m-tables: Representing Missing Data

Bruhathi Sundarmurthy[1], Paraschos Koutris[1], Willis Lang[3], Jeffrey Naughton[4], and Val Tannen[5]

1 **University of Wisconsin-Madison, Madison, WI, USA**
2 **University of Wisconsin-Madison, Madison, WI, USA**
3 **Microsoft Gray Systems Lab, Madison, WI, USA**
4 **University of Wisconsin-Madison, Madison, WI, USA**
5 **University of Pennsylvania, Pennsylvania, PA, USA**

Abstract

Representation systems have been widely used to capture different forms of incomplete data in various settings. However, existing representation systems are not expressive enough to handle the more complex scenarios of missing data that can occur in practice: these could vary from missing attribute values, missing a known number of tuples, or even missing an unknown number of tuples. In this work, we propose a new representation system called *m*-tables, that can represent many different types of missing data. We show that *m*-tables form a *closed*, *complete* and *strong* representation system under both set and bag semantics and are strictly more expressive than conditional tables under both the closed and open world assumptions. We further study the complexity of computing certain and possible answers in *m*-tables. Finally, we discuss how to "interpret" *m*-tables through a novel labeling scheme that marks a type of generalized tuples as certain or possible.

1998 ACM Subject Classification H.2.4 [Database Management] Systems

Keywords and phrases missing values, incomplete data, c tables, representation systems

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.21

1 Introduction

Advances in technology have resulted in a large amount of data being collected and integrated from numerous data sources. A popular way to store such massive data-sets is by sharding over multiple independent relational database instances. When a user given query spans over such a collection, the complete data required by the query may not always be available. For instance, due to inconsistencies and incompleteness during data integration pipelines, data-sets are frequently incomplete and missing data [7, 11, 21]. As another example, when a query spans a large collection of shards, it becomes likely that some nodes will be unavailable for query processing due to outages, misconfigurations, or network congestion. This scenario of missing data due to node failures has also received attention in the applied literature, with a recent work [19] advocating an interesting solution: to "ignore" failures during query evaluation, and to inform the user of problems that may exist by labeling the result with possible errors.

Missing data in such scenarios can be of varied types: a database could be missing attribute values, a set of tuples with partially known values along with bounds on the cardinality of missing tuples, or even an unknown number of tuples. Current techniques that process queries over incomplete databases cannot handle the numerous kinds of missing data described above. For this reason, we propose in this paper a new representation system called *m*-tables to represent and operate on incomplete databases.

20th International Conference on Database Theory (ICDT 2017).
Editors: Michael Benedikt and Giorgio Orsi; Article No. 21; pp. 21:1–21:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Table R.

Name	DOB	State	Salary	
William Smith	6/5/1976	CA	100000	certain tuple
John Doe	?	?	?	two records may be missing
?	?	WI	?	up to 50 employee records may be missing
?	?	PA	?	unknown number of employee records may be missing

Querying over incomplete databases has been widely studied in the literature [1, 22]. The standard approach is to model an incomplete database using a formal representation system, M, that can succinctly describe the incomplete database, and then query over M. For example, conditional tables (c-tables) [16], a widely used representation system, are expressive enough to efficiently capture the result of any relational algebra query over the incomplete database represented by the c-table (such a representation system is called *strong* for relational algebra). However, c-tables can only represent missing attribute values under the so-called closed world assumption, and under the open world assumption (where they can represent an unknown number of missing tuples, but in a very limited manner) they are not closed for the selection operator. Other representation systems, such as v-tables or Codd-tables are even less expressive than c-tables. We note here that, to the best of our knowledge, none of the existing representation systems can represent incomplete databases consisting of zero tuples (the zero information incomplete database), or tuples with cardinality constraints (including possibly infinite cardinality). A different approach than using representation systems is to compute *certain* answers: a tuple is considered to be a certain answer if it is in the result of executing a given query over every possible instance of the incomplete database. However, such an approach loses information about the incomplete database.

To give a more concrete example, consider a practical scenario of a user running analytical queries over a column store database that consists of information about a company's employees in various states. Due to missing values in the database and due to node failures during query execution, the data required to process the analytical queries may not be completely available. A toy version of a resulting incomplete database is shown in Table 1. The last column in table R indicates whether the tuple is certainly part of the table or not and if not, what constraints are placed on the missing data. The first tuple is definitely part of the dataset. The second tuple indicates that we may be missing two employees, both with name 'John Doe'. We could be missing up to fifty employees from the state of Wisconsin as indicated by the third tuple, and the fourth tuple indicates that we could be missing an unknown number of tuples from the state of Pennsylvania. None of the existing representation systems can handle the last three types of missing data.

