \times U)$
c	$\pi_{1,2,3,4}\sigma_{2=6}\sigma_{4=5}(S \times (T \times S))$	f	S	$\pi_{1,4}\sigma_{2=3}(S \times S)$

$\square$

The $\mathtt{construct}$ operator can be expressed in the meta algebra. For any $v \in \mathbf{V}$, define $v_m : m$ as the relational algebra expression

$$\underbrace{\{(v)\} \times \{(v)\} \times \ldots \times \{(v)\}}_{m \text{ times}}.$$

We then denote by e^{v_m} the meta algebra expression

$$\pi_4 \mathtt{extract}_{3:m}\mathtt{rewrite\text{-}one}_{2:(v) \to \pi_1(v_m)}\mathtt{wrap}_1(\{(v)\})$$

of type $[\langle m \rangle]$ which constructs the relation $\{(v_m)\}$. The operator $\mathtt{construct}_\alpha(R)$ now equals

$$\pi_{1,\ldots,n,n+2}\mathtt{rewrite\text{-}all}_{n+1:v_m \to \alpha}(R \times e^{v_m}).$$

For any relational algebra expression e, we denote by $\langle e \rangle$ the expression

$$\pi_2 \mathtt{construct}_e(\{(v)\}),$$

for some arbitrary data value v. We denote the induced operator by $\langle \cdot \rangle$. This derived operator will be used in the proof of Theorem 4.

3.4.2. Generalized Patterns

Next we generalize patterns with new expression variables: so-called $*$-variables. These expressions variables are instantiated by arbitrary subexpressions, in contrast to the $\square$-variables which are only instantiated by expressions occurring as components of tuples in relations.

So assume given additional expression variables of all possible arities, denoted by a possibly subscripted $*$. Generalizing Definition 5, a *generalized pattern* over a schema S with respect to a type τ is a relational algebra expression over the augmented schema $S \cup \{\square_j \mid j \in C\} \cup \mathcal{V}$, where $\mathcal{V}$ is a set of $*$-variables. A *generalized rewrite rule* over S with respect to τ is of the form $\alpha \to \beta$, where α and β are now generalized patterns such that all $*$-variables in β occur in α, C and the $\square_j$ are as in Definition 5.

We now define $\mathtt{extract}$, $\mathtt{rewrite\text{-}one}$ and $\mathtt{rewrite\text{-}all}$ operators in terms of these generalized patterns and show that they can be simulated in the meta algebra.

Let $R : \tau$ be a relation with $\tau = [\tau_1, \ldots, \tau_n]$, let i be an expression column of τ, and let $\alpha : m$ be a generalized pattern. Then $\mathtt{extract}_{i:\alpha}(R)$ is the relation of type $[\tau_1, \ldots, \tau_n, \langle m \rangle]$ consisting of all tuples $(x_1, \ldots, x_n, f(\alpha))$, where $(x_1, \ldots, x_n) \in R$ and f is a mapping on the expression variables of α such that

- $f(\alpha)$ is a subexpression of x_i;

- $*$ and $f(*)$ are of the same arity, for each $*$ in α; and

- $f(\Box_j) = x_j$, for each $j \in C$.

Example 5 Let $\mathcal{S} = \{S : 2, T : 2\}$ and let R be the following relation of type $[\langle 2 \rangle, \langle 4 \rangle]$:

T	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$
S	$\sigma_{1=4}(T \times S) \cup (T \times T)$

Then $\mathtt{extract}_{2:\sigma_{1=2}(*)}(R)$, where the arity of $*$ is 2, yields the following relation:

T	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$	$\sigma_{1=2}(T)$
T	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$	$\sigma_{1=2}(S)$

$\Box$

The operator $\mathtt{extract}_{i:\alpha}(R)$ can be expressed in the meta algebra. Suppose α contains the $*$-variables $*_1, \ldots, *_r$ where for each $j = 1, \ldots, r$, $*_j$ is of arity s_j. Let R^* be the expression

$$\mathtt{extract}_{i:s_r} \cdots \mathtt{extract}_{i:s_1} \mathtt{extract}_{i:m}(R).$$

Then $\mathtt{extract}_{i:\alpha}(R)$ is expressed by

$$\pi_{1,\ldots,n+1} \sigma_{n+r+1=n+1} \mathtt{construct}_{\alpha'}(R^*),$$

where α' is obtained from α by replacing each $*_j$ by $\Box_{n+j}$.

Let $R : \tau$ be a relation with $\tau = [\tau_1, \ldots, \tau_n]$, let i be an expression column of τ and let $\alpha \to \beta$ be a generalized rewrite rule over $\mathcal{S}$ with respect to τ. Then $\mathtt{rewrite\text{-}one}_{i:\alpha \to \beta}(R)$ is the relation of type $[\tau_1, \ldots, \tau_n, \tau_i]$ consisting of all tuples $(x_1, \ldots, x_n, x)$, where $(x_1, \ldots, x_n) \in R$, f is a mapping on the expression variables occurring in α such that

- $f(\alpha)$ is a subexpression of x_i;

- $*$ and $f(*)$ are of the same arity, for each $*$ in α; and

- $f(\Box_j) = x_j$, for each $j \in C$, and

x is obtained from x_i by replacing one occurrence of the subexpression $f(\alpha)$ in x_i by $f(\beta)$.

The restriction that all $*$-variables of β also have to occur in α makes sure that each mapping f on the variables of α uniquely determines $f(\beta)$.

Example 6 Consider relation R from Example 5 again. Then

$$\mathtt{rewrite\text{-}one}_{2:* \to S}(R),$$

where the arity of $*$ is 2, equals

T	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$	$\sigma_{1=2}(S) \times \sigma_{1=2}(S)$
T	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$	$S \times \sigma_{1=2}(S)$
T	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$
T	$\sigma_{1=2}(T) \times \sigma_{1=2}(S)$	$\sigma_{1=2}(T) \times S$
S	$\sigma_{1=4}(T \times S) \cup (T \times T)$	$\sigma_{1=4}(S \times S) \cup (T \times T)$
S	$\sigma_{1=4}(T \times S) \cup (T \times T)$	$\sigma_{1=4}(T \times S) \cup (T \times T)$
S	$\sigma_{1=4}(T \times S) \cup (T \times T)$	$\sigma_{1=4}(T \times S) \cup (S \times T)$
S	$\sigma_{1=4}(T \times S) \cup (T \times T)$	$\sigma_{1=4}(T \times S) \cup (T \times S)$

and the expression $\mathtt{rewrite\text{-}one}_{2:\square_1\times*\to\sigma_{1=2}(\square_1)\times*}(R)$, where the arity of $*$ is 2, yields

T	$\sigma_{1=4}(T\times S)\cup(T\times T)$	$\sigma_{1=4}(\sigma_{1=2}(T)\times S)\cup(T\times T)$
T	$\sigma_{1=4}(T\times S)\cup(T\times T)$	$\sigma_{1=4}(T\times S)\cup(\sigma_{1=2}(T)\times T)$

$\square$

To show that a rewrite operation involving generalized rewrite rules can be expressed in the meta algebra, consider $\mathtt{rewrite\text{-}one}_{i:\alpha\to\beta}(R)$, where $*_1,\ldots,*_r$ are the $*$-variables that occur in $\alpha\to\beta$ and for $j=1,\ldots,r$, $*_j$ is of arity s_j. Let R^* be the expression

$$\mathtt{extract}_{i:s_r}\ldots\mathtt{extract}_{i:s_1}(R),$$

Then $\mathtt{rewrite\text{-}one}_{i:\alpha\to\beta}(R)$ is expressed by

$$\pi_{1,\ldots,n,n+r+1}\mathtt{rewrite\text{-}one}_{i:\alpha'\to\beta'}(R^*),$$

where α' (β') is obtained from α (β) by replacing each variable $*_j$ by $\square_{r+j}$.

Let $R:\tau$ be a relation with $\tau=[\tau_1,\ldots,\tau_n]$, let i be an expression column of τ and let $\alpha\to\beta$ be a generalized rewrite rule over S with respect to τ. Then $\mathtt{rewrite\text{-}all}_{i:\alpha\to\beta}(R)$ is the relation of type $[\tau_1,\ldots,\tau_n,\tau_i]$ consisting of all tuples $(x_1,\ldots,x_n,x)$, where $(x_1,\ldots,x_n)\in R$, f is a mapping on the expression variables occurring in α such that

- $f(\alpha)$ is a subexpression of x_i;

- $*$ and $f(*)$ are of the same arity, for each $*$ in α; and

- $f(\square_j)=x_j$, for each $j\in C$, and

x is obtained from x_i by replacing all occurrences of the subexpression $f(\alpha)$ in x_i by $f(\beta)$.

Example 7 Consider relation R from Example 5. Then $\mathtt{rewrite\text{-}all}_{2:*\to S}(R)$, where the arity of $*$ is 2, equals

T	$\sigma_{1=2}(T)\times\sigma_{1=2}(S)$	$\sigma_{1=2}(S)\times\sigma_{1=2}(S)$
T	$\sigma_{1=2}(T)\times\sigma_{1=2}(S)$	$S\times\sigma_{1=2}(S)$
T	$\sigma_{1=2}(T)\times\sigma_{1=2}(S)$	$\sigma_{1=2}(T)\times\sigma_{1=2}(S)$
T	$\sigma_{1=2}(T)\times\sigma_{1=2}(S)$	$\sigma_{1=2}(T)\times S$
S	$\sigma_{1=4}(T\times S)\cup(T\times T)$	$\sigma_{1=4}(T\times S)\cup(T\times T)$
S	$\sigma_{1=4}(T\times S)\cup(T\times T)$	$\sigma_{1=4}(S\times S)\cup(S\times S)$

$\square$

Now $\mathtt{rewrite\text{-}all}_{i:\alpha\to\beta}(R)$ is expressed by

$$\pi_{1,\ldots,n,n+r+1}\mathtt{rewrite\text{-}all}_{i:\alpha'\to\beta'}(R^*),$$

where α' (β') is obtained from α (β) by replacing each variable $*_j$ by $\square_{r+j}$, and R^* is as above.

3.4.3. Matching

Our last two derived operators are straightforward abbreviations of expressions using $\mathtt{extract}$. For a relation $R:[\tau_1,\ldots,\tau_n]$

1. Define
$$\mathtt{match}_{i:\alpha}(R):=\pi_1\cdots\pi_n\sigma_{i=n+1}\mathtt{extract}_{i:\alpha}(R).$$
   This operator selects all tuples t for which the pattern α matches $t(i)$.

2. Define
$$\mathtt{submatch}_{i:\alpha}(R):=\pi_1\cdots\pi_n\mathtt{extract}_{i:\alpha}(R).$$
   This operator selects all tuples t for which the pattern α matches a subexpression of $t(i)$.

