A Uniform Data Model for Relational Data and Meta-Data Query Processing

Manoj Jain, Anurag Mendhekar, and Dirk Van Gucht *

Indiana University

Abstract

We introduce the *uniform data model* to reason about relational data and meta-data query processing. The principal property of the uniform data model is that both meta-data and data are organized in uniformly shaped *information objects*. This uniform treatment of meta-data and data allows for the design of query languages that are independent of the meta-data/data separation. We present two query languages that are designed in this manner: the *uniform calculus* and the *uniform algebra*, which are a logical and algebraic query languages, respectively, for the uniform data model. We establish that the uniform calculus has at least the expressive power of the uniform algebra and prove that, besides providing meta-data query processing capabilities, these languages can efficiently simulate conventional relational query languages. We also give upper-bounds on the expressive power of these languages and since these languages have higher data complexity than the relational calculus and algebra, we introduce sublanguages that have AC^0 data complexity.

1 Introduction

The concept of "meta-data/data" separation is ubiquitous in computer science (as it is in other disciplines). For example, in programming languages, this concept is manifested in the form "Programming Language P / Programs written in P", in document processing we have "Grammar G / Documents structured according to G", in databases we have "Database schema S / Database instances over S", etc.

The benefits of the separation between meta-data and data are well established, and we will not repeat them here. There is, however, a significant cost involved. Operations on data objects are dictated by this structure, which, in turn, is defined at the meta-level by the meta-data. It thus becomes harder to express operations which are independent of specific meta-data context. The example of this dependence that we study in this paper concerns querying relational databases.

Consider a user, `Bob`, who wishes to query a database `foo` maintained in a relational database management system. Assume that `Bob` has the query: "*Find the facts in* `foo` *related to* `Alice`."[1]

* Computer Science Department, Indiana University, Bloomington, IN 47405. e-mail: {mjain,anurag,vgucht}@cs.indiana.edu

[1] Observe that this is a feasible query because it can be computed in linear time in the size of the database.

Before `Bob` can attempt this query, he is obligated to discover `foo`'s schema.[2] Having found this schema, `Bob` can begin to query `foo`. He, however, immediately faces another obstacle: SQL, or any other conventional relational query language, only allows him to formulate queries which have *access-paths* specified in strict accordance with `foo`'s schema. Hence, `Bob` will have to express his schema-*independent* query in a schema dependent way. Furthermore, if `Bob` is interested in running this query in the future, it may no longer have the desired effect (`foo`'s schema may have changed), i.e., `Bob` may have to adapt his query to the current schema. We will call this phenomenon the *meta-data dependence problem* of relational query languages.

In recent years, several papers have appeared in the literature that address the meta-data dependence problem [11, 10, 4, 16, 12, 26, 9]. The solutions proposed there augment the query language with mechanisms that allow it to query both meta-data and ordinary data. In Section 4.4, we will review these papers. Here we only want to state that the solutions advocated in [11, 10, 4], though elegant, are embedded in very powerful object-oriented query languages, whereas the solution in [16], on the contrary, does not go far enough because certain reasonable meta-data/data queries still cannot be expressed in a meta-data independent way. The approach adopted in [12] is the most advanced and comes the closest to solving the meta-data dependence problem while remaining in the low data complexity class. This approach, however, allows untyped relational queries, thereby moving away from the conventional definition of relational query [5] as it is used in relational database management systems. Finally, the solution advocated in [26], which relies on adding *reflection* to the query language and is sufficiently powerful, is too complex for the task at hand.

In this paper we present another approach to overcome the meta-data dependence problem in relational data processing. The key insight is to retain the meta-data/data separation, but to insist that meta-data and data be maintained in uniformly shaped *information objects*. This uniform treatment of meta-data and data has the liberating consequence that the query languages for this data model can be designed in a manner independent of the meta-data/data separation. Obviously, the challenging part is to propose a concept of information object that is general enough to model relational meta-data and data, but that is sufficiently constrained to remain close in design to the relational data model. Furthermore, it is important to be able to define query languages for this model which are similar to conventional relational query languages, but which are immune to the *meta-data dependence problem*. The *uniform data model* (with its query languages), proposed in

[2] If `Bob` is knowledgeable about RDBMS, he can discover this information by querying the catalog (of course to do this, he will need to know the schema of the catalog; often users accomplish this by browsing through the RDBMS manuals.)

this paper, satisfies all these constraints.

Another area of database research, though we will not emphasize it here in this paper, that our paper addresses relates to the *heterogeneous database systems* (HDBS) [2, 18, 13, 7, 20]. An HDBS is a distributed database system that includes component databases which may be different at the database level (such as data model, query language, and schema). An HDBS is required to access these diverse databases in a unified manner by hiding the heterogeneity of the constituent databases from the users while preserving the autonomy of the constituent databases at the same time. Two of the main challenges in the development of HDBS are: to define a strong integrating model that has sufficient power to capture the conceptual relationships among the information objects in the component databases, and to provide *schema integration*, that is, to unify the representation of semantically similar information that is represented in a heterogeneous way across the component databases. This paper can also provide some insights in solving a major problem inherent in the development of heterogeneous databases due to the semantic issues [17] concerning *semantic heterogeneity* (which occurs when two objects that represent the same real world entity are represented differently), and *semantic discrepancy* (which occurs when two objects that represent the same real world entity have inconsistent information - for example, one component database's data corresponds to meta-data in other component databases). The problem is how to integrate data with semantic heterogeneity and semantic discrepancy in a HDBS. In [8], Kent gives a introduction to the factors that cause semantic heterogeneity and semantic discrepancy in a HDBS, and also presents some approaches to solve these problems. It is well-known that these two problems exist even when all the component databases in a HDBS follow the same data model and query language [10]. We do not give details in this paper on how all these issues related to HDBS can be solved using our approach because our main aim was to present a data model which is simple and elegant, and yet allows for the design of query languages that are independent of the meta-data/data separation. However, as should be clear from this paper, our work has some applications in the area of HDBS (such as providing uniform views of heterogeneous database). For example, if data in all component databases is organized using the relational data model, then it is possible to extend the uniform data model presented in this paper to serve as an integrating model which acts as an 'interpreter' among the component relational databases.

This paper is organized as follows. We begin with a brief review of the relational model. We then describe the uniform data model and its associated query languages. We also discuss their expressiveness and state some complexity results. Finally, we present related work and some conclusions.

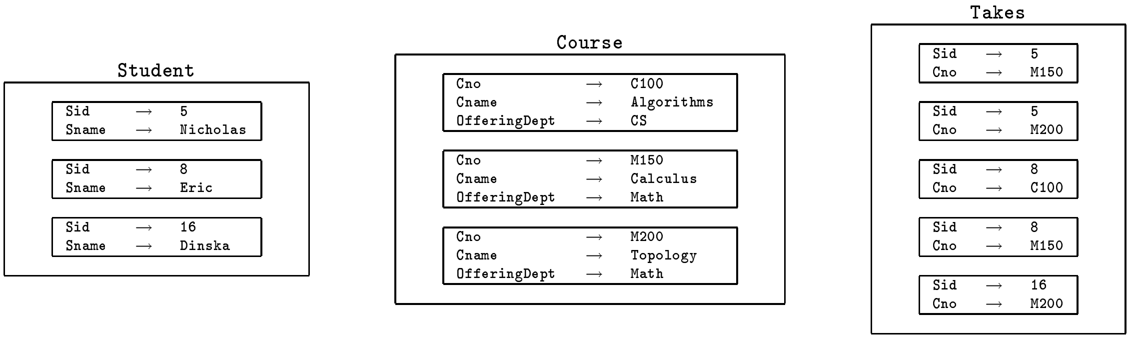

Figure 1: A relational database instance

2 The Relational Data Model

Here we review the basic relational database concepts that will be used in this paper.

Definition 2.1 Let $\mathcal{A}$ and $\mathcal{R}$ be denumerable sets of attribute names and relation names, respectively. A *relation schema* over $\mathcal{A}$ is a finite subset of $\mathcal{A}$. (The set of relation schemas over $\mathcal{A}$ will be denoted by $sch(\mathcal{A})$.) A *relational database schema* Σ over $(\mathcal{A}, \mathcal{R})$ is a finite set of pairs (R, S), where $R \in \mathcal{R}$ and $S \in sch(\mathcal{A})$; furthermore, Σ is required to be a function.

