From Complete to Incomplete Information and Back

Lyublena Antova, Christoph Koch, and Dan Olteanu
Saarland University Database Group
Saarbrücken, Germany
{lublena, koch, olteanu}@infosys.uni-sb.de

ABSTRACT

Incomplete information arises naturally in numerous data management applications. Recently, several researchers have studied query processing in the context of incomplete information. Most work has combined the syntax of a traditional query language like relational algebra with a nonstandard semantics such as certain or ranked possible answers. There are now also languages with special features to deal with uncertainty. However, to the standards of the data management community, to date no language proposal has been made that can be considered a natural analog to SQL or relational algebra for the case of incomplete information.

In this paper we propose such a language, World-set Algebra, which satisfies the robustness criteria and analogies to relational algebra that we expect. The language supports the contemplation on alternatives and can thus map from a complete database to an incomplete one comprising several possible worlds. We show that World-set Algebra is conservative over relational algebra in the sense that any query that maps from a complete database to a complete database (a complete-to-complete query) is equivalent to a relational algebra query. Moreover, we give an efficient algorithm for effecting this translation. We then study algebraic query optimization of such queries.

We argue that query languages with explicit constructs for handling uncertainty allow for the more natural and simple expression of many real-world decision support queries. The results of this paper not only suggest a language for specifying queries in this way, but also allow for their efficient evaluation in any relational database management system.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms

Design, languages

Keywords

Incomplete information, hypothetical queries, query rewriting, world-set algebra, use cases

SIGMOD'07, June 11–14, 2007, Beijing, China.

1. INTRODUCTION

Incomplete information arises naturally in numerous data management applications like data integration [18], data cleaning [3], or data exchange [11]. In the last decades the research community has shown a vivid interest in efficiently managing incomplete information viewed as a *set of possible worlds* [16, 12, 17, 2, 13, 19, 6, 7, 10, 11, 14, 8, 4]. When it comes to expressing queries on incomplete information, these contributions mostly consider standard languages for complete data such as relational algebra or SQL. While [16] uses a compositional semantics for relational algebra in which a query transforms each world individually, most recent work has assumed nonstandard, noncompositional query semantics.

A significant amount of research has attempted to find the right balance between the succinctness of world-set representations and the efficiency of query evaluation on top of them (e.g., [2, 8, 4, 5]). However there is a lack of expressive query languages which are well tailored for sets of possible worlds.

In this paper we address the issue of supporting queries on incomplete information. A query language for incomplete information should fulfill at least the following four desiderata.

- It must be **generic**, i.e., it must preserve the independence of the data from its representation. Query results must not depend on details of how the data is stored.
- It must be **expressive** enough to support common queries on incomplete information. The expressiveness of the language should be proven for a reasonable load of use cases.
- It should be **conservative** over existing query languages such as relational algebra in the sense that any query that maps from a complete database to a complete database admits an equivalent relational algebra query. Conservativity provides an argument that the language is an analog of relational algebra for the new data model. It formalizes our desire for the language not to be overly expressive, at the expense of high computational complexity or difficulty at adapting established query processing techniques to the new language.
- It should allow for **efficient evaluation**.

To date, no proposal for a query language for incomplete information has been made that satisfies all of the above desiderata. SQL lacks explicit constructs for dealing with uncertainty, though there are queries on incomplete information that can be expressed as SQL queries on relational representations of incomplete databases with complicated nesting and aggregations (as shown later in the paper). Extensions of relational algebra or SQL with limited constructs, such as *certain* or *top-k*, that close the possible worlds semantics are presented in, e.g., [17, 6] and [10] respectively. Such extensions

are not expressive enough, as they do not allow for the convenient construction of new worlds or for the use of data correlations across worlds. Other recent query languages provide constructs that depend to a large extent on the representation model. An example is TriQL [23], the query language of the Trio system for managing possible worlds [8]. Its constructs provide explicit access to the internal artifacts of the used representation model, which cannot be interpreted independently from the model.

Languages for querying incomplete information can be motivated even for querying complete data. This is the case for hypothetical ("what if") queries that are important in decision support. The TPC-H (Decision Support) Benchmark Specification [21] includes two "what if" queries (Q_6 and Q_{17}) [24]. For example, Q_6 asks for the amount of revenue increase that *would have resulted* from eliminating certain company-wide discounts in a given percentage range and year. This query thus reasons in alternative worlds that contain counterfactual data.

Hypothetical queries have also been addressed in previous research. For instance, [15] develops a language for determining in queries what result would have been obtained from the database if certain updates had been applied. While this work develops interesting techniques for rewriting and optimizing queries, the language itself does not use a possible worlds semantics and cannot be used to map from or to uncertain data.

The technical contributions of this paper are as follows.

- We propose a new language, called I-SQL, which is a natural analog to SQL for the case of incomplete information. In contrast to SQL queries, I-SQL queries can exploit the possible worlds interpretation of incomplete data.
- We motivate our language using scenarios from planning and decision-support applications. We also argue that query languages with explicit constructs for handling incompleteness allow for the simpler expression of many real-world queries of the traditional one-world-to-one-world kind.
- We define an algebra for a clean fragment of I-SQL, which we call *world-set algebra*. World-set algebra is to I-SQL what relational algebra is to SQL. Our algebra focuses on the new constructs that deal with incomplete information. Like relational algebra, it does not consider SQL aggregations and bag semantics.
- We show that world-set algebra is generic.
- We show that world-set algebra is conservative over relational algebra. This means that any world-set algebra query that maps from a complete database to a complete database (a "complete-to-complete" query) is equivalent to a relational algebra query.
- We give an efficient algorithm for effecting this translation. It follows that complete-to-complete world-set algebra queries have the same low data complexity as relational algebra. Traditional techniques for efficiently processing relational algebra queries can be directly employed to evaluate world-set algebra queries.
- We establish equivalences and rewrite rules that hold for the operations of world-set algebra and study algebraic query optimization.

Thus we show that world-set algebra fulfills all the above desiderata for a query language for incomplete information.

The structure of the paper follows the list of contributions. Due to lack of space, we will treat I-SQL informally, mostly in examples. We will focus on world-set algebra in the formal treatment. To make it easy to see the close connection between I-SQL and the algebra, it is safe to assume a set-based semantics for I-SQL.

2. APPLICATION SCENARIOS

We next motivate our language I-SQL by examples from areas such as decision support, trip planning and data cleaning.

Business decision support queries. Decision support queries assist decision makers in various domains of business analysis, like pricing and promotions, profit and revenue management, or studies on the market and customer satisfaction. Usually, such queries are hypothetical (or "what if") in the sense that they contemplate possible alternatives based on various hypothetical assumptions of decision makers.

Let us consider an example in which we have a (complete, i.e. single-world) database containing information about companies, their employees, and the various skills of these employees.

Company_Emp	CID	EID
	ACME	e1
	ACME	e2
	HAL	e3
	HAL	e4
	HAL	e5

Emp_Skills	EID	Skill
	e1	Web
	e2	Web
	e3	Java
	e3	Web
	e4	SQL
	e5	Java

Suppose we consider buying a single of these companies in order to gain the competency 'Web'. However, we want to take into consideration that one of the employees might be disgruntled by the takeover and leave; we want to guarantee that we acquire the skill 'Web' nevertheless. We can phrase this query in the I-SQL language as follows. In the following, we proceed constructing the query step by step, and always also show the resulting relations.

- "*Suppose I choose* to buy exactly one company."

```
U ← select      *
    from        Company_Emp
    choice of   CID;
```

This results in two possible worlds, obtained by taking the two input relations and adding U^1 for the first world and U^2 for the second:

U^1	CID	EID
	ACME	e1
	ACME	e2

U^2	CID	EID
	HAL	e3
	HAL	e4
	HAL	e5

- "*Assume* that one (key) employee leaves that company."

```
V ← select   R1.CID, R1.EID
    from     Company_Emp R1,
             (select * from U choice of EID) R2
    where    R1.CID = R2.CID and R1.EID != R2.EID;
```

The result is five worlds (Company_Emp, Emp_Skills, U^i, $V^{i.j}$):

$V^{1.1}$	CID	EID
	ACME	e1

$V^{1.2}$	CID	EID
	ACME	e2

$V^{2.1}$	CID	EID
	HAL	e3
	HAL	e4

$V^{2.2}$	CID	EID
	HAL	e3
	HAL	e5

$V^{2.3}$	CID	EID
	HAL	e4
	HAL	e5

- "If I acquire that company, which skills can I obtain for *certain*?"

W ←

select **certain**	CID, Skill
from	V, Emp_Skill
where	V.EID = Emp_Skill.EID
group worlds by	(select CID from V);

We stay at five worlds, and extend them by one of the two following relations.

