What Can Programming Languages Say About Data Exchange?

Michael Johnson
Macquarie University
Michael.Johnson@mq.edu.au

Jorge Pérez
DCC, Universidad de Chile
jperez@dcc.uchile.cl

James F. Terwilliger
Microsoft Research
james.terwilliger@microsoft.com

ABSTRACT

Data Exchange, defined generally, is the process of taking data structured under one schema and transforming it into data structured under another independent schema. This process is present in enough scenarios both theoretical and practical that it has been addressed in many different ways. Most prominent amongst the solutions to the problem is that proposed by database literature, in which one constructs schema mappings, using (a subset of) first-order predicate calculus, to establish the high-level relationship among the database schemas participating in the exchange. From a schema mapping an executable process is derived to perform the exchange. This line of research has made significant progress and come to impressive findings, but has some theoretical and practical shortcomings as well. For instance, there are theoretical limitations as to how to compose or invert such mappings in a complete and unique way, which is a barrier to making such mappings bidirectional. It is possible to address some of these shortcomings by looking to solutions from a different discipline—a construct from the programming language literature called a *lens*—that addresses similar problems from a different perspective. By combining solutions from these two disciplines, one ends up with a new direction of research as well as a result that might be greater than the sum of its parts.

Keywords

Bidirectional Transformations, Data Exchange, Lenses

1. INTRODUCTION

Data Exchange—the process of taking data in one structure and processing it into another—is a problem that is widely seen across nearly all fields of computer science. From databases (moving data instances between schemas) to visualization (moving graphics between formats) to software engineering (moving software artifacts between models), many disciplines face a familiar problem, where given source and target descriptions of data:

- How does one specify a transformation of source instances into target instances,
- execute that transformation, and
- demonstrate that the transformation has been done as faithfully as possible?

Even simple data exchange problems can prove to be difficult. For instance, consider a trivial example of mapping data from a schema $\texttt{Person}_1(\mathit{Id}, \mathit{Name}, \mathit{Age}, \mathit{City})$ to another schema $\texttt{Person}_2(\mathit{Id}, \mathit{Name}, \mathit{Salary}, \mathit{ZipCode})$. Some aspects of this transformation are clear, such as that every instance of $\texttt{Person}_1$ should correspond to an instance of $\texttt{Person}_2$ with the same name and identifier post-transformation. However:

- How does one populate the *Salary* field? Should it be filled in by nulls, or as a function of the *ZipCode* field?
- How does one populate the *ZipCode* field? Should it be filled in by nulls, or as a function of the *City* attribute of the corresponding $\texttt{Person}_1$ instance?
- In the event that one wants to make changes to the instances in their $\texttt{Person}_2$ form, how are those changes migrated back to the corresponding $\texttt{Person}_1$ instance? Is the *Age* field preserved? How does one calculate the value of the *City* attribute?

One could always leave blank the attributes that do not exactly match. In practice, one wants to preserve as much data through a transformation as possible. With networked and cloud-enabled applications, one wants such transformations to be bidirectional to enable updates to propagate between instances. And data-exchange scenarios are rarely so simple as the example above, as anyone who has written a financial or healthcare application may attest.

The database field has been developing a solution to the data exchange problem over the past decade. This solution uses at its core a subset of first-order predicate calculus, with success in both practical [9] and theoretical applications [1, 11]. Essentially, the transformation is defined by using a logical implication $\alpha \longrightarrow \beta$, in which α and β are first-order predicate calculus expressions, stating that whenever a source database satisfies α then the transformed database should satisfy β. One can use a graphical tool based on drawing associations using box-and-line diagrams; those diagrams map cleanly to these predicate calculus expressions. The elegance of this solution and its extensively proven formal properties have led to this solution becoming the *lingua franca* of data exchange for database researchers.

The predicate calculus approach has some notable limitations, however. It is commonplace in database research to take a language or tool designed for one-way mappings (as predicate calculus is) and attempt to reuse it in bidirectional scenarios for which is may not be suited. There are theoretical results demonstrating that a calculus-based data exchange mapping may not have an inverse, and when it does, that inverse may not be unique [1, 2, 10]. Even in traditional unidirectional settings, a calculus expression may involve existential quantifiers, which mean that target instances will need to fill in attributes with labeled nulls. There may be environment information, domain policy, or other sources that one can use to fill in values that are inaccessible to the current formal treatment.