Note that the definitions stated above do not give immediate rise to efficient implementations; there are more direct ways to implement matching and sub-matching. A similar remark applies to the other derived operators presented in this section.

4. APPLICATIONS OF THE RELATIONAL META ALGEBRA

In this section we discuss several applications of the meta algebra which also serve to illustrate its expressive power. To this end, we will first introduce an example situation already hinted upon in Example 2.

4.1. An Example

Consider the schema S of a bookstore database which can be queried by users over the Internet. The bookstore puts relations it gets from publishers into this database, but the database administrator does some integration of these relations into views before the database goes on-line. The bookstore provides access to several such predefined views; at the same time, it wants to monitor the usage made of the database by users.

More specifically, let schema S contain publisher-specific information such as the following: Publisher A has a distinct relation for each subject area he is active in, e.g., $ACompSci$ for computer science books, $AMath$ for mathematics, $APhys$ for physics and so on. Each of these relations keeps track of *subarea, author, title, price,* and *ISBN*. Publisher B provides one relation $BSubject$ for information on *subject, area* and *ISBN*; another relation $BBook$ specifies *ISBN, price, title,* and *author*. So $S = \{ACompSci, AMath, APhys, BSubject, BBook, \dots\}$.

We assume querying is done through a predefined form into which selection criteria can be entered. The form provides access to views such as *all computer science books, all books, all database books,* etc. Views definitions and queries over these views are naturally formalized as $\mathcal{A}$-expressions over S. For example, the view regarding 'databases' could be defined as follows:

$$\pi_{3,2,4,5}\sigma_{1='databases'}(ACompSci)$$
$$\cup \ \pi_{5,6,4,3}(\sigma_{1='computer\ science'}\sigma_{2='pers.\ prog.\ lang.'}(BSubject \bowtie BBook)$$
$$\cup \ \sigma_{1='computer\ science'}\sigma_{2='info.\ systems'}(BSubject \bowtie BBook)).$$

Similarly, the view regarding 'all' could have the following definition:

$$\pi_{3,2,4,5}(ACompSci) \cup \ \pi_{5,6,4,3}(BSubject \bowtie BBook).$$

Notice that we make use of shorthand notations here; for example, the natural join is not formally part of the relational algebra but is a well-known short-hand.

For the purposes mentioned above, the bookstore maintains the meta schema $\mathcal{M} = \{$ *Views, Log*$\}$, where both meta relations are of type $[0, \langle 4 \rangle]$. *Views* contains pairs (n, e) where n is the name of a view and e is the expression (of arity 4) defining the view. Whenever a user poses a query, such as "show all books by author Smith", an entry in relation *Log* is made, which contains pairs (u, q) where u is the name of a user and q is a query u has posed.

4.2. View Management Queries

We now give several examples of how to use the meta algebra to express view management queries. For the sample application described above, suppose that publisher A changes the name of relation $ACompSci$ to ACS; a corresponding update of all entries in relation *Views* can be done by

$$\pi_{1,3}\texttt{rewrite-all}_{2:ACompSci \to ACS}(Views).$$

The next query retrieves all views that do not use relations from publisher B:

$$Views - \bigcup_{R \in \mathcal{R}} (\texttt{submatch}_{2:R}(Views)),$$

where $\mathcal{R}$ denotes the set of all relations from publisher B. With the following meta algebra query we can select those pairs of views that evaluate to the same relation, based on the current state of the database:

$$\pi_{1,3}(Views \times Views) - \pi_{1,3}\texttt{eval}_5\texttt{construct}_{(\square_2 - \square_4) \cup (\square_4 - \square_2)}(Views \times Views)$$

This query constructs an expression computing the symmetric difference of two views and evaluates it. The resulting pairs are of views that are distinct; hence the application of the difference operator to get the equal (i.e., non-distinct) pairs.

Adding, a new view can be done easily. If e is the definition of a new view called 'logic', then we add ('logic') $\times$ ($\langle e \rangle$) to *Views*.

Now suppose publisher A ships, in a meta-relation *Updates*, pairs of the form (e_R, R), where R is the name of a relation coming from publisher A, and e_R is an expression that evaluates to a new content for relation R. For example, a pair in relation *Updates* could be

$$A\mathit{CompSci} - (\{(\text{`AI'})\} \times (\ldots) \times \cdots) \cup (\{(\text{`Java'})\} \times (\ldots) \times \cdots) \quad , \quad A\mathit{CompSci}.$$

To find out which views will be affected by these updates, provided they have not yet been applied, we can use the following meta algebra query:

$$\pi_1 \mathtt{eval}_2 \pi_{5,9} \mathtt{construct}_{(\square_8 - \square_6) \cup (\square_8 - \square_6)} \sigma_{2 \neq 4} \mathtt{rewrite\text{-}all}_{7:\square_4 \to \square_3}$$
$$\mathtt{rewrite\text{-}all}_{6:\square_2 \to \square_1}(\mathit{Updates} \times \mathit{Updates} \times \mathit{Views}).$$

This query tentatively replaces each relation appearing in a view definition by its updated version. Then the old and the new view are compared in the same way as already done earlier.

Similar techniques can be used to express query expansion: given a set of queries over the views, expand the view names by their definition. We leave this as an exercise to the reader.

Finally, suppose we want to compute all pairs (x, y) such that x is an expression stored in *Views* and $y : 2$ is a subexpression of x occurring at least twice in x. The naive attempt

$$\pi_{2,3} \sigma_{3=4} \mathtt{extract}_{2:2} \mathtt{extract}_{2:2}(\mathit{Views})$$

is incorrect; to distinguish different occurrences of the same subexpression we have to mark them in some way. This can be done using $\mathtt{rewrite\text{-}one}$. Assume we have some dummy relation name $D \in \mathcal{S}$ of arity 0. An occurrence x of a subexpression can now be "marked" by rewriting it into $x \times D$. So if $\mathtt{mark}$ is the following meta algebra expression:

$$\pi_{2,3,4} \mathtt{rewrite\text{-}one}_{2:\square_3 \to \square_3 \times D} \mathtt{extract}_{2:2}(\mathit{Views}),$$

then the wanted query is expressible as

$$\pi_{1,2} \sigma_{3 \neq 6} \sigma_{2=5} \sigma_{1=4}(\mathtt{mark} \times \mathtt{mark}).$$

4.3. Queries on Queries

We next illustrate how the meta algebra can be used to query the queries stored in the *Log* relation, in order to find out the behavior of the users of the Internet site.

If we want to see for every user u the results of all queries posed by u (as recorded in *Log*) evaluated on the current instance, we simply write

$$\pi_{1,3,4,5,6} \mathtt{eval}_2(\mathit{Log}).$$

To determine all queries that gave no result, we use

$$\pi_2(\mathit{Log}) - \pi_2 \mathtt{eval}_2(\mathit{Log}).$$

To find out all users that used relation $A\mathit{CompSci}$ in a query but never relation $A\mathit{Math}$, the following expression can be used:

$$\pi_1 \mathtt{submatch}_{2:A\mathit{CompSci}}(\mathit{Log}) - \pi_1 \mathtt{submatch}_{2:A\mathit{Math}}(\mathit{Log}).$$

The following query expression will return the union of the results of the queries posed by user Jones:

$$\pi_{3,4,5,6} \mathtt{eval}_2 \sigma_{1=\text{`Jones'}}(\mathit{Log}).$$

To obtain the intersection of the results of the queries posed by Jones, the following can be done. Let adom be the relational algebra expression that computes the active domain of the database. Then the query is expressed by

$$\pi_{3,4,5,6}\texttt{eval}_2\sigma_{1=\text{`Jones'}}(Log) - \pi_{4,5,6,7}\texttt{eval}_3\texttt{construct}_{adom-\square_2}\sigma_{1=\text{`Jones'}}(Log).$$

The next query retrieves the items that occur as a result in at least two queries posed by Jones:

$$\pi_{6,7,8,9}\texttt{eval}_5\texttt{construct}_{\square_2\cap\square_4}\sigma_{1=\text{`Jones'}\wedge 1=3\wedge 2\neq 4}(Log \times Log).$$

The last query returns all the data values appearing as constants in queries asked by Jones:

$$\pi_4\sigma_{3=5}(\texttt{extract}_{2:1}\sigma_{1=\text{`Jones'}}(Log) \times \texttt{wrap}_1(adom)).$$

5. EXPRESSIVE POWER OF THE RELATIONAL META ALGEBRA

5.1. An Equivalent Calculus

Codd's classical theorem [3, 9, 24] says that the queries expressible in the relational algebra are precisely the queries definable in first-order logic (in this context referred to as the *relational calculus*). We now show how this equivalence can be extended to the meta algebra by introducing $\mathcal{MC}$, the *relational meta calculus*. This equivalent calculus will also prove to be helpful in establishing properties of the expressive power of the meta algebra (see Sections 5.2 and 5.3).

Fix a combined schema $(\mathcal{S}, \mathcal{M})$. Our calculus uses two kinds of variables: *data variables* and *expression variables*. Data variables will range over $\mathbf{V}$ (the universe of data values). Expression variables have an associated arity and range over the $\mathcal{A}$-expressions over $\mathcal{S}$ of that arity.

Terms are defined as follows:

- data variables and data values are terms *of sort* 0;

- an $\mathcal{A}$-expression over the augmentation of $\mathcal{S}$ with a finite set of expression variables is a term *of sort* $\langle n \rangle$, where n is the arity of the expression; and

- an expression of the form $\{(x)\}$, where x is a data variable or a data value, is a term *of sort* $\langle 1 \rangle$.

Atomic formulas can be of one of the following forms:

- $S(x_1, \ldots, x_n)$, where $S : n \in \mathcal{S}$ and each x_i is a data variable;

- $R(t_1, \ldots, t_n)$, where $R : [\tau_1, \ldots, \tau_n] \in \mathcal{M}$ and each t_i is a term of sort τ_i;

- $t_1 = t_2$ and $t_1 \leq t_2$, where t_1 and t_2 are terms of the same expression sort;

- $\texttt{rewrite-one}(t_1, t_2, t_3, t_4)$ and $\texttt{rewrite-all}(t_1, t_2, t_3, t_4)$, where $t_1, \ldots, t_4$ are terms such that t_1 and t_4, and t_2 and t_3 have the same expression sort, respectively; and

- $\texttt{eval}(t, x_1, \ldots, x_n)$, where t is a term of sort $\langle n \rangle$ and $x_1, \ldots, x_n$ are data variables.

Formulas, finally, are built from atomic formulas in the standard manner using Boolean connectives and quantifiers. The set of all formulas is denoted by $\mathcal{MC}$.

