Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 150 (2006) 37–54
www.elsevier.com/locate/entcs

A Calculus for Data Mapping

George H. L. Fletcher,[a,1] Catharine M. Wyss,[a,b,2]
Edward L. Robertson,[a,b,3] and Dirk Van Gucht[a,4]

[a] *Computer Science Department*
Indiana University, Bloomington, USA

[b] *School of Informatics*
Indiana University, Bloomington, USA

Abstract

Technologies for overcoming heterogeneities between autonomous data sources are key in the emerging networked world. In this paper we discuss the initial results of a formal investigation into the underpinnings of technologies for alleviating structural heterogeneity. At the core of structural heterogeneity is the *data mapping problem*: discovering effective mappings between structured representations of data. Automating the discovery of these mappings is one of the fundamental unsolved challenges for data interoperability, integration, and sharing. We introduce a novel data model and calculus for expressing data mappings between relational data sources, laying the ground for a better understanding of the data mapping problem. This research uncovers several new safety issues in data mapping languages. We discuss ongoing investigations of syntactic and semantic restrictions on the calculus to deal with these issues.

Keywords: Data Calculus, Data Mapping, Database Interoperability

1 Introduction

The emerging networked world promises new opportunities and possibilities for information dissemination, wide-scale collaboration, and knowledge construction. These opportunities will be fostered in large part by technologies which bring together autonomous data sources. Since these data sources were created

[1] gefletch@cs.indiana.edu
[2] cmw@cs.indiana.edu
[3] edrbtsn@cs.indiana.edu. Work supported by NSF grant IIS 82407.
[4] vgucht@cs.indiana.edu. Work supported by NSF grant IIS 82407.

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.11.033

and have evolved in isolation, they are each maintained differently according to local constraints and usage patterns. Consequently, facilitating technologies must bridge a wide variety of heterogeneities. These heterogeneities encompass differences at the system level, differences in the structuring of data, and semantic pluralism in the interpretation of data.

We are investigating an approach for overcoming *structural* heterogeneity between *relational* data sources [5]. Although research in databases has led to great practical successes in the storage and management of structured data, it has made limited progress in technologies for alleviating structural heterogeneity. At the heart of overcoming structural heterogeneity is the *data mapping problem*: automating the discovery of mappings between structured data sources [5]. These mappings are the basic "glue" for building information sharing systems [7], and automating their discovery is one of the fundamental unsolved challenges for data interoperability, integration, and sharing. This ubiquitous problem arises in almost any information system involving multiple structured data sources. Consequently, the problem has many traditional manifestations: schema matching [17] and schema mapping [15] in databases, ontology mapping on the Semantic Web [9], schema mediation in peer-to-peer systems [7], and matching of information models [13], to name a few. In general, the problem is considered to be "AI-complete" in the sense that it is as hard as any of the hardest problems of artificial intelligence [13].

Example 1.1 To illustrate the data mapping problem, consider the three databases containing student grade information in Figure 1. In this example, each database contains the same student information. As shown, there are many natural ways to organize even the simplest datasets such as these. For example, `G1` and `G2` maintain the student information in a single table, while `G3` contains a table for each assignment. Note that to move between these representations of student data, schema matchings and both data-data and data-metadata transformations must be performed. For example, mapping data from `G1` to `G2` involves promoting the values in attribute `Assignment` in `G1` to attribute names in `G2` and "matching" the `Name` and `Student` attributes. Also note that this mapping dynamically creates as many new attributes as there are values in `Assignment`. To move information from `G3` to `G1`, relation names must be demoted to data values. This will be our running example in the paper.

Any solution for overcoming relational structural heterogeneity must consider the full data mapping problem space for relational data sources. Both schema matchings ("traditional" metadata-metadata mappings between schema

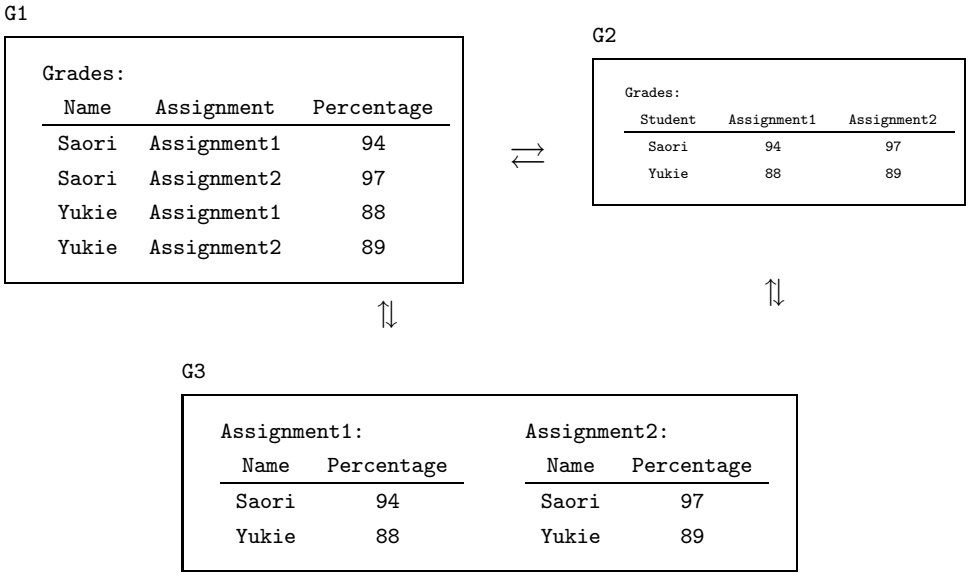

Fig. 1. Mappings between student grade representations.

elements [17]) and data-metadata/metadata-data mappings (where data elements in one structure serve as metadata components in the other, or vice versa [10,14]) must be expressible. It is important to note that consideration of the full mapping space blurs the distinction between schema matching and schema mapping [15], since data-metadata mapping encompasses schema matching as a special case. When metadata itself is seen as data, the entirety of schema matching and schema mapping is encompassed in *data mapping*.