To address these issues, this paper draws from a sequence of recent efforts to bring together researchers from different fields, all working on bidirectional transformations [6, 18]. Part of the charter of that ongoing effort is to find commonalities between different solutions to similar problems. One commonality that has been identified, but only briefly [23], is between data exchange and a construct from the programming languages community called a *lens* [14]. This paper takes that germ of an idea and expounds upon it to lay out a potential research path that brings those fields together.

Section 2 briefly describes the state of the art in the data exchange literature, giving an overview of the formalism used, the successes of the approach, and its limitations. Section 3 discusses the lens construct in detail, with a particular focus on lens properties that could potentially serve to address the limitations from Section 2. In Section 4, the paper describes what data exchange mappings could look like when leveraging the work of both communities, as well as a few other recent advances. Finally, Section 5 concludes the paper with some additional thoughts on related work and future directions.

2. DATA EXCHANGE

In the database context, data exchange is the problem that arises when data must be transferred between independent database applications that do not have the same data format. In the typical data exchange settings, one is given a source schema and a target schema, usually relational schemas, a *schema mapping* $\mathcal{M}$ that specifies the relationship between the source and the target, and a database instance I of the source schema. The basic problem that one wants to address is how to materialize an instance of the target schema that reflects the source data as accurately as possible [1, 11].

The building block of data exchange, as well as of many other data-interoperability tasks, is the notion of schema mapping, which is a high-level specification, usually stated in a logical language, that describes the relationship between database schemas. The most extensively studied schema-mapping language in the database literature, is the language of source-to-target tuple-generating dependencies (st-tgds). An st-tgd is a sentence of the form

$$\forall \bar{x} \big(\exists \bar{y} \varphi_S(\bar{x}, \bar{y}) \longrightarrow \exists \bar{z} \psi_T(\bar{x}, \bar{z}) \big), \tag{1}$$

where $\varphi_S(\bar{x}, \bar{y})$ and $\psi_T(\bar{x}, \bar{z})$ are conjunctions of relational atoms (names of tables with variables) over the source and target schemas, respectively. The semantics of st-tgds is inherited from the semantics of first order logic. A pair of database instances I and J of the source and target schemas, respectively, satisfies (1) whenever the following holds for every tuple $\bar{a}$: if formula $\varphi_S(\bar{a}, \bar{b})$ is true in I for some elements

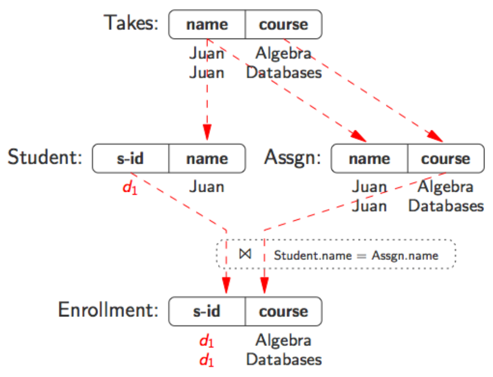


Figure 1: Visual representation of st-tgds

$\bar{b}$, then there exists elements $\bar{c}$ such that $\psi_T(\bar{a}, \bar{c})$ is true in J. Moreover, given a fixed I and a set of st-tgds $\mathcal{M}$, every target instance J such that (I, J) satisfies all the st-tgds in $\mathcal{M}$, is called *a solution for I under $\mathcal{M}$*. Thus, given I and $\mathcal{M}$, the data exchange problem can be formalized as how to materialize *the best* solution for I under $\mathcal{M}$.

EXAMPLE 1. Consider a source schema composed of a single relation $\texttt{Emp}(\cdot)$ to store employees, and a target schema $\texttt{Manager}(\cdot,\cdot)$. A possible st-tgd between these schemas is

$$\forall x \big(\; \texttt{Emp}(x) \;\longrightarrow\; \exists y \; \texttt{Manager}(x, y) \; \big), \tag{2}$$

which essentially states that every employee in the source schema should have a manager in the target schema.