Contributions. We summarize below the contributions of this work:

1. We present in detail a new representation system called m-tables (Section 3). To construct our new representation we use several ideas, among which is the use of annotations in the form of polynomials (similar to provenance polynomials [13]). We give several examples of how m-tables can be used to express various types of missing data that can arise in practice (Section 3.3).
2. We show (in Section 4) that m-tables are a strong representation system for positive relational algebra $\mathcal{RA}^+$ (which includes selection, projection, join, union and renaming) under both set and bag semantics. This means that we can efficiently compute a new m-table that represents the result of a $\mathcal{RA}^+$ query over underlying m-tables. We should

emphasize here that an important feature of m-tables is that it can capture bag semantics, in contrast to other representation systems and current approaches, where operations under bag semantics are not clearly defined [15].

3. We prove that m-tables are strictly more expressive than c-tables, under both the closed and open world assumptions (Section 5). We also perform a detailed study of the expressive power of m-tables (also Section 5).
4. Given a query result as an m-table, interpreting the semantics to determine whether a tuple is a certain or a possible answer is not an easy task. We thus propose a labeling scheme (Section 6) that interprets the annotations and labels a type of "generalized" tuples with either certain or possible labels, along with other possible information. We show that, as a consequence of the labeling process, we can study the complexity of computing the certain and possible tuples of an m-table representation.

2 Background

In this section, we present the necessary background on representation systems of incomplete data. We will focus on *c-tables*, a representation system that will be of interest to us. We further provide an overview of semiring algebras and provenance semirings [13].

2.1 Representation Systems

We assume that a relational instance is defined over a countably infinite domain $\mathbb{D}$. For the sake of simplicity, we will present the definitions over a relational schema with a single relation with attribute set U; the definitions extend in a straightforward way to any database schema. We use the convention that a *tuple* is a function $t : U \to \mathbb{D}$, and we let $\mathbb{D}^U$ denote the set of all possible tuples with schema U.

An *incomplete database* $\mathcal{I}$ is any set of finite instances $I \subseteq \mathbb{D}^U$ (an instance of an incomplete database refers to a possible completion of the incomplete database). The standard definition of a complete database corresponds to a singleton set $\{I\}$. Representation systems can concisely describe an incomplete database: a *representation system* consists of a set of elements $\mathcal{M}$, and a function Mod that maps each $M \in \mathcal{M}$ to an incomplete database $\mathcal{I}$. For a query $q \in \mathcal{L}$, where $\mathcal{L}$ is a query language, we define the answer of q on the incomplete database $\mathcal{I}$ as $q(\mathcal{I}) = \{q(I) \mid I \in \mathcal{I}\}$. In this work, we will mainly focus on *positive Relational Algebra*, or $\mathcal{RA}^+$, which contains queries that are formed using the selection, projection, join, union and renaming, and the full *Relational Algebra*, $\mathcal{RA}$, which additionally uses difference.

▶ **Definition 1** (Closure)**.** A representation system is *closed* under a query language $\mathcal{L}$ if for any query $q \in \mathcal{L}$ and any $M \in \mathcal{M}$, there exists some $M' \in \mathcal{M}$ such that $q(\mathsf{Mod}(M)) = \mathsf{Mod}(M')$. We further say that it is *strong* for $\mathcal{L}$ if M' is computable.

In addition to the closure property, we are interested in representation systems where we can efficiently perform the following tasks [14, 24]:

Instance Membership. Given an instance I and $M \in \mathcal{M}$, is $I \in \mathsf{Mod}(M)$?

Tuple Membership. Given a tuple $t \in \mathbb{D}^U$ and $M \in \mathcal{M}$, does there exist some instance $I \in \mathsf{Mod}(M)$ such that $t \in I$?

Tuple Certainty. Given a tuple $t \in \mathbb{D}^U$ and $M \in \mathcal{M}$, does $t \in I$ for every $I \in \mathsf{Mod}(M)$?

We should mention here that tuple and instance membership, as well as tuple certainty, can be extended to be defined with respect to a given query q. For example, for tuple membership we can ask whether for a given tuple t, $M \in \mathcal{M}$ and a query q, there exists $I \in q(\mathsf{Mod}(M))$ such that $t \in I$.

A	B	C	
10	2	y	$x = 100 \vee y = 210$
40	x	x	$x \neq 100$

Figure 1 Example of a c-table.

Conditional Tables. A *conditional table, or c-table,* is expressed as (S, ϕ), where S is a table that contains variables along with constant values from $\mathbb{D}$, and ϕ is a function that associates a local condition ϕ_s (a boolean combination of equalities involving variables and constants) with each tuple $s \in S$.

As an example c-table, consider $C(A, B, C)$ with two tuples, t_1 and t_2. $t_1 = (10, 2, y)$ with conditions $(x = 100 \vee y = 210)$ and $t_2 = (40, x, x)$ with condition $(x \neq 100)$.