Our investigation of the data mapping problem is part of the *Modular Integration of Queryable Information Sources* (MIQIS) project at Indiana University [22], a framework for data interoperability on the Semantic Web and in peer-to-peer data management systems [7]. Among the distinguishing features of MIQIS is a focus on the *modular* nature of information systems, encompassing XML, relational, text, and other data sources. The framework fully respects the autonomy of peers to manage locally their schemata and concepts. The two major components of MIQIS are a formal investigation of data interoperability and a practical implementation of modules driven by this investigation. In this paper we present initial results on a formal study of data interoperability for the the relational module of MIQIS.

1.1 Related Work

Overcoming structural heterogeneity is a long standing problem in database research. Our work in this paper is motivated by relational languages for database interoperability such as FISQL [21], data mapping solutions, and

recent work on the data exchange problem.

1.1.1 Languages for Relational Federations

It has been known for some time that first order logic (i.e., SQL) is not adequate for expressing many useful transformations for data interoperability [10,14]. This led to work over the last decade on relational languages for federated data models. Most influential for our work are Chen et al. [2] on higher order logic programming, Grant et al. [6], Krishnamurthy et al. [10], and Lakshmanan et al. [11] on multidatabase languages, Sattler et al. [19] on languages for database federations, and Jain et al. [8] on a general data model and languages for data-metadata querying. Our work complements and extends this line of research with a *logical* characterization of the full space of relational data mapping transformations.

1.1.2 Federated Interoperable SQL

Our work is a direct outgrowth of research on the relational metadata query language *Federated Interoperable SQL* (FISQL) of Wyss et al. [21]. This principled extension of SQL naturally expresses all of the database transformations in Fig. 1.

Example 1.2 To illustrate FISQL, consider the following query to restructure data in G1 into the format of G2:

```
SELECT G.Name AS ‘‘STUDENT’’,
G.Percentage ON G.Assignment INTO ‘‘G2’’
FROM G1.Grades AS G
```

This query language is *transformationally complete* in the sense that it precisely expresses all natural relational data-metadata transformations [21]. FISQL also has an equivalent query algebra, FIRA [21]. This algebra has, in addition to the normal relational operators, simple operators for data-metadata querying, for example: $\uparrow$ to promote metadata, $\downarrow$ to demote metadata, and $\rightarrow$ to dereference data. FISQL will serve as our benchmark for relational interoperability; a successful relational data mapping language should have at least the expressivity of FISQL.

1.1.3 Data Mapping Solutions

The formal investigation initiated in this paper is concurrent with a successful implementation of a data mapping solution [5] which uses the FIRA operators and a restricted merge operator. This solution views data mapping as a search problem, a novel perspective that allows us to leverage traditional techniques from artificial intelligence. The present study lays the groundwork for a formal investigation of data mapping solutions.

1.1.4 The Data Exchange Problem

A problem closely related to the data mapping problem is the *data exchange problem*, presented recently by Fagin et al. [3] to formalize aspects of the Clio schema mapping system developed at IBM [15]. Briefly, the data exchange problem is as follows: given a source schema S, target schema T, source instance I, and a set $\Sigma_{S,T}$ of source-to-target dependencies in some logical formalism, find a target instance J that satisfies $\Sigma_{S,T}$ [3]. Fagin et al. have characterized solutions to the data exchange problem and have explored query answering in data exchange settings [3]. A limitation of these results is a restriction of the logical formalism for expressing $\Sigma_{S,T}$ to fragments of first order logic which do not always adequately express naturally occurring data mappings. Furthermore, in data exchange it is assumed (1) that these dependencies are given as input and (2) the target schema T is fixed. In the data *mapping* problem we are concerned precisely with discovering meaningful source to target constraints, given S, T, and perhaps (I, J) as input where the target schema T is potentially dynamic [5], as we saw in the mapping from `G1` to `G2`, which creates as many new assignment names in `G2` as there are `Assignment` values in `G1`.

1.2 Contributions and Outline of Paper

In this paper we introduce a data mapping calculus for reasoning about the data mapping problem. This language is a descendant of the Uniform Calculus developed by Jain et al. [8], specialized to investigate relational data mapping; it is a novel development of Jain's language in that it clearly captures a very minimal extension of the standard relational model for structural transformations for database interoperability. This simple logical formalism expresses the transformations for data mapping, and complements our work on automating data mapping solutions [5]. A primary motivation for the calculus is that it facilitates an investigation into the complexity and/or decidability of aspects of the data mapping problem.

In this paper, our contributions are as follows:

- A novel data model for data mapping is introduced in Section 2.1.
- The data mapping calculus is presented in Sections 2.2 and 2.3.
- Several important safety issues which arise in structural data mapping, such as schema safety and functionality, are introduced in Sections 2.4-2.6 In these sections we also discuss ongoing work on characterizing an appropriate safe fragment of the calculus which is equivalent to tractable query languages for relational data interoperability.

We give concluding remarks and indications for future work in Section 3.

