Chapter 2

MODELS FOR INCOMPLETE AND PROBABILISTIC INFORMATION

Todd J. Green
Department of Computer and Information Science
University of Pennsylvania
tjgreen@cis.upenn.edu

Abstract We discuss, compare and relate some old and some new models for incomplete and probabilistic databases. We characterize the expressive power of c-tables over infinite domains and we introduce a new kind of result, algebraic completion, for studying less expressive models. By viewing probabilistic models as incompleteness models with additional probability information, we define completeness and closure under query languages of general probabilistic database models and we introduce a new such model, probabilistic c-tables, that is shown to be complete and closed under the relational algebra. We also identify fundamental connections between query answering with incomplete and probabilistic databases and data provenance. We show that the calculations for incomplete databases, probabilistic databases, bag semantics, lineage, and why-provenance are particular cases of the same general algorithms involving semi-rings. This further suggests a comprehensive provenance representation that uses semi-rings of polynomials. Finally, we show that for positive Boolean c-tables, containment of positive relational queries is the same as for standard set semantics.

Keywords: Incomplete databases, probabilistic databases, provenance, lineage, semi-rings

1. Introduction

This chapter provides a survey of models for incomplete and probabilistic information from the perspective of two recent papers that the author has written with Val Tannen [28] and Grigoris Karvounarakis and Val Tannen [27]. All the concepts and technical developments that are not attributed specifically to another publication originate in these two papers.

The representation of incomplete information in databases has been an important research topic for a long time, see the references in [25], in Ch.19 of [2], in [43], in [48, 36], as well as the recent [45, 42, 41, 4]. Moreover, this work is closely related to recently active research topics such as inconsistent databases and repairs [5], answering queries using views [1], data exchange [20], and data provenance [9, 8]. The classic reference on incomplete databases remains [30] with the fundamental concept of c-table and its restrictions to simpler tables with variables. The most important result of [30] is the query answering algorithm that defines an algebra on c-tables that corresponds exactly to the usual relational algebra ($\mathcal{RA}$). A recent paper [41] has defined a hierarchy of incomplete database models based on finite sets of choices and optional inclusion. We shall give below **comparisons** between the models [41] and the tables with variables from [30].

Two criteria have been provided for comparisons among all these models: [30, 41] discuss *closure* under relational algebra operations, while [41] also emphasizes *completeness*, specifically the ability to represent all finite incomplete databases. We point out that the latter is not appropriate for tables with variables over an infinite domain, and we describe another criterion, **$\mathcal{RA}$-completeness**, that fully characterizes the expressive power of c-tables.

We outline a method for the study of models that are not complete. Namely, we consider combining existing models with queries in various fragments of relational algebra. We then ask how big these fragments need to be to obtain a combined model that is complete. We give a number of such **algebraic completion** results.

Early on, probabilistic models of databases were studied less intensively than incompleteness models, with some notable exceptions [10, 6, 39, 34, 17]. Essential progress was made independently in three papers [22, 33, 47] that were published at about the same time. [22, 47] assume a model in which tuples are taken independently in a relation with given probabilities. [33] assumes a model with a separate distribution for each attribute in each tuple. All three papers attacked the problem of calculating the probability of tuples occurring in query answers. They solved the problem by developing more general models in which rows are **annotated** with additional information ("event expressions," "paths," "traces"), and they noted the similarity with the conditions in c-tables.

We go beyond the problem of individual tuples in query answers by defining **closure** under a query language for probabilistic models. Then we describe **probabilistic c-tables** which add *to the c-tables themselves* probability distributions for the values taken by their variables. Here is an example of such a representation that captures the set of instances in which Alice is taking a course that is Math with probability 0.3; Physics (0.3); or Chemistry (0.4), while Bob takes the same course as Alice, provided that course is Physics or

Chemistry and Theo takes Math with probability 0.85:

Student	*Course*	*Condition*
Alice	x	
Bob	x	$x = \text{phys} \lor x = \text{chem}$
Theo	math	$t = 1$

$$x = \begin{cases} \text{math} & : 0.3 \\ \text{phys} & : 0.3 \\ \text{chem} & : 0.4 \end{cases}$$

$$t = \begin{cases} 0 : & 0.15 \\ 1 : & 0.85 \end{cases}$$

The concept of probabilistic c-table allows us to solve the closure problem by using the same algebra on c-tables defined in [30].

We also give a **completeness** result by showing that probabilistic Boolean c-tables (all variables are two-valued and can appear only in the conditions, not in the tuples) can represent *any* probabilistic database.

An important conceptual point is that, at least for the models we consider, the probabilistic database models can be seen, as **probabilistic counterparts** of incomplete database models. In an incompleteness model a tuple or an attribute value in a tuple may or may not be in the database. In its probabilistic counterpart, these are seen as elementary events with an assigned probability. For example, the models used in [22, 33, 47] are probabilistic counterparts of the two simplest incompleteness models discussed in [41]. As another example, the model used in [17] can be seen as the probabilistic counterpart of an incompleteness model one in which tuples sharing the same key have an exclusive-or relationship.

A consequence of this observation is that, in particular, query answering for probabilistic c-tables will allow us to solve the problem of calculating probabilities about query answers for any model that can be defined as a probabilistic counterpart of the incompleteness models considered in [30, 41].

Besides the models for incomplete and probabilistic information, several other forms of **annotated relations** have appeared in various contexts in the literature. Query answering in these settings involves generalizing $\mathcal{RA}$ to perform corresponding operations on the annotations.

In data warehousing, [14] and [15] compute lineages for tuples in the output of queries, in effect generalizing $\mathcal{RA}$ to computations on relations annotated with sets of contributing tuples. For curated databases, [9] proposes decorating output tuples with their why-provenance, essentially the set of sets of contributing tuples. Finally, $\mathcal{RA}$ on bag semantics can be viewed as a generalization to annotated relations, where a tuple's annotation is a number representing its multiplicity.

We observe that in all of these cases, the calculations with annotations are strikingly similar. This suggests looking for an algebraic structure on annotations that captures the above as particular cases. It turns out that the right structure to use for this purpose is that of **commutative semi-rings**. In fact, one can show that the laws of commutative semi-rings are *forced* by certain expected identities in $\mathcal{RA}$. Having identified commutative semi-rings as the right algebraic structure, we argue that a symbolic representation of semi-ring calculations is just what is needed to record, document, and track $\mathcal{RA}$ querying from input to output for applications which require rich **provenance** information. It is a standard philosophy in algebra that such symbolic representations form *the most general* such structure. In the case of commutative semi-rings, just as for rings, the symbolic representation is that of polynomials. This strongly suggests using polynomials to capture provenance.

The rest of this chapter is organized as follows:

- We develop the basic notions of **representation systems** for **incomplete information databases**, and we give several examples (Section 2).
- We define two measures of expressive power for representation systems, $\mathcal{RA}$**-Completeness** and **finite completeness**. $\mathcal{RA}$-com-pleteness characterizes the expressiveness of c-tables, and finite completeness the expressiveness of a restricted system which we call finite c-tables (Section 3).
- We examine the related notion of **closure** of representation systems under relational operations (Section 4).
- We define the notion of **algebraic completion**, and we give a number of results showing, for various representation systems not closed under the full relational algebra, that "closing" them under (certain fragments of) the relational algebra yields expressively complete representation systems (Section 5).
- We develop the basic notions of **probabilistic representation systems** (Section 6) and present **probabilistic counterparts** of various representation systems for incomplete databases (Sections 7 and 8).
- We observe patterns in the calculations used in incomplete and probabilistic databases, bag semantics, and why-provenance which motivate the more general study of **annotated relations** (Section 9).
- We define K**-relations**, in which tuples are annotated (tagged) with elements from K. We define a generalized positive algebra on K-relations and argue that K must be a **commutative semi-ring** (Section 10).
- For **provenance semi-rings** we use polynomials with integer coefficients, and we show that positive algebra semantics for any commutative semi-rings **factors** through the provenance semantics (Section 11).

- We consider **query containment** w.r.t. K-relation semantics and we show that for unions of conjunctive queries and when K is a **distributive lattice**, query containment is the same as that given by standard set semantics (Section 12).

2. Incomplete Information and Representation Systems

Our starting point is suggested by the work surveyed in [25], in Ch. 19 of [2], and in [43]. A database that provides incomplete information consists of a *set of possible instances*. At one end of this spectrum we have the conventional single instances, which provide "complete information." At the other end we have the set of *all* allowable instances which provides "no information" at all, or "zero information."

We adopt the formalism of relational databases over a fixed countably infinite domain $\mathbb{D}$. We use the unnamed form of the relational algebra. To simplify the notation we will work with relational schemas that consist of a single relation name of arity n. Everything we say can be easily reformulated for arbitrary relational schemas. We shall need a notation for the set of *all* (conventional) instances of this schema, i.e., all the finite n-ary relations over $\mathbb{D}$:

$$\mathcal{N} := \{I \mid I \subseteq \mathbb{D}^n,\ I \text{ finite}\}$$

DEFINITION 2.1 *An* **incomplete(-information) database** *(***i-database** *for short), $\mathcal{I}$, is a set of conventional instances, i.e., a subset $\mathcal{I} \subseteq \mathcal{N}$.*

The usual relational databases correspond to the cases when $\mathcal{I} = \{I\}$. The **no-information** or **zero-information database** consists of *all* the relations: $\mathcal{N}$.

Conventional relational instances are finite. However, because $\mathbb{D}$ is infinite incomplete databases are in general infinite. Hence the interest in finite, syntactical, representations for incomplete information.

DEFINITION 2.2 *A* **representation system** *consists of a set (usually a syntactically defined "language") whose elements we call tables, and a function Mod that associates to each table T an incomplete database Mod(T).*

The notation corresponds to the fact that T can be seen as a logical assertion such that the conventional instances in *Mod*(T) are in fact the *models* of T (see also [38, 44]).