Assume given an $\mathcal{MC}$-formula φ, a combined instance $\mathcal{K} = (\mathcal{I}, \mathcal{J})$ of $(\mathcal{S}, \mathcal{M})$, and a valuation ρ of the free variables of φ mapping data variables to data values and expression variables to $\mathcal{A}$-expressions over $\mathcal{S}$ of the right arity. The truth of φ in $\mathcal{K}$ under ρ, denoted by $\mathcal{K} \models \varphi[\rho]$, is defined in the standard way given the following semantics for the above terms and predicates:

- the term $\rho(\{(x)\})$ equals the constant $\mathcal{A}$-expression $\{(\rho(x))\}$;

- $t_1 \leq t_2$ means that $\rho(t_1)$ is a subexpression of $\rho(t_2)$;

- `rewrite-one`(t_1, t_2, t_3, t_4), respectively `rewrite-all`(t_1, t_2, t_3, t_4), means that $\rho(t_4)$ is obtained from $\rho(t_1)$ by replacing one, respectively every, occurrence of $\rho(t_2)$ in $\rho(t_1)$ by $\rho(t_3)$; and

- `eval`$(t, x_1, \ldots, x_n)$ means that $(x_1, \ldots, x_n)$ is in the result of evaluating $[\![\rho(t)]\!]^{\mathcal{I}}$.

An $\mathcal{MC}$-formula φ with free variables $x_1, \ldots, x_n$ of sorts $\tau_1, \ldots, \tau_n$, respectively, defines the query q of type $[\tau_1, \ldots, \tau_n]$ defined by $q(\mathcal{K}) = \{(\rho(x_1), \ldots, \rho(x_n)) \mid \mathcal{K} \models \varphi[\rho]\}$. Of course this is only well-defined if $q(\mathcal{K})$ is finite for every $\mathcal{K}$. However, a syntactical restriction called *safety* can be put on $\mathcal{MC}$-formulas such that finiteness is guaranteed. Our definition of safety is a natural extension of the definition given by Ullman [24] for the classical relational calculus.

Definition 6 A meta calculus formula is *safe* if:

- It does not contain $\forall$;

- Any subformula of the form $\varphi \vee \psi$ is such that φ and ψ have the same free variables;

- Let σ be a maximal subformula of the form $\delta_1 \wedge \ldots \wedge \delta_n$. (Maximal in the sense that there is no longer subformula of the form $\delta \wedge \sigma$ or $\sigma \wedge \delta$.) Then we must be able to deduce that every free variable of σ is *limited* using the following deduction rules: If x is a data variable then x is limited if

  1. x occurs free in one of the δs that is not negated (i.e., not of the form $\neg \xi$) and that is not $x = y$ or `eval`;

  2. one of the δs is of the form $x = v$, where $v \in \mathbf{V}$;

  3. one of the δs is of the form $x = y$ and y is already limited;

  4. one of the δs is of the form $\{(x)\} = y$, where y is already limited; or

  5. one of the δs is of the form `eval`$(t, y_1, \ldots, y_m)$, where x is one of the ys and all variables occurring in t are already limited.

  If x is an expression variable, then x is limited if

  6. x occurs free in one of the δs that is not negated (i.e., not of the form $\neg \xi$) and that is not of the form $t_1 = t_2$, $t_1 \leq t_2$, `rewrite-one`, `rewrite-all`, or `eval`;

  7. one of the δs is of the form $t_1 = t_2$ or $t_2 = t_1$, where x occurs in t_1 and all the variables occuring in t_2 are already limited;

  8. one of the δs is of the form $t_1 \leq t_2$, where x occurs in t_1 and all the variables of t_2 are already limited;

  9. one of the δs is of the form $\{(y)\} = x$, where y is already limited; or

  10. one of the δs is of the form `rewrite-one`(t_1, t_2, t_3, t_4) or `rewrite-all`(t_1, t_2, t_3, t_4), where x occurs in t_4 and all variables occurring in t_1, t_2 and t_3 are already limited.

Before we prove that the meta algebra and the safe meta calculus are equivalent we show how some of the example queries in Section 4 can be expressed in this calculus.

Example 8 The query that changes in each view definition the name of relation *ACompSci* to *ACS*:

$$\varphi(x, y) \equiv (\exists z)(\textit{Views}(x, z) \wedge \texttt{rewrite-all}(z, ACompSci, ACS, y)).$$

The query that retrieves the pairs of views that evaluate to the same relation:

$$\varphi(x, y) \equiv (\exists v)(\exists w)(\textit{Views}(x, v) \wedge \textit{Views}(y, w) \wedge (\forall \bar{z})(\texttt{eval}(v, \bar{z}) \leftrightarrow \texttt{eval}(w, \bar{z}))).$$

The query that selects all users that have used the relation $ACompSci$ in a query but never relation $AMath$:

$$\varphi(x) \equiv (\exists y)(ACompSci \le y \land Log(x,y)) \land \neg(\exists y)(AMath \le y \land Log(x,y)).$$

The query that returns the intersection of the results of the queries posed by user Jones:

$$\varphi(\bar{x}) \equiv (\forall y)(Log(Jones, y) \to \texttt{eval}(y, \bar{x})).$$

$\square$

We now show:

Theorem 1 *The meta algebra and the safe meta calculus are equivalent.*

Proof. It is very easy to construct for every meta algebra expression an equivalent safe meta calculus formula. The proof goes along the lines of the well known proof that the standard relational algebra can be expressed in the safe relational calculus [9, 24] and is therefore omitted.

The essential part of the proof of the other direction is the construction, for any safe calculus formula $\varphi(\bar{x})$, of an algebra expression dom_φ such that on any instance $\mathcal{K}$ and for any valuation ρ, $\mathcal{K} \models \varphi[\rho]$ implies $\rho(\bar{x}) \in [\![\text{dom}_\varphi]\!]^{\mathcal{K}}$. In the well-known proof of the simulation of the standard relational calculus by the standard relational algebra, dom_φ is simply the expression computing the active domain. In our case, however, new expressions can be generated by calculus expressions, so the definition of dom_φ is a bit more complicated.

Let φ be a meta calculus formula. Define $\text{dom}_\varphi^{0,1}$ as the meta algebra expression that computes the union of the set of data values in the active domain with the set of data values occurring in φ. Let p be the maximum arity of a term occurring in φ, let T be the set of all terms occurring in φ, and let T^{ground} be the set consisting of all ground terms that are subexpressions of terms occurring in φ. For $j = 1, \ldots, p$, define the meta algebra expression $\text{dom}_\varphi^{\langle j \rangle, 1}$ of type $[\langle j \rangle]$ by

$$
\begin{aligned}
\text{dom}_\varphi^{\langle j \rangle, 1} \quad := \quad & \bigcup \{\langle e \rangle \mid e : \langle j \rangle \in T^{\text{ground}}\} \\
& \cup \bigcup \{\pi_\ell(R) \mid R : [\tau_1, \ldots, \tau_k] \in \mathcal{M}, \ell \in \{1, \ldots, k\} \text{ and } \tau_\ell = \langle j \rangle\}.
\end{aligned}
$$

For $i > 1$, define the meta algebra expression $\text{dom}_\varphi^{0, i+1}$ of type $[0]$ by

$$\text{dom}_\varphi^{0, i+1} := \bigcup \{\pi_{k+1} \texttt{eval}_1(\text{dom}_\varphi^{\langle \ell \rangle, i}) \mid \ell \in \{1, \ldots, p\} \text{ and } k \in \{1, \ldots, \ell\}\};$$

define the meta algebra expression $\text{dom}_\varphi^{\langle 1 \rangle, i+1}$ of type $[\langle 1 \rangle]$ by

$$
\begin{aligned}
\text{dom}_\varphi^{\langle 1 \rangle, i+1} := \quad & \text{dom}_\varphi^{\langle 1 \rangle, i} \\
& \cup \bigcup_{\ell=1}^{p} \pi_2 \texttt{extract}_{1:1}(\text{dom}_\varphi^{\langle \ell \rangle, i}) \\
& \cup \bigcup_{\ell=1}^{p} \pi_4 \texttt{rewrite-one}_{1:\square_2 \to \square_3}(\text{dom}_\varphi^{\langle 1 \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i}) \\
& \cup \bigcup_{\ell=1}^{p} \pi_4 \texttt{rewrite-all}_{1:\square_2 \to \square_3}(\text{dom}_\varphi^{\langle 1 \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i}) \\
& \cup \pi_2 \texttt{wrap}_1(\text{dom}_\varphi^{0, i}) \\
& \cup U^{\langle 1 \rangle, i},
\end{aligned}
$$

where $U^{\langle 1 \rangle, i}$ is the union of the expressions $\langle t(e_1, \ldots, e_n) \rangle$ such that $t(x_1, \ldots, x_n) : 1 \in T$, $n \ge 1$, and for each $\ell = 1, \ldots, n$, x_ℓ is an expression variable of sort $\langle j_\ell \rangle$ and $e_\ell \in \text{dom}_\varphi^{\langle j_\ell \rangle, i}$. Finally, for $i > 1$ and $j = 2, \ldots, p$, define the meta algebra expression $\text{dom}_\varphi^{\langle j \rangle, i+1}$ of type $[\langle j \rangle]$ by

$$
\begin{aligned}
\text{dom}_\varphi^{\langle j \rangle, i+1} := \quad & \text{dom}_\varphi^{\langle j \rangle, i} \\
& \cup \bigcup_{\ell=1}^{p} \pi_2 \texttt{extract}_{1:j}(\text{dom}_\varphi^{\langle \ell \rangle, i}) \\
& \cup \bigcup_{\ell=1}^{p} \pi_4 \texttt{rewrite-one}_{1:\square_2 \to \square_3}(\text{dom}_\varphi^{\langle j \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i}) \\
& \cup \bigcup_{\ell=1}^{p} \pi_4 \texttt{rewrite-all}_{1:\square_2 \to \square_3}(\text{dom}_\varphi^{\langle j \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i} \times \text{dom}_\varphi^{\langle \ell \rangle, i}) \\
& \cup U^{\langle j \rangle, i},
\end{aligned}
$$

where $U^{\langle j \rangle, i}$ is the union of the expressions $\langle t(e_1, \ldots, e_n) \rangle$ such that $t(x_1, \ldots, x_n) : j \in T$, $n \geq 1$, and for each $\ell = 1, \ldots, n$, x_ℓ is an expression variable of sort $\langle j_\ell \rangle$ and $e_\ell \in \mathrm{dom}_\varphi^{\langle j_\ell \rangle, i}$.

Let m be the number of occurrences of variables in φ. Then define for $j = 1, \ldots, p$: $\mathrm{dom}_\varphi^{\langle j \rangle} := \mathrm{dom}_\varphi^{\langle j \rangle, m}$. If $\tau = [\tau_1, \ldots, \tau_k]$ then define $\mathrm{dom}_\varphi^\tau$ as the meta algebra expression $\mathrm{dom}_\varphi^{\tau_1} \times \cdots \times \mathrm{dom}_\varphi^{\tau_k}$.

A proof of the following lemma can be found in Appendix A.