2 Data Mapping: Data Model and Calculus

This section presents a new data model and declarative *data mapping calculus* (DMC) to address the data mapping problem. Throughout the presentation, we illustrate the effectiveness of DMC with several examples. After presenting the data model underlying DMC in Section 2.1, we present DMC syntax in Section 2.2 and semantics in Section 2.3. It turns out that unrestricted DMC is too powerful, and the natural semantics for DMC is problematic. In Sections 2.4-2.6, we illustrate the problems that occur and suggest ways of refining DMC to a fragment which is expressive enough for data mapping and yet is restricted enough to admit equivalence with existing efficient languages for data mapping.

2.1 Data Model

This data model underpinning DMC is closely related to both the Federated Data Model of Wyss et al. [21] and to the Uniform Data Model of Jain et al. [8]. Similarly to the relational data model, databases are finite structured objects over a countably infinite set **DOM** of atoms (e.g., alphanumeric strings). However, in the DMC data model, tuples are simply finite sets of ordered pairs tagged explicitly with a relation name. A database is then a finite set of such tuples. Formally, the model is defined as follows. Let $\mathcal{P}^{\mathtt{fin}}(X)$ denote the set of all finite subsets of a given set X.

- A *tuple*, T, is an ordered pair

$$\langle \mathtt{r}, \{\langle \mathtt{a}_1, \mathtt{v}_1 \rangle, \ldots, \langle \mathtt{a}_n, \mathtt{v}_n \rangle\} \rangle$$

  where $\mathtt{r}, \mathtt{a}_1, ..., \mathtt{a}_n, \mathtt{v}_1, ..., \mathtt{v}_n \in \mathbf{DOM}$ and $n \geqslant 0$. We will use the notation $name(T)$ to denote $\mathtt{r}$ and $body(T)$ to denote the set of attribute-value pairs. Let $\mathscr{T}$ denote the set of all possible tuples over **DOM**.
- A *database*, D, is an ordered pair

$$\langle \mathtt{d}, \mathbf{T} \rangle \in (\mathbf{DOM} \times \mathcal{P}^{\mathtt{fin}}(\mathscr{T})).$$

  As shorthand, we write $name(D)$ and $body(D)$ as above to denote the first and second elements of D, respectively. Let $\mathscr{D}$ denote the set of all databases over **DOM**.
- A *relation* $\mathtt{r}$ in a database D is the set of tuples $\{t \in body(D) \mid name(t) = \mathtt{r}\}$. Note that every database D has a finite set of relations which partition $body(D)$.

- A *federation* $\bar{D}$ is a finite set of databases:

$$\bar{D} = \{D_1, \ldots, D_n\} \in \mathcal{P}^{\mathtt{fin}}(\mathscr{D})$$

where $name(D_i) \neq name(D_j)$, for $i \neq j$. Let $\mathbf{ADOM}(\bar{D})$ denote the set of all atoms appearing in $\bar{D}$. Also, let $\mathscr{T}_{\bar{D}}$ denote the finite set of all possible tuples over $\mathbf{ADOM}(\bar{D})$.

Example 2.1 In this data model, the representation of the federation of Figure 1 is the set of databases $\{\mathtt{G1}, \mathtt{G2}, \mathtt{G3}\}$. Furthermore, `G2` (for example) has the following representation:

$$\langle \mathtt{G2},\ \{\langle \mathtt{Grades}, \{\langle \mathtt{Student}, \mathtt{Saori}\rangle, \langle \mathtt{Assignment1}, 94\rangle, \langle \mathtt{Assignment2}, 97\rangle\}\rangle,$$
$$\langle \mathtt{Grades}, \{\langle \mathtt{Student}, \mathtt{Yukie}\rangle, \langle \mathtt{Assignment1}, 88\rangle, \langle \mathtt{Assignment2}, 89\rangle\}\rangle\}\rangle.$$

2.1.1 Comments on the Data Model

Note that our data model is a proper extension of the relational data model in that it allows *non-functional* tuples, i.e., tuples that have more than one pair in their body with the same first element. This flexibility is necessary to accommodate natural transformations which occur during data mapping; enforcing strict functionality (i.e., the relational data model) is an issue which we discuss at length in this paper and is also the focus of ongoing research. This phenomenon imitates "nests" (in the sense of Nested Relational Model [1]) of depth 1 and thus our data model has affinities with the data model underlying the Pack-Unpack Calculus [16].

Another interesting feature is that the data model naturally supports missing information. It is common that during data mapping, a query may dynamically allow input data to become output metadata. Special "null" values have typically been introduced to indicate missing or inapplicable information. DMC data model cleanly and simply avoids problems associated with this approach by viewing a null value as *absence*. As an example, if a single student in `G2` has the attribute-value pair $\langle \mathtt{extra\text{-}credit}, 83\rangle$, it is not necessary to explicitly have null values for the attribute `extra-credit` in the tuples of all other students.

These flexible features of the data model render it particularly suitable for use in dynamic data mapping. We now turn to a presentation of the syntax of DMC.

2.2 Calculus Syntax

2.2.1 Terms

We assume the following basic building blocks of the language:

- A countably infinite set of *constants* $\mathcal{C} = \{c_1, c_2, \ldots\}$,
- A countably infinite set of *domain variables* $\mathcal{DV} = \{x_1, x_2, \ldots\}$, and
- A countably infinite set of *tuple variables* $\mathcal{TV} = \{t_1, t_2, \ldots\}$.

Furthermore, we assume that $\mathcal{C}$, $\mathcal{DV}$, and $\mathcal{TV}$ are pairwise disjoint.

The set of DMC terms $\mathcal{T}$ is built up from constants and variables as follows.