Let v denote a valuation that maps the variables in a c-table to values in $\mathbb{D}$. Under the *closed world assumption (CWA)*, a c-table $C = (S, \phi)$ represents: $\mathsf{Mod}_C(C) = \{I \mid \exists$ valuation v s.t. $I = \{v(t) \mid t \in S, v$ satisfies $\phi_t\}\}$. The CWA for missing data assume that all information about an incomplete database is modeled by its representation. Alternatively, a c-table can be defined under the *open world assumption (OWA)*, where an instance of an incomplete database can contain any number of tuples, not necessarily justified by the presence of a tuple in its representation. Under OWA: $\mathsf{Mod}_O(C) = \{I \mid \exists$ valuation v s.t. $I \supseteq \{v(t) \mid t \in S, v$ satisfies $\phi_t\}\}$. c-tables form a closed and strong representation system for $\mathcal{RA}$ [16] under CWA. However, c-tables are not closed even for $\mathcal{RA}^+$ under OWA. We will present a detailed comparison of c-tables with m-tables in Section 3.

2.2 Semiring Algebras and Provenance

A *commutative monoid* is a structure $(M, +_M, 0_M)$ where $+_M$ is an associative and commutative binary operation and 0_M is the identity for $+_M$. A *commutative semiring* is a structure $(K, +_K, \cdot_K, 0_K, 1_K)$, where $(K, +_K, 0_K)$ and $(K, \cdot_K, 1_K)$ are commutative monoids, $\cdot_K$ is distributive over $+_K$, and $a \cdot_K 0_K = 0_K \cdot_K a = 0_K$. Examples of commutative semirings are the natural number semiring $(\mathbb{N}, +, \cdot, 0, 1)$ and the boolean semiring $(\mathbb{B}, \vee, \wedge, \bot, \top)$. A *semiring homomorphism* is a mapping $h : K \to K'$ where K, K' are semirings and $h(0_K) = 0_{K'}$, $h(1_K) = 1_{K'}$, $h(a +_K b) = h(a) +_{K'} h(b)$, $h(a \cdot_K b) = h(a) \cdot_{K'} h(b)$.

The work on provenance semirings [13, 4, 12, 18] established the theoretical foundations and implementations for representing, computing and querying *annotated relations*. Many applications need to manipulate some "property" of tuples. These properties, viewed as annotations, and operations on these tuple annotations together form the *semiring* algebraic structure. These semiring structures adequately capture enough information for a variety of applications including obtaining *what* and *how* [5] provenance information. In particular, the *how* provenance of result tuples is captured by annotations from the provenance semiring, $(\mathbb{N}[X], +, \cdot, 0, 1)$, where $\mathbb{N}[X]$ represents the multivariate polynomials with indeterminates from X (a set of *provenance tokens* that we use to annotate input tuples).

Let U be a finite set of attributes and $(K, +, \cdot, 0, 1)$ be a commutative semiring. A *K-relation* is a function $R : \mathbb{D}^U \to K$, whose support, $\mathsf{supp}(R) = \{t \mid R(t) \neq 0\}$ is finite. The $+$ operation in the semiring represents alternate derivations for the same tuple, and the $\cdot$ operation represents the joint use of data to obtain the tuple. 1 represents a tuple that is present in the result and 0 represents the absence of that tuple. We refer the reader to [13] for details on how to execute relational algebra queries over K-relations.

3 How to Represent Missing Data

In this section, we develop a representation system, called *m-tables*, which allows us to represent relational data with missing tuples. Although there exists extensive literature on representation systems for incomplete and uncertain data (from *c*-tables [16], to more recent research [14, 24]), existing representation systems cannot represent and operate on missing data that may consist of zero to any number of tuples. Consider the following example.

▶ **Example 2.** Let $R(A, B, C)$ be a ternary relation, and suppose that the domain $\mathbb{D}$ is the natural numbers $\mathbb{N}$. Our goal is to represent the incomplete database $\mathcal{I}$ that contains all instances that satisfy the following conditions:

- The tuple $(1, 2, 3)$ must be included in any instance.
- The tuple $(2, 3, 4)$ may be present in an instance; if present, its multiplicity should be 2.
- Any other tuple, if present, must be of the form $(x, y, 3)$, where $x, y \in \mathbb{D}$ and $3 \leq x \leq 10$.

$\{R(1, 2, 3), R(2, 3, 4), R(2, 3, 4), R(4, 5, 3)\}$ and $\{R(1, 2, 3), R(4, 5, 3), R(5, 5, 3)\}$ are possible instances of $\mathcal{I}$, but $\{R(2, 3, 4), R(4, 5, 3)\}$ and $\{R(1, 2, 3), R(2, 3, 4), R(4, 5, 3)\}$ are not. Observe that an instance in $\mathcal{I}$ is a *bag*, and not a set. Each possible instance of this incomplete database $\mathcal{I}$ can contain zero to any number of tuples of the form $(x, y, 3)$, and this cannot be represented by a *c*-table (under either open or closed world semantics), *v*-table or other representation system.