The classical reference [30] considers three representation systems: **Codd tables**, v**-tables**, and c**-tables**. v-tables are conventional instances in which variables can appear in addition to constants from $\mathbb{D}$. If T is a v-table then[1]

$$\mathit{Mod}(T) := \{\nu(T) \mid \nu : \mathit{Var}(T) \to \mathbb{D} \text{ is a valuation for the variables of } T\}$$

Codd tables are v-tables in which all the variables are distinct. They correspond roughly to the current use of nulls in SQL, while v-tables model "labeled" or "marked" nulls. c-tables are v-tables in which each tuple is annotated with a *condition* — a Boolean combination of equalities involving variables and constants. The tuple condition is tested for each valuation ν and the tuple is discarded from $\nu(T)$ if the condition is not satisfied.

EXAMPLE 2.3 *A v-table and its possible worlds.*

$$R := \begin{array}{|ccc|}\hline 1 & 2 & x \\ 3 & x & y \\ z & 4 & 5 \\ \hline\end{array} \qquad \mathit{Mod}(R) = \left\{ \begin{array}{|ccc|}\hline 1 & 2 & 1 \\ 3 & 1 & 1 \\ 1 & 4 & 5 \\ \hline\end{array}, \begin{array}{|ccc|}\hline 1 & 2 & 2 \\ 3 & 2 & 1 \\ 1 & 4 & 5 \\ \hline\end{array}, \ldots, \begin{array}{|ccc|}\hline 1 & 2 & 77 \\ 3 & 77 & 89 \\ 97 & 4 & 5 \\ \hline\end{array}, \ldots \right\}$$

EXAMPLE 2.4 *A c-table and its possible worlds.*

$$S := \begin{array}{|ccc|c|}\hline 1 & 2 & x & \\ 3 & x & y & x = y \wedge z \neq 2 \\ z & 4 & 5 & x \neq 1 \vee x \neq y \\ \hline\end{array}$$

$$\mathit{Mod}(S) = \left\{ \begin{array}{|ccc|}\hline 1 & 2 & 1 \\ 3 & 1 & 1 \\ \hline\end{array}, \begin{array}{|ccc|}\hline 1 & 2 & 2 \\ 1 & 4 & 5 \\ \hline\end{array}, \ldots, \begin{array}{|ccc|}\hline 1 & 2 & 77 \\ 97 & 4 & 5 \\ \hline\end{array}, \ldots \right\}$$

Several other representation systems have been proposed in a recent paper [41]. We illustrate here three of them and we discuss several others later. A **?-table** is a conventional instance in which tuples are optionally labeled with "?," meaning that the tuple may be missing. An **or-set-table** looks like a conventional instance but or-set values [31, 37] are allowed. An or-set value $\langle 1, 2, 3\rangle$ signifies that exactly one of 1, 2, or 3 is the "actual" (but unknown) value. Clearly, the two ideas can be combined yielding another representation systems that we might (awkwardly) call **or-set-?-tables**.[2]

EXAMPLE 2.5 *An or-set-?-table and its possible worlds.*

$$T := \begin{array}{|ccc|c|}\hline 1 & 2 & \langle 1,2\rangle & \\ 3 & \langle 1,2\rangle & \langle 3,4\rangle & \\ \langle 4,5\rangle & 4 & 5 & ? \\ \hline\end{array} \qquad \mathit{Mod}(T) = \left\{ \begin{array}{|ccc|}\hline 1 & 2 & 1 \\ 3 & 1 & 3 \\ 4 & 4 & 5 \\ \hline\end{array}, \begin{array}{|ccc|}\hline 1 & 2 & 1 \\ 3 & 1 & 3 \\ \hline\end{array}, \ldots, \begin{array}{|ccc|}\hline 1 & 2 & 2 \\ 3 & 2 & 4 \\ \hline\end{array} \right\}$$

3. $\mathcal{RA}$-Completeness and Finite Completeness

"Completeness" of expressive power is the first obvious question to ask about representation systems. This brings up a fundamental difference between the representation systems of [30] and those of [41]. The presence of

[1] We follow [2, 41] and use the *closed-world assumption (CWA)*. [30] uses the *open-world assumption (OWA)*, but their results hold for CWA as well.

[2] In [41] these three systems are denoted by $\mathcal{R}_?$, $\mathcal{R}^A$ and $\mathcal{R}^A_?$.

variables in a table T and the fact that $\mathbb{D}$ is infinite means that $\mathit{Mod}(T)$ may be infinite. For the tables considered in [41], $\mathit{Mod}(T)$ is always finite.

[41] defines completeness as the ability of a representation system to represent "all" possible incomplete databases. For the kind of tables considered in [41] the question makes sense. But in the case of the tables with variables in [30] this is hopeless for trivial reasons. Indeed, in such systems there are only countably many tables while there are uncountably many incomplete databases (the subsets of $\mathcal{N}$, which is infinite). We will discuss separately below *finite completeness* for systems that only represent finite databases. Meanwhile, we will develop a different yardstick for the expressive power of tables with variables that range over an infinite domain.

c-tables and their restrictions (v-tables and Codd tables) have an inherent limitation: the cardinality of the instances in $\mathit{Mod}(T)$ is at most the cardinality of T. For example, the zero-information database $\mathcal{N}$ *cannot* be represented with c-tables. It also follows that among the incomplete databases that are representable by c-tables the "minimal"-information ones are those consisting for some m of all instances of cardinality up to m (which are in fact representable by Codd tables with m rows). Among these, we make special use of the ones of cardinality 1:

$$\mathcal{Z}_k := \{\{t\} \mid t \in \mathbb{D}^k\}.$$

Hence, $\mathcal{Z}_k$ consists of *all* the one-tuple relations of arity k. Note that $\mathcal{Z}_k = \mathit{Mod}(Z_k)$ where Z_k is the Codd table consisting of a single row of k distinct variables.

DEFINITION 3.1 *An incomplete database $\mathcal{I}$ is $\mathcal{RA}$-**definable** if there exists a relational algebra query q such that $\mathcal{I} = q(\mathcal{Z}_k)$, where k is the arity of the input relation name in q.*

THEOREM 3.2 *If $\mathcal{I}$ is an incomplete database representable by a c-table T, i.e., $\mathcal{I} = \mathit{Mod}(T)$, then $\mathcal{I}$ is $\mathcal{RA}$-definable.*

Proof: Let T be a c-table, and let $\{x_1, \dots, x_k\}$ denote the variables in T. We want to show that there exists a query q in $\mathcal{RA}$ such that $q(\mathit{Mod}(Z_k)) = \mathit{Mod}(T)$. Let n be the arity of T. For every tuple $t = (a_1, \dots, a_n)$ in T with condition $T(t)$, let $\{x_{i_1}, \dots, x_{i_j}\}$ be the variables in $T(t)$ which do not appear in t. For $1 \le i \le n$, define C_i to be the singleton $\{c\}$, if $a_i = c$ for some constant c, or $\pi_j(Z_k)$, if $a_i = x_j$ for some variable x_j. For $1 \le j \le k$, define C_{n+j} to be the expression $\pi_{i_j}(Z_k)$, where x_j is the jth variable in $T(t)$ which does not appear in t. Define q to be the query

$$q := \bigcup_{t \in T} \pi_{1,\dots,n}(\sigma_{\psi(t)}(C_1 \times \cdots \times C_{n+k})),$$

where $\psi(t)$ is obtained from $T(t)$ by replacing each occurrence of a variable x_i with the index j of the term C_j in which x_i appears. To see that $q(\mathit{Mod}(Z_k)) = \mathit{Mod}(T)$, since Z_k is a c-table, we can use Theorem 4.2 and check that, in fact, $\bar{q}(Z_k) = T$ where $\bar{q}$ is the translation of q into the c-tables algebra (see the proof of Theorem 4.2). Note that we only need the *SPJU* fragment of $\mathcal{RA}$. ■

EXAMPLE 3.3 *The c-table from Example 2.4 is definable as Mod*$(S) = q(\mathcal{Z}_3)$ *where q is the following query with input relation name V of arity 3:* $q(V) := \pi_{123}(\{1\} \times \{2\} \times V) \cup \pi_{123}(\sigma_{2=3,4\neq\text{'2'}}(\{3\} \times V)) \cup \pi_{512}(\sigma_{3\neq\text{'1'},3\neq 4}(\{4\} \times \{5\} \times V))$.

REMARK 3.4 *It turns out that the i-databases representable by c-tables are also definable via $\mathcal{RA}$ starting from the absolute zero-information instance, $\mathcal{N}$. Indeed, it can be shown (Proposition 15.1) that for each k there exists an $\mathcal{RA}$ query q such that $\mathcal{Z}_k = q(\mathcal{N})$. From there we can apply Theorem 3.2. The class of incomplete databases $\{\mathcal{I} \mid \exists q \in \mathcal{RA}\, s.t.\, \mathcal{I} = q(\mathcal{N})\}$ is strictly larger than that representable by c-tables, but it is still countable hence strictly smaller than that of all incomplete databases. Its connections with FO-definability in finite model theory might be interesting to investigate.*

Hence, c-tables are in some sense “no more powerful” than the relational algebra. But are they “as powerful”? This justifies the following:

DEFINITION 3.5 *A representation system is* $\mathcal{RA}$-**complete** *if it can represent any $\mathcal{RA}$-definable i-database.*

Since Z_k is itself a c-table the following is an immediate corollary of the fundamental result of [30] (see Theorem 4.2 below). It also states that the converse of Theorem 3.2 holds.

THEOREM 3.6 *c-tables are $\mathcal{RA}$-complete.*

This result is similar in nature to Corollary 3.1 in [25]. However, the exact technical connection, if any, is unclear, since Corollary 3.1 in [25] relies on the certain answers semantics for queries.

We now turn to the kind of completeness considered in [41].