Consider a source instance $I = \{\texttt{Emp}(\mathit{Alice}), \texttt{Emp}(\mathit{Bob})\}$. Two possible solutions for I under the mapping specified by the above st-tgd are

$$\begin{aligned} J_1 &= \{\texttt{Manager}(\mathit{Alice}, \mathit{Alice}), \texttt{Manager}(\mathit{Bob}, \mathit{Alice})\}, \\ J_2 &= \{\texttt{Manager}(\mathit{Alice}, \mathit{Bob}), \texttt{Manager}(\mathit{Bob}, \mathit{Ted})\}. \end{aligned}$$

Notice that there is no constraint over the possible values used as managers in the target. Actually the following is also a solution

$$J^* \;=\; \{\texttt{Manager}(\mathit{Alice}, \bot_1), \texttt{Manager}(\mathit{Bob}, \bot_2)\},$$

where $\bot_1$ and $\bot_2$ are *labelled nulls* used to represent that a value should be there but the mapping does not provide enough information to completely determine it. Actually, J^* is considered as the preferred solution for the exchange as it is the *most general* among all the possible solutions [1].

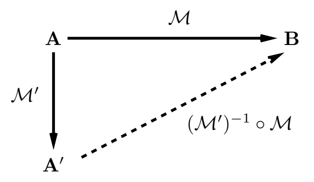
In a practical setting, an end user does not directly specify a mapping by writing down an st-tgd, but by specifying some simple correspondences usually exploiting some visual interface [9]. Figure 1 shows an example of correspondences established by *drawing arrows* between database schemas. These visual representations are then *compiled* into sets of st-tgds. For example, the upper part in Figure 1 can be represented by the st-tgd

$$\forall x \forall y \big(\; \texttt{Takes}(x, y) \;\longrightarrow\; \exists z (\texttt{Student}(z, x) \wedge \texttt{Assgn}(x, y)) \; \big),$$

while the lower part is represented by

$$\forall x \forall z \big(\; \exists y (\texttt{Student}(x, y) \wedge \texttt{Assgn}(y, z)) \longrightarrow \texttt{Enrollment}(x, z) \; \big).$$

In the last few years, a lot of attention has been paid to the development of solid foundations for the problem of exchanging data using schema mappings. These developments



$$\begin{tikzcd}\mathbf{A} \arrow[r, "\mathcal{M}"] \arrow[d, "\mathcal{M}'"'] & \mathbf{B} \\ \mathbf{A}' \arrow[ur, dashed, "(\mathcal{M}')^{-1} \circ \mathcal{M}"'] & \end{tikzcd}$$

Figure 2: The schema evolution problem

are a first step towards providing a general framework for exchanging information, but they are definitely not the last one. In fact, many information system problems involve not only the design and integration of complex application artifacts, but also their subsequent manipulation. This has motivated the need for the development of a general infrastructure for managing schema mappings. In particular, several operators for manipulating schema mappings have been shown to be fundamental in this area.

Two of the most fundamental operators on schema mappings are *composition* and *inversion* [3]. Given a mapping $\mathcal{M}_1$ from a schema $\mathbf{A}$ to a schema $\mathbf{B}$, and a mapping $\mathcal{M}_2$ from $\mathbf{B}$ to a schema $\mathbf{C}$, the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is a new mapping that describes the relationship between schemas $\mathbf{A}$ and $\mathbf{C}$. This new mapping must be *semantically consistent* with the relationships previously established by $\mathcal{M}_1$ and $\mathcal{M}_2$. On the other hand, *an inverse* of $\mathcal{M}_1$ is a new mapping that describes the *reverse* relationship from $\mathbf{B}$ to $\mathbf{A}$, and is semantically consistent with $\mathcal{M}_1$. In practical scenarios, these operators can have several applications one of them being the *schema evolution* problem (Figure 2). Consider a mapping $\mathcal{M}$ between schemas $\mathbf{A}$ and $\mathbf{B}$, and assume that schema $\mathbf{A}$ evolves into a schema $\mathbf{A}'$. This evolution can be expressed as a mapping $\mathcal{M}'$ between $\mathbf{A}$ and $\mathbf{A}'$. The relationship between the new schema $\mathbf{A}'$ and schema $\mathbf{B}$ can be obtained by inverting mapping $\mathcal{M}'$ and then composing the result with mapping $\mathcal{M}$.