$W^{1,*}$	CID	Skill
	ACME	Web

$W^{2,*}$	CID	Skill
	HAL	Java

- "Now list the *possible* acquisition targets if I want to guarantee to gain the skill "Web" by the acquisition."

select **possible**	CID
from	W
where	Skill = 'Web';

This results in the following relation, added to each of the five worlds:

Result*,*	CID
	ACME

Trip planning. Consider the relation

```
Flights(Fid, Dep, Arr, Dtime, Atime)
```

encoding information about daily flights with id Fid from departure airport Dep to arrival airport Arr. The departure and arrival time are given by Dtime and Atime, respectively. Suppose we want to schedule a meeting of a group of people from a set of cities given by unary relation 'Hometowns'. We will use the view

```
create view HFlights as
select * from Flights where Dep in Hometowns;
```

below to save space. The individuals would like to meet by taking direct flights to a single common city (in which none of them lives). We express this query for eligible destinations using an extension of SQL by two new constructs, choice-of and 'certain'.

```
select certain Arr from HFlights choice of Dep;
```

For each of the departure airports (expressed by, intuitively, nondeterministically choosing a departure airport and selecting all the flights with that departure airport), we select the possible destinations (attribute Arr), and then compute the destinations common to all departures (using 'certain'). Using choice-of we interpret each departure as an alternative world. In each world we find the destinations, and then close the possible worlds semantics by computing the certain destinations.

Assuming the existence of a division operator in SQL [22], we can express the query in SQL as (assuming set-based semantics for SQL and I-SQL):

```
select Arr
from (select Arr, Dep from HFlights) as F1
     divide by
     (select Dep from HFlights) as F2
     on F1.Dep = F2.Dep;
```

This computes all arrival cities that appear in combination with all departures. Division can be simulated in SQL using a nested subquery with two not-exists constructs:

```
select Arr from HFlights F1
where not exists
     (select * from HFlights F2
      where not exists
           (select * from HFlights F3
            where F3.Dep = F2.Dep and F3.Arr = F1.Arr));
```

This shows that at least in certain cases, I-SQL allows to phrase decision support queries more concisely than plain SQL.

TPC-H. A different example for decision support queries is query Q_6 in TPC-H [21] which quantifies the amount of revenue increase that *would* have resulted from eliminating certain company-wide discounts in a given percentage range in a given year. Also, query Q_{17} in TPC-H determines how much revenue would be lost if orders were no longer filled for some *given* quantities. Both queries are expressible in SQL using fairly simple select-from-where statements on one relation [24].

We next discuss one query similar in spirit to Q_{17}. Assume we have a simplified version of the TPC-H Lineitem relation

```
Lineitem(Product, Quantity, Price, Year)
```

containing information about products sold in fixed quantities or package sizes, e.g., one hundred grams or one kilogram.

We would like to compute the years with a revenue loss over 1.000.000$ if *any* quantity of the sold products is no longer available. To answer this query, we first define all pairs of year and missing quantity as possible worlds and compute the revenue for each of these pairs.

```
create view YearQuantity as
select A.Year, sum(A.Price) as Revenue
from  (select * from Lineitem choice of Year) as A
where Quantity not in
      (select * from Lineitem choice of Quantity)
group by A.Year;
```

The view YearQuantity creates a world for each year (by making a choice of the year), and then for each year it creates a world for each missing quantity (by specifying a choice of the quantity in the subquery). Finally, in each of the created worlds we compute the revenue, i.e., the sum of prices of all sold products.

For each pair of year and missing quantity, we can now compare its computed revenue with the revenue of the year without missing quantities. If the difference in revenues is greater that our threshold, then we report the year.

```
select possible Year from YearQuantity as Y
where (select sum(Price) from Lineitem
               where Lineitem.Year = Y.Year)
    - Y.Revenue > 1000000;
```

Consistent views of inconsistent data. Consider the relation Census (SSN, Name, POB, POW) containing simplified information about social security number, name, place of birth, and place of work for a set of persons. When such data is manually entered, it is highly likely that certain constraints are initially violated. For example, the mistyping of the SSN can violate the functional dependency SSN → Name, POB, POW. In such cases, one particularly useful view of this inconsistent relation is to consider the set of possible Census relations that are consistent w.r.t. the given functional dependency. We can support such a view using the new construct repair-by-key.

```
select * from Census repair by key SSN;
```

The above query creates all possible relations that are consistent w.r.t. our functional dependency and are contained in the relation Census. This query construct fits naturally into data cleaning scenarios and provides support for deduplication based on key constraints. The above query can produce exponentially many worlds

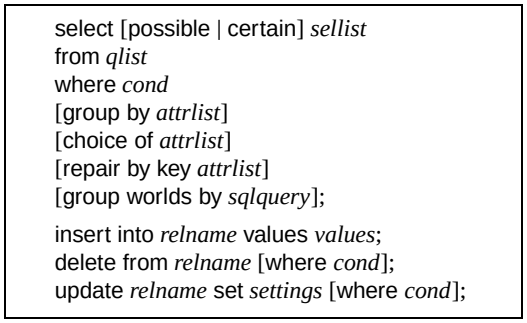


Figure 1: Syntax of I-SQL queries and data manipulation commands.

(representing all consistent combinations of social security numbers), and is thus not expressible in SQL (or relational algebra). In fact, NP-hard problems can be expressed as queries with the repair-by-key construct.

3. I-SQL

This section describes an SQL-like language, called I-SQL, for querying possible worlds, part of which was already introduced in Section 2 using examples. For the purposes of this paper we assume set semantics for SQL and I-SQL. The generalization to bag semantics is a subject of future work.

A main motivation of our work is to find a natural extension of relational algebra and SQL to the context of incomplete information. Traditionally, query evaluation in this context is defined as mapping between sets of possible worlds. The query is evaluated in each world independently, and the world is extended with the result of the query in it. A different approach is followed in work on computing certain or (ranked) possible answers which results in closing the possible worlds semantics.

Here we combine the two approaches and also add support for creating new worlds. The structure of an I-SQL query is summarized in Figure 1. We next detail on the syntax and semantics of the constructs separated in four groups.

Standard SQL constructs. In accordance to the possible worlds semantics, a query is evaluated in each world independently and the result is added as a new relation to that world. For example the query

```
select * from Flights where Arr = 'BCN';
```

will retrieve the flights to Barcelona in each world.

Merging worlds. This group contains constructs that go across world borders to collect information that appears in other worlds as well.

- possible and certain. These constructs compute the tuples that appear in some, respectively all worlds. The result is then added to each world of the input world-set.
- group-worlds-by. This construct is used in combination with 'possible' and 'certain' and allows specifying a condition on which the worlds are grouped. The condition is given in form of an SQL query; worlds that produce the same result of that query are then put into the same group. Then, 'possible' or 'certain', respectively, are computed within each of the created groups. Sometimes, when the query on which we group is a projection on a set of attributes, we will write directly only the set of attributes, as is done in the group-by construct of SQL.

Even though our language has the power to combine results from different worlds into one, this is still accomplished in an "inside-out" manner. This means that a query, as before, is evaluated in each world independently; however it can also occasionally look "outside" the current world to obtain the necessary information. The result of the query remains local to the currently considered world. This differs from the approach where a query can access the worlds "from outside", and preserves the spirit of traditional query processing on sets of possible worlds.

EXAMPLE 3.1. Consider as input the set of three worlds $\mathcal{A}, \mathcal{B}$, and C of Figure 2 (b) showing the flights from Frankfurt, Paris, and Barcelona, respectively and the query that finds the certain arrivals:

```
select certain Arr from Flights;
```

Evaluated in the first world, the query produces a new relation F which is the intersection of the Flights.Arr tuples from all worlds, and this relation is added to the first world. The result in the remaining two worlds is computed in the same way, see Figure 2 (d).

Even though we used the closing construct 'certain', the result is again the set of three input worlds, where each of them is extended with a new relation F. Only if the input is a single world, or if one is interested only in the result of the operation and not in the input relations, will a 'possible' or 'certain' construct produce a single world. □

Splitting up worlds. In addition to the merging constructs, the language we propose enables the creation of new worlds using the choice-of or repair-by-key operations.

- choice-of. This construct is used to freeze the values for a given set of attributes and analyze each such combination of values in a separate world. For example the following choice-of query

  ```
  select * from Flights choice of Dep;
  ```

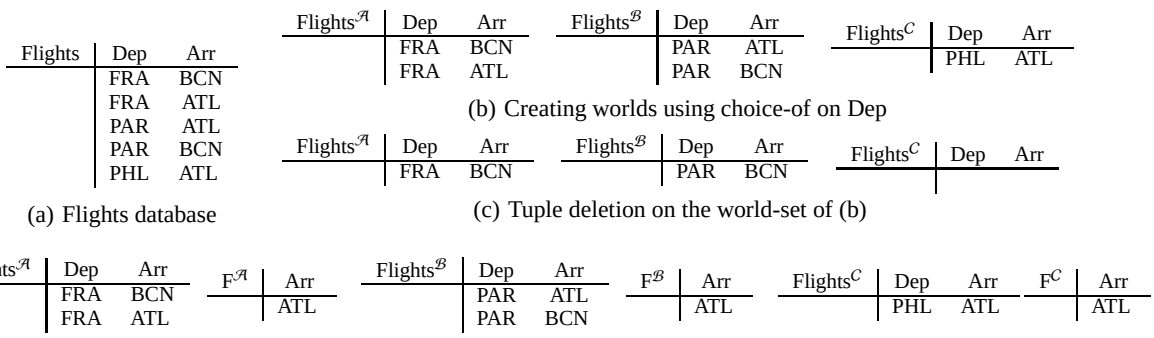
  applied on the world-set of Figure 2 (a) will produce a world for each possible value of the Dep attribute of Flights, which will contain all tuples with that value. The result of the query is given in Figure 2 (b).
- repair-by-key. The repair-by-key construct generates, as its name suggests, the possible repairs of a relation that violates a uniqueness constraint for the values of a given set of attributes. This makes sense in the context of cleaning inconsistent data, where tuples overlap on a set of key attributes and each choice of a distinct tuple for each combination of values is a possible repair of the database. As discussed in the introduction, repair-by-key can also be used for generating possible configurations of items where each configuration contains only one item of a type. For example,

  ```
  select * from R repair by key A;
  ```

  creates a world-set where each tuple has a unique A-value.