DEFINITION 3.7 *A representation system is* **finitely complete** *if it can represent any finite i-database.*

The finite incompleteness of ?-tables, or-set-tables, or-set-?-tables and other systems is discussed in [41] where a finitely complete representation system $\mathcal{R}^A_{\text{prop}}$ is also given (we repeat the definition in the Appendix). Is finite completeness a reasonable question for c-tables, v-tables, and Codd tables? In general, for such tables *Mod*(T) is infinite (all that is needed is a tuple with at least one variable and with an infinitely satisfiable condition). To facilitate comparison with the systems in [41] we define *finite-domain* versions of tables with variables.

DEFINITION 3.8 *A* **finite-domain** *c-table (v-table, Codd table) consists of a c-table (v-table, Codd table) T together with a* finite *dom*$(x) \subset \mathbb{D}$ *for each variable x that occurs in T.*

Note that finite-domain Codd tables are equivalent to or-set tables. Indeed, to obtain an or-set table from a Codd table, one can see $\mathrm{dom}(x)$ as an or-set and substitute it for x in the table. Conversely, to obtain a Codd table from an or-set table, one can substitute a fresh variable x for each or-set and define $\mathrm{dom}(x)$ as the contents of the or-set.

In light of this connection, finite-domain v-tables can be thought of as a kind of "correlated" or-set tables. Finite-domain v-tables are strictly more expressive than finite Codd tables. Indeed, every finite Codd table is also a finite v-table. But, the set of instances represented by e.g. the finite v-table $\{(1, x), (x, 1)\}$ where $\mathrm{dom}(x) = \{1, 2\}$ cannot be represented by any finite Codd table. Finite-domain v-tables are themselves finitely incomplete. For example, the i-database $\{\{(1, 2)\}, \{(2, 1)\}\}$ cannot be represented by any finite v-table.

It is easy to see that finite-domain c-tables are finitely complete and hence equivalent to [41]'s $\mathcal{R}^A_{\text{prop}}$ in terms of expressive power. In fact, this is true even for the fragment of finite-domain c-tables which we will call *Boolean c-tables*, where the variables take only Boolean values and are only allowed to appear in conditions (never as attribute values).

THEOREM 3.9 *Boolean c-tables are finitely complete (hence finite-domain c-tables are also finitely complete).*

Proof: Let $\mathcal{I} = \{I_1, \ldots, I_m\}$ be a finite i-database. Construct a Boolean c-table T such that $Mod(T) = \mathcal{I}$ as follows. Let $\ell := \lceil \lg m \rceil$. For $1 \leq i < m$, put all the tuples from I_i into T with condition φ_i, defined

$$\varphi_i := \bigwedge_j \neg x_j \wedge \bigwedge_k x_k,$$

where the first conjunction is over all $1 \leq j \leq \ell$ such that jth digit in the ℓ-digit binary representation of $i - 1$ is 0, and the second conjunction is over all $1 \leq k \leq \ell$ such that the kth digit in the ℓ-digit binary representation of $i - 1$ is 1. Finally, put all the tuples from I_m into T with condition $\varphi_m \vee \cdots \vee \varphi_{2^\ell}$. ■
Although Boolean c-tables are complete there are clear advantages to using variables in tuples also, chief among them being *compactness* of representation.

EXAMPLE 3.10 *Consider the finite v-table* $\{(x_1, x_2, \ldots, x_m)\}$ *where* $dom(x_1) = dom(x_2) = \cdots = dom(x_m) = \{1, 2, \ldots, n\}$. *The equivalent Boolean c-table has* n^m *tuples.*

If we additionally restrict Boolean c-tables to allow conditions to contain only true or a single variable which appears in no other condition, then we obtain a representation system which is equivalent to ?-tables.

Since finite c-tables and $\mathcal{R}^A_{\text{prop}}$ are each finitely complete there is an obvious naïve algorithm to translate back and forth between them: list all the instances the one represents, then use the construction from the proof of finite completeness for the other. Finding a more practical "syntactic" algorithm is an interesting open question.

4. Closure Under Relational Operations

DEFINITION 4.1 *A representation system is* **closed** *under a query language if for any query* q *and any table* T *there is a table* T' *that represents* $q(Mod(T))$.

(For notational simplicity we consider only queries with one input relation name, but everything generalizes smoothly to multiple relation names.)

This definition is from [41]. In [2], a *strong* representation system is defined in the same way, with the significant addition that T' should be *computable* from T and q. It is not hard to show, using general recursion-theoretic principles, that there exist representation systems (even ones that only represent finite i-databases) which are closed as above but not strong in the sense of [2]. However, the concrete systems studied so far are either not closed or if they are closed then the proof provides also the algorithm required by the definition of strong systems. Hence, we see no need to insist upon the distinction.

THEOREM 4.2 ([30]) *c-tables, finite-domain c-tables, and Boolean c-tables are closed under the relational algebra.*

Proof: (Sketch.) We repeat here the essentials of the proof, including most of the definition of the c-table algebra. For each operation u of the relational algebra [30] defines its operation on the c-table conditions as follows. For projection, if V is a list of indexes, the condition for a tuple t in the output is given by

$$\bar{\pi}_V(T)(t) := \bigvee_{t' \in T \text{ s.t. } \pi_V(t')=t} T(t')$$

where $T(t')$ denotes the condition associated with t' in T. For selection, we have

$$\bar{\sigma}_P(T)(t) := T(t) \wedge P(t)$$

where $P(t)$ denotes the result of evaluating the selection predicate P on the values in t (for a Boolean c-table, this will always be true or false, while for

c-tables and finite-domain c-tables, this will be in general a Boolean formula on constants and variables). For cross product and union, we have

$$\begin{array}{rcl}(T_1 \bar{\times} T_2)(t) & := & T_1(t) \wedge T_2(t) \\ (T_1 \bar{\cup} T_2)(t) & := & T_1(t) \vee T_2(t)\end{array}$$

Difference and intersection are handled similarly. By replacing u's by $\bar{u}$ we translate any relational algebra expression q into a c-table algebra expression $\bar{q}$ and it can be shown that

LEMMA 4.3 *For all valuations ν, $\nu(\bar{q}(T)) = q(\nu(T))$.*

From this, $Mod(\bar{q}(T)) = q(Mod(T))$ follows immediately. ■

In Section 10, we shall see a generalization of the (positive) c-table algebra and Lemma 4.3 in the context of annotated relations.

5. Algebraic Completion

None of the incomplete representation systems we have seen so far is closed under the full relational algebra. Nor are two more representation systems considered in [41], $\mathcal{R}_{\text{sets}}$and $\mathcal{R}_{\oplus\equiv}$ (we repeat their definitions in the Appendix).

PROPOSITION 5.1 ([30, 41]) *Codd tables and v-tables are not closed under e.g. selection. Or-set tables and finite v-tables are also not closed under e.g. selection. ?-tables, $\mathcal{R}_{sets}$, and $\mathcal{R}_{\oplus\equiv}$ are not closed under e.g. join.*

We have seen that "closing" minimal-information one-row Codd tables (see before Definition 3.5) $\{Z_1, Z_2, \ldots\}$, by relational algebra queries yields equivalence with the c-tables. In this spirit, we will investigate "how much" of the relational algebra would be needed to complete the other representation systems considered. We call this kind of result *algebraic completion*.

DEFINITION 5.2 *If $(\mathcal{T}, Mod)$ is a representation system and $\mathcal{L}$ is a query language, then the representation system* obtained by closing $\mathcal{T}$ under $\mathcal{L}$ *is the set of tables $\{(T, q) \mid T \in \mathcal{T}, q \in \mathcal{L}\}$ with the function $Mod : \mathcal{T} \times \mathcal{L} \to \mathcal{N}$ defined by $Mod(T, q) := q(Mod(T))$.*

We are now ready to state the results regarding algebraic completion.

THEOREM 5.3 ($\mathcal{RA}$-COMPLETION)

1 *The representation system obtained by closing Codd tables under $SPJU$ queries is $\mathcal{RA}$-complete.*

2 *The representation system obtained by closing v-tables under SP queries is $\mathcal{RA}$-complete.*

Proof: (Sketch.) For each case we show that given a arbitrary c-table T one can construct a table S and a query q of the required type such that $\bar{q}(S) = T$. Case 1 is a trivial corollary of Theorem 3.2. The details for Case 2 are in the Appendix. ■

Note that in general there may be a "gap" between the language for which closure fails for a representation system and the language required for completion. For example, Codd tables are not closed under selection, but at the same time closing Codd tables under selection does not yield an $\mathcal{RA}$-complete representation system. (To see this, consider the incomplete database represented by the v-table $\{(x, 1), (x, 2)\}$. Intuitively, selection alone is not powerful enough to yield this incomplete database from a Codd table, as, selection operates on one tuple at a time and cannot correlate two un-correlated tuples.) On the other hand, it is possible that some of the results we present here may be able to be "tightened" to hold for smaller query languages, or else proved to be "tight" already. This is an issue which may be worth examining in the future.

We give now a set of analogous completion results for the finite case.

Theorem 5.4 (Finite-Completion)

1 *The representation system obtained by closing or-set-tables under PJ queries is finitely complete.*

2 *The representation system obtained by closing finite v-tables under PJ or S^+P queries is finitely complete.*

3 *The representation system obtained by closing $\mathcal{R}_{sets}$ under PJ or PU queries is finitely complete.*

4 *The representation system obtained by closing $\mathcal{R}_{\oplus\equiv}$ under S^+PJ queries is finitely complete.*

Proof:(Sketch.) In each case, given an arbitrary finite incomplete data-base, we construct a table and query of the required type which yields the incomplete database. The details are in the Appendix. ■

Note that there is a gap between the $\mathcal{RA}$-completion result for Codd tables, which requires $SPJU$ queries, and the finite-completion result for finite Codd tables, which requires only PJ queries. A partial explanation is that proof of the latter result relies essentially on the finiteness of the i-database.

More generally, if a representation system can represent arbitrarily-large i-databases, then closing it under $\mathcal{RA}$ yields a finitely complete representation system, as the following theorem makes precise (see Appendix for proof).