The intuitive description of composition and inversion presented in the previous paragraph has turned out to be very difficult to formalize, and several different semantics have been proposed in the last few years [2, 4, 10, 12, 13]. For every semantics, several questions need to be answered, among them, questions about computability, existence, and expressiveness. This last topic is very important in practice. Given that mappings are usually specified using st-tgds, one main question is whether the composition of two of these mappings can also be specified by using st-tgds. A similar question arises for the inverse operator. As shown in the following example st-tgds are not strong enough to specify neither composition nor inversion of st-tgds for most of the semantics proposed so far in the literature.

Example 2 ([12]). Let $\mathcal{M}$ be the mapping specified by the st-tgd (2) in Example 1. Consider a new schema composed of relations $\texttt{Boss}(\cdot,\cdot)$ and $\texttt{SelfMngr}(\cdot)$ and the st-tgds

$$\begin{aligned}\forall x \forall y \big(\, \texttt{Manager}(x,y) &\longrightarrow \texttt{Boss}(x,y)\,\big),\\ \forall x \big(\, \texttt{Manager}(x,x) &\longrightarrow \texttt{SelfMngr}(x)\,\big).\end{aligned}$$

It can be shown [12] that if the mapping in Example 1 is composed with the above mapping, the result can be specified by the logical sentence

$$\begin{aligned}\exists \mathtt{f}\,\big[\quad &\forall x \big(\, \texttt{Emp}(x) \longrightarrow \texttt{Boss}(x,\mathtt{f}(x))\,\big) \;\wedge\\ &\forall x \big(\, \texttt{Emp}(x) \wedge x = \mathtt{f}(x) \longrightarrow \texttt{SelfMngr}(x)\,\big)\quad\big].\end{aligned}$$

This sentence essentially states that there exists a function $\mathtt{f}(\cdot)$ that assigns a manager/boss to every employee, and moreover, if the manager/boss assigned to an employee e equals $\mathtt{f}(e)$, then e should be in the table $\texttt{SelfMngr}$. This logical formula is not an st-tgd, actually it is not even in first-order logic, as it has a *second-order quantification* over a function symbol. It was shown by Fagin et al. [12] that second-order quantification, plus the equality in the left side of the above formula, is unavoidable if one wants to faithfully represent the composition of the two mappings. That is, a composition specified by st-tgds does not exist in this case.

Example 3. To show some of the problems that arise with inversion, consider a mapping specified by the st-tgds

$$\begin{aligned}\forall x \forall y \big(\, \texttt{Father}(x,y) &\longrightarrow \texttt{Parent}(x,y)\,\big),\\ \forall x \forall y \big(\, \texttt{Mother}(x,y) &\longrightarrow \texttt{Parent}(x,y)\,\big),\end{aligned}$$

and let $I = \{\texttt{Father}(\mathit{Leslie}, \mathit{Alice})\}$. Then the best solution for I is the instance $J = \{\texttt{Parent}(\mathit{Leslie}, \mathit{Alice})\}$. When trying to establish the reverse relationship, it is not clear what source instance should be assigned to J. In fact, according to Fagin's initial definition of inverse [10], the above mapping is not invertible. Subsequent works on relaxed notions of inverses for mappings have tried to solve this problem by applying a *best effort* approach [2, 13]. In particular, under the definition of Arenas et al. [2], the *best possible inverse* for the above mapping is given by the sentence

$$\forall x \forall y \big(\, \texttt{Parent}(x,y) \longrightarrow \texttt{Father}(x,y) \vee \texttt{Mother}(x,y)\,\big).$$

Notice that under the inverse mapping both instances $I_1 = \{\texttt{Father}(\mathit{Leslie}, \mathit{Alice})\}$ and $I_2 = \{\texttt{Mother}(\mathit{Leslie}, \mathit{Alice})\}$, are equally good as solutions for $J = \{\texttt{Parent}(\mathit{Leslie}, \mathit{Alice})\}$. Also notice that the above mapping is not an st-tgd as it has a disjunction in the right-hand side, which was shown by Arenas et al. [2] to be unavoidable in this case. This example shows that inverses in general may lose information when going back from target to source, and more importantly, that inverses specified by st-tgds might not exist.