Data Manipulation. I-SQL uses the standard operations of SQL 'insert', 'update' and 'delete' to manipulate the data. Again, the semantics follows the possible worlds scheme, where the query is executed in each world of the world-set independently. For example, a query that inserts a tuple into a relation will insert the tuple in each world of the world-set. In case that inserting the tuple violates a constraint in some worlds, the update is discarded in all worlds.

The syntax of the data manipulation operations corresponds directly to the one in SQL and is given in Figure 1.



(a) Flights database

(b) Creating worlds using choice-of on Dep

(c) Tuple deletion on the world-set of (b)

(d) Result of "`select certain Arr from Flights;`" on the world-set of (b)

Figure 2: Trip planning.

EXAMPLE 3.2. Consider the world-set in Figure 2 (b) and the command that deletes all entries containing 'ATL' as the Arr attribute:

```
delete from Flights where Arr = 'ATL';
```

The result is given in Figure 2 (c). □

Order of evaluation. The skeleton of each I-SQL query is an SQL select-from-where statement which is conceptually evaluated in the standard way by (1) computing the product of the relations produced by the subqueries in the from-clause, (2) applying the conditions of the where-clause on top, and (3) projecting on the attributes given in the select list. If any of the new operators choice-of, repair-by-key and group-worlds-by are specified, they are applied after step (2) in the order given by Figure 1. In other words we first use choice-of to create a world for each combination of values for the specified attributes, and then repair-by-key in each of the created worlds. The group-worlds-by operation is applied on the world-set created after the repair-by-key construct. Only then we apply step (3) to project the attributes given in the select-clause and if 'possible' or 'certain' are present we union, respectively intersect, the tuples in that projection.

4. WORLD-SET ALGEBRA

In this section we define an algebra for the fragment of I-SQL without SQL grouping and aggregation constructs. This algebra, called World-set Algebra, is an extension of relational algebra with the new constructs poss, cert, choice-of, and group-worlds-by. We then show that world-set algebra is generic in the sense that the semantics of a query is independent of the world-set representation. Recall that genericity is a fundamental property of query languages like relational algebra and SQL [1].

4.1 Syntax and Semantics

We consider the named perspective of the relational model and relational algebra operators selection σ, projection π, product $\times$, union $\cup$, difference $-$, and renaming δ. Sometimes we will also use the operators intersect $\cap$ and division $\div$ expressible using the six base operators. Sets of attributes are denoted by capital letters U and V. In addition to relational algebra operators, we consider a fragment of the I-SQL operators, namely the unary operators poss, cert, χ_U (choice-of), $p\gamma_U^V$ (possible-group-worlds-by), and $c\gamma_U^V$ (certain-group-worlds-by).

Figure 3 gives the semantics of world-set algebra defined as a function $[\![\cdot]\!]$ mapping between world-sets. The semantics function is defined inductively on the structure of world-set algebra queries. Let the initial world-set $\mathbf{A}$ contain worlds over schema $\Sigma = \langle R_1, \ldots, R_k \rangle$. By applying a query q to $\mathbf{A}$ we add to each world of $\mathbf{A}$ a new relation R_{k+1} that represents the answer to q in that world. Thus we obtain a new world-set over schema $\langle R_1, \ldots, R_{k+1} \rangle$. We next discuss the semantics of each of our operators.

If the query to evaluate is the identity on a relation (i.e., of the form R_i), we add a copy of that relation to each world.

To evaluate $f(q)$, where f is a unary relational algebra operator (selection, projection, or renaming), we first evaluate q and produce a relation R_{k+1} in each world. Then, in each world, f is evaluated on R_{k+1} and the answer, obtained in standard way for this relational algebra operation, replaces R_{k+1}.

The case of relational algebra binary operators is a bit more involved, because its operands can produce different world-sets. We first evaluate the operands and obtain two world-sets $\mathbf{A}'$ and $\mathbf{A}''$, both representing sets of databases with $k+1$ relations. We then perform the binary operation in those combinations of one world from $\mathbf{A}'$ and one world from $\mathbf{A}''$ that agree on the relations $R_1, \ldots, R_k$. This condition ensures that we forbid operations between relations that occur in different worlds in the original world-set.

The choice-of operator χ_U creates a new world for each choice of the values in the projection π_U on R_{k+1} in each world. The relation R_{k+1} is then replaced in each of the new worlds by the subset of R_{k+1} consisting of those tuples that agree on the values of U. Thus there are no two new worlds created from the same world $\langle R_1, \ldots, R_{k+1} \rangle$ with the same values of U. When applied to the empty relation, choice-of produces an empty relation. Note that each newly created world also contains the relations $R_1, \ldots, R_k$ of the world from which it was derived. This assures compositionality.

The group-worlds-by operators $p\gamma_U^V$ and $c\gamma_U^V$ group worlds in a world-set such that all worlds in a group agree on $\pi_U(R_{k+1})$. We then replace R_{k+1} by $\pi_V(R_{k+1})$ in each world. In the case of $p\gamma_U^V$, R_{k+1} in each world $\mathcal{A}$ is replaced by the union of the relations R_{k+1} from the group of worlds associated with $\mathcal{A}$. Analogously, in the case of $c\gamma_U^V$, the new relation R_{k+1} in a world $\mathcal{A}$ becomes the intersection of the relations R_{k+1} from the group of worlds associated with $\mathcal{A}$.

In case f is poss, then R_{k+1} is replaced by the union of all its instances across all worlds. (That is, this replacement is carried out in all worlds.) If f is cert, then R_{k+1} is replaced by the intersection of all its instances across all worlds.

In Figure 3, $p\gamma_U^V$, $c\gamma_U^V$, poss and cert are all defined using the auxiliary operators $p\gamma_\sim^V$ and $c\gamma_\sim^V$ (which we assume not to be part of world-set algebra).

$$[\![R_i]\!](\mathbf{A}) := \{\langle R_1, \ldots, R_k, R_i\rangle \mid \langle R_1, \ldots, R_k\rangle \in \mathbf{A}\}, \quad \text{where } 1 \le i \le k$$

$$[\![f(q)]\!](\mathbf{A}) := \{\langle R_1, \ldots, R_k, f(R_{k+1})\rangle \mid \langle R_1, \ldots, R_{k+1}\rangle \in [\![q]\!](\mathbf{A})\}, \quad \text{where } f \in \{\pi_U, \sigma_\phi, \delta_{U\to V}\}$$

$$[\![q_1\ Op\ q_2]\!](\mathbf{A}) := \{\langle R_1, \ldots, R_k, R^1_{k+1}\ Op\ R^2_{k+1}\rangle \mid \langle R_1, \ldots, R_k, R^1_{k+1}\rangle \in [\![q_1]\!](\mathbf{A}), \langle R_1, \ldots, R_k, R^2_{k+1}\rangle \in [\![q_2]\!](\mathbf{A})\}, \quad \text{where } Op \in \{\times, \cup, \cap, -\}$$

$$[\![\chi_A(q)]\!](\mathbf{A}) := \{\langle R_1, \ldots, R_k, \sigma_{A=v}(R_{k+1})\rangle \mid \langle R_1, \ldots, R_{k+1}\rangle \in [\![q]\!](\mathbf{A}), (v \in \pi_A(R_{k+1}) \vee (R_{k+1} = \emptyset \Rightarrow v = 1))\}$$

$$[\![p\gamma^V_\sim(q)]\!](\mathbf{A}) := \left\{\left\langle R_1, \ldots, R_k, \bigcup\{\pi_V(R'_{k+1}) \mid \mathcal{A}' = \langle R_1, \ldots, R_k, R'_{k+1}\rangle \in [\![q]\!](\mathbf{A}),\ \mathcal{A} \sim \mathcal{A}'\}\right\rangle \mid \mathcal{A} = \langle R_1, \ldots, R_{k+1}\rangle \in [\![q]\!](\mathbf{A})\right\}$$

$$[\![c\gamma^V_\sim(q)]\!](\mathbf{A}) := \left\{\left\langle R_1, \ldots, R_k, \bigcap\{\pi_V(R'_{k+1}) \mid \mathcal{A}' = \langle R_1, \ldots, R_k, R'_{k+1}\rangle \in [\![q]\!](\mathbf{A}),\ \mathcal{A} \sim \mathcal{A}'\}\right\rangle \mid \mathcal{A} = \langle R_1, \ldots, R_{k+1}\rangle \in [\![q]\!](\mathbf{A})\right\}$$

$$[\![p\gamma^V_U(q)]\!](\mathbf{A}) := [\![p\gamma^V_{\pi_U(R_{k+1})=\pi_U(R'_{k+1})}(q)]\!](\mathbf{A}) \qquad [\![poss(q)]\!](\mathbf{A}) := [\![p\gamma^*_{\text{true}}(q)]\!](\mathbf{A})$$

$$[\![c\gamma^V_U(q)]\!](\mathbf{A}) := [\![c\gamma^V_{\pi_U(R_{k+1})=\pi_U(R'_{k+1})}(q)]\!](\mathbf{A}) \qquad [\![cert(q)]\!](\mathbf{A}) := [\![c\gamma^*_{\text{true}}(q)]\!](\mathbf{A})$$

Figure 3: Semantics of world-set algebra (with auxiliary definitions of $p\gamma^V_\sim$ and $c\gamma^V_\sim$).