Theorem 5.5 (General Finite-Completion) *Let $\mathcal{T}$ be a representation system such that for all $n \geq 1$ there exists a table T in $\mathcal{T}$ such that*

$|Mod(T)| \geq n$. Then the representation system obtained by closing $\mathcal{T}$ under $\mathcal{RA}$ is finitely-complete.

COROLLARY 5.6 *The representation system obtained by closing ?-tables under $\mathcal{RA}$ queries is finitely complete.*

6. Probabilistic Databases and Representation Systems

Finiteness assumption For the entire discussion of probabilistic database models we will assume that *the domain of values $\mathbb{D}$ is finite.* Infinite domains of values are certainly interesting in practice; for some examples see [33, 45, 41]. Moreover, in the case of incomplete databases we have seen that they allow for interesting distinctions.[3] However, finite probability spaces are much simpler than infinite ones and we will take advantage of this simplicity. The issues related to probabilistic databases over infinite domains are nonetheless interesting and worth pursuing in the future.

We wish to model probabilistic information using a probability space whose possible outcomes are all the conventional instances. Recall that for simplicity we assume a schema consisting of just one relation of arity n. The finiteness of $\mathbb{D}$ implies that there are only finitely many instances, $I \subseteq \mathbb{D}^n$.

By **finite probability space** we mean a probability space (see e.g. [18]) $(\Omega, \mathcal{F}, \Pr[\,])$ in which the set of outcomes Ω is *finite* and the σ-field of events $\mathcal{F}$ consists of *all* subsets of Ω. We shall use the equivalent formulation of pairs (Ω, p) where Ω is the finite set of outcomes and where the *outcome probability assignment* $p : \Omega \to [0, 1]$ satisfies $\sum_{\omega \in \Omega} p(\omega) = 1$. Indeed, we take $\Pr[A] = \sum_{\omega \in A} p(\omega)$.

DEFINITION 6.1 *A* **probabilistic(-information) database** *(sometimes called in this paper a p-***database***) is a finite probability space whose outcomes are all the conventional instances, i.e., a pair $(\mathcal{N}, p)$ where $\sum_{I \in \mathcal{N}} p(I) = 1$.*

Demanding the direct specification of such probabilistic databases is unrealistic because there are 2^N possible instances, where $N := |\mathbb{D}|^n$, and we would need that many (minus one) probability values. Thus, as in the case of incomplete databases we define **probabilistic representation systems** consisting of "probabilistic tables" (prob. tables for short) and a function *Mod* that associates to each prob. table T a probabilistic database *Mod*(T). Similarly, we define **completeness** (finite completeness is the only kind we have in our setting).

To define closure under a query language we face the following problem. Given a probabilistic database $(\mathcal{N}, p)$ and a query q (with just one input relation name), how do we define the probability assignment for the instances in $q(\mathcal{N})$?

[3] Note however that the results remain true if $\mathbb{D}$ is finite; we just require an infinite supply of *variables*.

It turns out that this is a common construction in probability theory: image spaces.

DEFINITION 6.2 *Let (Ω, p) be a finite probability space and let $f : \Omega \to \Omega'$ where Ω' is some finite set. The* **image** *of (Ω, p) under f is the finite probability space (Ω', p') where* [4] *$p'(\omega') := \sum_{f(\omega)=\omega'} p(\omega)$.*

Again we consider as query languages the relational algebra and its sublanguages defined by subsets of operations.

DEFINITION 6.3 *A probabilistic representation system is* **closed** *under a query language if for any query q and any prob. table T there exists a prob. table T' that represents $q(Mod(T))$, the image space of $Mod(T)$ under q.*

7. Probabilistic ?-Tables and Probabilistic Or-Set Tables

Probabilistic ?-tables (p-?-tables for short) are commonly used for probabilistic models of databases [47, 22, 23, 16] (they are called the "independent tuples" representation in [42]). Such tables are the probabilistic counterpart of ?-tables where each "?" is replaced by a probability value. Example 7.4 below shows such a table. The tuples not explicitly shown are assumed tagged with probability 0. Therefore, we define a p-?-table as a mapping that associates to each $t \in \mathbb{D}^n$ a probability value p_t. In order to represent a probabilistic database, papers using this model typically include a statement like "every tuple t is in the outcome instance with probability p_t, independently from the other tuples" and then a statement like

$$\Pr[I] = \Big(\prod_{t \in I} p_t\Big)\Big(\prod_{t \notin I} (1 - p_t)\Big).$$

In fact, to give a rigorous semantics, one needs to define the events $E_t \subseteq \mathcal{N}$, $E_t := \{I \mid t \in I\}$ and then to prove the following.

PROPOSITION 7.1 *There exists a unique probabilistic database such that the events E_t are jointly independent and $\Pr[E_t] = p_t$.*

This defines p-?-tables as a probabilistic representation system. We shall however provide an equivalent but more perspicuous definition. We shall need here another common construction from probability theory: product spaces.

DEFINITION 7.2 *Let $(\Omega_1, p_1), \ldots, (\Omega_n, p_n)$ be finite probability spaces. Their* **product** *is the space $(\Omega_1 \times \cdots \times \Omega_n, p)$ where*[5] *we have:*

$$p(\omega_1, \ldots, \omega_n) := p_1(\omega_1) \cdots p_n(\omega_n)$$

[4] It is easy to check that the $p'(\omega')$'s do actually add up to 1.
[5] Again, it is easy to check that the outcome probability assignments add up to 1.

This definition corresponds to the intuition that the n systems or phenomena that are modeled by the spaces $(\Omega_1, p_1), \ldots, (\Omega_n, p_n)$ behave without "interfering" with each other. The following formal statements summarize this intuition.

PROPOSITION 7.3 *Consider the product of the spaces* $(\Omega_1, p_1), \ldots, (\Omega_n, p_n)$. *Let* $A_1 \subseteq \Omega_1, \ldots, A_n \subseteq \Omega_n$.

1 *We have* $\Pr[A_1 \times \cdots \times A_n] = \Pr[A_1] \cdots \Pr[A_n]$.

2 *The events* $A_1 \times \Omega_2 \times \cdots \times \Omega_n$, $\Omega_1 \times A_2 \times \cdots \times \Omega_n$, $\ldots$, $\Omega_1 \times \Omega_2 \times \cdots \times A_n$ *are jointly independent in the product space.*

Turning back to p-?-tables, for each tuple $t \in \mathbb{D}^n$ consider the finite probability space $B_t := (\{\mathsf{true}, \mathsf{false}\}, p)$ where $p(\mathsf{true}) := p_t$ and $p(\mathsf{false}) = 1 - p_t$. Now consider the product space

$$P := \prod_{t \in \mathbb{D}^n} B_t$$

We can think of its set of outcomes (abusing notation, we will call this set P also) as the set of functions from $\mathbb{D}^n$ to $\{\mathsf{true}, \mathsf{false}\}$, in other words, predicates on $\mathbb{D}^n$. There is an obvious function $f : P \to \mathcal{N}$ that associates to each predicate the set of tuples it maps to true.

All this gives us a p-database, namely the image of P under f. It remains to show that it satisfies the properties in Proposition 7.1. Indeed, since f is a bijection, this probabilistic database is in fact *isomorphic* to P. In P the events that are in bijection with the E_t's are the Cartesian product in which there is exactly one component $\{\mathsf{true}\}$ and the rest are $\{\mathsf{true}, \mathsf{false}\}$. The desired properties then follow from Proposition 7.3.

We define now another simple probabilistic representation system called **probabilistic or-set-tables** (p-or-set-tables for short). These are the probabilistic counterpart of or-set-tables where the attribute values are, instead of or-sets, finite probability spaces whose outcomes are the values in the or-set. p-or-set-tables correspond to a simplified version of the ProbView model presented in [33], in which plain probability values are used instead of confidence intervals.

EXAMPLE 7.4 *A p-or-set-table S, and a p-?-table T.*

$S :=$

1	$\langle 2 : 0.3, 3 : 0.7 \rangle$
4	5
$\langle 6 : 0.5, 7 : 0.5 \rangle$	$\langle 8 : 0.1, 9 : 0.9 \rangle$

$T :=$

1	2	0.4
3	4	0.3
5	6	1.0

A p-or-set-table determines an instance by choosing an outcome in each of the spaces that appear as attribute values, *independently*. Recall that or-set tables are equivalent to finite-domain Codd tables. Similarly, a p-or-set-table corresponds to a Codd table T plus for each variable x in T a finite probability space $\mathrm{dom}(x)$ whose outcomes are in $\mathbb{D}$. This yields a p-database, again by image space construction, as shown more generally for c-tables next in Section 8.

Query answering The papers [22, 47, 33] have considered, independently, the problem of calculating the probability of tuples appearing in query answers. This does *not* mean that in general $q(\mathit{Mod}(T))$ can be represented by another tuple table when T is some p-?-table and $q \in \mathcal{RA}$ (neither does this hold for p-or-set-tables). This follows from Proposition 5.1. Indeed, if the probabilistic counterpart of an incompleteness representation system $\mathcal{T}$ is closed, then so is $\mathcal{T}$. Hence the lifting of the results in Proposition 5.1 and other similar results.

Each of the papers [22, 47, 33] recognizes the problem of query answering and solves it by developing a more general model in which rows contain additional information *similar in spirit* to the conditions that appear in c-tables (in fact [22]'s model is essentially what we call probabilistic Boolean c-tables, see next section). It turns out that one can actually use a probabilistic counterpart to c-tables themselves together with the algebra on c-tables given in [30] to achieve the same effect.

8. Probabilistic c-tables

DEFINITION 8.1 *A* **probabilistic c-table** *(*pc**-tables** *for short) consists of a c-table T together with a* finite probability space *$\mathit{dom}(x)$ (whose outcomes are values in $\mathbb{D}$) for each variable x that occurs in T.*

To get a probabilistic representation system consider the product space

$$V := \prod_{x \in \mathit{Var}(T)} \mathrm{dom}(x)$$

The outcomes of this space are in fact the *valuations* for the c-table T! Hence we can define the function $g : V \to \mathcal{N}, g(\nu) := \nu(T)$ and then define $\mathit{Mod}(T)$ as the image of V under g.