The fact that st-tgds cannot faithfully represent the composition or inversion of st-tgds has motivated the search for a *closed* mapping language; a language L such that the repeated application of the composition and inversion operators over mappings specified in L, can always be specified in L. The existence of such a language is still one of the most challenging open problems in the area.

Several features of st-tgds are in the focus of the problems mentioned above. For instance, the existential quantification both on the source and the target-side of mappings, makes them *non-deterministic* which makes more difficult not only the processes of exchanging data, but also composing and inverting mappings. Actually, Fagin et al. showed that the language of st-tgds that do not use existential quantification on the target side, is closed under composition [12]. Similarly, Arenas et al. proposed a weak semantics for inversion for which the extension of st-tgds with inequalities (plus a predicate to differentiate nulls from constant values) is closed under inversion [4]. Nevertheless, the search for a language closed under both operators has been somewhat elusive for the database community. An addition to st-tgds which has been very common in the literature are *target dependencies* used to enforce properties only over the target side (for example, key and foreign-key constraints on the target). Target dependencies add expressive power and can be used to decrease the level of non-determinism when exchanging data, but at the same time, they complicate the managing of mappings at a high level.

Although the data exchange problem has been successfully studied from a database perspective, several challenges remain open, among the most important, to find a practical mapping language with good properties for specifying exchange processes, and at the same time with good properties for computing and expressing high-level schema mapping operators. We think that in the search for answers to these open questions, the database community should be open to new ways of attacking the problem, borrowing tools and techniques from areas that have faced similar challenges but from different perspectives.

3. LENSES

Lenses [14] are a useful technique developed in the programming language community (although it is noteworthy that some lens concepts appear in work in other communities including work on data storage and caching). Lenses have played an important role in an international research initiative aimed at developing and understanding techniques for *bidirectional transformations* [6, 18].

The motivation for the development of lenses closely parallels some of the open issues raised in the previous section. Bidirectional transformations, unless they happen to be bijections,

- have to provide a candidate for a "reverse" transformation, even when an inverse transformation doesn't exist
- have to deal with potential non-determinism in the reverse transformation
- should be closed under composition, and ultimately
- should form a closed mapping language as described above.

Lenses, in their various guises, provide such bidirectional transformations.

The most basic form of a lens, called a *set-based lens*, consists of two sets S and V and two functions g (pronounced *get*) $S \longrightarrow V$, and p (pronounced *put*) $V \times S \longrightarrow S$. Set-based lenses are *asymmetric* and can be thought of as an abstraction of a view updating problem — S should be thought of as the set of states of a system, and V as the set of states of a view of that system. Since a view state contains less information than a system state, a view state can easily be calculated from any system state, and that's what g does. However given just a view state it is not in general possible to determine a single corresponding system state because of the missing information. Instead, the lens provides a way, p, of calculating a new system state from an old system state and a specified view state. The function p can be thought of as *view updating*: Given a system state s and its corresponding view $g(s)$, if the view is modified to some new view state v, then the system state s should be updated to a new system state $p(v, s)$.

A lens is called *well-behaved* if it satisfies two conditions

(i) (PutGet) the updated system state really does correspond to the view state v: $g(p(v, s)) = v$, and
(ii) (GetPut) the Put for a trivially updated state is trivial: $p(g(s), s) = s$.

Some examples of work on well-behaved asymmetric lenses include quotient lenses [15], which allow the properties of a lens to be relative to equivalence classes; delta lenses [8, 21], which enrich the situation by using the nature of the modification, the delta, from $g(s)$ to v to compute a delta which can be used to update s; edit lenses [16], which take as input edit operations rather than simple deltas; and so on. In each case the lenses are composable, and lenses are in a sense a schema mapping, but, sadly, inversion of asymmetric lenses is, except in trivial cases, not defined.