Lemma 1 *For any safe $\mathcal{MC}$-formula $\varphi(x_1, \ldots, x_k)$, where each x_i is of type τ_i, and for any valuation ρ, if $\mathcal{K} \models \varphi[\rho]$ then $\rho(x_i) \in [\![\mathrm{dom}_\varphi^{\tau_i}]\!]^{\mathcal{K}}$ for all $i \in \{1, \ldots, k\}$.*

For each safe meta calculus formula $\varphi(\bar{x})$, we construct a meta algebra expression e_φ such that for every instance $\mathcal{K}$ and for every valuation ρ: $\mathcal{K} \models \varphi[\rho] \Leftrightarrow \rho(\bar{x}) \in [\![e_\varphi]\!]^{\mathcal{K}}$. W.l.o.g., we assume only terms that are variables appear in atomic formulas that are not equalities and that all variables appearing in the same occurrence of an atomic formula are different. Furthermore, we assume that all equalities are of the form $x = t$ where x is a variable and t is a term. In the following, every variable x_j is of sort $\langle i_j \rangle$.

We transform all subformulas σ by the following inductive process:

1. If $\sigma = R(\bar{x})$, where $R \in \mathcal{S} \cup \mathcal{M}$, then $e_\sigma := R$.

2. If $\sigma = x_0 = t(x_1, \ldots, x_n)$ then

$$e_\sigma := \pi_{1, \ldots, n+1} \sigma_{1=n+2} \mathbf{construct}_{t'}(\mathrm{dom}_\varphi^{[\langle i_0 \rangle, \ldots, \langle i_n \rangle]}),$$

   where t' is the pattern obtained from t by replacing each x_j by $\square_{j+1}$.

3. If $\sigma = x_1 \leq x_2$ then $e_\sigma := \pi_{2,1} \mathbf{extract}_{1:i_1}(\mathrm{dom}_\varphi^{[\langle i_2 \rangle]})$.

4. If $\sigma = \mathbf{rewrite\text{-}one}(x_1, x_2, x_3, x_4)$ then

$$e_\sigma := \mathbf{rewrite\text{-}one}_{1:\square_2 \to \square_3}(\mathrm{dom}_\varphi^{[\langle i_1 \rangle, \langle i_2 \rangle, \langle i_3 \rangle, \langle i_4 \rangle]}).$$

5. If $\sigma = \mathbf{rewrite\text{-}all}(x_1, x_2, x_3, x_4)$ then

$$e_\sigma := \mathbf{rewrite\text{-}all}_{1:\square_2 \to \square_3}(\mathrm{dom}_\varphi^{[\langle i_1 \rangle, \langle i_2 \rangle, \langle i_3 \rangle, \langle i_4 \rangle]}).$$

6. If $\sigma = \mathbf{eval}(x_0, y_1, \ldots, y_n)$ then $e_\sigma := \mathbf{eval}_1(\mathrm{dom}_\varphi^{[\langle i_0 \rangle]})$.

7. If $\sigma = \neg \sigma'(x_1, \ldots, x_n)$ then $e_\sigma := \mathrm{dom}_\varphi^{[\langle i_1 \rangle, \ldots, \langle i_n \rangle]} - e_{\sigma'}$.

8. If $\sigma = \sigma_1 \vee \sigma_2$ then $e_\sigma := e_{\sigma_1} \cup e_{\sigma_2}$. $\square$

Remark 1 It follows from the above proof that any meta algebra expression is equivalent to one where every rewrite operator uses patterns that consists of only boxes; i.e., every rewrite operator is of the form $\mathbf{rewrite\text{-}one}_{i:\square_j \to \square_k}$ or $\mathbf{rewrite\text{-}all}_{i:\square_j \to \square_k}$.

5.2. Limitations of the Typed Approach

The meta algebra and the meta calculus are strictly typed formalisms. It is impossible to define relations with columns containing expressions of different arities. However, we can give an example of a natural and simple query that seems to have the property that computing it really requires such untyped intermediate results:

Theorem 2 *Let $R : [\langle 1 \rangle] \in \mathcal{M}$. Let q be the query of type $[\langle 1 \rangle]$ defined as follows: given an instance $\mathcal{K}$, $q(\mathcal{K})$ is the set of expressions in $\mathcal{K}(R)$ that are of the form $\pi_1(\ldots)$. This query is not definable in the meta algebra.*

Proof. The equivalence of the meta algebra with the meta calculus allows an elegant model-theoretic proof of this theorem. The $\mathcal{A}$-expressions over a schema $\mathcal{S}$ form a structure (in the sense of mathematical logic [10]) consisting of the relation names in $\mathcal{S}$ as constants, the operators as functions, and the relations $\leq$ (subexpression), `rewrite-one`, and `rewrite-all`. This structure is many-sorted: for example, we do not have one single function $\times$ but rather have a separate one $\times_{n,m}$ of sort $(\langle n \rangle, \langle m \rangle) \to \langle n + m \rangle$ for all arities n and m.

Now suppose, for the sake of contradiction, that there is an $\mathcal{MC}$-formula φ defining the query q from the theorem. Since the query is independent of the object-level instance we can as well assume that all object-level relations are empty. Hence, we may assume without loss of generality that φ neither uses data variables, object-level relation names, nor `eval`.

So φ is essentially a first-order logic formula, evaluated over the above-described structure of $\mathcal{A}$-expressions, call it $\mathcal{E}$, expanded with a relation R of sort $\langle 1 \rangle$. Let n be strictly larger than the arity of any term occurring in φ. Then φ looks only at $\mathcal{E}|_{<n}$, the restriction of $\mathcal{E}$ to sorts $\langle m \rangle$ with $m < n$.

Define the following function f on $\mathcal{A}$-expressions e: $f(e)$ is obtained from e by replacing each occurrence of a subexpression of the form $\pi_1(e')$, where e' is n-ary, by $\pi_2(e')$, and conversely, replacing each occurrence of a subexpression of the form $\pi_2(e')$, where e' is n-ary, by $\pi_1(e')$. This function is an automorphism of $\mathcal{E}|_{<n}$. It maps $\pi_1(S^n)$ to $\pi_2(S^n)$ and back, where S^n stands for $S \times \cdots \times S$ (n times).

Hence, on an instance in which R consists of the two expressions $\pi_1(S^n)$ and $\pi_2(S^n)$, the query defined by φ will either contain both expressions in the result, or none of them, since first-order logic formulas cannot distinguish between automorphic elements. This yields the desired contradiction, since $\pi_2(S^n)$ is not of the form $\pi_1(\ldots)$. □

Theorem 2 offers the most challenging direction for further research. How can our formalism (in particular its type system) be generalized so that queries of the kind mentioned in the theorem become expressible, *at the same time* not giving up on type-safety of `eval`?

Note that Theorem 2 may be compared to a similar situation in the design of computationally complete query languages. The language QL, proposed and studied by Chandra and Harel [6], is an adaptation of the relational algebra designed to work with "untyped" relations of variable width, to which a while-loop construct is added. QL is computationally complete. If, however, the ordinary "typed" relational algebra is extended with while-loops, one gets a language whose expressiveness remains within PSPACE [7, 3].

Another situation to which Theorem 2 may be compared to is that of the lambda calculus. Functions on the natural numbers, encoded as functions on Church numerals, are typed. But again the computation of many such functions requires intermediate results that are untyped: in the untyped lambda calculus all partial recursive functions are definable, while in the simply-typed lambda calculus only a restricted class of functions, the so-called extended polynomials, are definable [4, 19].

5.3. Non-Redundancy

A natural question to ask is whether the meta algebra is non-redundant, i.e., whether each operator provided in the meta algebra is primitive (not expressible using the other operators).

As a matter of fact, we obtain the following theorem.

Theorem 3 *The meta algebra is not redundant.*

Regarding primitivity of the five relational algebra operators, it is easily seen and well known that each of them is primitive within the standard relational algebra. Of course this does not automatically imply primitivity within the meta algebra. The latter follows nevertheless because the meta algebra admits a *conservative extension property* which we prove in the next section. In Appendix B we give detailed, but technical, proofs of the primitiveness of `wrap`, `extract`, `rewrite-one` and `rewrite-all`.

We now show the primitivity of `eval`. The equivalence with the meta calculus allows for an elegant model-theoretic proof. We make use of the structure $\mathcal{E}$ of $\mathcal{A}$-expressions introduced in

Section 5.2. Let $\mathcal{S} = \{E : 2, U : 1\}$ and $\mathcal{M} = \{Q : [\langle 0 \rangle]\}$. Define the instance $\mathcal{I}$ of $\mathcal{S}$ by $\mathcal{I}(E) = \{(1, 2)\}$ and $\mathcal{I}(U) = \{(1)\}$. For any natural number m, define the instance $\mathcal{J}_1^m$ over $\mathcal{M}$ by $\mathcal{J}_1^m(Q) = \{\pi_\emptyset(\pi_1(E^m) \cap U)\}$ and define $\mathcal{J}_2^m$ by $\mathcal{J}_2^m(Q) = \{\pi_\emptyset(\pi_2(E^m) \cap U)\}$. For any natural number m and $i \in \{1, 2\}$ define the combined instance $\mathcal{K}_i^m = (\mathcal{I}, \mathcal{J}_i^m)$. Then, for all m, $[\![\texttt{eval}_1(Q)]\!]^{\mathcal{K}_1^m} \neq [\![\texttt{eval}_1(Q)]\!]^{\mathcal{K}_2^m}$.

We now proceed like in Section 5.2. Suppose, for the sake of contradiction, that there is an $\mathcal{MA}$-expression φ without $\texttt{eval}$ that expresses the query $\texttt{eval}(Q)$. By Theorem 1 there exists an equivalent $\mathcal{MC}$-sentence φ. Moreover, φ does not contain $\texttt{eval}$. So φ is a first-order logic formula over $\mathcal{E}$ expanded with the relations E, U and Q. Let n be strictly larger than the arity of any term occurring in φ. Then φ looks only at $\mathcal{E}|_{<n}$, the restriction of $\mathcal{E}$ to sorts $\langle m \rangle$ with $m < n$.

Define the following function f on $\mathcal{A}$-expressions e: $f(e)$ is obtained from e by replacing each occurrence of a subexpression of the form $\pi_1(e')$, where e' is n-ary, by $\pi_2(e')$, and conversely, replacing each occurrence of a subexpression of the form $\pi_2(e')$, where e' is n-ary, by $\pi_1(e')$. This function is an automorphism of $\mathcal{E}|_{<n}$. It maps $\pi_\emptyset(\pi_1(E^n) \cap U)$ to $\pi_\emptyset(\pi_2(E^n) \cap U)$ and back.