- If $c \in \mathcal{C}$, then $c \in \mathcal{T}$.
- If $x \in \mathcal{DV}$, then $x \in \mathcal{T}$.
- If $t \in \mathcal{TV}$ and $c \in \mathcal{C}$, then $t.c \in \mathcal{T}$.
- If $t \in \mathcal{TV}$ and $x \in \mathcal{DV}$, then $t.x \in \mathcal{T}$.

Intuitively, the syntax $t.x$ represents the set of values for tuple t on attribute x.

2.2.2 Formulas

Formulas in DMC are built from terms as follows.

- If $\alpha, \beta \in \mathcal{T}$, then $\alpha = \beta \in$ DMC.
- If $\varphi, \psi \in$ DMC, then
  - $\neg\varphi \in$ DMC
  - $\varphi \vee \psi \in$ DMC
  - $\varphi \wedge \psi \in$ DMC
- If $t \in \mathcal{TV}, c \in \mathcal{C}, x \in \mathcal{DV}$, and $\varphi \in$ DMC, then
  - $(\exists x)\varphi \ \in$ DMC
  - $(\exists x : t \in c)\varphi \ \in$ DMC

Note that (1) the syntax $x : t$ emphasizes that x is the relation name coupled with tuple t, (2) the database name c is a *constant*, and (3) we will use $(\forall \ldots)\varphi$ as shorthand for $\neg(\exists \ldots)\neg\varphi$, and $\varphi \rightarrow \psi$ as shorthand for $\neg\varphi \vee \psi$, as usual.

2.2.3 Queries

A DMC *query* has the form

$$\{x : t \mid \varphi(x, t)\},$$

where $x \in \mathcal{DV}$ and $t \in \mathcal{TV}$ are the only free variables in $\varphi \in$ DMC.

2.2.4 Examples

A few examples will illustrate the expressive power of this syntax.

Example 2.2 We begin with a standard relational query on database G1 that returns all student names appearing in the Grades relations, placing them in an output relation named StudentNames:

$$\{x : t \mid x = \texttt{StudentNames} \wedge (\exists r : s \in \texttt{G1})\ (r = \texttt{Grades} \wedge t.\texttt{Name} = s.\texttt{Name})\}.$$

Example 2.3 Next, consider a query mapping G2 to G3. This query must promote attribute names to relation names, which can be done as follows:

$$\{x : t \mid (\exists r : s \in \texttt{G2})(\exists a)(\exists v)(r = \texttt{Grades} \wedge a \neq \texttt{Student} \wedge s.a = v \wedge x = a \wedge t.\texttt{Name} = s.\texttt{Student} \wedge t.\texttt{Percentage} = v)\}.$$

Example 2.4 Finally, a query mapping G2 to G1 must demote attribute names to data values:

$$\{x : t \mid x = \texttt{Grades} \wedge (\exists r : s \in \texttt{G2})(\exists a)(\exists v)\ (r = \texttt{Grades} \wedge s.a = v \wedge a \neq \texttt{Student} \wedge t.\texttt{Name} = s.\texttt{Student} \wedge t.\texttt{Assignment} = a \wedge t.\texttt{Percentage} = v)\}.$$

Note that DMC can cleanly express all of the mappings among databases in Fig. 1. Examples of other transformations are given in subsequent sections.

2.3 Calculus Semantics

DMC has an *active domain* semantics, a natural choice which avoids problems with domain safety of queries [1]. Under this semantics, quantifiers only range over $\mathbf{ADOM}(\bar{D})$, the active domain of an input federation $\bar{D}$. A formula $\varphi \in$ DMC is interpreted under a valuation. Define a *valuation* with respect to a federation $\bar{D}$ to be a function of one of the following types:

$$\nu^{\mathcal{C}}_{\bar{D}} : \mathcal{C} \to \mathbf{ADOM}(\bar{D})$$
$$\nu^{\mathcal{DV}}_{\bar{D}} : \mathcal{DV} \to \mathbf{ADOM}(\bar{D})$$
$$\nu^{\mathcal{TV}}_{\bar{D}} : \mathcal{TV} \to \mathscr{T}_{\bar{D}}.$$

Since $\mathcal{C}$, $\mathcal{DV}$, and $\mathcal{TV}$ are pairwise disjoint, we can use $\nu_{\bar{D}}$ to refer to any of these functions without confusion (and just ν when the context is clear).

2.3.1 Terms

The *meaning* of a DMC term α with respect to a valuation ν (denoted $[\![\alpha]\!]^{\nu}$) is given as follows:

- For $c \in \mathcal{C}$, $[\![c]\!]^{\nu} = \{\nu(c)\}$, i.e., the singleton value of constant c.
- For $x \in \mathcal{DV}$, $[\![x]\!]^{\nu} = \{\nu(x)\}$, i.e., the singleton value of domain variable x.

- For $t \in \mathcal{TV}$ and $c \in \mathcal{C}$, $[\![t.c]\!]^\nu = \{y \in \mathbf{DOM} \mid \langle \nu(c), y\rangle \in body(\nu(t))\}$, i.e., the set of values of tuple t on attribute name $\nu(c)$.
- For $t \in \mathcal{TV}$ and $x \in \mathcal{DV}$, $[\![t.x]\!]^\nu = \{y \in \mathbf{DOM} \mid \langle \nu(x), y\rangle \in body(\nu(t))\}$, i.e., the set of values of tuple t on attribute name $\nu(x)$.

2.3.2 Formulas

Given a formula $\varphi \in \mathsf{DMC}$ and a valuation $\nu_{\bar{D}}$, we can define a standard notion of *satisfaction* inductively based on the syntactic form of φ as follows. We write "$\nu \models \varphi$" to mean ν satisfies φ.