EXAMPLE 4.1. The first query of Section 2 asking for possible acquisition targets can be expressed in world set algebra as

$$poss(\pi_{CID}(\sigma_{\text{Skill}=\text{'Web'}}(c\gamma^*_{CID}(\pi_{1.CID,1.EID}(\chi_{CID,EID}(\text{Company\_Emp}) \bowtie_{1.CID=2.CID \wedge 1.EID \neq 2.EID} \text{Company\_Emp}) \bowtie \text{Emp\_Skills}))))). \quad \square$$

Operator Typing. We type the operators based on the cardinality of their input and output world-sets. An operator has type $1 \mapsto 1$ if it is a mapping between singleton world-sets, $1 \mapsto m$ if it maps singleton world-sets to world-sets, $m \mapsto 1$ if it maps world-sets to singleton world-sets, and $m \mapsto m$ if it is a mapping between world-sets. We also allow type overloading for operators.

The operators of the world-set algebra can then be characterized as follows. The relational algebra operators and the world grouping operators are either $1 \mapsto 1$ or $m \mapsto m$. The operators poss and cert have the type $m \mapsto 1$. Finally, the type of the operator choice-of is either $1 \mapsto m$ or $m \mapsto m$. The type of a world-set algebra query can then be checked statically using standard typechecking inference rules. For example, all queries from Section 2 have type $1 \mapsto 1$, because they start with a singleton world-set and their outermost operators are either poss or cert.

The usefulness of query typing becomes evident in Section 5, where we study query evaluation on an inlined representation of world-sets. There, we use the type to decide whether the query maps to a singleton world-set.

Extending World-set Algebra. The I-SQL operator repair-by-key enforces a uniqueness constraint for the values of a given set of attributes. We note here that by adding this operator to world-set algebra, we can express complex guess and check programming tasks. For example, one can easily reduce the 3-colorability problem to the evaluation of a world-set algebra query.

PROPOSITION 4.2. The evaluation of queries in world-set algebra with repair-by-key is NP-hard.

In contrast, the evaluation of queries in relational algebra is in PTIME w.r.t. data complexity.

4.2 Genericity

Genericity is a fundamental property of query languages. It guarantees that query results are independent from the representation of the data and interpretation of domain values. While relational query languages such as relational algebra and SQL are generic, this is not always obvious in possible worlds query languages. In fact, as we will see later, this property does not hold for some languages for querying incompleteness. World-set algebra is generic, i.e., its semantics does not depend on the world-set representation. We next define the notion of isomorphism on world-sets, which we later use to formally define genericity.

DEFINITION 4.3. Given two world-sets $\mathbf{A} = \{I_1, \ldots, I_n\}$ and $\mathbf{A}' = \{I'_1, \ldots, I'_n\}$ and a bijection $\theta : \mathbf{dom}^{\mathbf{A}} \to \mathbf{dom}^{\mathbf{A}'}$ between their domains, $\mathbf{A}$ and $\mathbf{A}'$ are *isomorphic under* θ, denoted $\mathbf{A} \cong_\theta \mathbf{A}'$, iff

$$(\forall I)\ I \in \mathbf{A} \Rightarrow \theta(I) \in \mathbf{A}' \quad \text{and} \quad (\forall I')\ I' \in \mathbf{A}' \Rightarrow \theta^{-1}(I') \in \mathbf{A}.$$

Analogous to the relational case [1], we now define genericity for the case of world-sets.

DEFINITION 4.4. A query q is *generic* iff for all world-sets $\mathbf{A}$, $\mathbf{A}'$ there is a domain value bijection $\theta : \mathbf{dom}^{\mathbf{A}} \to \mathbf{dom}^{\mathbf{A}'}$ such that

$$\mathbf{A} \cong_\theta \mathbf{A}' \Rightarrow q(\mathbf{A}) \cong_\theta q(\mathbf{A}').$$

A query language is generic iff all of its queries are generic.

The above definition ignores the issue of constants in queries (in selection conditions). However, it can be easily generalized, cf. [1].

PROPOSITION 4.5. World-set algebra is generic.

PROOF SKETCH. Consider the world-sets $\mathbf{A} = \{I_1, \ldots, I_n\}$ and $\mathbf{A}' = \{I'_1, \ldots, I'_n\}$, which are isomorphic under a bijection θ: $\mathbf{A} \cong_\theta \mathbf{A}'$. W.l.o.g. let the schema of I_j and I'_j be $\langle R_1, \ldots, R_k\rangle$ and $\theta(I_j) = I'_j$, for $1 \le j \le n$. It can be shown using induction on the query structure that any query q in world-set algebra is generic, i.e., $q(\mathbf{A}) \cong_\theta q(\mathbf{A}')$. We show this for two operators: projection and poss.

The query $\pi_B(R)$ creates in each world of $\mathbf{A}$ (or $\mathbf{A}'$) a new relation R_{k+1} (R'_{k+1}) representing the projection of B on R. Then, for any world I_j we have $\theta(\pi_B(R^{I_j})) = \pi_B(\theta(R^{I_j})) = \pi_B(R^{I'_j})$. The projection commutes with θ because relational algebra is generic on complete databases.

The query $poss(R)$ creates in each world of $\mathbf{A}$ (or $\mathbf{A}'$) a new relation R_{k+1} (R'_{k+1}) representing the union of relations R over all worlds: $R_{k+1} = \bigcup(R^{I_j})$ and $R'_{k+1} = \bigcup(R^{I'_j})$. By taking θ on R_{k+1} we obtain

$$\theta(\bigcup(R^{I_j})) = \theta(R^{I_1}) \cup \ldots \cup \theta(R^{I_k}) = R^{I'_1} \cup \ldots \cup R^{I'_k} = \bigcup(R^{I'_j}).$$

Again, θ commutes with union because relational algebra is generic. $\square$

REMARK 4.6. To demonstrate that genericity can be a subtle issue in the context of queries on world-sets, we show that the language TriQL [9, 23] lacks genericity. Consider two ULDBs [8] (i.e., databases with lineage and uncertainty) U_1 and U_2:

R^T	A	V
	1	1
	3	1
	1	2

W^T	V
	1
	2
	3

R^1	A
	1
	3

R^2	A
	1

R^3	A

(a) Inlined representation. (b) Represented worlds.

Figure 4: Inlined representation of a world-set.

U_1	R(A)	λ	
t_1	(1) ‖ (2)	{}	?

U_2	R(A)	λ	
t_1	(1)	$\{(s_1, 1)\}$	?
t_2	(2)	$\{(s_1, 2)\}$	?

U_1 has one maybe x-tuple with id t_1, no lineage λ, and two alternatives (1) and (2). U_2 has two maybe x-tuples, each with one alternative. The lineage of the first x-tuple points to the first alternative of an external x-tuple with id s_1, whereas the lineage of the second x-tuple points to the second alternative of the same x-tuple s_1. Alternatives of the same x-tuple, or x-tuples whose lineage points to different alternatives of the same tuple cannot appear in the same world. Also, maybe x-tuples may be missing from a world. Both U_1 and U_2 represent the same set of three worlds $\mathcal{A}, \mathcal{B}, C$:

$R^{\mathcal{A}}$	A
	1

$R^{\mathcal{B}}$	A
	2

R^C	A

Let q be a TriQL query which uses horizontal selection $[\cdot]$:

```
select * from R where
    exists [select * from R r1, R r2 where r1.A <> r2.A];
```

The meaning of this query is that an x-tuple from R is selected if it has at least two different alternatives. It is an adaptation of a query example from [23] which uses count to express the same condition. On U_1 the query q is the identity, while on U_2 it does not select anything:

$q(U_1)$	R(A)	λ	
t_1	(1) ‖ (2)	{}	?

$q(U_2)$	R(A)	λ	

It is easy to see that the identity isomorphism on the input world-sets does not hold on the world-sets representing the answers to q in the two cases. □

5. FROM WORLD-SET ALGEBRA TO RELATIONAL ALGEBRA

In this section we show that any world-set algebra query can be efficiently translated to an equivalent relational algebra query over a *complete* representation of the input world-set. We propose such a representation in Section 5.1, the *inlined representation*, where the tuples of a relation over all worlds are represented in one table that has special attributes to denote the identifier of the world each tuple belongs to.

In the case of complete-to-complete world-set algebra queries (i.e., those with type $1 \mapsto 1$), the input database is not encoded as an inlined representation. Complete-to-complete world-set algebra queries admit equivalent relational algebra queries that operate directly on standard databases (thus the inlined representation is not needed). However, the intermediate results of the operators present in the relational algebra query can make use of the inlined representation to keep track of which tuples belong to which world. This result is in the spirit of the translation of flat-to-flat nested algebra queries to relational algebra queries [20], and shows that world-set algebra is conservative over relational algebra.

5.1 Inlined Representation of World-sets

We next define a representation of world-sets, where all instances of a relation over all worlds are inlined into one single relation. The main idea behind the representation is to assign identifiers to worlds and use them to identify the tuples of any relation in a world. As we will show in Section 5.2, such identifiers need not be provided from the outside. Our world-set algebra queries can gracefully identify worlds based on the choices of values used to create them.