The structures $(\mathcal{E}|_{<n}, \mathcal{K}_1^n)$ and $(\mathcal{E}|_{<n}, \mathcal{K}_2^n)$ are isomorphic via f', where f' is the identity on $\mathcal{I}$ and is defined as f on $\mathcal{A}$-expressions. Since first-order logic cannot distinguish between isomorphic structures, we have that φ holds in the instance $\mathcal{K}_1^m$ if and only if φ holds in the instance $\mathcal{K}_2^m$. This yields the desired contradiction.

5.4. Conservative Extension

In this section, we prove that the meta algebra provides no power above that of the relational algebra if only classical queries not involving meta-level relations are under consideration. More concretely, we show:

Theorem 4 *Let S be a schema and let q be a query over $(S, \emptyset)$ of type $[0, \ldots, 0]$ (n zeros). If q is definable in the meta algebra then q is already definable in the relational algebra.*

We prove this theorem in two lemmas. To state the first lemma, we introduce the following notions. Let $\mathcal{D}$ be a schema consisting of unary relation names only. We call such a schema a "dummy schema". Dummy relation names will serve as representatives of relational algebra expressions of the form $\{(v)\}$ with $v \in \mathbf{V}$. Let $\nu = [\nu_1, \ldots, \nu_m]$ be a type. A *column assignment of $\mathcal{D}$ in ν* is a function $\gamma : \mathcal{D} \to \{1, \ldots, m\}$ such that for every $D \in \mathcal{D}$, $\gamma(D)$ is a data column. A *column sequence of ν* is a sequence $i_1, \ldots, i_n$, with $i_1, \ldots, i_n \in \{1, \ldots, m\}$. We denote the type $[\nu_{i_1}, \ldots, \nu_{i_n}]$ by $\pi_{i_1, \ldots, i_n}(\nu)$. Let γ be a column assignment of $\mathcal{D}$ in γ, and let δ be a column sequence of ν. For any meta-relation R of type ν containing $\mathcal{A}$-expressions over the schema $\mathcal{S} \cup \mathcal{D}$, define the following relation of type $\pi_\delta(\nu)$:

$$\Delta^{\gamma, \delta}(R) := \pi_\delta(R'),$$

where

$$R' := \{(x_1', \ldots, x_m') \mid (x_1, \ldots, x_m) \in R \text{ and for each } i = 1, \ldots, m,$$
$$x_i' \text{ is obtained from } x_i \text{ by replacing each occurrence of a name } D \in \mathcal{D}$$
$$\text{by the expression } \{(x_{\gamma(D)})\}\}$$

If $\mathcal{I}$ is an instance of $\mathcal{S}$ then we extend $\mathcal{I}$ to an instance $\mathcal{I}_\mathcal{D}$ of $\mathcal{S} \cup \mathcal{D}$ by setting $\mathcal{I}_\mathcal{D}(D) := \emptyset$ for each $D \in \mathcal{D}$. We denote the relational algebra extended with the $\langle \cdot \rangle$ operator by $\mathcal{A}^{\langle \cdot \rangle \dagger}$.

Lemma 2 *Let $e : \tau$, with $\tau = [\tau_1, \ldots, \tau_n]$, be a meta algebra expression over $(\mathcal{S}, \emptyset)$. There exists*

- *a dummy schema $\mathcal{D}_e$;*

- *a natural number m_e;*

---

†The operator $\langle \cdot \rangle$ is defined in Section 3.4.1.

- an $\mathcal{A}^{\langle\cdot\rangle}$-expression $\omega_e : \nu$, with $\nu = [\nu_1, \ldots, \nu_{m_e}]$, over the combined schema $(S \cup \mathcal{D}_e, \emptyset)$;

- a column assignment γ_e of $\mathcal{D}_e$ in ν;

- a column sequence δ_e of ν such that $\pi_{\delta_e}(\nu) = \tau$;

- a finite set A_e of $\mathcal{A}$-expressions over $S \cup \mathcal{D}_e$;

such that for any instance $\mathcal{I}$ of S

- $[\![e]\!]^{\mathcal{I}} = \Delta^{\gamma_e, \delta_e}([\![\omega_e]\!]^{\mathcal{I}\nu})$; and

- for every expression column j, $[\![\pi_j(\omega_e)]\!]^{\mathcal{I}_D} \subseteq A_e$.

Proof. The proof proceeds by induction on the structure of e. Suppose e is an $\mathcal{M}\mathcal{A}$-expression as in the lemma, for which we assume the lemma already holds. Let V_e be the set of all data values that appear as constant expressions in expressions in A_e, let E be an equivalence relation on $\mathcal{D}_e$, and let η be a partial, injective function from E-equivalence classes into V_e. We call (E, η) an *instantiation of* e. For any $D \in \mathcal{D}_e$, if the value of η on D's equivalence class in E equals $v \in V_e$, then we also write $\eta(D) = v$. If η is undefined on D's equivalence class, then we also write $\eta(D) = \perp$. Intuitively, if $\eta(D) = v$, then D is a representative of the constant expression $\{(v)\}$, and if $\eta(D) = \perp$, then D is a representative of a constant expression $\{(v')\}$ for some data value v' appearing in the database but not in V_e. The equivalence relation E then specifies which dummy relation names represent the same constant expression. We now define $=_{E,\eta}$ as the smallest equivalence relation on $\mathcal{A}$-expressions over $S \cup \mathcal{D}_e$ such that:

- For $D, D' \in \mathcal{D}_e$, if $(D, D') \in E$, then $D =_{E,\eta} D'$.

- For $D \in \mathcal{D}_e$ and $v \in \mathbf{V}$, if $\eta(D) = v$, then $D =_{E,\eta} \{(v)\}$.

- For expressions e_1 and e_2 and any unary $\mathcal{A}$-operator op, if $e_1 =_{E,\eta} e_2$, then $\mathrm{op}(e_1) =_{E,\eta} \mathrm{op}(e_2)$.

- For expressions e_1, e_2, e_3, e_4 and any binary $\mathcal{A}$-operator op, if $e_1 =_{E,\eta} e_3$ and $e_2 =_{E,\eta} e_4$, then $e_1 \text{ op } e_2 =_{E,\eta} e_3 \text{ op } e_4$.

We then introduce the following notations:

- The sequence consisting of all selection operators in

$$\{\sigma_{\gamma_e(D)=\gamma_e(D')} \mid (D, D') \in E\}$$

  or

$$\{\sigma_{\gamma_e(D)\neq\gamma_e(D')} \mid (D, D') \in (\mathcal{D}_e \times \mathcal{D}_e) - E\}$$

  is denoted by σ_E^e.

- For any $D \in \mathcal{D}_e$ such that $\eta(D) = \perp$, we denote the sequence consisting of all selection operators in

$$\{\sigma_{\gamma_e(D)\neq v} \mid v \in V_e\}$$

  by $\sigma_{\eta,D}^e$. If $\eta(D) \neq \perp$, we define $\sigma_{\eta,D}^e$ as $\sigma_{\gamma(e)(D)=\eta(D)}$.

- The sequence consisting of all $\sigma_{\eta,D}^e$ with $D \in \mathcal{D}_e$ is denoted by σ_η^e.

We can now start with the actual proof. In the following, we denote the jth element of a column sequence δ by $\delta(j)$. The cases where $e = \{(v)\}$ or $e = R$, with $R \in S$, are trivial.

- $e = \mathtt{wrap}_i(e')$: Define $\omega_e := \omega_{e'} \times \langle D' \rangle$, where $D' \notin D_{e'}$; $m_e := m_{e'} + 1$; $D_e := D_{e'} \cup \{D'\}$; for each $j = 1, \ldots, n$ define

$$\delta_e(j) := \begin{cases} \delta_{e'}(j) & \text{if } 1 \le j < n, \\ m_e' + 1 & \text{if } j = n; \end{cases}$$

for each $D \in D_e$ define

$$\gamma_e(D) := \begin{cases} \delta_{e'}(i) & \text{if } D = D', \\ \gamma_{e'}(D) & \text{otherwise}; \end{cases}$$

and define $A_e := A_{e'} \cup \{D'\}$.

- $e = \pi_{i_1, \ldots, i_p}(e')$: $\omega_e := \omega_{e'}$; $m_e := p$; $D_e := D_{e'}$; for $j = 1, \ldots, p$, define $\delta_e(j) := \delta_{e'}(i_j)$; $\gamma_e := \gamma_{e'}$; and define $A_e := A_{e'}$.

- $e = e_1 \cup e_2$: We assume the following:

  - $\delta_{e_1} = \delta_{e_2}$ (if not, we rearrange columns in $\omega(e_2)$);

  - $m_{e_1} = m_{e_2}$ (suppose $m_{e_1} < m_{e_2}$, then take a new dummy relation name D and redefine ω_{e_1} as $\omega_{e_1} \times \langle D \rangle \times \ldots \times \langle D \rangle$ ($m_{e_2} - m_{e_1}$ times)); and

  - $D_{e_1} \cap D_{e_2} = \emptyset$.

Define $\omega_e := \omega_{e_1} \cup \omega_{e_2}$; $m_e := m_{e_1}$; $D_e := D_{e_1} \cup D_{e_2}$; $\delta_e := \delta_{e_1}$; $\gamma_e := \gamma_{e_1} \cup \gamma_{e_2}$; and $A_e := A_{e_1} \cup A_{e_2}$.

- $e = e_1 \times e_2$: We assume $D_{e_1} \cap D_{e_2} = \emptyset$. Let e_1 be of arity n_1 and e_2 of arity n_2. Define $\omega_e := \omega_{e_1} \times \omega_{e_2}$; $m_e := m_{e_1} + m_{e_2}$; $D_e := D_{e_1} \cup D_{e_2}$; for $i = 1, \ldots, n_1 + n_2$, define

$$\delta_e(i) := \begin{cases} \delta_{e_1}(i) & \text{if } 1 \le i \le n_1, \\ \delta_{e_2}(i) + m_{e_1} & \text{if } n_1 + 1 \le i \le n_1 + n_2; \end{cases}$$

for each $D \in D_e$ define

$$\gamma_e(D) := \begin{cases} \gamma_{e_1}(D) & \text{if } D \in D_{e_1}, \\ \gamma_{e_2}(D) + m_{e_1} & \text{if } D \in D_{e_2}; \end{cases}$$

and define $A_e := A_{e_1} \cup A_{e_2}$.

- $e = \sigma_{i=j}(e')$: If i and j are data columns this is easy. So suppose i and j are expression columns with $\tau_i = \tau_j = \langle \ell \rangle$, define ω_e as

$$\bigcup \{ \sigma_\eta^{e'} \sigma_E^{e'} \sigma_{\delta_{e'}(i)=t_1} \sigma_{\delta_{e'}(j)=t_2}(\omega_{e'}) \mid$$
$$E, \eta \text{ an instantiation of } e', t_1 : \ell \in A_{e'}, t_2 : \ell \in A_{e'}, \text{ and } t_1 =_{E,\eta} t_2 \}.$$

Define $m_e := m_{e'}$; $D_e := D_{e'}$; $\delta_e := \delta_{e'}$; $\gamma_e := \gamma_{e'}$; and define $A_e := A_{e'}$.