These lenses are all called *asymmetric* to emphasise the difference between S and V: S contains all the information and V some information that can be derived from S. The lenses themselves are sometimes said to go *from S to V*. They are bidirectional, but asymmetrically so. Data exchange, and many other notions of data synchronisation, are in contrast symmetric — there is no master source of data like S, and no a priori "easy" direction for data exchange.

Recently, a notion of well-behaved *symmetric lens* has been introduced [17]. Although they are defined elementarily, a symmetric lens can be seen to be equivalent to a span of asymmetric lenses: A set-based symmetric lens between S and T amounts to a set U (sometimes called "universal" because it contains all the information of both S and T, and in general even more besides) and two asymmetric lenses, one from U to S and one from U to T. Symmetric lenses are composable and each symmetric lens has an inversion obtained by exchanging the roles of S and T.

(A *span* is, as just described, a pair of lenses (or functions, or more generally other agreed operations) with a common starting position viz $S \longleftarrow U \longrightarrow T$. A *cospan* (cospans will be referred to at the end of the paper) is similarly a pair of lenses (or functions, etc) with common ending positions viz $S \longrightarrow X \longleftarrow T$.)

There is an analogy here with functions and relations — functions do not in general have uniquely defined inversions, but instead we might consider spans of functions that form generalised relations and do have inversions given by reversing the relation. The analogy is in fact the mathematical basis of the symmetric lens constructions: in both cases inversion is obtained the same way, and composition is calculated by pullback (the basis of relational composition).

As a concrete example, consider the work to date on *relational lenses* [5]. Relational lenses have a strong correlation with relational algebra; for instance, there is a "projection" lens corresponding to the projection operator π. Because each lens must specify put and get functions p and g, each lens not only describes how to retrieve data as does its relational algebra counterpart, but also how to update and replace it. Thus, relational lenses can be considered a solution to the view-update problem [7].

Each given relational operator does not have a unique update policy, however. For the given simple example of the projection operator π, if the operator drops a column c, and a new row is added to the output (view) state, there are several possibilities as to how to populate that column c when adding the row to the input state. Some possibilities are:

- Always use a null value.

- Always use a constant value.
- Always insert an environment value, e.g., the current time or user.
- Use a functional dependency $c' \rightarrow c$ from another column c' to determine the value.

The original work on relational lenses treats the last of those options as the proper one in the sense that it is the least lossy, but requires the presence of a functional dependency to operate. Each of these choices of update policy is equally valid based on the requirements of the user and the available data. Therefore, one can equally consider a *relational lens template* as a way to describe a family of potential lenses corresponding to a specific relational operator but missing its update policy.

The update policy for the projection operator is very similar to a way to specify how to resolve existential quantifiers in st-tgds, but update policies for other operator templates need not be. For example, the join and union lens templates must have update policies specifying whether updates are propagated to the left or right inputs, or to both.

It is important to note that relational lenses to date are *asymmetric*. As noted before, to be a complete solution to data exchange scenarios, an important first step would be to develop symmetric versions of these lenses. However, the potential association between lenses, especially relational lenses, and data exchange is powerful. There is a rich and evolving body of work on lenses and their generalisations. Lenses are an abstraction of schema mappings. And symmetric lenses provide a closed mapping language since they have inversions and compositions. Thus lenses seem set to solve outstanding problems in data exchange including the development of a closed mapping language and the schema evolution problem. Furthermore, of course, the development of lenses will likewise be positively influenced by closer and more explicit links with the database community and practical data exchange.

4. BRINGING THEM TOGETHER

What is hopefully clear by this point is that the data exchange problem is relevant to multiple fields of study within computer science, and that different fields have developed solutions to the problem that could potentially learn from each other. To summarize:

An st-tgd is a declarative, unidirectional language. It has tool support in the form of mapping builders [9]. The formal treatment of st-tgds demonstrates a great many positive angles, but has practical problems when it comes to existential quantifiers and completeness under inversion and composition.

Lenses combine to become operationally specified, domain-specific bidirectional languages. One such language is relational lenses, which have general parity with relational algebra.