An incomplete database like $\mathcal{I}$ is not only of theoretical interest, since it can be the result of answering a query over a partitioned database (can also be a column store), where some partition has failed during evaluation (additionally, some columns may be unavailable for processing). Example 2 will be the running example throughout this section.

3.1 Basic Definitions

In this section, we define the components of an *m*-table in a bottom-up manner and then proceed to discuss *m*-table semantics. Informally, the construction involves two requirements. First, we need to represent 'missing' values along with the associated domain and cardinality constraints; we also need to encode correlation constraints between the missing values. Second, we require that the representation allows for systematic propagation of the constraints of missing values through the algebraic operators. To achieve this, we define a schema over the missing values (encoding correlations) with cardinality constraints defined over the schema elements. Next, we observe that propagating the schema information along with domain constraints is akin to propagating the provenance information for tuples. So, we introduce suitable annotations for tuples and propagate these annotations using machinery from [13].

We now begin the construction of *m*-tables. Let $\mathbf{U}$ denote the set of possible attribute names and $\mathbb{D}$ denote the set of all constants. $\mathbb{D}$ represents the domain of all the attributes.

Missing values. The first component is a distinguished symbol m, which represents any *missing value*. Define $\hat{\mathbb{D}} = \mathbb{D} \cup \{m\}$. This notation is similar in spirit to other types of representation systems; the difference in our setting is that m can potentially represent multiple possible values, or even no values at all, instead of exactly one. One should think of m as representing a set or a bag (depending on the semantics) of possible values.

We introduce the notion of an *extended tuple* $\hat{t} \in \hat{\mathbb{D}}^U$, where $U \subseteq \mathbf{U}$, to represent tuples in the representation $M \in \mathcal{M}$. This notation is meant to distinguish from a tuple t in an instance of an incomplete database. Notice that the distinguished symbol m can only be part of a tuple $\hat{t}$ and not t. We define $\mathsf{m}(\hat{t}) = \{A \in U \mid \hat{t}[A] = m\}$ and $\bar{\mathsf{m}}(\hat{t}) = \{A \in U \mid \hat{t}[A] \neq m\}$.

In other words, $\mathsf{m}(\hat{t})$ includes the attributes in the tuple that have a missing value instead of a constant value. For example, to represent the incomplete database in Example 2, we introduce three extended tuples: $\hat{t}_1 = (1, 2, 3)$, $\hat{t}_2 = (2, 3, 4)$ and $\hat{t}_3 = (m, m, 3)$.

The set of extended tuples is not enough by itself to represent the incomplete database $\mathcal{I}$ of Example 2. Indeed, there is no way to distinguish that $(1, 2, 3)$ is a *certain* tuple, in the sense that it appears in all instances, and $(2, 3, 4)$ is a *possible* tuple with multiplicity 2, in the sense that it appears in some instances. Furthermore, we want to ensure that for the tuple $(m, m, 3)$, the first m can only take values between 3 and 10, which is not yet captured. We thus augment the table with annotations.

However, simply annotating tuples with domain conditions will not be sufficient. A structure is necessary to encode correlations between different missing values that appear in different tuples or attributes of the same tuple. For instance, consider the tuple $\hat{t}_3 = (m, m, 3)$; consider another relation $S(C, D, E)$ with tuples $\hat{s}_1 = (3, 4, 5)$ and $\hat{s}_2 = (3, 9, 10)$. If $\hat{t}_3$ were to be joined with the relation S on attribute C, then we will have two tuples in the result: $r_1 = (m, m, 3, 4, 5)$ and $r_2 = (m, m, 3, 9, 10)$. Observe that $\mathsf{m}(\hat{r}_1) = \mathsf{m}(\hat{r}_2) = \{A, B\}$; the pairs of m values across tuples are not independent and are bound by the values taken by $\mathsf{m}(\hat{t}_3)$. To capture this, we need to relate annotations to the m values in the tuples; we do so by introducing a second component, a database schema $\Sigma = \{T_1(U_1), T_2(U_2), \ldots, T_N(U_N)\}$, where $U_i \subseteq \mathbf{U}$ for each $i = 1, \ldots, N$.

For the running example, we need a schema that allows the presence of the tuple $(2, 3, 4)$ (with multiplicity 2) to be toggled and allows the tuple $(m, m, 3)$ to take on multiple values for m. We construct the schema $\Sigma = \{T_1(), T_2(A, B)\}$. Observe that relation T_1 has no attributes: this means that T_1 will behave like a boolean variable depending on whether T_1 is empty or contains the empty tuple $()$ (this is also because of the additional cardinality constraints that we introduce next).