$\nu \models \alpha = \beta$	iff	$[\![\alpha]\!]^\nu \cap [\![\beta]\!]^\nu \neq \varnothing$
$\nu \models \neg\varphi$	iff	it is not the case that $\nu \models \varphi$
$\nu \models \varphi \vee \psi$	iff	$\nu \models \varphi$ or $\nu \models \psi$
$\nu \models \varphi \wedge \psi$	iff	$\nu \models \varphi$ and $\nu \models \psi$
$\nu \models (\exists x)\varphi$	iff	$\exists a \in \mathbf{ADOM}(\bar{D})$ such that $\nu[x \leftarrow a] \models \varphi$
$\nu \models (\exists x : t \in c)\varphi$	iff	$\exists D \in \bar{D}\ \exists T \in body(D), name(D) = \nu(c)$ and $\nu[x \leftarrow name(T), t \leftarrow T] \models \varphi$

Note that the semantics of equality in DMC is *set based*. This is because tuples are not necessarily functional, as noted above. While it may seem odd to define equality of terms as non-empty intersection, this is a direct reflection of the fact that our data model can include non-functional tuples. One consequence of this is that an atom x is equated with the singleton set $\{x\}$. This approach is becoming more accepted in the literature recently, since the same phenomenon occurs with *itemsets* when defining data models and query languages for XML data [4].

2.3.3 Queries

Given a DMC query, Q, and input federation $\bar{D}$, we can define the semantics of Q with respect to $\bar{D}$, denoted $[\![Q]\!]^{\bar{D}}$, as follows.

$$[\![\{x : t \mid \varphi(x,t)\}]\!]^{\bar{D}} =$$
$$\{\ [\![t]\!]^\nu \mid \nu \text{ is some valuation over } \bar{D} \text{ such that } \nu \models \varphi(name(\ [\![t]\!]^\nu),\ [\![t]\!]^\nu)\}.$$

This semantics is a natural extension of the relational tuple calculus semantics [1]. This semantics gives a unique, well-defined output database as the result

of any DMC query.

However, there are several inadequacies that are associated with the natural semantics. These arise because (i) the output of a DMC query is dynamic and thus cannot be explicitly typed, leading to extraneous tuples in the output, (ii) the DMC data model allows non-functional tuples, and (iii) the calculus is too powerful to be used as a practical query language. In the next sections we illustrate what can go wrong and provide successive restrictions of DMC. Note that a consideration in these restrictions is to develop a fragment of DMC that is equivalent to transformationally complete relational data mapping languages such as FISQL [21].

2.4 Schema Safety

One foremost consideration is that the natural semantics of a DMC query forces unintuitive results since DMC output schemata are dynamic. This means that *any* tuples in $\mathscr{T}_{\bar{D}}$ which satisfy φ appear in the output, not just those of "appropriate" schema.

Example 2.5 Consider input federation $\bar{D} := \{\langle \mathtt{a}, \{\langle \mathtt{a}, \{\langle \mathtt{b}, \mathtt{b}\rangle\}\rangle\}\rangle\}$ and the query $Q := \{x : t \mid x = \mathtt{a} \wedge (\exists r : s \in \mathtt{a})\ (r = \mathtt{a} \wedge t.\mathtt{b} = s.\mathtt{b})\}$. In this case, we have:

$$\begin{aligned}[\![Q]\!]^{\bar{D}} = \{\ & \langle \mathtt{a}, \{\langle \mathtt{b}, \mathtt{b}\rangle\}\rangle, \\ & \langle \mathtt{a}, \{\langle \mathtt{b}, \mathtt{b}\rangle, \langle \mathtt{a}, \mathtt{a}\rangle\}\rangle, \\ & \langle \mathtt{a}, \{\langle \mathtt{b}, \mathtt{b}\rangle, \langle \mathtt{a}, \mathtt{b}\rangle\}\rangle, \\ & \langle \mathtt{a}, \{\langle \mathtt{b}, \mathtt{b}\rangle, \langle \mathtt{b}, \mathtt{a}\rangle\}\rangle, \\ & \langle \mathtt{a}, \{\langle \mathtt{b}, \mathtt{b}\rangle, \langle \mathtt{a}, \mathtt{a}\rangle, \langle \mathtt{a}, \mathtt{b}\rangle\}\rangle, \ldots, \\ & \langle \mathtt{a}, \{\langle \mathtt{b}, \mathtt{b}\rangle, \langle \mathtt{a}, \mathtt{a}\rangle, \langle \mathtt{a}, \mathtt{b}\rangle, \langle \mathtt{b}, \mathtt{a}\rangle\}\rangle\}.\end{aligned}$$

Query Q on $\bar{D}$ stipulates that the output database must contain the relation a with a tuple containing attribute-value pair $\langle \mathtt{b}, \mathtt{b}\rangle$. *Every* tuple in $\mathscr{T}_{\bar{D}}$ that has name a and contains $\langle \mathtt{b}, \mathtt{b}\rangle$ will be included in the result, clearly an unintended consequence of the natural semantics in the face of dynamic output schemas.

In this sense, the natural semantics for DMC queries is *schema unsafe*, i.e., query results potentially contain unintuitive, unexpected tuples. Our first restriction on DMC is thus to provide *schema safety*. We can do this either *syntactically* (Section 2.4.1) or *semantically* (Section 2.4.2).

2.4.1 Syntactic Schema Safety

Syntactically, it is a simple matter to restrict DMC queries to schema-safe queries. All that is needed is to include in every query a parameterized formula $\sigma \in$ DMC that states explicitly that no extraneous attribute-value pairs exist

in the output. We illustrate the process on the first two queries from Section 2.2.4.