Similarly, we can talk about Boolean pc-tables, pv-tables and probabilistic Codd tables (the latter related to [33], see previous section). Moreover, the p-?-tables correspond to restricted Boolean pc-tables, just like ?-tables.

THEOREM 8.2 *Boolean pc-tables are complete (hence pc-tables are also complete).*

Proof: Let $I_1, \ldots, I_k$ denote the instances with non-zero probability in an arbitrary probabilistic database, and let $p_1, \ldots, p_k$ denote their probabilities. Construct a probabilistic Boolean c-table T as follows. For $1 \leq i \leq k-1$, put the tuples from I_i in T with condition $\neg x_1 \wedge \cdots \wedge \neg x_{i-1} \wedge x_i$. Put the tuples from I_k in T with condition $\neg x_1 \wedge \cdots \wedge \neg x_{k-1}$. For $1 \leq i \leq k-1$,

A B C	
$a\ b\ c$	b_1
$d\ b\ e$	b_2
$f\ g\ e$	b_3

(a)

A C	
$a\ c$	$(b_1 \wedge b_1) \vee (b_1 \wedge b_1)$
$a\ e$	$b_1 \wedge b_2$
$d\ c$	$b_1 \wedge b_2$
$d\ e$	$(b_2 \wedge b_2) \vee (b_2 \wedge b_2) \vee (b_2 \wedge b_3)$
$f\ e$	$(b_3 \wedge b_3) \vee (b_3 \wedge b_3) \vee (b_2 \wedge b_3)$

(b)

A C	
$a\ c$	b_1
$a\ e$	$b_1 \wedge b_2$
$d\ c$	$b_1 \wedge b_2$
$d\ e$	b_2
$f\ e$	b_3

(c)

Figure 2.1. Boolean c-tables example

set $\Pr[x_i = \textsf{true}] := p_i/(1 - \sum_{j=1}^{i-1} p_j)$. It is straightforward to check that this yields a table such that $\Pr[I_i] = p_i$. ∎

The previous theorem was independently observed in [42] and [28].

THEOREM 8.3 *pc-tables (and Boolean pc-tables) are closed under the relational algebra.*

Proof:(Sketch.) For any pc-table T and any $\mathcal{RA}$ query q we show that the probability space $q(\mathit{Mod}(T))$ (the image of $\mathit{Mod}(T)$ under q) is in fact the same as the space $\mathit{Mod}(\bar{q}(T))$. The proof of Theorem 4.2 already shows that the outcomes of the two spaces are the same. The fact that the probabilities assigned to each outcome are the same follows from Lemma 4.3. ∎

The proof of this theorem gives in fact an algorithm for constructing the answer as a p-database itself, represented by a pc-table. In particular this will work for the models of [22, 33, 47] or for models we might invent by adding probabilistic information to v-tables or to the representation systems considered in [41]. The interesting result of [16] about the applicability of an "extensional" algorithm to calculating answer tuple probabilities can be seen also as characterizing the conjunctive queries q which for any p-?-table T are such that the c-table $\bar{q}(T)$ is in fact equivalent to some p-?-table.

9. Queries on Annotated Relations

In this section we compare the calculations involved in query answering in incomplete and probabilistic databases with those for two other important examples. We observe similarities between them which will motivate the general study of annotated relations.

As a first example, consider the Boolean c-table in Figure 2.1(a), and the following $\mathcal{RA}$ query, which computes the union of two self-joins:

$$q(R) := \pi_{\mathrm{AC}}\big(\pi_{\mathrm{AB}}R \bowtie \pi_{\mathrm{BC}}R \cup \pi_{\mathrm{AC}}R \bowtie \pi_{\mathrm{BC}}R\big)$$

A B C	
$a\ b\ c$	2
$d\ b\ e$	5
$f\ g\ e$	1

(a)

A C	
$a\ c$	$2 \cdot 2 + 2 \cdot 2 = 8$
$a\ e$	$2 \cdot 5 = 10$
$d\ c$	$2 \cdot 5 = 10$
$d\ e$	$5 \cdot 5 + 5 \cdot 5 + 5 \cdot 1 = 55$
$f\ e$	$1 \cdot 1 + 1 \cdot 1 + 5 \cdot 1 = 7$

(b)

Figure 2.2. Bag semantics example

A B C	
$a\ b\ c$	$\{\{p\}\}$
$d\ b\ e$	$\{\{r\}\}$
$f\ g\ e$	$\{\{s\}\}$

(a)

A C	
$a\ c$	$\{\{p\}\}$
$a\ e$	$\{\{p, r\}\}$
$d\ c$	$\{\{p, r\}\}$
$d\ e$	$\{\{r\}\}$
$f\ e$	$\{\{s\}\}$

(b)

Figure 2.3. Minimal witness why-provenance example

The Imielinski-Lipski algorithm (cf. Theorem 4.2) produces the Boolean c-table shown in Figure 2.1(b), which can be simplified to the one shown in Figure 2.1(c). The annotations on the tuples of this c-table are such that it correctly represents the possible worlds of the query answer:

$$Mod(q(R)) = q(Mod(R))$$

Another kind of table with annotations is a *multiset* or *bag*. In this case, the annotations are natural numbers which represent the multiplicity of the tuple in the multiset. (A tuple not listed in the table has multiplicity 0.) Query answering on such tables involves calculating not just the tuples in the output, but also their multiplicities.

For example, consider the multiset shown in Figure 2.2(a). Then $q(R)$, where q is the same query from before, is the multiset shown in Figure 2.2(b). Note that for projection and union we add multiplicities while for join we multiply them. There is a striking similarity between the arithmetic calculations we do here for multisets, and the Boolean calculations for the c-table.

A third example involves the *minimal witness why-provenance* proposed in [9] for tracking the processing of scientific data. Here source tuples are annotated with their own tuple ids, and answering queries involves calculating the set of sets of ids of source tuples which "contribute together" for a given output tuple. The minimal witness why-provenance W for an output tuple t is required to be minimal in the sense that for any A, B in W neither is a subset of the other.

Figure 2.3(a) shows an example of a source table, where t_1, t_2, t_3 are tuple ids. Considering again the same query q as above, the algorithm of [9] produces the table with why-provenance annotations shown in Figure 2.3(b). Note again the similarity between this table and the example earlier with Boolean c-tables.

10. K-Relations

In this section we unify the examples above by considering generalized relations in which the tuples are annotated (*tagged*) with information of various kinds. Then, we will define a generalization of the positive relational algebra ($\mathcal{RA}^+$) to such tagged-tuple relations. The examples in Section 9 will turn out to be particular cases.

We use here the named perspective [2] of the relational model in which tuples are functions $t : U \to \mathbb{D}$ with U a finite set of attributes and $\mathbb{D}$ a domain of values. We fix the domain $\mathbb{D}$ for the time being and we denote the set of all such U-tuples by U-Tup. (Usual) relations over U are subsets of U-Tup.

A notationally convenient way of working with tagged-tuple relations is to model tagging by a function on all possible tuples, with those tuples not considered to be "in" the relation tagged with a special value. For example, the usual set-theoretic relations correspond to functions that map U-Tup to $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ with the tuples in the relation tagged by true and those not in the relation tagged by false.

DEFINITION 10.1 *Let K be a set containing a distinguished element 0. A* K**-relation** *over a finite set of attributes U is a function $R : U$-$\mathsf{Tup} \to K$ such that its* **support** *defined by* $\mathsf{supp}(R) := \{t \mid R(t) \neq 0\}$ *is finite.*

In generalizing $\mathcal{RA}^+$ we will need to assume more structure on the set of tags. To deal with selection we assume that the set K contains two distinct values $0 \neq 1$ which denote "out of" and "in" the relation, respectively. To deal with union and projection and therefore to combine different tags of the same tuple into one tag we assume that K is equipped with a binary operation "$+$". To deal with natural join (hence intersection and selection) and therefore to combine the tags of joinable tuples we assume that K is equipped with another binary operation "$\cdot$".

DEFINITION 10.2 *Let $(K, +, \cdot, 0, 1)$ be an algebraic structure with two binary operations and two distinguished elements. The operations of the* **positive algebra** *are defined as follows:*

empty relation *For any set of attributes* U*, there is* $\emptyset : U\text{-}\mathsf{Tup} \to K$ *such that* $\emptyset(t) = 0$.

union *If* $R_1, R_2 : U\text{-}\mathsf{Tup} \to K$ *then* $R_1 \cup R_2 : U\text{-}\mathsf{Tup} \to K$ *is defined by*

$$(R_1 \cup R_2)(t) \ := \ R_1(t) + R_2(t)$$

projection *If* $R : U\text{-}\mathsf{Tup} \to K$ *and* $V \subseteq U$ *then* $\pi_V R : V\text{-}\mathsf{Tup} \to K$ *is defined by*

$$(\pi_V R)(t) \ := \sum_{t=t' \text{ on } V \text{ and } R(t')\neq 0} R(t')$$

(here $t = t'$ on V *means* t' *is a* U*-tuple whose restriction to* V *is the same as the* V*-tuple* t*; note also that the sum is finite since* R *has finite support)*

selection *If* $R : U\text{-}\mathsf{Tup} \to K$ *and the selection predicate* **P** *maps each* U*-tuple to either* 0 *or* 1 *then* $\sigma_{\mathbf{P}} R : U\text{-}\mathsf{Tup} \to K$ *is defined by*

$$(\sigma_{\mathbf{P}} R)(t) \ := \ R(t) \cdot \mathbf{P}(t)$$

Which $\{0, 1\}$*-valued functions are used as selection predicates is left unspecified, except that we assume that* false*—the constantly* 0 *predicate, and* true*—the constantly* 1 *predicate, are always available.*

natural join *If* $R_i : U_i\text{-}\mathsf{Tup} \to K \ \ i = 1, 2$ *then* $R_1 \bowtie R_2$ *is the* K*-relation over* $U_1 \cup U_2$ *defined by*

$$(R_1 \bowtie R_2)(t) \ := \ R_1(t_1) \cdot R_2(t_2)$$

where $t_1 = t$ on U_1 *and* $t_2 = t$ on U_2 *(recall that* t *is a* $U_1 \cup U_2$*-tuple).*

renaming *If* $R : U\text{-}\mathsf{Tup} \to K$ *and* $\beta : U \to U'$ *is a bijection then* $\rho_\beta R$ *is a* K*-relation over* U' *defined by*

$$(\rho_\beta R)(t) \ := \ R(t \circ \beta)$$

PROPOSITION 10.3 *The operation of* $\mathcal{RA}^+$ *preserve the finiteness of supports therefore they map* K*-relations to* K*-relations. Hence, Definition 10.2 gives us an algebra on* K*-relations.*

This definition generalizes the definitions of $\mathcal{RA}^+$ for the motivating examples we saw. Indeed, for $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$ we obtain the usual $\mathcal{RA}^+$ with set semantics. For $(\mathbb{N}, +, \cdot, 0, 1)$ it is $\mathcal{RA}^+$ with bag semantics.