An instantiation of Σ determines an instance of the incomplete database. The size of the possible instantiations of Σ are constrained by two vectors, $\mathbf{min} = (min_1, \ldots, min_N)$ and $\mathbf{max} = (max_1, \ldots, max_N)$, where $min_i, \max_i \in \mathbb{N} \cup \{\infty\}$. The number of tuples in every instantiation of $T_i \in \Sigma$ are lower bounded by min_i and upper bounded by max_i. For our running example, we add the cardinality constraints $min_1 = 0, max_1 = 1$ and $min_2 = 0, max_2 = \infty$. The constraints enforce that T_1 behaves like a boolean variable and T_2 can be instantiated to anything.

We next introduce annotations that capture all necessary properties of an extended tuple; as we will argue in the next section, annotations are also necessary to make the representation system complete for SPJU queries.

Annotations. To construct a suitable set of annotations for m-tables, we first need to define two new kinds of expressions. The first kind has expressions of the form $\alpha_i(U)$, where $i = 1, \ldots, N$ corresponds to the relation $T_i(U_i)$ of the schema Σ, with $U \subseteq \mathbf{U}$ and $|U| = |U_i|$. The condition $|U| = |U_i|$ is sufficient, since as we will see in the next section, applying a renaming operator can change the attributes in α_i. We define $\mathcal{K} = \{\alpha_i(U) \mid T_i(U_i) \in \Sigma, U \subseteq \mathbf{U}, |U| = |U_i|\}$. In the preceding definition, not requiring $U_i = U$ allows the reuse of α expressions across multiple attributes and tables.

The second kind of expressions are symbolic equations, which will be used to capture selection and join conditions in query evaluation. We define $\mathcal{E} = \{[x \ \mathsf{op} \ y] \mid x, y \in \mathbb{D} \cup \mathbf{U}, \mathsf{op} \in \{=, <, >, \leq, \geq, \neq\}\}$.

For example, if $A, B \in \mathbf{U}$ and $\mathbb{D} = \mathbb{N}$, both $[A = B]$ and $[A > 3]$ are valid expressions in $\mathcal{E}$. This definition is similar to the technique used in [4] to capture provenance for

queries with aggregates. We will use the above expressions to annotate each extended tuple. Formally, let $\hat{K}$ be the *polynomial semiring* with variables from $\mathcal{K} \cup \mathcal{E}$ and coefficients from $\mathbb{N}$: $(\mathbb{N}[\mathcal{K} \cup \mathcal{E}], +, \cdot, 0, 1)$. We can then define a $\hat{K}$-relation R that maps each extended tuple in $\hat{\mathbb{D}}^U$ to an element in the semiring $\hat{K}$. The coefficients from $\mathbb{N}$ in the polynomial enable the annotations to handle both set and bag semantics.

▶ **Example 3.** Continuing the running example, the annotation for the tuple $(1, 2, 3)$ is 1 (the tuple is certain). The annotation for the tuple $(2, 3, 4)$ is $2 \cdot \alpha_1()$, with $\alpha_1()$ operating like a boolean variable. Alternately, we could have had two $(2, 3, 4)$ tuples, each with annotation $\alpha_1()$. Finally, the annotation for the tuple $(m, m, 3)$ is $\alpha_2(A, B) \cdot [A \leq 10] \cdot [A \geq 3]$. Intuitively, for each instance of the relation $T_2(A, B)$, we first filter the tuples using the conditions of the expressions $[A \leq 10]$ and $[A \geq 3]$. The result will define a set of valuations from (A, B) to $(\mathbb{D}, \mathbb{D})$; each such valuation will correspond to a tuple in the possible world.

An annotation $R(\hat{t}) \in \mathbb{N}[\mathcal{K} \cup \mathcal{E}]$ has a natural interpretation as a query in $\mathcal{RA}^+$. We first write the polynomial $R(\hat{t})$ in the following canonical form[1]: $R(\hat{t}) = \sum_{k=1}^{n} (\prod_{k_i} \alpha_{k_i}(U_{k_i}) \cdot \prod_{k_j} \theta_{k_j})$ where, $\theta_{k_j} \in \mathcal{E}$. We can now interpret each monomial in $R(\hat{t})$ as a query, which involves a renaming operation (to match attributes in α_i and T_i), followed by a natural join, followed by a selection condition specified by the equations θ_{k_j} and followed by a projection on $\mathsf{m}(\hat{t})$. Formally, for the k-th monomial in $R(\hat{t})$, we associate the query: $q_k(R(\hat{t})) = \pi_{\mathsf{m}(\hat{t})} \left(\sigma_{\bigwedge_{k_j} \theta_{k_j}} \left(\bowtie_{k_i} (\rho_{A_{k_i}/U_i} T_{k_i}) \right) \right)$. The result of this query will be a relation defined over the attributes in $\mathsf{m}(\hat{t})$. To obtain the final query associated with $R(\hat{t})$, we first extend this definition over all attributes in U, and then take the union over all monomials. Formally, we map each annotation $R(\hat{t})$ to:

$$q(R(\hat{t})) \stackrel{\text{def}}{=} \bigcup_{k=1}^{n} \left(\pi_{\bar{\mathsf{m}}(\hat{t})} \{\hat{t}(A)\} \times q_k(R(\hat{t})) \right) \tag{1}$$

The query $q(R(\hat{t}))$ returns a relation (set or bag) defined over the attribute set U. In the case where $R(\hat{t}) = 1$, $q(R(\hat{t}))$ is the constant query that returns a relation with an empty tuple $()$.