Example 2.6 First, we rewrite the simple query from Example 2.2 as follows.

$$\{x : t \mid x = \texttt{StudentNames} \wedge (\exists r : s \in \texttt{G1})\ (r = \texttt{Grades} \wedge t.\texttt{Name} = s.\texttt{Name} \wedge \sigma(t))\},$$

where $\sigma(t) := (\forall u)\ u \neq \texttt{Name} \rightarrow \neg(\exists v)\ t.u = v$.

Example 2.7 Similarly, for the query mapping G2 to G3 from Example 2.3, $\sigma(t) := (\forall u)\ (u \neq \texttt{Name} \wedge u \neq \texttt{Percentage}) \rightarrow \neg(\exists v)\ t.u = v$.

For queries having dynamic output, such as those in Examples 2.8 or 2.12 (below), it is slightly trickier to compose the schema-safe condition σ, but the method still works (by quantifying over the "active schema" of the output database). The details are omitted due to space limitations.

2.4.2 Semantic Schema Safety

Schema safety can also be enforced by adopting a stricter query semantics. Intuitively, we restrict the query result to a naturally defined minimal answer. The idea is to only include in the query result the "smallest" tuples on attribute names mentioned explicitly in the query. This semantics follows similarly to the standard model-theoretic semantics for the relational calculus. We will establish in future work that this minimal-answer semantics disallows spurious attribute-value pairs and is well defined.

2.5 Functionality

A similar oddity is that our data model and the natural query semantics allow non-functional tuples. It should be emphasized that this is not a problem, but rather a feature of the model and semantics. However, it prevents equivalence of DMC with *relational* query languages such as FISQL, which assume tuples are functions [21].

Example 2.8 Suppose that exam grades are added to the Grades relation in G2:

Student	Assignment1	Assignment2	Exam1
Saori	94	97	97
Yukie	88	89	88

Note that Saori made a 97 on both Assignment 2 and Exam 1, and Yukie made an 88 on both Assignment 1 and Exam 1. We can promote grade data

to attribute names with the following query:

$$\{x : t \mid x = \texttt{Grades} \wedge (\exists r : s \in \texttt{G2})(\exists a)(\exists v)\ (r = \texttt{Grades} \\ \wedge\ t.\texttt{Student} = s.\texttt{Student} \wedge a \neq \texttt{Student} \wedge v = s.a \wedge t.v = a \wedge \sigma(t,v))\},$$

where $\sigma(t,v) := (\forall p)\ (p \neq \texttt{Student} \wedge p \neq v) \rightarrow \neg(\exists w)t.p = w$. In this case we have the output database includes the following tuples (among others):

$$\{\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 94, \texttt{Assignment1}\rangle\}\rangle, \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 97, \texttt{Assignment2}\rangle\}\rangle, \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 97, \texttt{Exam1}\rangle\}\rangle, \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 94, \texttt{Assignment1}\rangle, \langle 97, \texttt{Assignment2}\rangle, \langle 97, \texttt{Exam1}\rangle\}\rangle, \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 97, \texttt{Assignment2}\rangle, \langle 97, \texttt{Exam1}\rangle\}\rangle, \ldots\}$$

Note that the output is schema-safe in the sense of Section 2.4, but includes non-functional tuples (e.g., the last two tuples) as well as intended tuples, since these satisfy the query conditions. This indicates the need for a "functional" restriction on DMC queries, in addition to the schema-safety restriction, if equivalence with relational languages is desired. Also note that the output will contain "merged" versions of the tuples as well as "unmerged" versions (e.g., the fourth tuple is a "merge" of the first three tuples on the `Student` attribute). We discuss this issue below in Section 2.6.

As with schema-safety, functionality can be strictly enforced either *syntactically* (Section 2.5.1) or *semantically* (Section 2.5.2). However, the functionality that is achieved disallows many natural queries that are possible in FISQL. We discuss a third approach to functionality in Section 2.5.3.

2.5.1 Syntactic Functionality

Functionality can be enforced syntactically by adding a formula θ to DMC queries that insists the output is functional. We illustrate the method by revisiting the query in Example 2.8.

Example 2.9 We can rewrite the query in Example 2.8 as follows.

$$\{x : t \mid x = \texttt{Grades} \wedge (\exists r : s \in \texttt{G2})(\exists a)(\exists v)\ (r = \texttt{Grades} \wedge \\ t.\texttt{Student} = s.\texttt{Student} \wedge a \neq \texttt{Student} \wedge s.a = v \wedge t.v = a \wedge \sigma(t,v)) \wedge \theta(t)\},$$

where $\theta(t) := \neg((\exists a)(\exists v)(\exists u)\ (t.a = v \wedge t.a = u \wedge u \neq v))$. This will force the following output (which is reasonably in conformance to our expectations):

$$\{\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 94, \texttt{Assignment1}\rangle\}\rangle, \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 97, \texttt{Assignment2}\rangle\}\rangle, \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 97, \texttt{Exam1}\rangle\}\rangle, \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 94, \texttt{Assignment1}\rangle, \langle 97, \texttt{Exam1}\rangle\}\rangle \\ \langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle 94, \texttt{Assignment1}\rangle, \langle 97, \texttt{Assignment2}\rangle\}\rangle, \ldots\}$$

Note, however, that both merged and unmerged tuples remain.