DEFINITION 5.1. An *inlined representation* of a world-set $\mathbf{A}$ over a schema $\Sigma = \langle R_1[U_1], \ldots, R_k[U_k] \rangle$ is a structure

$$T = \langle R_1^T[U_1 \cup V], \ldots R_k^T[U_k \cup V], W[V] \rangle$$

where V is a possibly empty set of attributes disjoint with the sets $U_1, \ldots, U_k$. The attributes in V represent world identifiers. Each table R_i^T encodes the instances of a relation R_i and the table W contains all world identifiers that appear in any table R_i^T.

We allow the world table W to contain identifiers that do not appear in any of the tables R_i^T. (Thus $R_i^T[V] \subseteq W^T[V]$.) In this way, we can encode the existence of an empty world. The empty world-set is encoded by an empty world table. It is easy to see that our representation can encode any finite set of possible worlds.

The set of possible worlds represented by an inlined representation T is defined by the function *rep*

$$rep(T) = \{\langle \pi_{U_1}(\sigma_{V=w}(R_1^T)), \ldots, \pi_{U_k}(\sigma_{V=w}(R_k^T)) \rangle \mid w \in W\}$$

We obtain a possible world by taking all tuples from $R_1^T, \ldots, R_k^T$ with a given world identifier from the world table W. Note that T can encode several equivalent worlds under different world ids.

EXAMPLE 5.2. The inlined representation of Figure 4 (a) encodes the three possible worlds depicted in Figure 4 (b). □

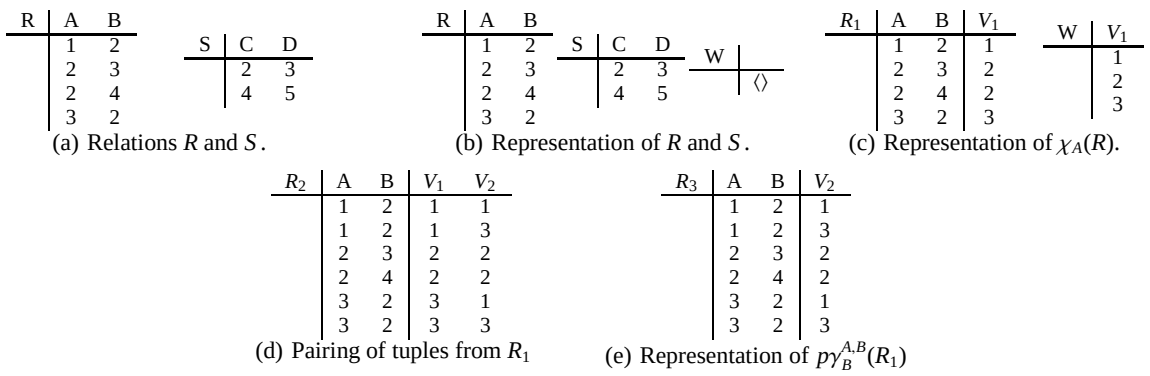
REMARK 5.3. The inlined representation of world-sets supports our encoding of world-set queries as relational queries and is not meant as a compact representation system for very large sets of worlds, as considered in, e.g., [8, 4]. The evaluation of a relational query on an inlined representation can only produce a world-set whose cardinality is polynomial in the input size. This is due to the polynomial data complexity of relational algebra. □

We next show how world-set algebra queries can be evaluated on inlined representations.

EXAMPLE 5.4. Consider the database $\langle R, S \rangle$ of Figure 5 (a) and its inlined representation T of Figure 5 (b).

We first show how to evaluate $R_1 = \chi_A(R)$. By definition, choice-of creates a world for each distinct value of A. An A-value becomes also the id of the world it determines. The world table W is then updated with the new world ids (using a join of R_1 and W) and each other relation (here R and S) is copied in the newly created worlds (using a join of the new W and each of R and S). Figure 5 (c) shows R_1 and the effect on the world tables in the inline representation (R and S are not shown for reasons of space).

Let us now evaluate $R_3 = p\gamma_B^{AB}(R_1)$. Our first step is to pair each tuple with the ids of worlds that have the same projection on B as the world the tuple belongs to. This is accomplished in a division-like manner and is detailed in the next subsection. The set of such pairs is represented by R_2 in Figure 5 (d). Note that the attribute V_2 now represents a group id for the groups of worlds we create. To compute the possible tuples within each group, we only need to remove the old world id column, see relation R_3 in Figure 5 (e). We obtain the result by renaming V_2 to V_1 to match the input set of world-id attributes. The other tables remain unchanged. □



R	A	B
	1	2
	2	3
	2	4
	3	2

S	C	D
	2	3
	4	5

(a) Relations R and S.

R	A	B
	1	2
	2	3
	2	4
	3	2

S	C	D
	2	3
	4	5

W
⟨⟩

(b) Representation of R and S.

R_1	A	B	V_1
	1	2	1
	2	3	2
	2	4	2
	3	2	3

W	V_1
	1
	2
	3

(c) Representation of $\chi_A(R)$.

R_2	A	B	V_1	V_2
	1	2	1	1
	1	2	1	3
	2	3	2	2
	2	4	2	2
	3	2	3	1
	3	2	3	3

(d) Pairing of tuples from R_1

R_3	A	B	V_2
	1	2	1
	1	2	3
	2	3	2
	2	4	2
	3	2	1
	3	2	3

(e) Representation of $p\gamma_B^{A,B}(R_1)$

Figure 5: Evaluating world-set algebra queries on inlined representation of world-sets.

5.2 World-set to relational algebra translation

Figure 6 defines the translation function $[\![\cdot]\!]_\tau$ recursively on the structure of world-set queries. The function $[\![\cdot]\!]_\tau$ takes a world-set query and an inlined representation $T = \langle R_1, \ldots, R_k, W\rangle$ to a new inlined representation $T' = \langle R'_1, \ldots, R'_{k+1}, W'\rangle$, where each table R'_i is obtained from R_i using a relational algebra query, and R'_{k+1} encodes the answer to the input query. In case an operator creates new worlds, the world table W is updated to also represent their ids. Also, the tables R_1 to R_k and W are copied in each of the new worlds. The result of the translation is then the composition of those relational algebra queries generated for each operator and used to define the answer of the world-set query.

We make use of the following naming convention. For any table in the inlined representation, the set of attributes defining the world identifiers is denoted by V (possibly with indices) and called id attributes. The remaining attributes, which define the values in the relations encoded by the tables, are denoted by D and called value attributes. This is with no loss of generality, as the id and value attributes can be statically inferred from the input query.

REMARK 5.5. For the translation of choice-of we use a slightly modified left outer join (denoted $\Rightarrow\!\bowtie$):

$$R \Rightarrow\!\bowtie S = R \bowtie S \cup (R - R \ltimes S) \times \{\langle c, \ldots, c\rangle\}$$

The result of the left outer join contains all combinations of tuples of R and S that fulfill the join condition, and the tuples from R without a join partner from S padded[1] with a special constant c up to size $|R \bowtie S|$. □

We next explain our translation. The base case of the translation is the identity on a relation R, in which case we create the $(k+1)$st table encoding R. This is the only case where the representation is extended. In all other cases, the operators are translated to relational algebra queries that modify the $(k+1)$st table. The relational algebra unary operators selection, projection, and renaming are applied to the table R_{k+1} from the representation obtained by translating their subqueries. In the case of projection, we also keep the id attributes of R_{k+1}.

According to the semantics, choice-of is the only operator that can create new worlds, see Section 4. On inlined representations, we simulate a choice-of $\chi_B(q)$ by extending the table R_{k+1} with new id attributes $V_{R.B}$, whose values are the same as for B. Then the tuples with the same values for B are in the same world (as ensured by the semantics of choice-of) or, equivalently, with the same world identifier. Note that in this way, we do not need the expressive power to create new world identifiers, as our world ids are values already existing in the data. We also update the world table with the new world ids and copy each other relation in the new worlds created by the choice-of operator. An evaluation example with choice-of is given in Example 5.4.

A query $poss(q)$ can be expressed on inlined representations by simply dropping the id attributes from the table R_{k+1}. The obtained relation is then copied in all worlds using a product of this relation and the world table W.

A query $cert(q)$ requires to compute the tuples in R_{k+1} that appear in all worlds. This can be expressed by a division of R_{k+1} and the world table W, which consists of all world ids. The certain tuples are then copied in all worlds.

EXAMPLE 5.6. Consider the trip-planning I-SQL query from Section 2 that finds the cities that are a common destination to all departures. We can express the query in world-set algebra as

$$cert(\pi_{\text{Arr}}(\chi_{\text{Dep}}(\text{HFlights}))).$$

We next show how this query can be translated to a relational algebra query.

1. We represent the initial database as an inlined representation $\langle \text{HFlights}, W\rangle$, where $W = \{\langle\rangle\}$ is a nullary world-id table with a single tuple.

2. The innermost query, the reference to the HFlights relation extends the input database with a new relation F which is a copy of the relation HFlights: $\langle \text{HFlights}, F, W\rangle$.