▶ **Example 4.** For the tuples $\hat{t}_2$, $\hat{t}_3$ we have $q_2 = q(R(\hat{t}_2)) = (\pi_{A,B,C}\{\hat{t}_2\} \times \pi_{()}(T_1)) \cup (\pi_{A,B,C}\{\hat{t}_2\} \times \pi_{()}(T_1))$ and $q_3 = q(R(\hat{t}_3)) = \pi_C\{\hat{t}_3\} \times \pi_{A,B}(\sigma_{A \geq 3 \wedge A \leq 10}(T_2))$. Since $\hat{t}_2$ has two monomials, $\alpha_1() + \alpha_1()$, in its annotation, its final annotation is a union over the queries of each of its monomials.

We could equivalently define an annotation directly as a query in $\mathcal{RA}^+$. The choice to use semirings instead is because they form a more compact annotation and work seamlessly for both set and bag semantics. For instance, the simple annotation $100 \cdot \alpha_1()$ would have to be written as a union of 100 expressions T_1.

The query $q(\gamma)$ may not be well-defined for a given polynomial γ, since a selection condition θ_{k_j} or a projection operator may include an attribute that does not appear in any of the α_{k_i} terms. For example, the annotation $\alpha_2(A, B) \cdot [C = 1]$ for the tuple $(m, m, 3)$ would correspond to the query $\pi_C\{\hat{t}_3\} \times \pi_{A,B}(\sigma_{C=1}(T_2))$, which is not a valid expression.

It is easy to see that an annotation $R(\hat{t})$ is valid if and only if for every monomial $\prod_{k_i} \alpha_{k_i}(A_{k_i}) \cdot \prod_{k_j} \theta_{k_j}$ in the annotation the following hold: (1) $\mathsf{m}(\hat{t}) \subseteq \bigcup_{k_i} A_{k_i}$, and (2) for any $\theta_{k_j} = [x_1 \ \mathsf{op} \ x_2]$ such that $x_i \in \mathbf{U}$, $x_i \in \bigcup_{k_i} A_{k_i}$.

[1] A monomial with coefficient > 1 can be easily split into multiple monomials, thus conforming to the canonical form. For example, the annotation $2 \cdot \alpha_1()$ can be written as $\alpha_1() + \alpha_1()$.

Table 2 The m-table for the running example.

A	B	C	
1	2	3	1
2	3	4	$2 \cdot \alpha_1()$
m	m	3	$\alpha_2(A,B) \cdot [A \leq 10] \cdot [A \geq 3]$

▶ **Definition 5** (m-proper)**.** A $\hat{K}$-relation R over a set of attributes U is *m-proper* if for every extended tuple $\hat{t} \in \hat{\mathbb{D}}^U$, $R(\hat{t})$ is a valid annotation.

Using the conditions for a valid annotation, given a $\hat{K}$-relation, we can efficiently determine whether it is m-proper. We now formally define m-tables.

▶ **Definition 6.** A set of *m-tables*, or an *m-multitable*, is a tuple $(\mathbf{R}, \Sigma, \mathbf{min}, \mathbf{max})$ such that:

1. each $R_j \in \mathbf{R}$ is an m-proper $\hat{K}$-relation, where $\hat{K}$ is the polynomial semiring $(\mathbb{N}[\mathcal{K} \cup \mathcal{E}], +, \cdot, 0, 1)$ (recall that the elements of $\mathcal{K}$ are constructed from the elements of Σ),
2. $\Sigma = \{T_1(U_1), \ldots, T_N(U_N)\}$ is a database schema,
3. $\mathbf{min}, \mathbf{max} \in (\mathbb{N} \cup \{\infty\})^N$ are vectors of cardinality constraints.

To define a single m-table for a relation R, we can simply write it as $M = (R, \Sigma, \mathbf{min}, \mathbf{max})$.

As discussed before, the cardinality constraints min_i, max_i provide a lower and upper bound on the cardinality of an instance of the relation $T_i \in \Sigma$. In the case where $min_i = 0$ and $max_i = \infty$ for every i, we say that the m-table is *free* and for simplicity we omit $\mathbf{min}, \mathbf{max}$ from the m-table definition. We denote by $\mathcal{M}^f$ the set of all free m-tables. When $min_i = max_i = 1$ for every i, we can equivalently view each relation T_i as a function from attributes to values in $\mathbb{D}$; as we will see later, this will allow us to capture the semantics of c-tables. We denote by $\mathcal{M}^c$ the set of all such m-tables. Finally, we define an *$m^{\not=}$-table* as an m-table where the expressions in the annotation are restricted to use only $=, \neq$.