2.5.2 Semantic Functionality

Functionality can also be achieved by stratifying tuple variables into those allowed to be non-functional and those that may only range over functional tuples, denoted by $\hat{t}$. Under this new stratification, a query has the form $\{x : \hat{t} \mid \varphi(x, \hat{t})\}$ and the semantics is developed appropriately. Note that if arbitrary tuple variables are allowed in φ (as opposed to just functional ones), the language is strictly more powerful. We defer the details of this alternative to a longer paper.

2.5.3 Operational Functionality

The syntactic and semantic approaches to functionality run into difficulty when several data-to-metadata promotions occur within a single DMC query. These promotions may conflict with each other, resulting in multiple "order of execution" possibilities reflected in the output database. In this case, a strict approach to functionality would force the query answer to be empty. The FISQL query language, however, does return functional tuples in this case since it has a natural operational semantics for generating query results that follows the syntax of a query.

Example 2.10 Consider the following `Grades` relation in a database `G1'`:

Name	Assignment	AGrade	Exam	EGrade
Sanae	1	100	1	98

Suppose now that both `Assignment` and `Exam` values are promoted to attribute names with the following query:

$$\{x : t \mid x = \texttt{Grades} \wedge (\exists r : s \in \texttt{G1'})(\exists y_1)(\exists y_2)\ (r = \texttt{Grades} \wedge t.\texttt{Name} = s.\texttt{Name} \wedge y_1 = s.\texttt{Assignment} \wedge y_2 = s.\texttt{Exam} \wedge t.y_1 = s.\texttt{AGrade} \wedge t.y_2 = s.\texttt{EGrade} \wedge \sigma(t)) \wedge \theta(t)\}$$

The result of this query will be the empty set, an undesired result. If we remove the $\theta(t)$ clause, the result will reflect the two possible values for attribute `1`:

$$\{\langle \texttt{Grades}, \{\langle \texttt{Name}, \texttt{Sanae}\rangle, \langle \texttt{1}, \texttt{100}\rangle, \langle \texttt{1}, \texttt{98}\rangle\}\rangle\}$$

Clearly having two different values for attribute `1` is also an undesired result. For such queries, we can explicitly specify an "order of execution" with a parameterized syntactic clause, $\omega(t)$. For equivalence with FISQL, it is important that the general form of $\omega(t)$ be compatible with the order of operations of the language.

Example 2.11 In the previous example,

$$\omega(t) := (y_1 = y_2) \rightarrow (t.y_1 = s.\texttt{AGrade})$$

and a deterministic, functional output is achieved:

$$\{\langle \texttt{Grades}, \{\langle \texttt{Name}, \texttt{Sanae}\rangle, \langle 1, 100\rangle\}\rangle\}$$

With this modification, we prevent the "indeterminacy" in the natural DMC semantics that arises from having several distinct canonical relations (all legitimately satisfying the query) reflected in the output database. This operational approach to functionality naturally captures the behavior of FISQL.

2.6 Complexity Reduction

As shown in Example 2.8, the semantics of a query can include both "merged" and "unmerged" output. This is an artifact of promoting data to attribute names. Commonly, the merged output is desired. The OLAP operation, PIVOT, for example, expects the pivoting columns to form a key so that a single, well-defined merged relation can be output. In general, the problem of merging tuples in dynamically typed output is known to be NP-complete [20].

Example 2.12 Consider a query mapping G1 to G2:

$$\{x : t \mid x = \texttt{Grades} \wedge (\exists r : s \in \texttt{G1})(\exists y)\ (r = \texttt{Grades} \wedge t.\texttt{Student} = s.\texttt{Name} \wedge y = s.\texttt{Assignment} \wedge t.y = s.\texttt{Percentage})\}$$

The output of this query is as follows:

$$\begin{aligned}\{&\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle \texttt{Assignment1}, 94\rangle\}\rangle,\\ &\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle \texttt{Assignment2}, 97\rangle\}\rangle,\\ &\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Yukie}\rangle, \langle \texttt{Assignment1}, 88\rangle\}\rangle,\\ &\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Yukie}\rangle, \langle \texttt{Assignment2}, 89\rangle\}\rangle,\\ &\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Saori}\rangle, \langle \texttt{Assignment1}, 94\rangle, \langle \texttt{Assignment2}, 97\rangle\}\rangle,\\ &\langle \texttt{Grades}, \{\langle \texttt{Student}, \texttt{Yukie}\rangle, \langle \texttt{Assignment1}, 88\rangle, \langle \texttt{Assignment2}, 89\rangle\}\rangle\}\end{aligned}$$

We would expect a Student value to be associated with all of its Assignment values "merged" into a single tuple, as in the last two tuples of the query result. In the case of the input federation in Figure 1, the following query accomplishes this:

$$\begin{aligned}\{x : t \mid\ & x = \texttt{Grades} \wedge (\exists r : s \in \texttt{G1})\ (r = \texttt{Grades} \wedge t.\texttt{Student} = s.\texttt{Name}\\ & \wedge (\forall r' : s' \in \texttt{G1})(\forall a)\ (r' = r \wedge t.\texttt{Student} = s'.\texttt{Name})\\ & \rightarrow (a = s'.\texttt{Assignment} \wedge t.a = s'.\texttt{Percentage}))\}\end{aligned}$$

It is important to note that there is a single merge for this particular input federation, but in general this will *not* be the case. The DMC semantics for such a query will reflect all merges that satisfy the query conditions.