- $e = e_1 - e_2$: As in the case $e = e_1 \cup e_2$, we can assume that $\delta_{e_1} = \delta_{e_2}$, $m_{e_1} = m_{e_2}$ and $D_{e_1} \cap D_{e_2} = \emptyset$. Let I denote the set of data columns in τ. Then define ω_e as

$$\omega_{e_1} - \pi_{1, \ldots, m_{e_1}} \left(\sigma_{\bigwedge_{i \in I} \delta_{e_1}(i) = m_{e_1} + \delta_{e_2}(i)}(\omega_{e_1} \times \omega_{e_2}) \cap \bigcap_{i \notin I} \omega_{\sigma_{\delta_{e_1}(i) = m_{e_1} + \delta_{e_2}(i)}(e_1 \times e_2)} \right).$$

Define $m_e := m_{e_1}$; $D_e := D_{e_1}$; $\delta_e := \delta_{e_1}$; $\gamma_e := \gamma_{e_1}$; and $A_e := A_{e_1}$.

- $e = \mathtt{rewrite\text{-}one}_{i:\alpha \to \beta}(e')$: By Remark 1 we can assume that $\alpha = \square_j$ and $\beta = \square_k$. Let $\tau_i = \langle \ell_1 \rangle$, $\tau_j = \langle \ell_2 \rangle$, and $\tau_k = \langle \ell_3 \rangle$. Define ω_e as

$$\bigcup \{ \sigma_\eta^{e'} \sigma_E^{e'} \sigma_{\delta_{e'}(i)=t_1} \sigma_{\delta_{e'}(j)=t_2} \sigma_{\delta_{e'}(k)=t_3}(\omega_{e'}) \times T_{E,\eta}^{t_1, t_2, t_3} \mid t_1 : \ell_1 \in A_{e'},$$
$$t_2 : \ell_2 \in A_{e'}, t_3 : \ell_3 \in A_{e'}, \text{ and } E, \eta \text{ is an instantiation of } e' \},$$

where

$$T_{E,\eta}^{t_1,t_2,t_3} = \bigcup \{ \langle t' \rangle \mid t' \text{ is obtained by replacing in } t_1$$

$$\text{a subexpression } t \text{ by } t_3 \text{ for some } t =_{E,\eta} t_2 \}.$$

Define $m_e := m_{e'} + 1$; $D_e := D_{e'}$; for $k := 1, \ldots, n$, define

$$\delta_e(k) := \begin{cases} \delta_{e'}(k) & \text{if } 1 \leq k < n, \\ m_{e'} + 1 & \text{if } k = n; \end{cases}$$

define $\gamma_e := \gamma_{e'}$; and define A_e as the union of A_e' with all the $T_{E,\eta}^{t_1,t_2,t_3}$s that appear in ω_e.

- $e = \texttt{rewrite-all}$: similar to the previous case.

- $e = \texttt{extract}_{i:m}(e')$: Let $\tau_i = \langle \ell \rangle$. Define ω_e as

$$\bigcup \{ \sigma_{\delta_{e'}(i)=t}(\omega_{e'}) \times T^t \mid t : \ell \in A_{e'} \},$$

where $T^t = \{ \langle t' \rangle \mid t' : m \text{ is a subexpression of } t \}$.

Define $m_e := m_{e'} + 1$; $D_e := D_{e'}$; for $k := 1, \ldots, n$, define

$$\delta_e(k) := \begin{cases} \delta_{e'}(k) & \text{if } 1 \leq k < n, \\ m(e') + 1 & \text{if } k = n; \end{cases}$$

define $\gamma_e := \gamma_{e'}$; and define A_e as the union of $A_{e'}$ with all the T^ts that appear in ω_e.

- $e = \texttt{eval}_i(e')$: Let $\tau_i = \langle \ell \rangle$, and let the arity of e' be n'; so $n = n' + \ell$. Define $D_e := D_{e'}$; $m_e := m_{e'} + \ell$; for $k := 1, \ldots, n'$, define $\delta_e(k) := \delta_{e'}(k)$, and for $k := 1, \ldots, \ell$, define $\delta_e(n' + k) := m_{e'} + k$. Define $\gamma_e := \gamma_{e'}$ and $A_e := A_{e'}$. Let $t : \ell$ be a relational algebra expression over $\mathcal{S} \cup D_e$. We construct an $\mathcal{A}^{\langle \cdot \rangle}$-expression ω_e^t such that for any $\mathcal{I}$:

$$\Delta^{\gamma_e,\delta_e}(\llbracket \omega_e^t \rrbracket^{\mathcal{I}_D}) \quad = \quad \llbracket \texttt{eval}_i \Delta^{\gamma_e,\delta_e}(\llbracket \sigma_{\delta_{e'}(i)=t}(\omega_{e'}) \rrbracket^{\mathcal{I}_D}) \rrbracket^{\mathcal{I}}.$$

The construction goes by induction on the structure of t.

- $t = (v)$ or $t \in \mathcal{S}$: define ω_e^t as $\sigma_{\delta_{e'}(i)=t}(\omega_{e'} \times t)$;

- $t = D$: define ω_e^t as $\pi_{1,\ldots,m_{e'},\gamma_{e'}}(D) \sigma_{\delta_{e'}(i)=t} \, \omega_{e'}$;

- $t = \pi_{i_1,\ldots,i_p}(t')$: define ω_e^t as $\pi_{1,\ldots,m_{e'},i_1+m_{e'},\ldots,i_p+m_{e'}} \omega_e^{t'}$;

- $t = \sigma_{j=j'}(t')$: define ω_e^t as $\sigma_{j+m_{e'}=j'+m_{e'}} \omega_e^{t'}$;

- $t = t_1 \times t_2$. Let the arity of t_1 be m_1 and let the arity of t_2 be m_2. Define ω_e^t as

$$\pi_{1,\ldots,m_{e'}+m_1,2\cdot m_{e'}+m_1+1,\ldots,2\cdot m_{e'}+m_1+m_2} \bigcap_{i=1}^{m_{e'}} \sigma_{i=i+m_{e'}+m_1}(\omega_e^{t_1} \times \omega_e^{t_2}).$$

- $t = t_1 - t_2$. Let the arity of t be m. Define ω_e^t as

$$\omega_e^{t_1} - \pi_{1,\ldots,m_{e'}+m} \bigcap_{i=1}^{m_{e'}} \sigma_{i=i+m_{e'}+m}(\omega_e^{t_1} \times \omega_e^{t_2}).$$

Now, define ω_e as $\bigcup \{ \omega_e^t \mid t : \ell \in A_{e'} \}$. $\square$

We prove a normal form for $\mathcal{A}^{\langle \cdot \rangle}$.

Lemma 3 *Assume every meta-level relation is of a type having only expression columns. Then every $\mathcal{A}^{(\cdot)}$-expression e is equivalent up to reordering of columns[†], to a union of "base expressions", of the form $\ell_e \times r_e$, where ℓ_e is a relational algebra expression and r_e is of the form $\langle t_1 \rangle \times \ldots \times \langle t_n \rangle$.*

Proof. The proof proceeds by induction on the structure of e. The lemma trivially holds for the base case where $e = S$ with $S \in \mathcal{S} \cup \mathcal{M}$, $e = \{(v)\}$ with $v \in \mathbf{V}$, or $e = \langle e' \rangle$. In the following, $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}'$ are sets of base expressions.

- If $e = \bigcup \mathcal{E} \cup \bigcup \mathcal{E}'$ then e is clearly in the right form.

- If $e = \bigcup \mathcal{E} \times \bigcup \mathcal{E}'$ then e is equivalent to

$$\bigcup \{\ell_{e_1} \times \ell_{e_2} \times r_{e_1} \times r_{e_2} \mid e_1 \in \mathcal{E}_1, e_2 \in \mathcal{E}_2\}.$$

- If $e = \pi_{i_1,\ldots,i_p} \bigcup \mathcal{E}'$ then e is equivalent to

$$\bigcup \{\pi_{S_1} \ell(e') \times \pi_{S_2} r(e') \mid e' \in \mathcal{E}'\},$$

  where

$$S_1 = \{i_j \mid j \in \{1,\ldots,p\} \text{ and } i_j \text{ is a data column}\}$$

  and

$$S_2 = \{i_j \mid j \in \{1,\ldots,p\} \text{ and } i_j \text{ is an expression column}\}.$$

- If $e = \sigma_{i=j}(\bigcup \mathcal{E}')$ then e is equivalent to

$$\bigcup \{\sigma_{i=j}(\ell_{e'}) \times r_{e'} \mid e' \in \mathcal{E}'\}$$

  if i and j are data columns; otherwise, e is equivalent to

$$\bigcup \{\ell_{e'} \times \sigma_{i=j}(r_{e'}) \mid e' \in \mathcal{E}'\}.$$

- If $e = \bigcup \mathcal{E}_1 - \bigcup \mathcal{E}_2$ then it suffices to show that the difference of two sets of base expressions can be represented by a set of base expressions. Since $\bigcup \mathcal{E}_1 - \bigcup \mathcal{E}_2$ can be written as $\bigcup \{e_1 - \bigcup \mathcal{E}_2 \mid e_1 \in \mathcal{E}_1\}$, we only have to show that $e_1 - \bigcup \mathcal{E}_2$ is equivalent to a set of base expressions. We prove the latter by induction on the size of $\mathcal{E}_2$.

  - If $\mathcal{E}_2 = \{e_2\}$ then $e_1 - e_2$ is equivalent to $((\ell_{e_1} - \ell_{e_2}) \times r_{e_2}) \cup (\ell_{e_1} \times (r_{e_1} - r_{e_2}))$.

  - If $\mathcal{E}_2 = \{b_1,\ldots,b_n\}$ then $e_1 - \mathcal{E}_2$ can be written as $(e_1 - \{b_1,\ldots,b_{n-1}\}) - b_n$. By the inductive hypothesis, this is equivalent to $\{c_1,\ldots,c_m\} - b_n$ for some base expressions $c_1, \ldots, c_m$. This expression can then be rewritten as $(c_1 - b_n) \cup \ldots \cup (c_m - b_n)$; by the inductive hypothesis we get the result. □

Proof of Theorem 4. Let q be defined by the meta algebra expression e. By applying Lemma 2 and Lemma 3, e is equivalent to a union of base expressions e'. Now, since the output of e contains no expression columns, e' is equivalent to a relational algebra expression. □

[†]Note that reordering of columns is expressible using projection.

REFERENCES

[1] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, second edition (1996).

[2] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. *The VLDB Journal,* **4**(4):727–794, Originally INRIA Research Report 846, 1988 (1995).