3. The choice-of operator χ_{Dep} adds a copy of the Dep attribute of F as a new id attribute V_{Dep} to F: $F' = \pi_{*,\text{Dep as } V_{\text{Dep}}}(F)$. The new world table is computed as $W' = W \Rightarrow\!\bowtie F'$ and the world ids are propagated to the input HFlights table: $\langle \text{HFlights} \bowtie W', F', W'\rangle$.

4. The projection computes $F'' = \pi_{\text{Arr},V_{\text{Dep}}}(F')$ and outputs $\langle \text{HFlights} \bowtie W', F'', W'\rangle$.

5. Finally cert performs the division on F'' with the current world table W': $F''' = (F'' \div W') \times W'$ and outputs $\langle \text{HFlights} \bowtie W', F''', W'\rangle$.

6. Since the query is a $1 \mapsto 1$ query, we obtain the final result by dropping the id attributes of the last computed relation: $\pi_{\text{Arr}}(F''')$.

[1] In the standard definition of outer joins the tuples are padded with null values, whereas here we use a constant for practical reasons.

$[\![R_i]\!]_\tau(\langle R_1,\ldots,R_k,W\rangle) = \langle R_1,\ldots,R_k,R_i,W\rangle$

$[\![f(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![q]\!]_\tau(T)$
  in $\langle R_1,\ldots,R_k,f(R),W\rangle$
where $f \in \{\sigma_{A=x}, \delta_{A\to A'}\}$

$[\![\pi_A(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![q]\!]_\tau(T)$
  in $\langle R_1,\ldots,R_k,\pi_{A,V}(R),W\rangle$

$[\![\chi_B(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![q]\!]_\tau(T)$,
    D be the value attributes of R (and $B \subseteq D$),
    $W' = W \rtimes \delta_{B\to V_B}(\pi_B(R))$
  in $\langle R_1 \bowtie W',\ldots,R_k \bowtie W', \pi_{D,V,B \text{ as } V_B}(R), W'\rangle$

$[\![poss(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![q]\!]_\tau(T)$,
  in $\langle R_1,\ldots,R_k,\pi_D(R)\times W, W\rangle$

$[\![cert(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![q]\!]_\tau(T)$
  in $\langle R_1,\ldots,R_k,(R \div W)\times W, W\rangle$

$[\![\gamma^B_A(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![q]\!]_\tau(T)$,
    $S = \pi_{V,V_2}(\pi_{A,V}(R)\times\pi_{V_2}(\delta_{V\to V_2}(R)) -$
      $\pi_{A,V,V_2}(R \bowtie_{A=A'} (\delta_{A\to A',V\to V_2}(R))))$,
    $S' = \pi_V(R)\times\pi_{V_2}(\delta_{V\to V_2}(R)) - S$,
    $R' = \pi_{B,V,V_2}(R\times\pi_{V_2}(\delta_{V\to V_2}(R)) \bowtie_{R.V=S'.V\wedge R.V_2=S'.V_2} S')$
  in $\langle R_1,\ldots,R_k,R',W\rangle$

$[\![p\gamma^B_A(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![\gamma^B_A(q)]\!]_\tau(T)$,
    V be the attributes of W,
    V, V_2 be the id attributes of R,
  in $\langle R_1,\ldots,R_k,\pi_{A,V}(\delta_{V_2\to V}(R)),W\rangle$

$[\![c\gamma^B_A(q)]\!]_\tau(T) =$
  let $\langle R_1,\ldots,R_k,R,W\rangle = [\![\gamma^B_A(q)]\!]_\tau(T)$,
    V be the attributes of W,
    V, V_2 be the id attributes of R,
    $P = \pi_{B,V,V_2}(R \bowtie_{B=B'\wedge V_2=V'_2} \delta_{B\to B',V\to V',V_2\to V'_2}(R))$,
    $P' = \pi_{B,V,V_2}(R \bowtie_{V_2=V'_2} \delta_{B\to B',V\to V',V_2\to V'_2}(R)) - P$,
    $R' = \pi_{B,V}(\delta_{V_2\to V}(R)) - \pi_{B,V}(\delta_{V_2\to V}(P'))$
  in $\langle R_1,\ldots,R_k,R',W\rangle$

$[\![q_1 \times q_2]\!]_\tau(\langle R_1,\ldots,R_k,W\rangle) =$
  let V be the attributes of W,
    $\langle R'_1,\ldots,R'_k,R',W'\rangle = [\![q_1]\!]_\tau(\langle R_1,\ldots,R_k,W\rangle)$,
    $\langle R''_1,\ldots,R''_k,R'',W''\rangle = [\![q_2]\!]_\tau(\langle R_1,\ldots,R_k,W\rangle)$,
    $W_0 = W' \bowtie W''$
  in $\langle R_1 \bowtie W_0,\ldots,R_k \bowtie W_0, R' \bowtie_{R'.V=R''.V} R'', W_0\rangle$

$[\![q_1 \,\Theta\, q_2]\!]_\tau(\langle R_1,\ldots,R_k,W\rangle) =$
  let $\langle R'_1,\ldots,R'_k,R',W'\rangle = [\![q_1]\!]_\tau(\langle R_1,\ldots,R_k,W\rangle)$,
    $\langle R''_1,\ldots,R''_k,R'',W''\rangle = [\![q_2]\!]_\tau(\langle R_1,\ldots,R_k,W\rangle)$,
    $W_0 = W' \bowtie W''$,
    $R = (R' \bowtie W_0)\,\Theta\,(R'' \bowtie W_0)$
  in $\langle R_1 \bowtie W_0,\ldots,R_k \bowtie W_0, R, W_0\rangle$
where $\Theta \in \{\cup,\cap,-\}$

Figure 6: Translation of world-set queries to relational queries.

We obtain the relational algebra query by composing the queries we used to compute each of the intermediate results that produced the final result:

$$\pi_{\text{Arr}}((\pi_{\text{Arr},V_{\text{Dep}}}(\pi_{*,\text{Dep as } V_{\text{Dep}}}(\text{HFlights})) \div q_w) \times (q_w))$$

where $q_w = (\{\langle\rangle\} \rtimes (\pi_{*,\text{Dep as } V_{\text{Dep}}}(\text{HFlights})))$. □

The group-worlds-by operators first group the instances of R_{k+1} that agree on the values of the grouping attributes A. This grouping can be expressed using universal quantification over all pairs of worlds and over the values of the grouping attributes of the paired worlds. For this, we first compute all pairs of world ids that belong to different groups (as given by S in Figure 6). These are the ids of worlds that produce different answers to the projection on A. We then compute all valid pairs of world ids by computing the possible pairings and subtracting those that are invalid (as given by S'). Note that S' is an equivalence relation over world ids. A pairs of world ids (w_1, w_2) is in the equivalence relation iff w_1 and w_2 are in the same group. Then w_1 represents the id of a world, whereas w_2 represents the id of the group. Finally, to compute the possible tuples in each group (as given by R'), we join R_{k+1} with the set of valid pairings we computed as S'.

To simulate poss-group-worlds-by, we keep the ids of the group and drop the ids of the worlds. To simulate cert-group-worlds-by, we further define relation P as the tuples with matches in some world of their group, and then P' as the tuples that do not have a match for some world of their group. Then, we obtain the tuples that appear in each world within a group by subtracting those defined by P' from those defined by R_{k+1}. An evaluation example with group-worlds-by is given in Example 5.4.

The translation of a product of two queries is the join on the world id attributes of the translations of each of the two queries. Thus we ensure that only tuples coming from the same original world are combined. As a desirable side effect, this join also produces the possible combinations of worlds from the two queries. In the case of union, we need to ensure that the union occurs in each world defined by any of the two operands. Therefore, we need to define the two sets of possible worlds of the operands, copy the relations represented by one operand in each world of the other, and then perform the union. The cases of intersection and difference are treated similarly.

Figure 6 shows how to translate world-set algebra operators into relational algebra queries on inlined representations. Given a $1 \mapsto 1$ world-set algebra query, we obtain the equivalent relational algebra query by composing the queries used at intermediate steps towards the final result. Since the typing guarantees that the query produces a single world as output, the last operator of the relational algebra query projects away the id attributes created by any nested operators.

Finally, we can state the main result of this section.

THEOREM 5.7. *World-set algebra is conservative over relational algebra, that is, for each $1 \mapsto 1$ query q in world-set algebra and a complete database $\mathcal{A}$, there exists a relational algebra query q' such that*

$$\{q'(\mathcal{A})\} = q(\{\mathcal{A}\}).$$

Note that using the translation function $[\![\cdot]\!]_\tau$, any world-set algebra query can be translated into a relational algebra query of polynomial size.

5.3 Optimized translation for complete-to-complete queries

Section 5.2 gives a general translation for world-set queries to relational algebra queries. We notice that in the case of complete-to-complete queries, this translation can be changed so as to produce more optimized equivalent relational algebra queries.

These optimizations are based on the following observations. First, the general translation makes extensive use of the world table W, although this table is only needed in the case of cert and the binary operators union, intersect, and difference. We can adopt here a lazy approach and compute the world table on demand. Second, we can postpone and sometimes avoid the creation of copies of existing relations in all worlds. In this way in case the input world-set query is a relational algebra query, the translation naturally produces that relational algebra query as output, thus avoiding computation of world ids.