For our running example, the final m-table $M = (R, \Sigma, \mathbf{min}, \mathbf{max})$ will have $\Sigma = \{T_1(), T_2(A, B)\}$ and $min_1 = min_2 = 0$, $max_1 = 1, \max_2 = \infty$. The annotated relation R can be seen in Table 2.

3.2 Semantics

We present here the semantics of m-tables. Given $M = (\mathbf{R}, \Sigma, \mathbf{min}, \mathbf{max})$, we formally define the incomplete database $\mathsf{Mod}(M)$ that it represents under both set and bag semantics.

To explain the semantics behind m-tables, we draw a parallel with c-tables. For a c-table C, each possible instance of the incomplete database is produced by computing $v(C)$ for a valuation v over the variables in the c-table. In m-tables, instead of a valuation, we will use an instance $\mathbf{T}$ on the schema Σ, which satisfies the cardinality constraints; each such instance will produce a possible instance I of the incomplete database: $M[\mathbf{T}]$ in $\mathsf{Mod}(M)$. Under set semantics, the instance I will be a set, and under bag semantics it will be a bag.

We start by looking at a single extended tuple $\hat{t}$ with annotation $R(\hat{t})$, for some $R \in \mathbf{R}$ with attribute set U. Let $J = q(R(\hat{t}))(\mathbf{T})$. As we discussed in the previous section, each tuple $v \in J$ can be equivalently viewed as a total function $v : U \to \mathbb{D}$. We say that $v(\hat{t})$ is an *instantiation* of the extended tuple $\hat{t}$.

▶ **Definition 7** (Derivation Set/Bag)**.** Let $\hat{t}$ be an extended tuple with a valid annotation $R(\hat{t})$. The *derivation set (bag)* of $\hat{t}$ for a set (bag) instance $\mathbf{T}$ of Σ is defined as $q(R(\hat{t}))(\mathbf{T})$.

▶ **Example 8.** Consider the schema Σ and the cardinality constraints of our running example. Consider the following set instance $\mathbf{T}$ of Σ: $\mathbf{T} = \{T_2(2,4), T_2(3,4), T_1()\}$. Then, the derivation set of the tuple $(1,2,3)$ with annotation 1 is $\{(1,2,3)\}$. For the tuple $\hat{t}_2 = (2,3,4)$ with annotation $2 \cdot \alpha_1()$, the derivation set w.r.t. $\mathbf{T}$ will be $\{(2,3,4)\}$. Notice that because of the set semantics, the coefficient 2 in the annotation is effectively ignored. Finally, for $\hat{t}_3 = (m,m,3)$ with annotation $\alpha_2(A,B) \cdot [A \leq 10] \cdot [A \geq 3]$, we compute $q_3(\mathbf{T}) = \{(3,4,3)\}$.

If we switch to bag semantics, we can start with a bag instance of the schema Σ: $\mathbf{T} = \{T_2(2,4), T_2(3,4), T_2(3,4), T_1()\}$. The derivation bag of $(1,2,3)$ will be as before $\{(1,2,3)\}$. For the tuple $\hat{t}_2 = (2,3,4)$ with annotation $2 \cdot \alpha_1()$, the derivation bag w.r.t. $\mathbf{T}$ will now be $\{(2,3,4),(2,3,4)\}$. Observe that the coefficient 2 is now critical for the correct interpretation. Finally for $\hat{t}_3 = (m,m,3)$ with annotation $\alpha_2(A,B) \cdot [A \leq 10] \cdot [A \geq 3]$, we compute $q_3(\mathbf{T}) = \{(3,4,3),(3,4,3)\}$ as its derivation bag.

▶ **Definition 9** (m-table Semantics)**.** Let $M = (\mathbf{R}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-multitable. For $R \in \mathbf{R}$, define the query $\mathcal{Q}_R \stackrel{\text{def}}{=} \bigcup_{\hat{t}:R(\hat{t})\neq 0} q(R(\hat{t}))$. The *instantiation* of M under a (set or bag) instance $\mathbf{T}$ of Σ is $M[\mathbf{T}] = \{\mathcal{Q}_R(\mathbf{T}) \mid R \in \mathbf{R}\}$. Under set (bag) semantics, the incomplete database $\mathsf{Mod}_S(M)$ ($\mathsf{Mod}_B(M)$) that is represented by M is

$$\mathsf{Mod}_{S/B}(M) = \{M[\mathbf{T}] \mid \forall i = 1,\ldots,N : \mathbf{T}^{T_i} \text{ is a set/bag, } min_i \leq |\mathbf{T}^{T_i}| \leq max_i\},$$

In other words, for each instance $\mathbf{T}$, we construct a possible world of the incomplete database by taking the union of all the derivation sets (or bags) for each extended tuple (w.r.t. $\mathbf{T}$) in the annotated relation. This of course is equivalent to computing the query $\mathcal{Q}_R(\mathbf{T})$ for each $R \in \mathbf{R}$. To construct the incomplete database, we compute all possible worlds that correspond to every instance of Σ that satisfies the cardinality constraints $\mathbf{min}, \mathbf{max}$ of the m-table. We present next an example that sheds more light on the semantics of m-tables.