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley (1995).

[4] H.P. Barendregt. *The Lambda Calculus.* Elsevier, North-Holland (1984).

[5] C. Beeri and T. Milo. On the power of algebras with recursion. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data,* Washington, D.C., *SIGMOD Record,* **22**(2):377–386, ACM Press (1993).

[6] A. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences,* **21**(2):156–178 (1980).

[7] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences,* **25**(1):99–128 (1982).

[8] W. Chen, M. Kifer, and D.S. Warren. HiLog: a foundation for higher-order logic programming. *Journal of Logic Programming,* **15**(3):187–230 (1993).

[9] E. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems,* pp. 65–98, Prentice-Hall (1972).

[10] H.B. Enderton. *A Mathematical Introduction to Logic.* Academic Press (1972).

[11] M. Gyssens, J. Paredaens, and D. Van Gucht. A grammar-based approach towards unifying hierarchical data models. *SIAM Journal on Computing,* **23**(6):1093–1137 (1994).

[12] J.-W. Klop. Term rewriting systems: a tutorial. *Bulletin of the EATCS,* **32**:143–183 (1987).

[13] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM,* **29**(3):699–717 (1982).

[14] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In S. Ceri, K. Tanaka, and S. Tsur, editors, *Deductive and Object-Oriented Databases,* volume 760 of *Lecture Notes in Computer Science,* pp. 81–100, Springer-Verlag (1993).

[15] G. Ozsoyoglu, Z.M. Ozsoyoglu, and V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Transactions on Database Systems,* **12**(4):566–592 (1987).

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

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

[18] K. Ross. Relations with relation names as arguments: algebra and calculus. In *Proceedings 11th ACM Symposium on Principles of Database Systems,* San Diego, California, pp. 346–353, ACM Press (1992).

[19] H. Schwichtenberg. Definierbare Funktionen im λ-Kalkül mit Typen. *Arch. Math. Logik Grundlagenforsch.,* **17**(3–4):113–114 (1975).

[20] T. Sheard and J. Hook. *Type Safe Meta-Programming.* Manuscript, Oregon Graduate Institute (1994).

[21] M. Stonebraker et al. QUEL as a data type. In B. Yormark, editor, *Proceedings of SIGMOD 84 Annual Meeting,* volume 14:2 of *SIGMOD Record,* Boston, Massachusetts, pp. 208–214, ACM Press (1984).

[22] M. Stonebraker et al. Extending a database system with procedures. *ACM Transactions on Database Systems,* **12**(3):350–376 (1987).

[23] D. Suciu. Bounded fixpoints for complex objects. *Theoretical Computer Science,* **176**(1–2):283–328 (1997).

[24] J. Ullman. *Principles of Database and Knowledge-Base Systems,* volume I. Computer Science Press (1988).

[25] J. Van den Bussche, D. Van Gucht, and G. Vossen. Reflective programming in the relational algebra. *Journal of Computer and System Sciences,* **52**(3):537–549 (1996).

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

APPENDIX A

Proof of Lemma 1. The proof proceeds by induction on the nesting depth of maximal conjunctions in φ. The base case of this induction is that of a conjunction consisting of atomic formulas only, which we can directly treat as a special case of the general case.

Since φ is safe, for each free variable x in each maximal conjunction σ of φ, there exists a proof for the limitedness of x. We next define the depth of x in σ with respect to this proof. If x is shown to be limited by an application of rule j then we just say that x is derived by rule j. We say that

- x is of depth 1 if x is derived by rule (1) or (6) and δ is an atomic formula, or x is derived by rule (2), or x is derived by rule (7) with $t_1 = x$ and t_2 a ground term;

- x is of depth $n + 1$

  - if x is derived by rule (1) or (6), δ is of the form $\delta_1 \vee \delta_2$, and $n + 1$ is the maximum of the depths of x in δ_1 and δ_2;

  - if x is derived by rule (1) or (6), δ is of the form $(\exists y)\delta'$, and $n + 1$ is the the depth of x in δ';

  - if x is derived by rule (3), (4) or (9) and y is of depth n;

  - if x is derived by rule (5) and the maximal depth of a variable in t is n;

  - if x is derived by rule (7) or (8) and the maximal depth of a variable in t_2 is n;

  - if x is derived by rule (10), where the maximal depth of a variable in t_1, t_2 or t_3 is n.

Clearly, the depth of every variable is bounded by the number of occurrences of variables in φ.

Let σ be a subformula of φ consisting of conjunctions that cannot be enlarged. W.l.o.g., we assume only terms that are variables appear in atomic formulas that are not equalities. Furthermore, we assume that all equalities are of the form $x = t$ where x is a variable and t is a term.

We now show that if $\mathcal{K} \models \sigma[\rho]$ then $\rho(x) \in [\![\mathrm{dom}_\varphi^{\tau,n}]\!]^{\mathcal{K}}$, for every variable x of sort τ and depth n. We do not have to discuss variables that are derived by rules (1) or (6) where δ is a disjunction or is of the form $(\exists y)\dots$: we then already may assume that $\rho(x) \in [\![\mathrm{dom}_\varphi^{\tau,n}]\!]^{\mathcal{K}}$.

1. If x is a data variable of depth 1, then $\rho(x)$ is a data value that occurs in the active domain, or $\rho(x)$ appears as a constant in φ. Hence, $\rho(x) \in [\![\mathrm{dom}_\varphi^{0,1}]\!]^{\mathcal{K}}$.

2. If x is an expression variable of sort $\langle \ell \rangle$ that is of depth 1, then $\rho(x)$ is an $\mathcal{A}$-expression that occurs in the active domain, or $\rho(x)$ occurs in φ. Hence, $\rho(x) \in [\![\mathrm{dom}_\varphi^{\langle \ell \rangle,1}]\!]^{\mathcal{K}}$.

Let n be a natural number greater than 1.

1. If x is a data variable of depth $n + 1$ then we consider the following cases:

   - If x is derived by rule (3), then $\rho(x) = \rho(y) \in [\![\mathrm{dom}_\varphi^{0,n}]\!]^{\mathcal{K}} \subseteq [\![\mathrm{dom}_\varphi^{0,n+1}]\!]^{\mathcal{K}}$.

   - If x is derived by rule (4), then $\{(\rho(x))\} = \rho(y)$ and $\rho(y) \in [\![\mathrm{dom}_\varphi^{\langle 1 \rangle,n}]\!]^{\mathcal{K}}$. Hence, $\rho(x) \in [\![\pi_2(\mathrm{eval}_1(\mathrm{dom}_\varphi^{\langle 1 \rangle,n}))]\!]^{\mathcal{K}} \subseteq [\![\mathrm{dom}_\varphi^{0,n+1}]\!]^{\mathcal{K}}$.

   - If x is derived by rule (5), then clearly $\rho(x) \in [\![\mathrm{dom}_\varphi^{0,n+1}]\!]^{\mathcal{K}}$.

2. If x is an expression variable of sort τ that is of depth $n + 1$, then we consider the following cases:

   - If x is derived by rule (7), then $\rho(x) = t_2(\rho(x_1), \dots, \rho(x_n))$, where for each $i = 1, \dots, n$, x_i is of sort τ_i and $\rho(x_i) \in [\![\mathrm{dom}_\varphi^{\tau_i,n}]\!]^{\mathcal{K}}$. By definition of $U^{\tau,n}$, $\rho(x) \in [\![U^{\tau,n}]\!]^{\mathcal{K}} \subseteq [\![\mathrm{dom}_\varphi^{\tau,n+1}]\!]^{\mathcal{K}}$.

- If x is derived by rule (8), then $x \leq y$ appears as a conjunct in σ, $\rho(x)$ is a subexpression of $\rho(y)$, and $\rho(y) \in [\![\mathrm{dom}_\varphi^{\tau',n}]\!]$, where y is of sort τ'. By definition of $\mathrm{dom}_\varphi^{\tau,n+1}$, $\rho(x) \in [\![\mathrm{dom}_\varphi^{\tau,n+1}]\!]^{\mathcal{K}}$.

- If x is derived by rule (9), then $\rho(x) = (\rho(y))$ and $\rho(y) \in [\![\mathrm{dom}_\varphi^{0,n}]\!]^{\mathcal{K}}$. Hence, $\rho(x) \in [\![\pi_2 \mathrm{wrap}_1(\mathrm{dom}_\varphi^{0,n})]\!]^{\mathcal{K}} \subseteq [\![\mathrm{dom}_\varphi^{\tau,n+1}]\!]^{\mathcal{K}}$.

- If x is derived by rule (10), then $\mathrm{rewrite\text{-}one}(y_1, y_2, y_3, x)$ or $\mathrm{rewrite\text{-}all}(y_1, y_2, y_3, x)$ appears as an atomic formula in σ. By definition of $\mathrm{dom}_\varphi^{\tau,n+1}$, $\rho(x) \in [\![\mathrm{dom}_\varphi^{\tau,n+1}]\!]$. □

APPENDIX B

We now show the primitiveness of the `wrap`, `extract`, `rewrite-one` and `rewrite-all` operators. In the following, if op denotes an operator of $\mathcal{MA}$, then $\mathcal{MA} - \{\mathrm{op}\}$ denotes the fragment of $\mathcal{MA}$ without the operator op.

B.1 Wrap

We start with `wrap`:

Lemma B.1 *Let $e : \tau$ be an expression of $\mathcal{MA} - \{\mathrm{wrap}\}$ over an empty meta-level schema, let $\tau = [\tau_1, \ldots, \tau_n]$ be a type, and let $C = \{1, \ldots, n\}$ be the set of expression columns of τ. Then there exists a finite set A_e of $\mathcal{A}$-expressions such that for every instance $\mathcal{K}$ and $j \in C$, $[\![\pi_j(e)]\!]^{\mathcal{K}} \subseteq A_e$.*

Proof. The proof proceeds by induction on the structure of e. If $e = e_1 \mathrm{\ op\ } e_2$, where op is $\cup$ or $\times$, then $A_e := A_{e_1} \cup A_{e_2}$. If $e = e_1 - e_2$ then $A_e := A_{e_1}$. If $e = \mathrm{op}(e_1)$, where op is σ or π then $A_e := A_{e_1}$. If $e = \mathrm{extract}_{i:m}(e_1)$ then

$$A_e := A_{e_1} \cup \{s \mid s : m \text{ is a subexpression of } s' \text{ and } s' \in A_{e_1}\}.$$

If $e = \mathrm{rewrite\text{-}one}_{i:\alpha \rightarrow \beta}(e_1)$ then, by Remark 1, we can assume that $\alpha = \square_j$ and $\beta = \square_k$. Define

$$A_e := A_{e_1} \cup \{s_4 \mid s_1 : \ell_1 \in A_{e_1}, s_2 : \ell_2 \in A_{e_1}, s_3 : \ell_2 \in A_{e_1}, s_4 : \ell_1 \in A_{e_1}, \text{ and}$$
$$s_4 \text{ is obtained from } s_1 \text{ by replacing one occurrence of } s_2 \text{ by } s_3\},$$

where the ith column and the jth column of e_1 are of arity ℓ_1 and ℓ_2 respectively. The case where $e = \mathrm{rewrite\text{-}all}_{i:\alpha \rightarrow \beta}(e_1)$ is similar to the previous one. □

Let S be the object-level schema $\{S : 1\}$ and let the meta-level schema be empty. Suppose there is an expression e of $\mathcal{MA} - \{\mathrm{wrap}\}$ that is equivalent to $\mathrm{wrap}_1(S)$. Then choose a $v \in \mathbf{V}$ such that $\{(v)\} \not\subseteq A_e$, where A_e is as defined in the previous lemma. Let $\mathcal{K}_v$ be the instance where $\mathcal{K}_v(S) = \{(v)\}$. Then $(v) \in [\![\mathrm{wrap}_1(S)]\!]^{\mathcal{K}_v}$, but $\{(v)\} \not\subseteq [\![e]\!]^{\mathcal{K}_v} \subseteq A_e$. Hence, e cannot be equivalent to $\mathrm{wrap}_1(S)$.