For the Imielinski-Lipski algebra on c-tables we consider the set of Boolean expressions over some set B of variables which are *positive*, i.e., they involve only disjunction, conjunction, and constants for true and false. Then we identify those expressions that yield the same truth-value for all Boolean assignments of the variables in B.[6] Denoting by $\mathsf{PosBool}(B)$ the result and applying Definition 10.2 to the structure $(\mathsf{PosBool}(B), \vee, \wedge, \mathsf{false}, \mathsf{true})$ produces exactly the Imielinski-Lipski algebra.

These three structures are examples of *commutative semi-rings*, i.e., algebraic structures $(K, +, \cdot, 0, 1)$ such that $(K, +, 0)$ and $(K, \cdot, 1)$ are commutative monoids, $\cdot$ is distributive over $+$ and $\forall a,\ 0 \cdot a = a \cdot 0 = 0$. Further evidence for requiring K to form such a semi-ring is given by

PROPOSITION 10.4 *The following $\mathcal{RA}$ identities:*

- *union is associative, commutative and has identity $\emptyset$;*
- *join is associative, commutative and distributive over union;*
- *projections and selections commute with each other as well as with unions and joins (when applicable);*
- $\sigma_{\mathsf{false}}(R) = \emptyset$ *and* $\sigma_{\mathsf{true}}(R) = R$.

hold for the positive algebra on K-relations if and only if $(K, +, \cdot, 0, 1)$ is a commutative semi-ring.

Glaringly absent from the list of relational identities are the idempotence of union and of (self-)join. Indeed, these fail for the bag semantics, an important particular case of the general treatment presented here.

Any function $h : K \to K'$ can be used to transform K-relations to K'-relations simply by applying h to each tag (note that the support may shrink but never increase). Abusing the notation a bit we denote the resulting transformation from K-relations to K'-relations also by h. The $\mathcal{RA}$ operations we have defined work nicely with semi-ring structures:

PROPOSITION 10.5 *Let $h : K \to K'$ and assume that K, K' are commutative semi-rings. The transformation given by h from K-relations to K'-relations commutes with any $\mathcal{RA}^+$ query (for queries of one argument) $q(h(R)) = h(q(R))$ if and only if h is a semi-ring homomorphism.*

Proposition 10.5 has some useful applications. For example, for Boolean c-tables and semi-ring homomorphisms $\mathsf{Eval}_\nu : \mathsf{PosBool}(B) \to \mathbb{B}$ corresponding to valuations of the variables $\nu : B \to \mathbb{B}$, Proposition 10.5 generalizes Lemma 4.3 and can be used to establish the closure of $\mathsf{PosBool}(B)$-annotated relations (in the sense of Section 4) under $\mathcal{RA}^+$ queries.

[6] in order to permit simplifications; it turns out that this is the same as transforming using the axioms of *distributive lattices* [13]

(a)

A B C	
$a\ b\ c$	p
$d\ b\ e$	r
$f\ g\ e$	s

(b)

A C	
$a\ c$	$\{p\}$
$a\ e$	$\{p,r\}$
$d\ c$	$\{p,r\}$
$d\ e$	$\{r,s\}$
$f\ e$	$\{r,s\}$

(c)

A C	
$a\ c$	$\{\{p\}\}$
$a\ e$	$\{\{p,r\}\}$
$d\ c$	$\{\{p,r\}\}$
$d\ e$	$\{\{r\},\{r,s\}\}$
$f\ e$	$\{\{s\},\{r,s\}\}$

(d)

A C	
$a\ c$	$2p^2$
$a\ e$	pr
$d\ c$	pr
$d\ e$	$2r^2 + rs$
$f\ e$	$2s^2 + rs$

Figure 2.4. Lineage, why-provenance, and provenance polynomials

11. Polynomials for Provenance

Lineage was defined in [14, 15] as a way of relating the tuples in a query output to the tuples in the query input that "contribute" to them. The lineage of a tuple t in a query output is in fact the *set* of all contributing input tuples.

Computing the lineage for queries in $\mathcal{RA}^+$ turns out to be exactly Definition 10.2 for the semi-ring $(\mathcal{P}(X) \cup \{\bot\}, +, \cdot, \bot, \emptyset)$ where X consists of the ids of the tuples in the input instance, $\bot + S = S + \bot = S$, $S \cdot \bot = \bot \cdot S = \bot$, and $S + T = S \cdot T = S \cup T$ if $S, T \neq \bot$[7]

For example, we consider the same tuples as in relation R used in the examples of Section 9 but now we tag them with their own ids p,r,s, as shown in Figure 2.4(a). The resulting R can be seen as a $\mathcal{P}(\{p,r,s\})$-relation by replacing p with $\{p\}$, etc. Applying the query q from Section 9 to R we obtain according to Definition 10.2 the $\mathcal{P}(\{p,r,s\})$-relation shown in Figure 2.4(b).

A related but finer-grained notion of provenance, called why-provenance, was defined in [9].[8] The why-provenance of a tuple t in a query output is the *set* of *sets* of input tuples which contribute together to produce t. The lineage of t can be obtained by flattening the why-provenance of t.

As with lineage, computing the why-provenance for queries in $\mathcal{RA}^+$ can be done [8] using Definition 10.2, this time for the semi-ring $(\mathcal{P}(\mathcal{P}(X)), \cup, \uplus, \emptyset, \{\emptyset\})$ where X is the set of tuple ids for the input instance and $A \uplus B$ is the pairwise union of A and B, i.e., $A \uplus B := \{a \cup b : a \in A, b \in B\}$. For example, the R in Figure 2.4(a) can be seen as a why-provenance relation by replacing p with $\{\{p\}\}$, etc. Applying the query q from Section 9 to R we obtain according to Definition 10.2 the why-provenance relation shown in Figure 2.4(c).

[7]This definition for lineage, due to [8], corrects the one which appeared in [27].

[8]The distinction between lineage and why-provenance, which went unnoticed in [9] and [27], was pointed out in [8].

Finally, to return to the third example of Section 9, the minimal witness why-provenance can be computed [8] using a semi-ring whose domain is $\mathsf{irr}(\mathcal{P}(X))$, the set of *irredundant* subsets of $\mathcal{P}(X)$, i.e., W is in $\mathsf{irr}(\mathcal{P}(X))$ if for any A, B in W neither is a subset of the other. We can associate with any $W \in \mathcal{P}(X)$ a unique irredundant subset of W by repeatedly looking for elements A, B such that $A \subset B$ and deleting B from W. Then we define a semi-ring $(\mathsf{irr}(\mathcal{P}(X)), +, \cdot, 0, 1)$ as follows:

$$\begin{array}{rclcrcl} I + J & := & \mathsf{irr}(I \cup J) & \qquad & I \cdot J & := & \mathsf{irr}(I \uplus J) \\ 0 & := & \emptyset & & 1 & := & \{\emptyset\} \end{array}$$

The table in Figure 2.3(b) is obtained by applying the query q from Section 9 to R of Figure 2.3(a) according to Definition 10.2 for the minimal why-provenance semi-ring. Note that this is a well-known semi-ring: the construction above is the construction for the free distributive lattice generated by the set X. Moreover, it is isomorphic $\mathsf{PosBool}(X)$! This explains the similarity between the calculations in Figure 2.1 and Figure 2.3.

These examples illustrate the limitations of lineage and why-provenance (also recognized in [12]). For example, in the query result in Figure 2.4(b) (f, e) and (d, e) have the same lineage, the input tuples with id r and s. However, the query can also calculate (f, e) from s alone and (d, e) from r alone. In a provenance application in which one of r or s is perhaps less trusted or less usable than the other the effect can be different on (f, e) than on (d, e) and this cannot be detected by lineage. Meanwhile, with why-provenance we do see that (f, e) can be calculated from s alone and (d, e) from r alone, but we have lost information about *multiplicities* (the number of times a tuple was used in a self-join, or the number of derivations of an output tuple in which a given set of tuples is involved) which may be needed to calculate a more refined notion of trust. It seems that we need to know not just *which* input tuples contribute but also exactly *how* they contribute.[9]

On the other hand, by using the different operations of the semi-ring, Definition 10.2 appears to fully "document" how an output tuple is produced. To record the documentation as tuple tags we need to use a semi-ring of symbolic expressions. In the case of semi-rings, like in ring theory, these are the *polynomials*.