Let $\mathcal{D}$ be an enumerable set. Let $S \in sch(\mathcal{A})$, and let Σ be a database schema over $(\mathcal{A}, \mathcal{R})$. A *tuple* over S with domain values in $\mathcal{D}$ is a mapping $t : S \to \mathcal{D}$. A *relation instance* over S with domain values in $\mathcal{D}$ is a *finite* set of tuples over S with domain values in $\mathcal{D}$. A *relational database instance* over Σ with domain values in $\mathcal{D}$ is a function σ with $dom(\sigma) = dom(\Sigma)$ and such that for each $r \in dom(\Sigma)$, $\sigma(r)$ is a relation instance over $\Sigma(r)$ with domain values in $\mathcal{D}$. □

Example 2.2 Assume that $\mathcal{A}$ contains the attribute names `Sid`, `Sname`, `Cno`, `Cname`, and `OfferingDept`, and assume that $\mathcal{R}$ contains the relation names `Student`, `Course`, and `Takes`. Then we define $\Sigma_{\texttt{enrollment}}$ as the relational database schema

$$\{(\texttt{Student}, \{\texttt{Sid}, \texttt{Sname}\}), (\texttt{Course}, \{\texttt{Cno}, \texttt{Cname}, \texttt{OfferingDept}\}), (\texttt{Takes}, \{\texttt{Sid}, \texttt{Cno}\})\}.$$

The tables in Figure 1 give a relational database instance over $\Sigma_{\texttt{enrollment}}$. □

As relational query languages, we consider the *tuple* relational calculus with the logical connectives $\wedge, \vee$ and $\neg$, and the quantifier $\exists$, and the relational algebra with operators $\times$, $\cup$, $-$, $\hat{\pi}$ (projecting an attribute out), σ, and ρ (attribute renaming). In this paper, we do not consider more powerful relational query languages such as Datalog, FO+fixpoint, and FO+while. These languages also suffer from the meta-data dependence problem. The approach advocated in this paper, however, can be applied to these languages so that they no longer have this problem.

3 The Uniform Data Model

In this section we define the uniform data model, argue that it is a "weak" extension of the relational data model, but show that it is rich enough to model relational meta-data and data.

In the uniform data model a database is modeled as a finite set of *information objects*. Each information object is a *finite binary relation* over some name space $\mathcal{N}$. Observe that in the uniform data model there is no distinction between meta-data and data. Formally:

Definition 3.1 Let $\mathcal{N}$ be a denumerable set which will be referred to as the *name space*.

- An *information object* over $\mathcal{N}$ is a *finite* binary relation over $\mathcal{N}$, i.e, a finite subset of $\mathcal{N} \times \mathcal{N}$. The set of all information objects over $\mathcal{N}$ will be called *information object space* and will be denoted by $IO(\mathcal{N})$.

- A *uniform database* over $\mathcal{N}$ is a finite set of information objects over $\mathcal{N}$. The set of all uniform databases over $\mathcal{N}$ will be called *database space* and will be denoted by $DB(\mathcal{N})$. □

The uniform data model is a weak extension of the relational model.[3] Indeed, if we examine the definitions of *relation schema* and *relation tuple*, we observe that a relation schema is a [*relation name, set of attribute names*]*-pair* and a relation tuple is a *function* from attribute names to data domain values. Clearly, such pairs and functions can be modeled as information objects. One can thus view the information object concept as a "weak" generalization of the relation schema and relation tuple concepts.

Furthermore, a *relational database schema* is (essentially) a set of relation schemas, and a *relational database instance* is a set of tuples. Because relation schemas and tuples can be modeled as information objects, these sets can be modeled as uniform databases. Since we can form a union of two or more uniform databases to obtain a single uniform database, a relational database (with its associated meta-data and data) can be modeled as a *single* uniform database. We conclude that the uniform data model is rich enough to model relational databases.

Remark 3.2 The relational data model allows the definition of combinatorial objects of set-height one and element-width two (tuples and relation schemas), and of set-height two and element-width two[4] (relation instances and database schemas). Observe that the uniform data model operates within these combinatorial constraints. These combinatorial consistencies between the relational

[3]Data models such as the nested relational model or the complex object model could also have been used, but we deemed them too powerful for our purposes. The uniform model can also be viewed as a restriction of the nested relational model to set-height two and element-width two.

[4]The reason that the element-width is two and not one, derives from the fact that a database schema is defined as a set of *pairs*.

Type	RelationSchema
RelationName	Student
AttributeName	Sid
AttributeName	Sname

Type	RelationSchema
RelationName	Course
AttributeName	Cno
AttributeName	Cname
AttributeName	OfferingDept

Type	RelationSchema
RelationName	Takes
AttributeName	Sid
AttributeName	Cno

Figure 2: Schema representation

Type	RelationTuple
RelationName	Student
Sid	5
Sname	Nicholas

...

Type	RelationTuple
RelationName	Course
Cno	C100
Cname	Algorithms
OfferingDept	CS

...

Type	RelationTuple
RelationName	Takes
Sid	5
Cno	M150

...

Figure 3: Representation of database instance

database model and the uniform data model are the technical reasons behind our statement: "the uniform data model is a weak extension of the relational data model." □

Example 3.3 Reconsider the relational database introduced in Example 2.2. We can represent the schema information of this database using the uniform database shown in Figure 2, and we can represent the database instance by the uniform database shown in Figure 3. By taking the union of these two uniform databases we obtain a single uniform database that represents the example relational database. □

Remark 3.4 (Encoding Relational Databases.) The previous example describes a general strategy for *encoding* relational databases in the uniform data model. Given a relational database $d = (\Sigma, \sigma)$, we will denote its encoding in the uniform data model by $enc(d)$.

Technically, when decoding relational databases defined over $\mathcal{R}$ (the relation name space), $\mathcal{A}$ (the attribute name space), and $\mathcal{D}$ (the data domain space), one needs to, in advance, set aside names in the name space $\mathcal{N}$ for the special names `Type`, `RelationSchema`, `RelationName`, `AttributeName`, and `RelationTuple`. The names in $\mathcal{N}$ that encode values in $\mathcal{A}$, $\mathcal{R}$, or $\mathcal{D}$ need to be different from these special names. This can always be accomplished because we assumed $\mathcal{N}$ to be a denumerable set. (It is also useful to observe that it is possible, via constraint specification, to ensure that a uniform database encodes a relational database. For work along these lines, we refer the reader to Colby's dissertation [6].) □

Remark 3.5 (Dual Nature of Information Objects.) The uniform model has a built-in dualism, which the relational model lacks, that allows it to overcome the meta-data/data separation.

Type	Key
RelationName	Course
KeyAttributeName	Cno

and

Type	SubsetConstraint
LeftRelationName	Takes
RightRelationName	Course
LeftAttributeName	Cno
RightAttributeName	Cno

Figure 4: Representation of relational constraints

This dualism is manifested in the definition of *information object*. Given an information object $\mathbf{o}$, its inverse, $\mathbf{o}^{-1}$, is also an information object.[5] As a consequence of this ability to "invert" an information object, we can "move" meta-data into data position and vice-versa. The relational model lacks this dualism: the "inverse" of a relation schema or of a relation tuple is an ill-defined concept in the relational model. □

Remark 3.6 (Modeling Other Meta-Data.) The uniform data model is richer than what Example 3.3 might suggest. For example, it is possible to encode multiple databases (with different schemas) into a *single* uniform database. Also, it is possible to encode *constraints* as information objects. For example, the information objects in Figure 4 capture a key constraint on the `Course` relation and a subset constraint `Takes[Cno]` $\subseteq$ `Course[Cno]` between the relations `Takes` and `Course`, respectively. □

4 Query Languages for the Uniform Data Model

We present two query languages for the uniform data model. These languages are, respectively, generalizations of the tuple relational calculus and the relational algebra. We illustrate, through examples, how these languages can be used to formulate data and meta-data queries. The ability to express both the relational data and meta-data queries in the same language and in the manner independent of the meta-data/data separation comes at the expense of slightly complex formulation of some simple, ordinary queries (e.g., *join*). However, as we will show in the examples, the new formulation of queries in our languages still strongly parallels the old formulations in the existing languages. We establish that the proposed languages are in `PSPACE`, but specify a (safe) sublanguage that is in $\texttt{AC}^0$ and show that this safe language remains powerful enough to support a wide range of data and meta-data queries.[6] We conclude by discussing some related work.