There are almost certainly many possible ways to interact between these two tools and formalisms. However, one concrete approach comes from an old idea in the relational database world, namely the relationship between relational calculus and relational algebra familiar to every database researcher and central to every relational database implementation. The SQL language was largely inspired by domain relational calculus and was intended as a declarative specification to be later translated into operational atoms. Specifically, the user workflow is as follows:

- (optional) Via some form of query builder, the user constructs a query that is translated into SQL.
- The SQL query is translated statically to a relational algebra expression.
- The relational algebra expression is translated to a query plan by associating algorithms with operators, and by applying optimization routines. This process is highly informed by gathered statistics, and may be informed by user intervention.

By associating st-tgds with lenses, one can imagine a similar workflow in a bidirectional or synchronization world:

- Via some form of visual interface, the user constructs a mapping that is translated into st-tgds.
- The collection of st-tgds is translated statically to a relational lens template.
- The relational lens template is translated to a mapping plan by associating algorithms with operators, and by applying optimization routines. This process is highly informed by gathered statistics, and in some instances *must* be informed by user intervention (in the same spirit as past work on updatable views [22]).

An added benefit to this approach is that a mapping would now have a "show plan" capability similar to that used in relational database engines. The designer of a mapping would be able to see not only how the mapping is specified (in language that is natural to st-tgds) but also how it will be evaluated (in language that is natural to anyone that has ever used a relational database).

One necessary difference between this approach and the relationship between SQL and relational algebra is the expected amount of necessary user intervention. With a typical relational engine, user hints are entirely optional; a user may be able to specify indexes or join algorithms, but without any such input, the engine still manages to determine an optimal query plan a vast majority of the time. With the data exchange scenario, one would need to somehow fill in the relational lens template parameters, needing answers to questions like "what do I do with this extra column". While reasonable defaults may exist, it is unclear as to how often those defaults will be optimal to the user's scenarios. Therefore, it is important to discover some way to map those questions to user input, which is primarily a usability question, and a non-trivial one at that. Thus, to successfully incorporate the two technologies along these lines, at the very least the following will need to be accomplished:

1. A symmetric version of the relational lens framework.
2. An st-tgd-to-lens compiler, and a completeness proof of that compiler.
3. A reasonable mapping of relational lens template parameters to user gestures — for instance, giving the user an understandable way to dictate through which inputs an update to a join should propagate.

One final solution of note is that one can use a combination of lenses and st-tgds to solve the schema evolution problem in Section 2 as well, and possibly in two different manners. The simplest of the two possible approaches is to note that composing mappings specified using lenses is as simple as concatenating them. So, if there is a mapping from S to T as $[m_1, m_2, m_3]$, and one can express a schema evolution operation against S to S' as a sequence of symmetric lenses $[\ell_1, \ell_2]$, then one can construct a mapping from S' to T as $[\ell_2^{-1}, \ell_1^{-1}, m_1, m_2, m_3]$.

It should be noted that lenses are not the only recent attempt to construct mappings from incremental components such [23]; one such language (called *channels*) allows schema evolution primitives to be propagated through mappings rather than appended to one end [24]. It may prove useful to end users that have control over both ends of a mapping to have a choice between adapting one schema and composing the mappings as in the previous paragraph, or propagate the evolution primitives through the mapping and construct a new, evolved target schema T'. To support the latter, one must add an additional research goal to the above list, namely to update the lens formalism to allow the possibility of schema evolution, or otherwise resolve the differences between lenses and channels.

5. CONCLUSION

This paper has described the status quo of research on data exchange on two different fronts in two different disciplines: st-tgds from database literature, and lenses from programming language literature. Both tools have rich publication histories and formal results; the vision laid out in this paper is not intended to replace either one. Rather, it seems likely that by combining the work done in both areas and following the spelled-out research agenda, a novel, complete, and practical solution to the data exchange problem may result. To realize the vision will require work at practical and formal levels and across interdisciplinary boundaries.