DEFINITION 11.1 *Let X be the set of tuple ids of a (usual) database instance I. The* **positive algebra provenance semi-ring** *for I is the semi-ring of polynomials with variables (a.k.a. indeterminates) from X and coefficients from $\mathbb{N}$,*

[9]In contrast to why-provenance, the notion of provenance we describe could justifiably be called **how-provenance**.

with the operations defined as usual[10]*:* $(\mathbb{N}[X], +, \cdot, 0, 1)$.

Example of provenance computation. Start again from the relation R in Figure 2.4(a) in which tuples are tagged with their own id. R can be seen as an $\mathbb{N}[p, r, s]$-relation. Applying to R the query q from Section 9 and doing the calculations in the provenance semi-ring we obtain the $\mathbb{N}[p, r, s]$-relation shown in Figure 2.4(c). The provenance of (f, e) is $2s^2 + rs$ which can be "read" as follows: (f, e) is computed by q in three different ways; two of them use the input tuple s *twice*; the third uses input tuples r and s. We also see that the provenance of (d, e) is different and we see *how* it is different! □

The following standard property of polynomials captures the intuition that $\mathbb{N}[X]$ is as "general" as any semi-ring:

PROPOSITION 11.2 *Let K be a commutative semi-ring and X a set of variables. For any valuation $v : X \to K$ there exists a unique homomorphism of semi-rings*

$$\mathsf{Eval}_v : \mathbb{N}[X] \to K$$

such that for the one-variable monomials we have $\mathsf{Eval}_v(x) = v(x)$.

As the notation suggests, $\mathsf{Eval}_v(P)$ evaluates the polynomial P in K given a valuation for its variables. In calculations with the integer coefficients, na where $n \in \mathbb{N}$ and $a \in K$ is the sum in K of n copies of a. Note that $\mathbb{N}$ is embedded in K by mapping n to the sum of n copies of 1_K.

Using the Eval notation, for any $P \in \mathbb{N}[x_1, \ldots, x_n]$ and any K the **polynomial function** $f_P : K^n \to K$ is given by:

$$f_P(a_1, \ldots, a_n) := \mathsf{Eval}_v(P) \quad v(x_i) = a_i, i = 1..n$$

Putting together Propositions 10.5 and 11.2 we obtain the following conceptually important fact that says, informally, that the semantics of $\mathcal{RA}^+$ on K-relations for any semi-ring K **factors** through the semantics of the same in provenance semi-rings.

THEOREM 11.3 *Let K be a commutative semi-ring, let R be a K-relation, and let X be the set of tuple ids of the tuples in $\mathsf{supp}(R)$. There is an obvious valuation $v : X \to K$ that associates to a tuple id the tag of that tuple in R.*

We associate to R an "abstractly tagged" version, denoted $\bar{R}$, which is an $X \cup \{0\}$-relation. $\bar{R}$ is such that $\mathsf{supp}(\bar{R}) = \mathsf{supp}(R)$ and the tuples in

[10]These are polynomials in commutative variables so their operations are the same as in middle-school algebra, except that subtraction is not allowed.

$\mathsf{supp}(\bar{R})$ *are tagged by their own tuple id. Note that as an* $X \cup \{0\}$*-relation,* $\bar{R}$ *is a particular kind of* $\mathbb{N}[X]$*-relation.*

Then, for any $\mathcal{RA}^+$ *query* q *we have*[11]

$$q(R) \;=\; \mathsf{Eval}_v(q(\bar{R}))$$

To illustrate an instance of this theorem, consider the provenance polynomial $2r^2 + rs$ of the tuple (d, e) in Figure 2.4(c). Evaluating it in $\mathbb{N}$ for $p = 2, r = 5, s = 1$ we get 55 which is indeed the multiplicity of (d, e) in Figure 2.2(a).

12. Query Containment

Here we present some results about query containment w.r.t. the general semantics in K-relations.

Definition 12.1 *Let* K *be a naturally ordered commutative semi-ring and let* q_1, q_2 *be two queries defined on* K*-relations. We define containment with respect to* K*-relations semantics by*

$$q_1 \sqsubseteq_K q_2 \;\overset{\texttt{def}}{\Leftrightarrow}\; \forall R\, \forall t\; q_1(R)(t) \leq q_2(R)(t)$$

When K is $\mathbb{B}$ and $\mathbb{N}$ we get the usual notions of query containment with respect to set and bag semantics.

Some simple facts follow immediately. For example if $h : K \to K'$ is a semi-ring homomorphism such that $h(x) \leq h(y) \Rightarrow x \leq y$ and q_1, q_2 are $\mathcal{RA}^+$ queries it follows from Prop. 10.5 that $q_1 \sqsubseteq_{K'} q_2 \Rightarrow q_1 \sqsubseteq_K q_2$. If instead h is a surjective homomorphism then $q_1 \sqsubseteq_K q_2 \Rightarrow q_1 \sqsubseteq_{K'} q_2$.

The following result allows us to use the decidability of containment of unions of conjunctive queries [11, 40].

Theorem 12.2 *If* K *is a* distributive lattice *then for any* q_1, q_2 *unions of conjunctive queries we have*

$$q_1 \sqsubseteq_K q_2 \;\; \textit{iff} \;\; q_1 \sqsubseteq_{\mathbb{B}} q_2$$

Proof:(sketch) One direction follows because $\mathbb{B}$ can be homomorphically embedded in K. For the other direction we use the existence of query body homomorphisms to establish mappings between monomials of provenance polynomials. Then we apply the factorization theorem (11.3) and the idempotence and absorption laws of K. ■

Therefore, if K is a distributive lattice for (unions of) conjunctive queries containment with respect to K-relation semantics is decidable by the same

[11]To simplify notation we have stated this theorem for queries of one argument but the generalization is immediate.

procedure as for standard set semantics. $\mathsf{PosBool}(B)$ is a distributive lattice, as is the semi-ring $([0,1], \max, \min, 0, 1)$ which is related to *fuzzy sets* [46] and could be referred to as the *fuzzy semi-ring*. A theorem similar to the one above is shown in [32] but the class of algebraic structures used there does not include $\mathsf{PosBool}(B)$ (although it does include the fuzzy semi-ring).

13. Related Work

Lineage and why-provenance were introduced in [14, 15, 9], (the last paper uses a tree data model) but the relationship with [30] was not noticed. The papers on probabilistic databases [22, 47, 33] note the similarities with [30] but do not attempt a generalization.

Two recent papers on provenance, although independent of our work, have a closer relationship to the approach outlined here. Indeed, [12] identifies the limitations of why-provenance and proposes *route-provenance* which is also related to derivation trees. The issue of infinite routes in recursive programs is avoided by considering only *minimal* ones. [7] proposes a notion of lineage of tuples for a type of incomplete databases but does not consider recursive queries. It turns out that we can also describe the lineage in [7] by means of a special commutative semi-ring.

The first attempt at a general theory of relations with annotations appears to be [32] where axiomatized *label systems* are introduced in order to study containment.

14. Conclusion and Further Work

The results on algebraic completion may not be as tight as they can be. Ideally, we would like to be able show that for each representation system we consider, the fragment of $\mathcal{RA}$ we use is minimal in the sense that closing the representation system under a more restricted fragment does not obtain a complete representation system.

We did not consider c-tables with global conditions [24] nor did we describe the exact connection to logical databases [38, 44]. Even more importantly, we did not consider complexity issues as in [3]. All of the above are important topics for further work, especially the complexity issues and the related issues of *succinctness/compactness* of the table representations.

As we see, in pc-tables the probability distribution is on the values taken by the variables that occur in the table. The variables are assumed independent here. This is a lot more flexible (as the example shows) than independent tuples, but still debatable. Consequently, to try to make pc-tables even more flexible, it would be worthwhile to investigate models in which the assumption that the variables take values independently is relaxed by using conditional probability distributions [21].

It would be interesting to connect this work to the extensive literature on *disjunctive databases*, see e.g., [35], and to the work on probabilistic object-oriented databases [19].

Probabilistic modeling is by no means the only way to model uncertainty in information systems. In particular it would be interesting to investigate *possibilistic* models [29] for databases, perhaps following again, as we did here, the parallel with incompleteness.

Query answering on annotated relations can be extended beyond $\mathcal{RA}^+$ to recursive Datalog programs, using semi-rings of formal power series (see [27] for details). These formal power series, which can be represented finitely using a system of equations, are the foundation of trust policies and incremental update propagation algorithms in the ORCHESTRA collaborative data sharing system [26].

Beyond the technical results, the approach surveyed above can be regarded also as arguing that various forms of K-relations, even multisets, provide coarser forms of provenance while the polynomial annotations are, by virtue of their “universality” (as illustrated by the factorization theorem) the most general form of annotation possible with the boundaries of semi-ring structures. This might be a perspective worth using when, in the future, we search for provenance structures for data models other than relational.

Acknowledgments

The author is grateful to Grigoris Karvounarakis and Val Tannen, his co-authors of the papers [28, 27] on which this survey is based.

References

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison–Wesley, Reading, MA, 1995.

[3] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):159–187, 1991.

[4] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.

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

[6] D. Barbara, H. Garcia-Molina, and D. Porter. A probabilistic relational data model. In *EDBT*, pages 60–74, New York, NY, USA, 1990.

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

[8] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *PODS*, pages 1–12, 2008.

[9] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[10] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB*, pages 71–81, 1987.

[11] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[12] L. Chiticariu and W.-C. Tan. Debugging schema mappings with routes. In *VLDB*, 2006.

[13] P. Crawley and R. P. Dilworth. *Algebraic Theory of Lattices*. Prentice Hall, 1973.

[14] Y. Cui. *Lineage Tracing in Data Warehouses*. PhD thesis, Stanford University, 2001.

[15] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *TODS*, 25(2), 2000.

[16] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[17] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM TODS*, 21(3):339–369, 1996.

[18] R. Durrett. *Probability: Theory and Examples*. Duxbury Press, 3rd edition, 2004.

[19] T. Eiter, J. J. Lu, T. Lukasiewicz, and V. S. Subrahmanian. Probabilistic object bases. *ACM Trans. Database Syst.*, 26(3):264–312, 2001.