[5] The inverse of an object $\mathbf{o}$, denoted $\mathbf{o}^{-1}$, is defined as $\{(a, b) \mid (b, a) \in \mathbf{o}\}$. (See Section 4.1.)

[6] $\texttt{AC}^0$ is the class of problems that can be solved using polynomially many processors in constant time. This implies that the queries in this sublanguage can be evaluated in *parallel* very efficiently (in constant time).

4.1 The Uniform Calculus

In this section we present the *uniform calculus* (*UC*) which is a query language for the uniform model. This language can be viewed as a generalization of the relational tuple calculus and also as a specialization of complex object calculus [1].[7] *UC* has two different types of variables, namely, *name variables* (which range over the name space $\mathcal{N}$) and *information object variables* (which are also called *object variables* and range over the information object space $IO(\mathcal{N})$). The object variables can be viewed as a generalization of *tuple variables* of the tuple relational calculus, and the name variables can be thought of as *domain variables* of the domain relational calculus except that these variables can range over the attribute names in addition to ranging over the domains of the attributes. Also, the relational predicate names of the tuple calculus are replaced by the single database predicate name $\mathbb{DB}$.

4.1.1 Syntax

The formal definition of the syntax of *UC* is as follows. Let $\mathbf{N}$, $\mathbf{O}$ and $\mathbf{C}$ be denumerable sets of *name variables*, *object variables* and *constants*, respectively, and let $\mathbb{DB}$ be the single uniform database predicate.

- **Terms**.

  1. If $n \in \mathbf{N}$, then n is a *name term*.
  2. If $c \in \mathbf{C}$, then c is a *name term*.
  3. If $x \in \mathbf{O}$, then x is an *object term*.
  4. If $c \in \mathbf{C}$, then $\widehat{c}$ is an *object term*.
  5. $()$ is an *object term*.
  6. If a is a name term, and t is an object term, then $t(a)$ is an *object term*.
  7. If $a_1, a_2, \ldots, a_r$ and $b_1, b_2, \ldots, b_r$ are name terms, then $\{(a_1, b_2), (a_2, b_2), \ldots, (a_r, b_r)\}$ is an object term.
  8. If t and u are object terms, then t^{-1}, $\overline{t}$, $t \cup u$, and $t \odot u$ are also object terms.[8]

[7] In complex object calculus there are variables of each *sort* type. *Sorts* (or types) indicate the structure of the data. Complex objects are relations in which the entries are not required to be atomic domain elements (as in the relational model) but are allowed to be themselves relations.

[8] Since information objects are binary relations, we have adopted Tarski's well known binary relation algebra [24] as our object term algebra. Tarski's algebra consists of four operations: inversion of a relation, relative complementation of a relation, union of two relations, and composition of two relations. Besides providing mechanisms to aggregate information objects, the object term algebra has operations that make meta-data query processing possible.

If t is an object term, then we write $t(x_1, \ldots, x_n)$ to denote that the free variables of t are among the variables $x_1, \ldots, x_n$. A similar notational convention is followed for formulas, which are defined as follows:

- **Formulas**.

  1. If t is an object term then $\mathbb{DB}(t)$ is a *formula*.
  2. If a and b are name terms, then $a = b$ is a *formula*.
  3. If t and u are object terms, then $t = u$ and $t \subseteq u$ are *formulas*.
  4. If a and b are name terms, and t is an object term, then $t(a, b)$ is a *formula*.
  5. If φ and ψ are formulas, then $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\varphi \Rightarrow \psi$ are *formulas*.
  6. If φ is a formula, then $\neg\varphi$ is a *formula*.
  7. If φ is a formula, and x is a object variable, then $\exists x\varphi$ and $\forall x\varphi$ are *formulas*.
  8. If φ is a formula, and n is a name variable, then $\exists n\varphi$ and $\forall n\varphi$ are *formulas*.

- **Queries**. A UC-*query* is an expression of the form

$$\{x \mid \phi(x)\}$$

  where ϕ is a formula in *UC* with $x \in \mathbf{O}$ as its only free variable. Naturally, this expression defines the uniform database containing all information objects x such that $\phi(x)$ is true.

4.1.2 Semantics

Let B be a uniform database over the name space $\mathcal{N}$. Furthermore, assume that the constants in $\mathbf{C}$ have been assigned a value in $\mathcal{N}$. We will denote the *finite* set containing those values of $\mathcal{N}$ that occur in database B by $adom(B)$, and the set of all information objects that can be constructed from elements in $adom(B)$ by $IO(B)$.[9] Notice that because $adom(B)$ is a finite set, $IO(B)$ is also a finite set. We define the set of all valuations, $\mathtt{Val_B}$, in the context of database B as follows.

$$\begin{aligned} \mathtt{Val_B} \; = \; \{ v : \mathbf{O} \cup \mathbf{N} \rightarrow IO(B) \cup adom(B) \mid \; & \textit{if } x \in \mathbf{O} \textit{ then } v(x) \in IO(B), \\ & \textit{and if } x \in \mathbf{N} \textit{ then } v(x) \in adom(B)\}. \end{aligned}$$

The semantics of terms, formulas, and queries is then defined with respect to valuations in $\mathtt{Val_B}$ as follows:

- **Terms**. Let $v \in \mathtt{Val_B}$, then $[\![t]\!]_v$ gives the semantics of the term t under the valuation v.

[9]In the semantics of terms and formulas that contain constants, we will implicitly assume that $adom(B)$ has been augmented with the values in $\mathcal{N}$ that correspond to these constants.

1. $[\![n]\!]_v = v(n)$.

2. $[\![x]\!]_v = v(x)$.

3. $[\![\widehat{c}]\!]_v = \{(c, c)\}$.

4. $[\![()]\!]_v = \{\}$, i.e., the empty information object.

5. $[\![t(a)]\!]_v = \{(d_1, d_2) \mid d_1 = [\![a]\!]_v \wedge (d_1, d_2) \in [\![t]\!]_v\}$.

6. $[\![\{(a_1, b_2), (a_2, b_2), \ldots, (a_r, b_r)\}]\!]_v = \{([\![a_1]\!]_v, [\![b_2]\!]_v), ([\![a_2]\!]_v, [\![b_2]\!]_v), \ldots, ([\![a_r]\!]_v, [\![b_r]\!]_v)\}$.

7. The semantics of the object term algebra operations is as follows: [10]

   $[\![t^{-1}]\!]_v = \{(d_1, d_2) \mid (d_2, d_1) \in [\![t]\!]_v\}$.

   $[\![\overline{t}]\!]_v = \{(d_1, d_2) \mid (d_1, d_2) \in adom(B) \times adom(B) \wedge (d_1, d_2) \notin [\![t]\!]_v\}$.

   $[\![t \cup u]\!]_v = \{(d_1, d_2) \mid (d_1, d_2) \in [\![t]\!]_v \vee (d_1, d_2) \in [\![u]\!]_v\}$.

   $[\![t \odot u]\!]_v = \{(d_1, d_2) \mid \exists d_3 (d_1, d_3) \in [\![t]\!]_v \wedge (d_3, d_2) \in [\![u]\!]_v\}$.

- **Formulas**. Let $v \in \mathtt{Val}_{\mathtt{B}}$. Then v satisfies the formula φ if and only if the condition on the right-hand side of the $\Longleftrightarrow$ in the following items is satisfied.

1. $v \models \mathbb{DB}(x) \iff v(x) \in B$.

2. $v \models a = b \iff [\![a]\!]_v = [\![b]\!]_v$.