Our observations are supported by the following new interpretation of the tables in an inlined representation. If a table has no world identifiers, then the relation it encodes appears in all worlds. Two tables R^T and S^T can have different sets of world identifiers, say a set of n_R worlds and the other set with n_S worlds, in which case the representation encodes $n_R \cdot n_S$ worlds for each possible combination of a world for R^T and a world for S^T. This is correct, as all worlds created by any operator derive from the same input world corresponding to the input database.

We next explain how the world table can be computed on demand. The binary operators and choice-of update the world table W. The choice-of construct creates new ids based on the choice attributes, which can be expressed using a projection. For example the world-ids created by the query $\chi_A(R)$ can be computed with $\pi_A(R)$. In the case of a binary operator $q_1 \ \Theta \ q_2$ the new world ids can be obtained by combining the ones produced by the two subexpressions. If q_1' and q_2' are the queries that compute the ids created in q_1 and q_2, respectively, the ids after the binary operator Θ can be retrieved using the query $q_1' \times q_2'$. Thus we avoid the creation and updates to W until a reference to it is made. When this happens, one can compute W with an expression which can be statically inferred from the structure of the subquery.

EXAMPLE 5.8. Using the translation optimized for complete-to-complete queries, the query of Example 5.6 can be translated to

$$\pi_{\text{Arr,Dep}}(\text{HFlights}) \div \pi_{\text{Dep}}(\text{HFlights}).$$

This relational algebra query is simpler than the one created using the general translation and discussed in Example 5.6. □

6. ALGEBRAIC EQUIVALENCES

Figure 7 gives equivalences of queries in world-set algebra. We turn each equivalence $l = r$ into a rewrite rule $l \rightarrow r$ and define logical optimizations in close analogy to the relational case.

We define two broad classes of equivalences. The first class covers the cases of pairs of operators that commute. The second class covers simplifications of operator compositions.

Commute rules. This class reminds of the standard optimization techniques of relational algebra like pushing selections and projections as down as possible in the logical query plan. The operator poss commutes with selection, projection, and union (Eq. (1) through (3)). The operator cert commutes with selection, intersection, and product (Eq. (4) through (6)). The operator choice-of commutes with product and with projection in the case the projection list includes the choice attributes (Eq. (7) and (8)). Finally, the operators group-worlds-by commute with selection in case the selection condition refers only to attributes among the grouping and projecting attributes (Eq. (9) and (10)).

Commute

$$poss(\sigma_\phi(q)) = \sigma_\phi(poss(q)) \quad (1)$$
$$poss(\pi_X(q)) = \pi_X(poss(q)) \quad (2)$$
$$poss(q_1 \cup q_2) = poss(q_1) \cup poss(q_2) \quad (3)$$
$$cert(\sigma_\phi(q)) = \sigma_\phi(cert(q)) \quad (4)$$
$$cert(q_1 \cap q_2) = cert(q_1) \cap cert(q_2) \quad (5)$$
$$cert(q_1 \times q_2) = cert(q_1) \times cert(q_2) \quad (6)$$
$$\pi_{X\cup Y}(\chi_X(q)) = \chi_X(\pi_{X\cup Y}(q)) \quad (7)$$
$$\chi_X(q_1) \times q_2 = \chi_X(q_1 \times q_2), \text{ if } X \subseteq Attrs(q_1) \quad (8)$$
$$\sigma_\phi(p\gamma^Y_X(q)) = p\gamma^Y_X(\sigma_\phi(q)), \text{ if } Attrs(\phi) \subseteq X \cap Y \quad (9)$$
$$\sigma_\phi(c\gamma^Y_X(q)) = c\gamma^Y_X(\sigma_\phi(q)), \text{ if } Attrs(\phi) \subseteq X \cap Y \quad (10)$$

Reduce

$$poss(\chi_X(q)) = poss(q) \quad (11)$$
$$p\gamma^X_{X\cup Y}(q) = c\gamma^X_{X\cup Y}(q) = \pi_X(q) \quad (12)$$
$$\pi_Z(p\gamma^{Y\cup Z}_{X\cup Z}(q)) = \pi_Z(q) \quad (13)$$
$$\pi_Z(p\gamma^{Y\cup Z}_X(q)) = p\gamma^Z_X(q), \text{ if } Z \not\subseteq X \quad (14)$$
$$poss(p\gamma^Y_X(q)) = poss(\pi_Y(q)) \quad (15)$$
$$cert(c\gamma^Y_X(q)) = cert(\pi_Y(q)) \quad (16)$$
$$\chi_X(\chi_Y(q)) = \chi_Y(\chi_X(q)) = \chi_{X\cup Y}(q) \quad (17)$$
$$p\gamma^Y_X(p\gamma^{X\cup Y\cup Z}_{X\cup V}(q)) = c\gamma^Y_X(p\gamma^{X\cup Y\cup Z}_{X\cup V}(q)) = p\gamma^Y_{X\cup V}(q) \quad (18)$$
$$p\gamma^Y_X(c\gamma^{X\cup Y\cup Z}_{X\cup V}(q)) = c\gamma^Y_X(c\gamma^{X\cup Y\cup Z}_{X\cup V}(q)) = \pi_Y(c\gamma^{X\cup Y\cup Z}_{X\cup V}(q)) \quad (19)$$
$$p\gamma^Y_{X\cup Z}(\chi_X(q)) = c\gamma^Y_{X\cup Z}(\chi_X(q)) = \pi_Y(\chi_X(q)) \quad (20)$$
$$poss(cert(q)) = cert(cert(q)) = cert(q) \quad (21)$$
$$poss(poss(q)) = cert(poss(q)) = poss(q) \quad (22)$$

Figure 7: Equivalences in World-set Algebra.

In the case of relational algebra the projections and selections are usually pushed down in the query plan. In world-set algebra the pushing down of the new operators poss and cert usually bears even greater potential for optimization, and these operators are pushed down even across selection and projection where this is possible. This is because the latter operators close the possible worlds semantics and can eliminate other world-set operators like choice-of or group-worlds-by, as discussed below for the reduce equivalences.

The most interesting cases of pairs of operators that do not commute are selection and choice-of on one hand and and product and poss on the other. The choice-of operator applied on top of a selection can create a world with an empty relation only in the case where the answet to the selection is empty. However, the selection applied on top of a choice-of operator creates an empty relation in a world if there exists at least one tuple that does not match the selection criteria. In the case of poss and product, tuples from different worlds can be joined if the worlds are flattened using poss before the product, and only tuples from the same world are paired if poss is applied after product. In both cases the operators poss and selection cannot be always pushed down to the relations in a plan. The apparent drawback of the former case has, however, an interesting and useful side-effect that allows to easily specify properties that must hold for all worlds.

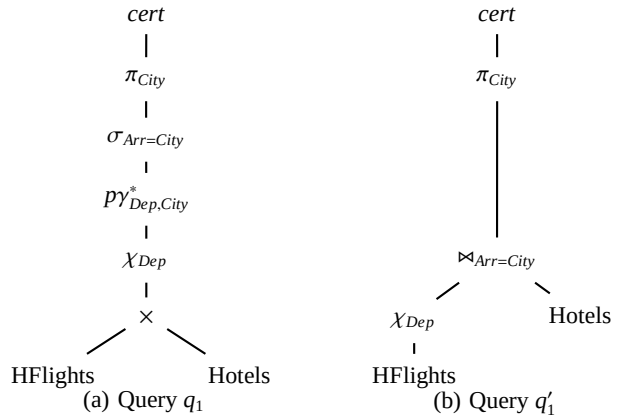


Figure 8: Equivalent queries q_1 and q'_1 of Example 6.1.

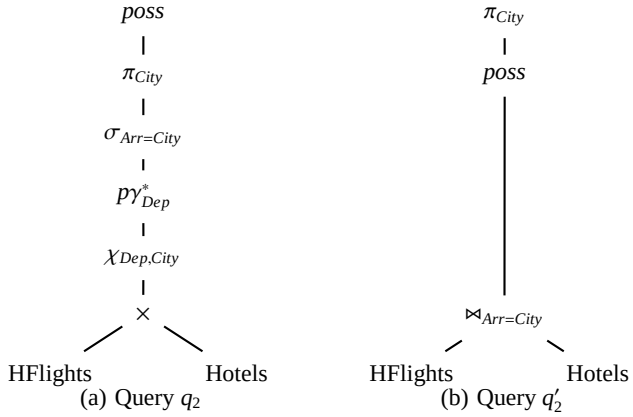


Figure 9: Equivalent queries q_2 and q'_2 of Example 6.2.

Reduce rules. The operator poss eliminates choice-of operators, because choice-of distributes tuples into a set of disjoint worlds, which is later flattened by the operator poss (Eq. (11)). In the same way, poss can undo world grouping (Eq. (15)). The operator cert can only undo world grouping expressed using certain-group-worlds-by, because each group is formed on projections that are certain in all worlds in that group (Eq. (16)). Similarly, Eq. (12) shows that special cases of world grouping can be expressed as simple projections. This is because if all worlds in a group agree on a projection $\pi_{X\cup Y}$, then π_X is the same in all these worlds and its evaluation in each world expresses precisely the computation of π_X in each group. Nested choice-of operators can be reduced to a single choice-of (Eq. (17)). By Eq. (8) and Eq. (17), we can derive that $\chi_X(q_1)\times\chi_Y(q_2) = \chi_{X\cup Y}(q_1\times q_2)$.