B.2. Extract

We now show the primitivity of the `extract`-operator.

Let $\mathcal{S} = \{S : 1\}$, and $\mathcal{M} = \{R : [\langle 2 \rangle]\}$. Define for each natural number n the relational algebra expression $e_n : 2$ as

$$\underbrace{\pi_1 \pi_1 \pi_1 \ldots \pi_1}_{n \text{ times}}(S) \times S.$$

Let $\mathcal{K}_n$ be the combined instance where $[\![S]\!]^{\mathcal{K}_n} = \emptyset$ and $[\![R]\!]^{\mathcal{K}_n} = \{e_n\}$. Clearly, $[\![\mathrm{extract}_{1:1}(R)]\!]^{\mathcal{K}_n}$ is the relation

e_n	$\underbrace{\pi_1 \pi_1 \pi_1 \ldots \pi_1}_{n \text{ times}}(S)$
e_n	$\underbrace{\pi_1 \pi_1 \ldots \pi_1}_{n-1 \text{ times}}(S)$
$\vdots$	$\vdots$
e_n	$\pi_1 \pi_1(S)$
e_n	$\pi_1(S)$
e_n	S

The following lemma says that this cannot be the case for an expression in $\mathcal{MA} - \{\texttt{extract}\}$, thus proving that $\texttt{extract}$ is primitive.

Lemma B.2 *Let e be an expression of $\mathcal{MA} - \{\texttt{extract}\}$. There exists a constant c_e such that for every natural number i, if*

$$\underbrace{\pi_1 \pi_1 \pi_1 \ldots \pi_1}_{i \text{ times}}(S) \tag{1}$$

occurs as an entry of a tuple in $[e]^{\mathcal{K}_n}$ then $i < c_e$.

Proof. The proof proceeds by induction on the structure of e.

- If $e = S$, $e = \{(v)\}$, or $e = R$ then $c_e := 1$.

- If $e = e_1 \text{ op } e_2$, where op is $\times$ or $\cup$, then $c_e := \max\{c_{e_1}, c_{e_2}\}$.

- If $e = e_1 - e_2$ then $c_e := c_{e_1}$.

- If $e = \text{op}(e')$, where op is σ, π, $\texttt{eval}$, or $\texttt{wrap}$, then $c_e := c_{e'}$.

- Let $e = \texttt{rewrite-all}_{j:\alpha \to \beta}(e')$, with $e' : [\tau_1, \ldots, \tau_n]$. If $\tau_j \neq \langle 1 \rangle$, we set $c_e := c_{e'}$. If $\tau_j = \langle 1 \rangle$, but β is not of the form

  (*i*) $\underbrace{\pi_1 \pi_1 \pi_1 \ldots \pi_1}_{m \text{ times}}(S)$, or

  (*ii*) $\underbrace{\pi_1 \pi_1 \pi_1 \ldots \pi_1}_{m \text{ times}}(\Box_k)$, with $1 \leq k \leq n$,

  (note that m can be zero) again $c_e := c_{e'}$.

  If $\tau_j = \langle 1 \rangle$ and β is of the form (*i*), then $c_e := c_{e'} + m$. If β is of the form (*ii*) and $\tau_k \neq \langle 1 \rangle$, again $c_e := c_{e'}$. Finally, if $\tau_k = \langle 1 \rangle$, then $c_e := c_{e'} + c_{e'} + m$.

- The case $e = \texttt{rewrite-one}_{j:\alpha \to \beta}(e')$ is analogous. $\qquad\square$

B.3 Rewrite-One

We next consider the $\texttt{rewrite-one}$ operator. To this end, let $\mathcal{S} = \{S : 1, R : 1\}$, and let $\mathcal{M} = \{Q : [(1)]\}$. For each natural number n, define $e_n : n$ as the relational algebra expression

$$\underbrace{R \times (R \times (R \times (\ldots \times (R \times R) \ldots))}_{n \text{ times } R}.$$

Define $\mathcal{K}_n$ as the combined instance where $[R]^{\mathcal{K}_n} = [S]^{\mathcal{K}_n} = \emptyset$ and $[Q]^{\mathcal{K}_n} = \{(e_n)\}$. It readily follows that $[\texttt{rewrite-one}_{1:R \to S}(Q)]^{\mathcal{K}_n}$ contains n tuples. The following lemma shows that for any expression e of $\mathcal{MA} - \{\texttt{rewrite-one}\}$, the number of tuples in $[e]^{\mathcal{K}_n}$ is always bounded by some constant independent of n, thus proving primitivity of $\texttt{rewrite-one}$.

Lemma B.3 *Let e be an expression of $\mathcal{MA} - \{\texttt{rewrite-one}\}$. There exist natural numbers c_e and f_e, and a sequence of numbers (d_e^i) such that*

1. *the number of tuples in $[\![e]\!]^{\mathcal{K}_n}$ is less than or equal to c_e;*

2. *the number of different data values occurring in $[\![e]\!]^{\mathcal{K}_n}$ (in data or expression columns) is less than or equal to f_e; and*

3. *for each i, for each t in $[\![e]\!]^{\mathcal{K}_n}$, and for each expression column j, the number of different subexpressions of arity i in $t(j)$ is less than or equal to d_e^i.*

Proof. The proof proceeds by induction on the structure of e.

- If $e = \{(v)\}$, $e = R$, or $e = S$, then $c_e := 1$, $f_e := 1$ and $d_e^i := 0$ for all i.

- If $e = Q$, then $c_e := 1$, $f_e := 0$ and

$$d_e^i := \left\{ \begin{array}{ll} 1 & \text{if } 1 \leq i \leq n; \\ 0 & \text{otherwise.} \end{array} \right.$$

- If $e = e_1 \cup e_2$, then $c_e := c_{e_1} + c_{e_2}$, $f_e := f_{e_1} + f_{e_2}$, and $d_e^i := \max\{d_{e_1}^i, d_{e_2}^i\}$ for all i.

- If $e = e_1 - e_2$, then $c_e := c_{e_1}$, $f_e := f_{e_1}$, and $d_e^i := d_{e_1}^i$ for all i.

- If $e = e_1 \times e_2$, then $c_e := c_{e_1} \cdot c_{e_2}$, $f_e := f_{e_1} + f_{e_2}$, and $d_e^i := \max\{d_{e_1}^i, d_{e_2}^i\}$ for all i.

- If $e = \mathbf{op}(e')$, where $\mathbf{op}$ is σ or π, then $c_e := c_{e'}$, $f_e := f_{e'}$, and $d_e^i := d_{e'}^i$ for all i.

- If $e = \mathbf{wrap}(e')$, then $c_e := c_{e'}$, $f_e := f_{e'}$, $d_e^1 := \max\{d_{e'}^1, 1\}$, and $d_e^i := d_{e'}^i$ for all $i > 1$.

- Let $e = \mathbf{rewrite\text{-}all}_{j:\alpha \to \beta}(e')$, where e' is of arity n. Let f be the number of data values occurring in β, and let d^i be the number of different subexpressions of β of arity i, for each i. Then $c_e := c_{e'}$, $f_e := f_{e'} + f$, and $d_e^i := d_{e'}^i + d^i$ for all i.

- If $e = \mathbf{extract}_{j:m}(e')$, then $c_e := d_{e'}^m \cdot c_{e'}$, $f_e := f_{e'}$, and $d_e^i := d_{e'}^i$ for all i.

- Let $e = \mathbf{eval}_i(e')$, and let the i-th column of e' be of type $\langle m \rangle$. Then $c_e := c_{e'} \cdot (f_{e'})^m$, $f_e := f_{e'}$, and $d_e^i := d_{e'}^i$ for all i. $\square$

B.4. Rewrite-All

Let $\mathcal{K}_n$ be defined as in Section 5.4. Then

$$[\![\mathbf{rewrite\text{-}all}_{1:R \to S}(Q)]\!]^{\mathcal{K}_n}$$

equals the relation

$$\left\{ \left(\underbrace{R \times (R \times (R \times (\ldots \times (R \times R) \ldots)))}_{n \text{ times } R} \right), \left(\underbrace{S \times (S \times (S \times (\ldots \times (S \times S) \ldots)))}_{n \text{ times } S} \right) \right\}.$$

The following lemma shows that this cannot be the case for an expression of $\mathcal{MA} - \{\mathbf{rewrite\text{-}all}\}$, thus proving primitivity of $\mathbf{rewrite\text{-}all}$.

Lemma B.4 *Let e be an expression of $\mathcal{MA} - \{\mathbf{rewrite\text{-}all}\}$. There exists a natural number c_e such that for all $n \geq 1$ every entry in each tuple of $[\![e]\!]^{\mathcal{K}_n}$ contains less than c_e occurrences of S.*

Proof. The proof proceeds by induction on the structure of e.

- If $e = \{(v)\}$, $e = R$, $e = S$, or $e = Q$ then $c_e := 0$.

- If $e = \mathbf{rewrite\text{-}one}_{i:\alpha \to \beta}(e')$, s is the number of occurrences of S in β, and k is the number of expression variables in β, then $c_e := \max\{s + k \cdot c_{e'}, c_{e'}\}$.

- If $e = \mathbf{extract}_{i:m}(e')$, then $c_e := c_{e'}$.

- If $e = e_1 \mathbf{op} e_2$, where $\mathbf{op}$ is $\cup$, $\times$, or $-$, then $c_e := \max\{c_{e_1}, c_{e_2}\}$.

- If $e = \mathbf{op}(e')$, where $\mathbf{op}$ is σ, π, $\mathbf{eval}$, or $\mathbf{wrap}$, then $c_e := c_{e'}$. $\square$