3. $v \models t = u \iff [\![t]\!]_v = [\![u]\!]_v$.

   $v \models t \subseteq u \iff [\![t]\!]_v \subseteq [\![u]\!]_v$.

4. $v \models t(a, b) \iff ([\![a]\!]_v, [\![b]\!]_v) \in [\![t]\!]_v$.

5. $v \models \varphi \wedge \psi \iff v \models \varphi \ \ \textit{and} \ \ v \models \psi$.

   $v \models \varphi \vee \psi \iff v \models \varphi \ \ \textit{or} \ \ v \models \psi$.

   $v \models \varphi \Rightarrow \psi \iff \textit{if}\ v \models \varphi \ \ \textit{then} \ \ v \models \psi$.

6. $v \models \neg\varphi \iff v \not\models \varphi$.

7. $v \models \exists x \varphi \iff \textit{for some}\ o \in \mathit{IO}(B),\ v[o/x] \models \varphi$.

   $v \models \forall x \varphi \iff \textit{for each}\ o \in \mathit{IO}(B),\ v[o/x] \models \varphi$.

8. $v \models \exists n \varphi \iff \textit{for some}\ d \in adom(B),\ v[d/n] \models \varphi$.

   $v \models \forall n \varphi \iff \textit{for each}\ d \in adom(B),\ v[d/n] \models \varphi$.

- **Queries**.

1. $[\![\{x \mid \phi(x)\}]\!] = \{v(x) \ \mid \ v \models \phi(x) \ \wedge \ v \in \mathtt{Val}_{\mathtt{B}}\}$.

[10] These operations are easily expressible in *UC* (and hence can be thought of as just providing syntactic sugar). We present them here because we will need them to define the *uniform algebra* in Section 4.2 and also because they are sometimes useful for concise expressibility of some term expressions.

Example 4.1 In this example, we assume that these queries are applied to the encoding of the relational database specified in Example 2.2 (see Example 3.3).

1. "List all students from the `Student` relation."

$$\{\ x \ \mid\ \mathbb{DB}(x)\ \wedge\ x(\texttt{Type}, \texttt{RelationTuple})\ \wedge\ x(\texttt{RelationName}, \texttt{Student})\}.$$

   The sub-formula $\mathbb{DB}(x)$ indicates that x must be in the database, the sub-formula $x(\texttt{Type}, \texttt{RelationTuple})$ indicates that the pair $(\texttt{Type}, \texttt{RelationTuple})$ must occur in x (which means that x is an information object that corresponds to a relation tuple), and the sub-formula $x(\texttt{RelationName}, \texttt{Student})$ indicates that the pair $(\texttt{RelationName}, \texttt{Student})$ must occur in x (so, x is an information object related to the relation `Student`).

   We will abbreviate a formula of the form

$$\mathbb{DB}(x)\ \wedge\ x(\texttt{Type}, \texttt{RelationTuple})\ \wedge\ x(\texttt{RelationName}, name)$$

   by the formula $name(x)$.

   Using this notation, the above query can be expressed simple as: $\{\ x \ \mid\ \texttt{Student}(x)\}$.

2. "Compute the *join* of the `Student` and `Takes` relations."

   In the relational tuple calculus, this query can be formulated as follows:

$$\begin{aligned}\{\ x \ \mid\ & \exists s\ \exists t\ \ \texttt{Student}(s)\ \wedge\ \texttt{Takes}(t)\ \wedge\ s.\texttt{Sid} = t.\texttt{Sid}\ \wedge \\ & x.\texttt{Sid} = s.\texttt{Sid}\ \wedge\ x.\texttt{Sname} = s.\texttt{Sname}\ \wedge\ x.\texttt{Cno} = t.\texttt{Cno}\}.\end{aligned}$$

   Observe that in this query, there are three tuple variables, x, s, and t, of which only x is a free variable. Semantically, the above tuple calculus query represents a relation whose value is a set of tuples which are formed by *aggregating* (i.e. *tupling*) appropriate values for each component of the tuple x. We want to write a query in *UC* which has a semantics corresponding to this tuple calculus query, i.e. we want to aggregate appropriate component values in the information object x and the way it can be done is by *unioning* the information objects containing these component values. Thus, the concept of aggregating tuples in the relational data model corresponds to unioning information objects in the uniform data model. We can, therefore, formulate this query[11] in *UC* as follows:

$$\begin{aligned}\{\ x \ \mid\ & \exists s\ \exists t\ \ \texttt{Student}(s)\ \wedge\ \texttt{Takes}(t)\ \wedge\ s(\texttt{Sid}) = t(\texttt{Sid})\ \wedge \\ & x = s(\texttt{Sid}) \cup s(\texttt{Sname}) \cup t(\texttt{Cno}) \cup \{(\texttt{Type}, \texttt{RelationTuple}), (\texttt{RelationName}, \texttt{ST})\}\}.\end{aligned}$$

[11] Here we called the resulting relation `ST`. Notice that there is more than one formulation for some formulas. (for example, $s(\texttt{Sid}) = t(\texttt{Sid})$ can also be formulated as $\exists n\ s(\texttt{Sid}, n) \wedge t(\texttt{Sid}, n)$, and, $\widehat{\texttt{Sid}} \odot s = \widehat{\texttt{Sid}} \odot t$.)

3. "List all courses from the Course relation that do not have any enrollment."

$$\{\ x \ \mid\ \texttt{Course}(x)\ \wedge\ \neg\ \exists t\ \exists c\ \texttt{Takes}(t)\ \wedge\ t(\texttt{Cno},c)\ \wedge\ x(\texttt{Cno},c)\}.$$

4. "Obtain the schema information of the Student relation."

$$\{\ x \ \mid\ \mathbb{DB}(x)\ \wedge\ x(\texttt{Type},\texttt{RelationSchema})\ \wedge\ x(\texttt{RelationName},\texttt{Student})\}.$$

5. "What is known about Dinska in the database?"

   Observe that this query is formulated in a meta-data independent fashion; so is its corresponding *UC* query.

$$\{\ x \ \mid\ \mathbb{DB}(x)\ \wedge\ \exists n\ (x(\texttt{Dinska},n)\ \vee\ x(n,\texttt{Dinska}))\}.$$

   The sub-formula $x(\texttt{Dinska},n)$ indicates that the value Dinska occurs as a left-value in a pair of x, and the sub-formula $x(n,\texttt{Dinska})$ indicates that the value Dinska occurs as a right-value in a pair of x. □

4.2 The Uniform Algebra

The *uniform algebra*, denoted as *UA*, is a functional language like the relational algebra. It provides two levels of operations: one for the manipulation of information objects and another for the manipulation of uniform databases. At the level of information objects, the uniform algebra offers operations corresponding to terms in the uniform calculus. At the level of databases, *UA* provides operations to union and subtract two databases, collapse the database into a database containing a single information object, construct a database containing all sub-objects of the information objects in the database, and a pair of *mapping* operators to apply a function to information objects (that satisfy a given condition) in a uniform database. This section first presents the family of core information object operators and various other derived operators that can be simulated by them. We then present a family of core database operations followed by several additional derived database operations that can be expressed in the uniform algebra. Finally, as with the uniform calculus, we illustrate the uniform algebra defined in this section with a series of examples involving meta-data and conventional data queries. The following sets are the ones on which the primary operations are defined: name space, $\mathcal{N}$, information object space, $IO(\mathcal{N})$, and database space, $DB(\mathcal{N})$. (See Definition 3.1. For simplicity, we will use *IO* and *DB* instead of $IO(\mathcal{N})$ and $DB(\mathcal{N})$, respectively.)