There are also tantalising opportunities that require further study. Among them there is work showing how certain kinds of lenses really do correspond concretely to schema mappings [20]. Can this be generalised to symmetric lenses in their various forms? Are symmetric lenses really an embodiment of schema mappings with a built-in calculus of data exchange? Also, there is already practical work in building data exchange via cospans of certain kinds of lenses [19]. That work has been used to concretely implement data exchange and systems interoperation. A *co*-span of asymmetric lenses is not a symmetric lens, but there must be a precise mathematical relationship between the co-span based data exchange and the span based schema mappings.

6. REFERENCES

[1] M. Arenas, P. Barceló, L. Libkin, F. Murlak. Relational and XML Data Exchange. Synthesis Lectures on Data Management, Morgan and Claypool Publishers, 2010.

[2] M. Arenas, J. Peréz, C. Riveros. The recovery of a schema mapping: bringing exchanged data back. *TODS* 34(4) (2009).

[3] M. Arenas, J. Pérez, J.L. Reutter, C. Riveros. Composition and inversion of schema mappings. *SIGMOD Record* 38(3), 17–28 (2009).

[4] M. Arenas, J. Pérez, J.L. Reutter, C. Riveros. Query Language based Inverses of Schema Mappings: Semantics, Computation, and Closure Properties *VLDBJ* 21(6), 823–842 (2012)

[5] A. Bohannon, B.C. Pierce, J.A. Vaughan. Relational lenses: a language for updatable views. *PODS 2006.*

[6] K. Czarnecki, J.N. Foster, Z. Hu, R. Lämmel, A. Schürr, J.F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. *ICMT 2009.*

[7] U. Dayal and P. Bernstein. On the Correct Translation of Update Operations on Relational Views. *ACM Transactions on Database Systems*, September 1982, 8(3).

[8] Z. Diskin, Y. Xiong, K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case, *Journal of Object Technology* 2011, 10(6), 1–25.

[9] R. Fagin, L.M. Haas, M.A. Hernández, R.J. Miller, L. Popa, Y. Velegrakis et al. Clio: Schema Mapping Creation and Data Exchange. *Conceptual Modeling: Foundations and Applications*, 2009, 198–236.

[10] R. Fagin. Inverting schema mappings. *TODS* 32(4) (2007).

[11] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa. Data exchange: semantics and query answering. *TCS* 336(1), 89–124 (2005).

[12] R. Fagin, P.G. Kolaitis, L. Popa, W.C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *TODS* 30(4), 994–1055 (2005).

[13] R. Fagin, P.G. Kolaitis, L. Popa, W.C. Tan. Quasi-inverses of schema mappings. *TODS* 33(2) (2008).

[14] J.N. Foster, M.B. Greenwald, J.T. Moore, B.C. Pierce, A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3): (2007).

[15] J.N. Foster, A. Pilkiewicz, B.C. Pierce. Quotient Lenses. *ICFP 2008.*

[16] M. Hofmann, B.C. Pierce, D. Wagner. Edit lenses. *POPL 2012.*

[17] M. Hofmann, B.C. Pierce, D. Wagner. Symmetric lenses. *POPL 2011.*

[18] Z. Hu, A. Schürr, P. Stevens, J.F. Terwilliger. Dagstuhl seminar on bidirectional transformations (BX). *SIGMOD Record* 40(1), (2011).

[19] M. Johnson. Enterprise Software with Half-Duplex Interoperations. In Doumeingts, Mueller, Morel and Vallespir (eds), Enterprise Interoperability: New Challenges and Approaches, 521–530, Springer-Verlag, (2007).

[20] M. Johnson and R. Rosebrugh. Fibrations and Universal View Updatability. Theoretical Computer Science, 388, 109–129, (2007).

[21] M. Johnson and R Rosebrugh. Delta lenses and opfibrations. *BX 2013*, to appear.

[22] A.M. Keller. Choosing a View Update Translator by Dialog at View Definition Time. *VLDB 1986*, 467–474.

[23] J.F. Terwilliger, A. Cleve, C. Curino. How Clean Is Your Sandbox? *ICMT 2012.*

[24] J.F. Terwilliger, L.M.L. Delcambre, D. Maier, J. Steinhauer, S. Britell. Updatable and Evolvable Transforms for Virtual Databases. *PVLDB 3(1) (VLDB 2010).*