In fact, allowing queries such as the second one of Example 2.12 enables us to express the *generalized join* [18] in DMC, an operation that is known to be PSPACE-complete [18]. In contrast, our paradigmatic language for relational data mapping FISQL is in LOGSPACE. Thus, to achieve equivalence with FISQL, we need to further restrict DMC to rule out such queries. This can be done syntactically by restricting the scope of the free variables in the query body to a negation-free *grounding* formula, γ. The general form of such queries is:

$$\{x : t \mid (\exists r_1 : s_1 \in c_1) \cdots (\exists r_n : s_n \in c_n)(\exists y_1) \cdots (\exists y_k) \\ \gamma(x, t, r_1, ..., r_n, s_1, ...s_n, y_1, ...y_k) \wedge \varphi(r_1, ...r_n, s_1, ...s_n, y_1, ...y_k)\},$$

where γ is negation-free and neither x nor t appears free in φ. This syntactic form of DMC queries directly reflects the scoping of a FISQL query. (A similar scoping occurs in SQL.)

We denote the query sub-language of DMC having appropriate σ (schema-safety), γ (grounding), and ω (ordering) clauses by the name *Federated Interoperable Relational Calculus*, or FIRC. One goal of future work is to show that under a natural mapping, this fragment of DMC gives us exactly the expressivity of FISQL.

3 Conclusions and Future Work

In this paper we presented ongoing work on a formal underpinning for investigations into the data mapping problem. After discussing related works, we developed DMC, a calculus for data mapping, and illustrated its appropriateness for expressing relational data mapping transformations. We then presented a natural semantics for DMC queries and introduced several thorny safety issues which arise with this semantics. Finally, we discussed syntactic and semantic safety restrictions with an eye towards developing a fragment of the language that is equivalent with known practical metadata query languages.

Currently, we are working out the full details of these issues and the restrictions on DMC which address them. In addition, with DMC in hand we are now in a position to clearly formalize the data mapping problem in terms of DMC decision problems. Our next major step in this research is to formally state these problems and establish their complexity and/or decidability.

References

[1] Abiteboul, Serge, Richard Hull, and Victor Vianu. *Foundations of Databases*, Addison-Wesley, Reading, MA, 1995.

[2] Chen, Weidong, Michael Kifer, and David Scott Warren. HILOG: A Foundation for Higher-Order Logic Programming. *J. Logic Prog.* 15(3): 187-230, 1993.

[3] Fagin, R., P.G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *Proc. ACM PODS*, pp. 90-101, San Diego, CA, 2003.

[4] Fernández, M.F., et al. *XQuery 1.0 and XPath 2.0 Data Model*. W3C, Feb. 2005.

[5] Fletcher, George H.L. and Catharine M. Wyss. Mapping Between Data Sources on the Web. *Proc. IEEE ICDE Workshop WIRI*, Tokyo, Japan, 2005.

[6] Grant, J., W. Litwin, N. Roussopoulos, and T. Sellis. Query Languages for Relational Multidatabases. *VLDB Journal* 2(2):153-171, 1993.

[7] Halevy, A. Y., Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. *Proc. IEEE ICDE*, pp. 505-516, Bangalore, India, 2003.

[8] Jain, M., A. Mendhekar, and D. Van Gucht. A Uniform Data Model for Relational Data and Meta-Data Query Processing. *Proc. COMAD*, Pune, India, 1995.

[9] Kalfoglou, Yannis and Marco Schorlemmer. Ontology Mapping: the State of the Art. *The Knowledge Engineering Review* 18(1):1-31, 2003.

[10] Krishnamurthy, R., W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. *Proc. ACM SIGMOD*, pp. 40-49, Denver, CO, USA, 1991.

[11] Lakshmanan, L.V.S., F. Sadri, and I.N. Subramanian. Logic and Algebraic Languages for Interoperability in Multidatabase Systems. *J. Logic Prog.* 33(2):101-149, 1997.

[12] Lenzerini, Maurizio. Data Integration: A Theoretical Perspective. *Proc. ACM PODS*, pp. 233-246, Madison, WI, USA, 2002.

[13] Melnik, Sergey. *Generic Model Management: Concepts and Algorithms, LNCS 2967*. Springer Verlag, Berlin, 2004.

[14] Miller, Renée J. Using Schematically Heterogeneous Structures. *Proc. ACM SIGMOD Conf. on Management of Data*, pp. 189-200, Seattle, WA, USA, 1998.

[15] Miller, Renée J., Laura M. Haas, and Mauricio A. Hernández. Schema Mapping as Query Discovery. *Proc. VLDB Conf.*, pp. 77-88, Cairo, Egypt, 2000.

[16] Özsoyoğlu, G., Z.M. Özsoyoğlu, and V. Matos. Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Function. *ACM Transactions on Database Systems*, 12(4):566-592, 1987.

[17] Rahm, Erhard and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal* 10(4):334-350, 2001.

[18] Rood, C.M., D. Van Gucht, and F. I. Wyss. MD-SQL: A Language for Meta-Data Queries over Relational Databases. Indiana Univ. CS Dept. TR528, July 1999.

[19] Sattler, Kai-Uwe, Stefan Conrad, and Gunter Saake. Interactive Example-Driven Integration and Reconciliation for Accessing Database Federations. *Information Systems* 28(5):393-414, July 2003.

[20] Wyss, Catharine M. and Edward Robertson. Optimal Tuple Merge is NP-Complete. Indiana Univ. Computer Science Dept. TR599, July 2004.

[21] Wyss, Catharine M. and Edward Robertson. Relational Languages for Metadata Integration. *ACM Transactions on Database Systems*, to appear June 2005.

[22] Wyss, Catharine M., George H. L. Fletcher, Fulya Erdinc, and Jeremy T. Engle. MIQIS: Modular Integration of Queryable Information Systems. *Proc. VLDB Workshop IIWeb*, pp. 136-140, Toronto, Canada, 2004.