The operations at the information object level are defined by the following rules:

Information object operations

Constants	$\dfrac{c : \mathcal{C}}{\hat{c} : \mathit{IO}}$	Empty Object	$\dfrac{}{() : \mathit{IO}}$	IO Inversion	$\dfrac{\mathbf{o} : \mathit{IO}}{\mathbf{o}^{-1} : \mathit{IO}}$
IO complement	$\dfrac{\mathbf{o} : \mathit{IO}}{\overline{\mathbf{o}} : \mathit{IO}}$	IO union	$\dfrac{\mathbf{o}_1, \mathbf{o}_2 : \mathit{IO}}{\mathbf{o}_1 \cup \mathbf{o}_2 : \mathit{IO}}$	IO compose	$\dfrac{\mathbf{o}_1, \mathbf{o}_2 : \mathit{IO}}{\mathbf{o}_1 \odot \mathbf{o}_2 : \mathit{IO}}$

The semantics of these operations on information objects follows that for *UC* terms and is described in the previous section. We can derive the following operations from this above set of information object operations.

$\{(\mathtt{a}, \mathtt{b})\}$	$=_{\text{def}}$	$\hat{\mathtt{a}} \odot (\overline{(\hat{\mathtt{a}} \cup \hat{\mathtt{b}})} \odot \hat{\mathtt{b}})$.	(Constant IO containing a single pair $(\mathtt{a}, \mathtt{b})$.)
$\mathbf{o}_1 \cap \mathbf{o}_2$	$=_{\text{def}}$	$\overline{\overline{\mathbf{o}_1} \cup \overline{\mathbf{o}_2}}$.	(Intersection.)
$\mathbf{o}_1 - \mathbf{o}_2$	$=_{\text{def}}$	$\mathbf{o}_1 \cap \overline{\mathbf{o}_2}$.	(Subtraction.)
$\hat{\pi}_{\mathtt{A}}(\mathbf{o})$	$=_{\text{def}}$	$\mathbf{o} - (\hat{\mathtt{A}} \odot \mathbf{o})$.	(Projecting out "attribute" $\mathtt{A}$.)
$\rho_{\mathtt{A}/\mathtt{B}}(\mathbf{o})$	$=_{\text{def}}$	$\hat{\pi}_{\mathtt{A}}(\mathbf{o}) \cup (\{(\mathtt{B}, \mathtt{A})\} \odot \mathbf{o})$.	(Renaming "attribute" $\mathtt{A}$ by "attribute" $\mathtt{B}$.)
$\mathbf{o} \ni \{(\mathtt{a}, \mathtt{b})\}$	$=_{\text{def}}$	$\hat{\mathtt{a}} \odot (\mathbf{o} \odot \hat{\mathtt{b}})$.	(Evaluates to $\{(\mathtt{a}, \mathtt{b})\}$ if a pair $(\mathtt{a}, \mathtt{b}) \in \mathbf{o}$; $()$ otherwise.)

The database operations are defined by the following rules:

Database operations

$\mathbb{DB}$	$\dfrac{}{\mathbb{DB} : \mathit{DB}}$	$\mathit{DB}_{\emptyset}$	$\dfrac{}{\emptyset : \mathit{DB}}$	$\mathit{DB}_{()}$	$\dfrac{}{\{()\} : \mathit{DB}}$	Universe	$\dfrac{}{\mathbb{DB}_{\mathtt{U}} : \mathit{DB}}$
Union	$\dfrac{E_1, E_2 : \mathit{DB}}{E_1 \cup E_2 : \mathit{DB}}$			Subtraction	$\dfrac{E_1, E_2 : \mathit{DB}}{E_1 - E_2 : \mathit{DB}}$		
Database Collapse	$\dfrac{E : \mathit{DB}}{\mathit{implode}(E) : \mathit{DB}}$			Sub-object Construction	$\dfrac{E : \mathit{DB}}{\mathit{explode}(E) : \mathit{DB}}$		
Map$_=$	$\dfrac{c, e : \mathit{IO} \times \mathit{IO} \to \mathit{IO} \quad E_1, E_2 : \mathit{DB}}{\mathit{map}_=[c, e; E_1, E_2] : \mathit{DB}}$						
Map$_{\neq}$	$\dfrac{c, e : \mathit{IO} \times \mathit{IO} \to \mathit{IO} \quad E_1, E_2 : \mathit{DB}}{\mathit{map}_{\neq}[c, e; E_1, E_2] : \mathit{DB}}$						

As in the case of uniform calculus these operators are interpreted in the context of a given database B. The $\mathbb{DB}$ operator returns the entire database B, $\emptyset$ returns the *empty database* (database containing no information object), $\{()\}$ returns the database containing *only* the *empty* information object, and $\mathbb{DB}_{\mathtt{U}}$ is a database consisting of all the information objects that can be constructed from the values in $\mathit{adom}(B)$ (i.e. a database containing the *powerset* of $\mathit{adom}(B) \times \mathit{adom}(B)$).

The semantics of the union and subtraction operations are as follows:

$$E_1 \cup E_2 \;=\; \{\, \mathbf{o} \;\mid\; \mathbf{o} \in E_1 \;\vee\; \mathbf{o} \in E_2 \}.$$

$$E_1 - E_2 \;=\; \{\, \mathbf{o} \;\mid\; \mathbf{o} \in E_1 \;\wedge\; \mathbf{o} \notin E_2 \}.$$

The *explode* operator constructs a database of all the sub-objects of the information objects of the input database:

$$\mathit{explode}(E) \;=\; \{\, \mathbf{o}' \;\mid\; \exists \mathbf{o} \in E \;\wedge\; \mathbf{o}' \subseteq \mathbf{o} \}.$$

and the *implode* operator collapses a database into a singleton set containing the object that is the union of all its information objects:

$$\mathit{implode}(E) \;=\; \{\, \bigcup_{\mathbf{o} \in E} \mathbf{o} \}.$$

The semantics of the *map* operators are as follows:

$$\mathit{map}_{=}[c, e;\; E_1, E_2] \;=\; \{\, e(\mathbf{o}_1, \mathbf{o}_2) \;\mid\; \mathbf{o}_1 \in E_1 \;\wedge\; \mathbf{o}_2 \in E_2 \;\wedge\; c(\mathbf{o}_1, \mathbf{o}_2) = () \}.$$

$$\mathit{map}_{\neq}[c, e;\; E_1, E_2] \;=\; \{\, e(\mathbf{o}_1, \mathbf{o}_2) \;\mid\; \mathbf{o}_1 \in E_1 \;\wedge\; \mathbf{o}_2 \in E_2 \;\wedge\; c(\mathbf{o}_1, \mathbf{o}_2) \neq () \}.$$

where c and e are object algebra term expressions which we will refer to as *mapping condition* and *mapping function*, respectively. The *map* operator is a higher-order mapping operator over databases. It allows manipulations to be done at the level of information objects and extends these manipulations to databases. We expect both c and e to be constructed out of elementary operations in the algebra. Therefore, we need to include the following functional operators:

Functional operations

Composition	$\dfrac{f, g_1, g_2 :\; \mathit{IO} \times \mathit{IO} \rightarrow \mathit{IO}}{f \circ \langle g_1, g_2 \rangle :\; \mathit{IO} \times \mathit{IO} \rightarrow \mathit{IO}}$
Argument Projection	$\dfrac{}{\mathbf{x}_i :\; \mathit{IO} \times \mathit{IO} \rightarrow \mathit{IO}}$
Constant IO	$\dfrac{}{K :\; \mathit{IO} \rightarrow (\mathit{IO} \times \mathit{IO} \rightarrow \mathit{IO})}$

The semantics of these operators are as follows.

$$f \circ \langle g_1, g_2 \rangle (x_1, x_2) = f(g_1(x_1, x_2), g_2(x_1, x_2))$$

From the definition of $\mathit{map}_{=}$ and $\mathit{map}_{\neq}$, it follows that two information objects will be made available to the functional argument. It then becomes necessary to select particular information objects. This is enabled by the following operator:

$$\mathbf{x}_i(x_1, x_2) = x_i \;\;\text{where } 1 \leq i \leq 2.$$

Similarly, it becomes necessary to inject an information object into a function that always returns that information object. We define:

$$Kx(x_1, x_2) = x$$

We will denote Kc by c. Also, $f \circ \langle g_1, g_2 \rangle$ will be written as $f(g_1, g_2)$.