Eq. (18) and (19) show that nested group-worlds-by can also be reduced to a single group-worlds-by in case all grouping and projecting attributes of the outer operator occur as grouping and projecting attributes, respectively, of the inner operator. Similarly, a projection applied to a world grouping can eliminate the grouping, in case the attributes of the projection occur as both grouping and projecting attributes of the group-worlds-by operator (Eq. (13)). In case the attributes of a projection occur as projecting (but not as grouping) attributes of the group-worlds-by operator, then the projection is removed and its attributes become the new projecting attributes (Eq. (14)). In the presence of choice-of operators, the group-worlds-by operators are reduced to simple projections in case the choice attributes are also grouping attributes, as shown in Eq. (20). Eq. (21) and (22) consider the case of redundant poss or cert operators.

We next use these equivalences to rewrite world-set algebra queries.

Example 6.1. Consider a possibly incomplete version of our HFlights database from Section 2, where additionally we have information on hotels given by relation Hotels(Name,City,Price). The participants of our meeting would all like to book a direct flight to and a hotel in the same city ('*' stands for all attributes of a relation):

$$q_1 = cert(\pi_{City}(\sigma_{Arr=City}(p\gamma^*_{Dep,City}(\chi_{Dep}(HFlights\times Hotels)))))$$

Query q_1 first considers each possible departure, then groups the possibilities on common departures and arrivals, and finally computes the arrivals (cities) common to the departures.

Query q_1 can be rewritten as follows.

$$\begin{aligned} q'_1 &\overset{(20)}{=} cert(\pi_{City}(\sigma_{Arr=City}(\pi_*(\chi_{Dep}(HFlights\times Hotels))))) \\ &\overset{(8)}{=} cert(\pi_{City}(\sigma_{Arr=City}(\pi_*(\chi_{Dep}(HFlights)\times Hotels)))) \\ &= cert(\pi_{City}(\chi_{Dep}(HFlights)\bowtie_{Arr=City} Hotels)). \end{aligned}$$

In constructing q'_1, we absorbed the group-worlds-by operator in the choice-of operator and a projection on all attributes (*), we pushed down the choice-of operator and transformed the product into a join. Figure 8 shows the logical query plans for q_1 and q'_1. □

Example 6.2. Let us now consider a slight modification of q_1 where the operator cert is replaced by poss, and the choice is on each combination of departures and arrivals, followed by grouping on departures:

$$q_2 = poss(\pi_{City}(\sigma_{Arr=City}(p\gamma^*_{Dep}(\chi_{Dep,City}(HFlights\times Hotels)))))$$

Query q_2 can be rewritten as follows:

$$\begin{aligned} &\overset{(1,2)}{=} \pi_{City}(\sigma_{Arr=City}(poss(p\gamma^*_{Dep}(\chi_{Dep}(HFlights\times Hotels))))) \\ &\overset{(15),(2)}{=} \pi_{City}(\sigma_{Arr=City}(\pi_*(poss(\chi_{Dep}(HFlights\times Hotels))))) \\ &\overset{(11)}{=} \pi_{City}(\sigma_{Arr=City}(\pi_*(poss(HFlights\times Hotels)))) \\ &\overset{(1)}{=} \pi_{City}(\pi_*(poss(HFlights\bowtie_{Arr=City} Hotels))) \\ &= \pi_{City}(poss(HFlights\bowtie_{Arr=City} Hotels)). \end{aligned}$$

Note that q'_2 has no grouping or choice constructs. In case the input data is complete, the operator poss can be dropped and q'_2 becomes a relational algebra query, as one would expect. Figure 9 shows the logical query plans for q_2 and q'_2. □

So far we have not addressed the difference operation of relational algebra. Apart from rewrite rules involving difference and the other operators of relational algebra, the following equivalence is of interest:

$$cert(R - S) = cert(cert(R) - S). \tag{23}$$

Furthermore, cert and poss are mutually expressible using difference and a domain relation D which holds the values that appear in the union of all the worlds.

Proposition 6.3.

$$\begin{aligned} cert(Q) &= Q - poss(D^{\text{arity(Q)}} - Q) \\ &= Q - poss(poss(Q) - Q) && (24) \\ poss(Q) &= D^{\text{arity(Q)}} - cert(D^{\text{arity(Q)}} - Q). && (25) \end{aligned}$$

7. FURTHER EXPRESSIVENESS ISSUES

We have shown that each world-set algebra query is generic and can be translated to an equivalent relational algebra query on inlined representations. The converse is not true: Consider world-sets whose schema contains just one relation R and a query that computes all pairs of worlds in a world-set, i.e., which, for each world I and every choice of *other world* J, creates a world containing the relation R^I and, renamed, the relation R^J. This query is generic and expressible in relational algebra on inlined representations. However, it is not expressible in world-set algebra: If $|poss(R)| = n$ and the world-set consists of all 2^n subsets of this set, the pairing query will compute a world-set of cardinality 2^{2n}, too great to be produced in world-set algebra using a fixed query and choice-of as the only operation to increase the number of worlds.

It is an interesting open question whether all queries of the types $1 \mapsto 1$ and $m \mapsto 1$ in world-set algebra extended by the world-pairing operation are expressible in world-set algebra.

We did not add a world-pairing operation to world-set algebra because it would take away from the intuition of queries being evaluated on each world individually, with an occasional look outside the world. Also, the pairing operation cannot replace choice-of: For example, starting with a single world, pairing will not increase the cardinality of the world-set, while choice-of in general does.

8. CONCLUSION

This paper introduces I-SQL, an analog to SQL for the case of incomplete information. An I-SQL query allows for the convenient formulation of "what if" queries and is thus even relevant for queries on complete data. We motivate I-SQL using several application scenarios and point out that many natural queries in such scenarios can be expressed easily in I-SQL and are rather complicated (or not even possible) in SQL.

We then formalize a clean fragment of I-SQL, World-set Algebra, and show its fundamental properties, like genericity and conservativity over relational algebra. From the more practical side of world-set algebra, we give a set of equivalences and show how they can be used to produce more efficient logical query plans. We also investigate the relationship between world-set algebra and relational algebra and give a translation of any world-set query to a relational query over an inlined representation of world-sets. For the case of world-set algebra queries that map between complete databases, we show how the general translation scheme can be improved so that the generated relational queries become shorter.

One future research direction is the implementation of I-SQL on top of a relational engine. In some sense, the optimized translation of complete-to-complete queries to relational queries can provide one way to evaluate such queries in any relational database engine. We believe, however, that query plans with dedicated physical operators for our I-SQL constructs should perform much better than the default relational algebra query over the (nonsuccinct, and thus in practice too large) inlined representation. Another research direction is to implement I-SQL on top of an existing representation system for finite world-sets, like databases with lineage and uncertainty [8] or world-set decompositions [4].

9. REFERENCES

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

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, **78**(1):158–187, 1991.

[3] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *Proc. ICDE*, 2006.

[4] L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *Proc. ICDE*, 2007.

[5] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *Proc. ICDT*, 2007.

[6] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. PODS*, 1999.

[7] M. Arenas, L. E. Bertossi, and J. Chomicki. "Answer sets for consistent query answering in inconsistent databases". *TPLP*, **3**(4–5):393–424, 2003.

[8] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. VLDB*, 2006.

[9] O. Benjelloun, A. D. Sarma, C. Hayworth, and J. Widom. An Introduction to ULDBs and the Trio System. *IEEE Data Engineering Bulletin*, **29**(1):5–16, Mar. 2006.

[10] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. VLDB*, 2004.

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

[12] G. Grahne. Dependency satisfaction in databases with incomplete information. In *Proc. VLDB*, pages 37–45, 1984.

[13] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Number 554 in LNCS. Springer-Verlag, 1991.

[14] T. J. Green and V. Tannen. "Models for Incomplete and Probabilistic Information". In *International Workshop on Incompleteness and Inconsistency in Databases (IIDB)*, 2006.

[15] T. Griffin and R. Hull. "A Framework for Implementing Hypothetical Queries". In *Proc. SIGMOD*, 1997.

[16] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of ACM*, **31**:761–791, 1984.

[17] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects — a data model for design and planning applications. In *Proc. SIGMOD*, pages 288–297, 1991.

[18] N. Leone, G. Greco, G. Ianni, V. Lio, G. Terracina, T. Eiter, W. Faber, M. Fink, G. Gottlob, R. Rosati, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszkis. "The INFOMIX system for advanced integration of incomplete and inconsistent data". In *Proc. SIGMOD*, pages 915–917, 2005.

[19] L. Libkin and L. Wong. Semantic representations and query languages for OR-sets. In *Proc. PODS*, pages 37–48, 1993.

[20] J. Paredaens and D. V. Gucht. Converting nested algebra expressions into flat algebra expressions. *TODS*, **17**(1):65–93, 1992.

[21] M. Poess and C. Floyd. New TPC Benchmarks for Decision Support and Web Commerce. *SIGMOD Record*, **29**(4), 2000.

[22] R. Rantzau and C. Mangold. Laws for rewriting queries containing division operators. In *Proc. ICDE*, 2006.

[23] Stanford Trio Project. *TriQL – The Trio Query Language*, Oct. 2006. http://infolab.stanford.edu/~widom/triql.html.

[24] Transaction Processing Performance Council. *TPC Benchmark H (Decision Support)*, revision 2.6.0 edition, 2006. http://www.tpc.org/tpch/spec/tpch2.6.0.pdf.