[20] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, London, UK, 2003. Springer-Verlag.

[21] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models with structural uncertainty. In *Proc. ICML*, 2001.

[22] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *TOIS*, 14(1), 1997.

[23] E. Grädel, Y. Gurevich, and C. Hirch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[24] G. Grahne. Horn tables - an efficient tool for handling incomplete information in databases. In *PODS*, pages 75–82. ACM Press, 1989.

[25] G. Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1991.

[26] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007.

[27] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semi-rings. In *PODS*, 2007.

[28] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *EDBT Workshops*, 2006.

[29] J. Y. Halpern. *Reasoning About Uncertainty*. MIT Press, Cambridge, MA, 2003.

[30] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

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

[32] Y. E. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: beyond relations as sets. *TODS*, 20(3), 1995.

[33] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM TODS*, 22(3):419–469, 1997.

[34] L. V. S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *ILPS*, pages 254–268, Cambridge, MA, USA, 1994. MIT Press.

[35] N. Leone, F. Scarcello, and V. S. Subrahmanian. Optimal models of disjunctive logic programs: Semantics, complexity, and computation. *IEEE Trans. Knowl. Data Eng.*, 16(4):487–503, 2004.

[36] L. Libkin. *Aspects of Partial Information in Databases*. PhD thesis, University of Pennsylvania, 1994.

[37] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. *J. Computer and System Sci.*, 52(1):125–142, 1996.

[38] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM*, 33(2):349–370, 1986.

[39] F. Sadri. Modeling uncertainty in databases. In *ICDE*, pages 122–131. IEEE Computer Society, 1991.

[40] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4), 1980.

[41] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[42] D. Suciu and N. Dalvi. Foundations of probabilistic answers to queries (tutorial). In *SIGMOD*, pages 963–963, New York, NY, USA, 2005. ACM Press.

[43] R. van der Meyden. Logical approaches to incomplete information: A survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer Academic Publishers, Boston, 1998.

[44] M. Y. Vardi. Querying logical databases. *JCSS*, 33(2):142–160, 1986.

[45] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, Jan. 2005.

[46] L. A. Zadeh. Fuzzy sets. *Inf. Control*, 8(3), 1965.

[47] E. Zimányi. Query evaluation in probabilistic databases. *Theoretical Computer Science*, 171(1–2):179–219, 1997.

[48] E. Zimányi and A. Pirotte. Imperfect information in relational databases. In *Uncertainty Management in Information Systems*, pages 35–88. Kluwer, 1996.

15. Appendix

PROPOSITION 15.1 *There exists a relational query q such that $q(\mathcal{N}) = \mathcal{Z}_n$.*

Proof: Define sub-query q' to be the relational query

$$q'(V) := V - \pi_\ell(\sigma_{\ell \neq r}(V \times V)),$$

where ℓ is short for $1, \ldots, n$ and $\ell \neq r$ is short for $1 \neq n+1 \vee \cdots \vee n \neq 2n$. Note that q' yields V if V consists of a single tuple and $\emptyset$ otherwise. Now define q to be the relational query

$$q(V) := q'(V) \cup (\{t\} - \pi_\ell(\{t\} \times q'(V))),$$

where t is a tuple chosen arbitrarily from $\mathbb{D}^n$. It is clear that $q(\mathcal{N}) = \mathcal{Z}_n$. ■

DEFINITION 15.2 *A table in the representation system $\mathcal{R}_{sets}$ is a multiset of sets of tuples, or* blocks, *each such block optionally labeled with a '?'. If T is an $\mathcal{R}_{sets}$ table, then Mod(T) is the set of instances obtained by choosing one tuple from each block not labeled with a '?', and at most one tuple from each block labeled with a '?'.*

DEFINITION 15.3 *A table in the representation system $\mathcal{R}_{\oplus\equiv}$ is a multiset of tuples $\{t_1, \ldots, t_m\}$ and a conjunction of logical assertions of the form $i \oplus j$ (meaning t_i or t_j must be present in an instance, but not both) or $i \equiv j$ (meaning t_i is present in an instance iff t_j is present in the instance). If T is an $\mathcal{R}_{\oplus\equiv}$ table then Mod(T) consists of all subsets of the tuples satisfying the conjunction of assertions.*

DEFINITION 15.4 *A table in the representation system $\mathcal{R}^A_{prop}$ is a multiset of or-set tuples $\{t_1, \ldots, t_m\}$ and a Boolean formula on the variables $\{t_1, \ldots, t_m\}$. If T is an $\mathcal{R}^A_{prop}$ table then Mod(T) consists of all subsets of the tuples satisfying the Boolean assertion, where the variable t_i has value* true *iff the tuple t_i is present in the subset.*

Theorem 5.3 ($\mathcal{RA}$-Completion).

1 *The representation system obtained by closing Codd tables under $SPJU$ queries is $\mathcal{RA}$-complete.*

2 *The representation system obtained by closing v-tables under SP queries is $\mathcal{RA}$-complete.*

Proof: In each case we show that given an arbitrary c-table T, one can construct a table S and a query q such that $\bar{q}(S) = T$.

1 Trivial corollary of Theorem 3.2.

2 Let k be the arity of T. Let $\{t_1, \ldots, t_m\}$ be an enumeration of the tuples of T, and let $\{x_1, \ldots, x_n\}$ be an enumeration of the variables which appear in T. Construct a v-table S with arity $k + n + 1$ as follows. For every tuple t_i in T, put exactly one tuple t'_i in S, where t'_i agrees with t_i on the first k columns, the $k + 1$st column contains the constant i, and the last m columns contain the variables $x_1, \ldots, x_m$. Now let q be the SP query defined

$$q := \pi_{1,\ldots,k}(\sigma_{\bigvee_{i=1}^{m} k+1=`i' \wedge \psi_i}(S))$$

where ψ_i is obtained from the condition $T(t_i)$ of tuple t_i by replacing variable names with their corresponding indexes in S.

■

Theorem 5.4 (Finite Completion).

1 The representation system obtained by closing or-set-tables under PJ queries is finitely complete.

2 The representation system obtained by closing finite v-tables under PJ or S^+P queries is finitely complete.

3 The representation system obtained by closing $\mathcal{R}_{sets}$ under PJ or PU queries is finitely complete.

4 The representation system obtained by closing $\mathcal{R}_{\oplus\equiv}$ under S^+PJ queries is finitely complete.

Proof: Fix an arbitrary finite incomplete database $\mathcal{I} = \{I_1, \ldots, I_n\}$ of arity k. It suffices to show in each case that one can construct a table T in the given representation system and a query q in the given language such that $q(Mod(T)) = \mathcal{I}$.

1 We construct a pair of or-set-tables S and T as follows. (They can be combined together into a single table, but we keep them separate to simplify the presentation.) For each instance I_i in $\mathcal{I}$, we put all the tuples of I_i in S, appending an extra column containing value i. Let T be the or-set-table of arity 1 containing a single tuple whose single value is the or-set $\langle 1, 2, \ldots, n\rangle$. Now let q be the S^+PJ query defined:

$$q := \pi_{1,\ldots,k}\sigma_{k+1=k+2}(S \times T).$$

2 Completion for PJ follows from Case 1 and the fact that finite v-tables are strictly more expressive than or-set tables. For S^+P, take the finite

v-table representing the cross product of S and T in the construction from Case 1, and let q be the obvious S^+P query.

3 Completion for PJ follows from Case 1 and the fact (shown in [41]) that or-set-tables are strictly less expressive than $\mathcal{R}_{\text{sets}}$. Thus we just need show the construction for PU. We construct an $\mathcal{R}_{\text{sets}}$ table T as follows. Let m be the cardinality of the largest instance in $\mathcal{I}$. Then T will have arity km and will consist of a single block of tuples. For every instance I_i in $\mathcal{I}$, we put one tuple in T which has every tuple from I_i arranged in a row. (If the cardinality of I_i is less than m, we pad the remainder with arbitrary tuples from I_i.) Now let q be the PU query defined as follows:

$$q := \bigcup_{i=0}^{m-1} \pi_{ki,\dots,ki+k-1}(T)$$

4 We construct a pair of $\mathcal{R}_{\oplus\equiv}$-tables S and T as follows. (S can be encoded as a special tuple in T, but we keep it separate to simplify the presentation.) Let $m = \lceil \lg n \rceil$. T is constructed as in Case 2. S is a binary table containing, for each i, $1 \leq i \leq m$, a pair of tuples $(0, i)$ and $(1, i)$ with an exclusive-or constraint between them. Let sub-query q' be defined

$$q' := \prod_{i=1}^{m} \pi_1(\sigma_{2=`i'}(S))$$

The S^+PJ query q is defined as in Case 2, but using this definition of q'.

■

Theorem 5.5 (General Finite Completion). *Let $\mathcal{T}$ be a representation system such that for all $n \geq 1$ there exists a table T in $\mathcal{T}$ such that $|Mod(T)| \geq n$. Then the representation system obtained by closing $\mathcal{T}$ under $\mathcal{RA}$ is finitely-complete.* **Proof:** Let $\mathcal{T}$ be a representation system such that for all $n \geq 1$ there is a table T in $\mathcal{T}$ such that $|Mod(T)| \geq n$. Let $\mathcal{I} = \{I_1, ..., I_k\}$ be an arbitrary non-empty finite set of instances of arity m. Let T be a table in $\mathcal{T}$ such that $Mod(T) = \{J_1, \dots, J_\ell\}$, with $\ell \geq k$. Define $\mathcal{RA}$ query q to be

$$q(V) := \bigcup_{1\leq i\leq k-1} I_i \times q_i(V) \cup \bigcup_{k\leq i\leq \ell} I_k \times q_i(V),$$

where I_i is the query which constructs instance I_i and $q_i(V)$ is the Boolean query which returns true iff V is identical to I_i (which can be done in $\mathcal{RA}$). Then $q(Mod(T)) = \mathcal{I}$. ■