Let E, E_1 and E_2 be uniform databases. We can simulate the following relational algebra operations on these databases in the uniform algebra:

- $\overline{E} \;=_{\text{def}}\; \mathbb{DB}_{\mathtt{U}} - E.$
- $E_1 \cap E_2 \;=_{\text{def}}\; \overline{\overline{E_1} \cup \overline{E_2}}.$
- $E_1 \times E_2 \;=_{\text{def}}\; \mathit{map}_{=}[(), \mathbf{x}_1 \cup \mathbf{x}_2; E_1, E_2].$
- $\sigma_{\mathtt{A}=\mathtt{c}}(E) \;=_{\text{def}}\; \mathit{map}_{\neq}[\mathbf{x}_1 \ni \{(\mathtt{A}, \mathtt{c})\}, \mathbf{x}_1; E, \{()\}].$
- $\hat{\pi}_{\mathtt{A}}(E) \;=_{\text{def}}\; \mathit{map}_{=}[(), \hat{\pi}_{\mathtt{A}}\mathbf{x}_1; E, \{()\}].$
- $\rho_{\mathtt{A/B}}(E) \;=_{\text{def}}\; \mathit{map}_{=}[(), \rho_{\mathtt{A/B}}\mathbf{x}_1; E, \{()\}].$

Example 4.2 In this example again, we assume that these queries are applied to the encoding of the relational database specified in Example 2.2 (see Example 3.3).

1. "List all students from the `Student` relation."

$$\begin{aligned}&\mathit{map}_{\neq}[\mathbf{x}_1 \ni \{(\mathtt{Type}, \mathtt{RelationTuple})\},\ \mathbf{x}_1;\\ &\qquad \mathit{map}_{\neq}[\mathbf{x}_1 \ni \{(\mathtt{RelationName}, \mathtt{Student})\},\ \mathbf{x}_1;\ \mathbb{DB},\ \{()\}],\\ &\qquad \{()\}].\end{aligned}$$

   The $\mathbb{DB}$ operator returns all the objects from the database B, and the inner map expression returns a database containing only those objects that contain the pair $(\mathtt{RelationName}, \mathtt{Student})$ (i.e., the objects that contain data and meta-data information for the relation `Student`). Then the outer map expression takes the output of the inner map expression as its input database and returns a database containing those objects which correspond to a relation tuple of the relation `Student` (i.e., the objects related to relation `Student` that contain the pair $(\mathtt{Type}, \mathtt{RelationTuple})$).

   We will abbreviate a database expression of the form

$$\begin{aligned}&\mathit{map}_{\neq}[\mathbf{x}_1 \ni \{(\mathtt{Type}, \mathtt{RelationTuple})\},\ \mathbf{x}_1;\\ &\qquad \mathit{map}_{\neq}[\mathbf{x}_1 \ni \{(\mathtt{RelationName}, \mathit{name})\},\ \mathbf{x}_1;\ \mathbb{DB},\ \{()\}],\\ &\qquad \{()\}].\end{aligned}$$

by the database expression $name(B)$.

Using this notation, the above query can be expressed simply as: $\texttt{Student}(B)$.

2. "List all courses from the `Course` relation that do not have any enrollment."

$$map_{=}[\mathbf{x}_1 \odot \mathbf{x}_2{}^{-1},\ \mathbf{x}_1;\ \texttt{Course}(B), implode(\texttt{Takes}(B))].$$

   The database expressions $\texttt{Course}(B)$ and $\texttt{Takes}(B)$ return databases containing the information objects that represent the tuples of the `Course` and `Takes` relations, respectively. Moreover, $implode(\texttt{Takes}(B))$ returns a database containing only one information object that is built by unioning all the information objects in the $\texttt{Takes}(B)$ database. The result of this *map* expression is then the database containing all those information objects from $\texttt{Course}(B)$ for which the *mapping condition*, $\mathbf{x}_1 \odot \mathbf{x}_2{}^{-1}$, returns an empty information object. In relational model terms, this means that the result contains all those tuples from the `Course` relation which have a `Cno` value that is different from all the `Cno` values in the `Takes` relation.

3. "For each course number `Cno` in the `Takes` relation build a relation with name `Cno`, and as tuples the student ids that are associated with `Cno` in `Takes`."

   This query is a meta-data query wherein ordinary data values (i.e, the course numbers) are *promoted* to meta-data. Furthermore, the number of relations created by this query depends on the current status of `Takes` and can, therefore, not be statically determined. (Obviously this query can not be expressed in the relational calculus or algebra.)

$$map_{=}[(),\ \rho_{\texttt{Cno}/\texttt{RelationName}}(\hat{\pi}_{\texttt{RelationName}}(\mathbf{x}_1));\ \texttt{Takes}(B),\ \{()\}].$$

   $\texttt{Takes}(B)$ returns a database containing all the information objects that represent tuples in the `Takes` relation. Then the promotion of the `Cno` values to the relation name status is done by first projecting out the (`RelationName`, `Takes`) pair from these information objects and then renaming the attribute name `Cno` of these information objects by `RelationName`. This query applied to the encoding of the relational database specified in Example 2.2 (see Example 3.3) produces the uniform database shown in Figure 5.

4. "List all the (`R`, `A`) pairs such that `R` is the name of a relation containing a tuple with value '`Dinska`' for attribute `A`."

   This is a query wherein meta-data values (relation names and attribute names satisfying the above query) are *demoted* to conventional data values. We will write this query in *UA* in two steps:

Type	RelationTuple
RelationName	M150
Sid	5

Type	RelationTuple
RelationName	M200
Sid	5

Type	RelationTuple
RelationName	C100
Sid	8

Type	RelationTuple
RelationName	M150
Sid	8

Type	RelationTuple
RelationName	M200
Sid	16

Figure 5: Result of query 3 from Example 4.2

(a) First we get all the information objects that contain '`Dinska`' in value position (i.e, as a right value in a pair of the information object.)

$$B_{\texttt{Dinska}} = map_{\neq}[\mathbf{x}_1 \odot \widehat{\texttt{Dinska}},\ \mathbf{x}_1;\ \mathbb{B},\ \{()\}].$$

(b) Next, for each information object that contains the value '`Dinska`', we demote the corresponding meta-data values (relation name and attribute name) to data values using the following *UA* expression (observe the usage of the inverse operator):

$$\begin{aligned} map_{=}[(),\ (\{(\texttt{Type}, \texttt{RelationTuple})\} \cup \{(\texttt{RelationName}, \texttt{DinskaInfo})\} \cup \\ (\{(\texttt{Rel}, \texttt{RelationName})\} \odot \mathbf{x}_1) \cup (\{(\texttt{Att}, \texttt{Dinska})\} \odot {\mathbf{x}_1}^{-1})); \\ B_{\texttt{Dinska}},\ \{()\}]. \end{aligned}$$

In this example, we called the resulting relation `DinskaInfo`, and used `Rel` and `Att` as the attribute names which will hold the `R` and `A` values, respectively. □

4.3 Complexity and Expressiveness

The uniform calculus and uniform algebra were introduced as generalizations of the relational calculus and relational algebra, respectively. In this section we establish that the uniform calculus has at least the expressive power of the uniform algebra and prove that these languages can efficiently simulate conventional relational query languages. We also give upper-bounds on the expressive power of these languages and, since these languages have higher data complexity than the relational calculus and algebra, we introduce sublanguages that have AC^0 data complexity.

Theorem 4.3 *Every query expressible in uniform algebra is expressible in uniform calculus.*

Proof: We show that for every expression E in the uniform algebra that is of type DB, there is a query $\{x \mid \phi(x)\}$ equivalent to E in the uniform calculus. The proof is by induction on E. The details of the proof are given in Appendix A. □

It is well-known that a relational calculus query can be evaluated in $\mathtt{AC}^0$. The following theorem reveals that the uniform query languages have higher complexity bounds.

Theorem 4.4 *Each query in* UC *(or, in* UA*) can be evaluated in* PSPACE.

Proof (sketch): The uniform calculus allows quantification over information objects (defined as binary relations over the name space) which is normally considered to be a second-order feature. Thus, the uniform calculus forms a (strict) subset of second-order logic. It is well-known that second-order logic queries can be evaluated in PSPACE [15]. □

Theorem 4.4 reveals that the uniform query languages are more powerful than their relational counterparts.[12] There exist however natural sublanguages of the uniform query languages which are in $\mathtt{AC}^0$. We next introduce the *safe uniform algebra* (safe *UA*) which is such a language.[13] Even though the safe uniform algebra is far less expressive than *UA*, it nevertheless still supports conventional as well as meta-data query formulation. The safe uniform algebra is obtained by replacing the *explode* operator by the *weakexplode* operator.

Singleton-Sub-object Construction	$\dfrac{E : DB}{weakexplode(E) : DB}$

The singleton-sub-object construction operator is defined as:

$$weakexplode(E) \ = \ \{\{(a,b)\} \mid \ \exists \mathbf{o} \in E \ \wedge \ (a,b) \in \mathbf{o}\}.$$

If one thinks of information objects as generalizing tuples, then *explode* is a mechanism to generate all the possible sub-tuples of the tuples in a database, and *weakexplode* is a mechanism to generate all the unary sub-tuples (i.e., the components) of the tuples in a database. And, if one thinks of information objects as generalizing relational schemas, then the *explode* is a mechanism to generate all the possible sub-schemas of all the schemas, and *weakexplode* is a mechanism to generate all the unary sub-schemas (i.e., the attributes) of all the schemas.

We have the following important properties of the safe uniform algebra.

Theorem 4.5 *Each query in the safe* UA *can be implemented in* $\mathtt{AC}^0$*. Furthermore, each relational algebra expression E (or relational calculus query) can be simulated in safe* UA*. In particular, there exists a safe* UA *expression* E^{UA} *such that if d is relational database, and $enc(d)$ is its encoding as a uniform database (see Section 3) then* $enc(E(d)) = E^{\mathrm{UA}}(enc(d))$.

[12] For example, it is possible to write a *UC* query that determines the parity of the number of names in $\mathbb{D}$.

[13] It is also possible to define a safe version of the uniform calculus and establish its equivalence to the safe uniform algebra. We do this in the full paper.

Proof (sketch). The safe *UA* can easily be seen to be a sublanguage of the nested relational algebra [25]. In a recent paper, Suciu [23] established that nested relational algebra expressions can be evaluated in $\mathtt{AC}^0$.

The simulation of the relational algebra in safe *UA* uses techniques similar to the ones used to define derived *UA* operators such as $\cap$, $-$, $\times$, etc. □

Remark 4.6 (On the meta-data querying power of safe *UA*.) Even though the safe uniform algebra is substantially less powerful than *UA*, it is still a powerful meta-data query language. For instance, all the examples of meta-data queries given in this paper can be expressed in safe *UA*. This provides evidence that a significant number of meta-data queries can already be formulated in $\mathtt{AC}^0$. □

4.4 Related Work

In an influential paper, Krishnamurthy and Naqvi [11] proposed a variety of syntactic extensions to datalog-like languages to make these languages more powerful with respect to complex-object manipulation and meta-data querying. Several subsequent papers by Chen, Kifer, Lausen, Warren, and Wu [4, 9] further elaborated on Krishnamurthy and Naqvi's work. In [10], Krishnamurthy, Litwin, and Kent extended the language presented in [11] and demonstrated its relational schema integration capabilities. An essential feature of the query languages proposed in these papers is that they allow predicate symbols and ordinary constants to be mixed. On the other hand, all these query languages have very high complexity bounds (in some cases they are even Turing complete).

Ross [16], and subsequently, Lakshmanan, Sadri and Subramanian [12], proposed tractable query languages based on the principles introduced in [11, 4, 9]. The language in [16], however, has limitations which prevent it from expressing natural meta-data queries. In particular, meta-data queries that produce a dynamic number of relations (e.g, Example 4.2 query 3) can not be specified. In other words, queries wherein ordinary data need to be promoted to meta-data status can not be expressed. Hence, in this language, there still is a sharp separation in the treatment of data and meta-data.

As a framework to study relational data and meta-data query processing, the uniform data model fits in between the approach advocated by Krishnamurthy *et.al.* [11, 10] and that advocated by Lakshmanan *et.al.* [12]. In the Lakshmanan approach, the data model is relational, but untyped relational queries can be expressed in the logic that they present. In particular, Schemalog allows the specification of queries which have an relational output schema that can depend on the database instance, i.e. when applied to two different database instances, the same query can yield differently typed output schemas. This notion of query, though considered in [3] in the context of complete relational query languages, is strictly broader than that of Codd [5] for the relational model and as it is used in conventional relational database systems. As a consequence, to extend a relational database system with a query language such as Schemalog, one will need to strongly couple it with the database system's *data definition* sublanguage. So even though Schemalog is to a large degree consistent with the relational database approach, the claim in [12] about the simplicity of the integration of Schemalog in a relational system has to be considered in view of untypedness of their notion of query. In the Krishnamurthy approach, the data model is the complex object data model which is very powerful. In our approach, the data model (uniform data model) is slightly more powerful than the relational data model and the concept of query is also typed. Our approach is similar to the Krishnamurthy approach in the manner the relational database objects (both meta-data and data) are represented. They explicitly represent the relational database objects as complex objects and we use relatively lightweight information objects.

Even though our approach has more in common with that just described, it also has affinities with that of Van den Bussche, Van Gucht and Vossen [26]. These authors extended the relational algebra with *reflection* [21, 22] mechanisms, i.e. *reification* (i.e., associating data with the meaning of a program) and *evaluation* (i.e., associating the meaning of a program with data). In a typical reflective-RA program, one dynamically constructs relational algebra queries with reification tools and subsequently evaluates these queries. The crucial observation in [26] was that, while a reflective-RA program is static, the relational algebra queries it creates can depend on the current data and meta-data status of the database. Thus reflection admits another mechanism to break the meta-data dependence problem.

The problem with reflective-RA programs however is that they become very complex even on simple meta-data queries. In particular, the reification mechanisms are an obstacle to elegant formulation of meta-data queries. Furthermore, from a complexity point of view, reflection is a very powerful programming paradigm, and it is non-trivial to syntactically control its expressive power. Our paper points out that meta-data query languages can be proposed without having to incorporate reflection capabilities into the languages.

Even though there is significant related work, our approach is distinguished by the simplicity of the data model. The uniform data model allows for a very simple and convenient framework to gain theoretical insights and understand the fundamental aspects of query languages that provide meta-data and conventional data querying facility.

5 Conclusions

We introduced the uniform data model and its query languages to overcome the meta-data dependence problem of query languages associated with the relational data model. Our definition of uniform data model only extends the relational data model weakly (see Remark 3.2) so as to remain in compliance with the latter's design restrictions. The uniform data model's query languages, *UC* and *UA*, correspond closely to their relational counterparts, but are more expressive (see Theorem 4.4). This leads to the definition of the safe sublanguages of *UC* and *UA* (we present the safe *UA* in this paper and the safe *UC* is discussed in the full paper) which reduces the data complexity of these languages from `PSPACE` to $\mathtt{AC}^0$, i.e., the complexity of the relational algebra. Furthermore, as motivated by examples, this reduction in expressiveness does not interfere with the ability to express meta-data queries.

A prototype system including the database and information object operations of the *UA* is being implemented. Our goal in this paper was to overcome the meta-data dependence problem in relational data processing by representing relational database objects (both meta-data and data) as uniformly shaped information objects. We feel that uniformity in the representation of meta-data and data is a desirable conceptual property to have, however, in the implementation of this model we can employ any of several structures and techniques that have been used in some advanced database applications to solve the problem of redundancy at the physical data representation level [19, 14].

A natural direction for future research is to adapt relational query languages with iteration constructs (such as datalog, FO + IFP, and FO + PFP) to uniform query languages with corresponding iteration mechanisms. (Technically, to limit the expressiveness of such languages one can use the ideas of Suciu [23] related to *bounded* fixpoints in nested relational query languages.) Such query languages will enable the formulation of meta-data queries which need to be expressed in terms of the path-structure of the meta-data space. An example of such a query is "*Are* `Bob` *and* `Alice` *related in the database?*"

Another natural problem is to extend our approach to richer data models than the relational data model. We are particularly interested in extending our work to data models for structured documents such as HTML documents (and more generally, SGML documents), because in that domain, the need for meta-data query capabilities is even greater than it is in relational databases.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley Publishing Company, 1995.

[2] ACM. *ACM Computing Surveys*, volume 22, September 1990. Special issue on HDBS.

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

[4] W. Chen, M. Kifer, and D.S. Warren. Hilog as a platform for database languages. In *Proc. DBPL 1989.*

[5] E.F. Codd. Relational completeness of data base sublanguages. In *Data Base Systems*, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, N.J., 1972.

[6] L.S. Colby. *Query Languages and a Unifying Framework for Non-Traditional Data Model.* PhD dissertation, Indiana University, 1992.

[7] D. Hsiao. Federated databases and systems: Part-one – a tutorial on their data sharing. *VLDB Journal*, 1:127–179, 1992.

[8] W. Kent. The breakdown of the information model in multidatabase systems. *SIGMOD Record*, special issue on Semantic Issues in Multidatabases, A. Sheth, ed., 20(4), December 1991.

[9] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. Tech. Rep. 90/14, Department of Computer Science, SUNY at Stony Brook, July 1990.

[10] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 40–49, 1991.

[11] R. Krishnamurthy and S. Naqvi. Towards a real horn clause languages. In *Proc. of the 14th VLDB Conf.*, pp. 252–263, 1988.

[12] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Proc. DOOD 1993.*

[13] W. Litwin, M. Leo, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3):267–293, September 1990.

[14] F. Olken, D. Rotem, A. Shoshani, and H. Wong. Scientific and statistical data management research at LBL. In *Proceedings of 3rd International Workshop on Statistical Database Management*, pages 1–20, 1986.

[15] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[16] K. Ross. Relations with relation names as arguments: Algebra and calculus. In *Proceedings 11th ACM Symposium on Principles of Database Systems*, pages 346–353, 1992.

[17] A. Sheth. Semantic issues in multidatabase systems. *SIGMOD Record*, special issue on Semantic Issues in Multidatabases, A. Sheth, ed., 20(4), December 1991.

[18] A. Sheth and J. Larson. Federated database system for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[19] A. Shoshani and H. Wong. Statistical and scientific database issues. *IEEE Transactions on Software Engineering*, SE-11(10):1040–1047, October 1985.

[20] Sigmod. *SIGMOD Record.* Special issue on Semantic Issues in Multidatabases, A. Sheth, ed., 20(4), December 1991.

[21] B.C. Smith. Reflection and semantics in LISP. In *Proceedings 11th ACM Symposium on Principles of Programming Languages*, pages 23–35, 1984.

[22] D. Stemple et al. Type-safe linguistic reflection: a generator technology. Research report CS/92/6, Univ. St Andrews, 1992.

[23] D. Suciu. Fixpoints and bounded fixpoints for complex objects. In *Proc. DBPL 1993*, pages 263–281.

[24] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.

[25] S.J. Thomas and P.C. Fischer. Nested relational structures. In *The Theory of Databases*, P.C. Kanellakis, ed., JAI Press, 1986, pp. 269–307.

[26] J. Van den Bussche, D. Van Gucht, and G. Vossen. Reflective programming in the relational algebra. In *Proc. Twelth PODS*, 1993, pp. 17–25.

Appendix

A Proof of Theorem 4.3

We show that for every expression E in the uniform algebra that is of type DB, there is a query $\{x \mid \phi(x)\}$ equivalent to E in the uniform calculus. The proof is by induction on E.

Basis: This covers the case where E is a 0-ary operator, i.e. one of the following:

1. $E = \mathbb{DB}$. The corresponding query is: $\{\ x \mid \mathbb{DB}(x)\}$.

2. $E = \emptyset$. The corresponding query is: $\{\ x \mid x \neq x\}$.

3. $E = \{()\}$. The corresponding query is: $\{\ x \mid x = ()\}$.

4. $E = \mathbb{DB}_{\mathbb{U}}$. The corresponding query is: $\{\ x \mid x = x\}$.

Induction: Here we have six cases corresponding to the six different database operations of the uniform algebra:

Case 1: $E = E_1 \cup E_2$. By the inductive hypothesis there are uniform calculus queries $\{x \mid \phi_1(x)\}$ and $\{x \mid \phi_2(x)\}$ that define the uniform databases of E_1 and E_2, respectively. Then the query for E is $\{x \mid \phi_1(x) \vee \phi_2(x)\}$.

Case 2: $E = E_1 - E_2$. By the inductive hypothesis there are uniform calculus queries $\{x \mid \phi_1(x)\}$ and $\{x \mid \phi_2(x)\}$ that define the uniform databases of E_1 and E_2, respectively. Then the query for E is $\{x \mid \phi_1(x) \wedge \neg\phi_2(x)\}$.

Case 3: $E = \mathit{explode}(E_1)$. By the inductive hypothesis there is a uniform calculus query $\{x \mid \phi_1(x)\}$ that produces the same uniform database as E_1. Then the query for E is $\{x \mid \exists x_1(\phi_1(x_1) \wedge x \subseteq x_1)\}$.

Case 4: $E = \mathit{implode}(E_1)$. By the inductive hypothesis there is a uniform calculus query $\{x \mid \phi_1(x)\}$ that produces the same uniform database as E_1. Then the query for E is $\{x \mid \forall x_1(\phi_1(x_1) \Rightarrow x_1 \subseteq x) \wedge \neg\exists x_2(\forall x_1(\phi_1(x_1) \Rightarrow x_1 \subseteq x_2) \wedge x_2 \subset x)\}$.[14]

Case 5: $E = \mathit{map}_{=}[c, e; E_1, E_2]$. By the inductive hypothesis there are uniform calculus queries $\{x \mid \phi_1(x)\}$ and $\{x \mid \phi_2(x)\}$ that define the uniform databases of E_1 and E_2, respectively. Moreover, the mapping condition c and the mapping function e are constructed using the functional operators and so we have to state this case by doing induction on the structure of c and e. The translations of c and e to their respective calculus forms follow the same style, and so we will show it for only some cases.

[14]The formula $x_2 \subset x$ is a derived formula and is defined as $(x_2 \subseteq x) \wedge (x_2 \neq x)$.

Basis: This covers the cases where c and e are either a constant IO or a argument projection operation. If c is $\mathbf{x}_1$ and e is $K\mathbf{o}$, then $E = map_{=}[\mathbf{x}_1, K\mathbf{o}; E_1, E_2]$, and the corresponding query is $\{x \mid \exists x_1 \exists x_2(\phi_1(x_1) \wedge \phi_2(x_2) \wedge x_1 = () \wedge x = \mathbf{o})\}$. If c is $K\mathbf{o}$ and e is $\mathbf{x}_1$, then $E = map_{=}[K\mathbf{o}, \mathbf{x}_1; E_1, E_2]$, and the corresponding query is $\{x \mid \exists x_1 \exists x_2(\phi_1(x_1) \wedge \phi_2(x_2) \wedge \mathbf{o} = () \wedge x = x_1)\}$.

Induction: The inductive step involves the cases where c and/or e are constructed using the composition operations. We show the translation for the case where e is constructed using the composition operations and state that the case where c is constructed using the composition operations follows the same pattern. Let f, g_1, and g_2 be term functions and let $e = f(g_1, g_2)$. Then $E = map_{=}[c, f(g_1, g_2); E_1, E_2]$, and using the inner induction assumption we have that $map_{=}[c, g_1; E_1, E_2]$ and $map_{=}[c, g_2; E_1, E_2]$ have equivalent calculus expressions, $\psi_1(x)$ and $\psi_2(x)$, respectively. Then the query for E is $\{x \mid \exists x_1 \exists x_2(\psi_1(x_1) \wedge \psi_1(x_2) \wedge x = f(x_1, x_2))\}$.

Case 6: $E = map_{\neq}[c, e; E_1, E_2]$. This case is similar to the previous case except that, instead of the formula $t(x_1, x_2) = ()$, where t is the translation of c in the calculus expressions, we have the formula $t(x_1, x_2) \neq ()$. □