▶ **Example 10.** Consider the binary relation $R(A,B)$ along with two different schemas: $\Sigma_1 = \{T_1(A,B)\}$ and $\Sigma_2 = \{T_2(A), T_3(B)\}$.

Consider first the free m-table $M_1 = (\{R_1\}, \Sigma_1)$, where R_1 contains the tuple (m,m) with annotation $\alpha_1(A,B)$. It is easy to see that $\mathsf{Mod}_S(M_1)$ is the set of all possible instances of R over the domain $\mathbb{D}$, otherwise known as the no-information instance.

Second, consider the free m-table $M_2 = (\{R_2\}, \Sigma_2)$, where R_2 contains, again, a single tuple (m,m) with annotation $\alpha_2(A) \cdot \alpha_3(B)$. Observe now that the incomplete database $\mathsf{Mod}_S(M_2)$ does not include all possible instances, since for example, the instance $\{(1,1),(1,2),(2,1)\}$ can not be produced from M_2.

3.3 Examples and Applications

In this section, we show how to use m-tables to represent different types of missing data that occur in a practical setting. Recall our original motivation: we have a cluster of nodes executing relational queries. Suppose that one of the tables in this cluster is $R(A_1,\ldots,A_k)$, and that during the execution of a query, several nodes become unresponsive. We look at three different cases:

Missing Arbitrary Data. We are certain about several tuples in the table R, but we are missing an arbitrary part, for which we have no information. To represent this instance, our underlying m-table schema needs a single relation: $\Sigma = \{T_1(A_1,\ldots,A_k)\}$. For the annotated relation, we first include in R all the certain tuples with annotation 1. Then, we introduce one more tuple $(m,m,\ldots,m)$ with annotation $\alpha_1(A_1,\ldots,A_k)$.

Missing Data in Range-Partitioned Databases. In this scenario, table R is initially range-partitioned across different nodes in the cluster using an attribute, say A_1. We construct an m-table for this case as follows. For each responsive node, we add the tuples in the nodes as certain tuples with annotation 1. Our underlying schema is the same as before: $\Sigma = \{T_1(A_1, \ldots, A_k)\}$. Let $[x_i, y_i]$ be the range of each missing node i. We then add a tuple $(m, m, \ldots, m)$ with annotation $\alpha_1(A_1, \ldots, A_k) \cdot \sum_i([A_1 \geq x_i] \cdot [A_1 \leq y_i])$.

Missing Data in Column Stores. Suppose that the table R is stored in columnar format. In this case, the columns may not be sharded, but observe that all columns may not be accessed at the same time. A particular node may be accessed several times during query processing while stitching the columns together. One of these accesses might fail and result in a missing column. Let's say we are missing the column corresponding to attribute A_1. We can use m-tables to represent this type of missing data as follows. Every tuple will be of the form $(m, a_2, \ldots, a_k)$, where $a_i \in \mathbb{D}$ and will have annotation $\alpha_j(A_1)$, where we introduce a distinguished unary relation T_j for every tuple. Moreover, we add cardinality constraints such that $min_j = max_j = 1$.

4 $\mathcal{RA}^+$ Algebra for m-tables

In this section we present the specifics of executing operators in the positive relational algebra ($\mathcal{RA}^+$) over m-tables, thus proving that m-tables form a strong representation system for $\mathcal{RA}^+$ under both set and bag semantics. Recall that an m-multitable M is a tuple $(\{R_1, \ldots, R_\ell\}, \Sigma, \mathbf{min}, \mathbf{max})$, where each R_i is an m-proper $\hat{K}$-relation. Thus, in order to define relational operators over m-tables we will need to modify the standard algebra operators over K-relations. We next present how each operator works.

Selection. Let $R : \hat{\mathbb{D}}^U \to \hat{K}$ and let θ be a selection predicate of the form $(A \ \mathsf{op} \ x)$, where $A \in U$, $x \in \mathbb{D}$ and $\mathsf{op} \in \{=, <, >, \leq, \geq, \neq\}$. Then, the selection $\sigma_\theta R : \hat{\mathbb{D}}^U \to \hat{K}$ is defined as

$$(\sigma_\theta R)(\hat{t}) = \begin{cases} R(\hat{t}) \cdot [A \ \mathsf{op} \ x] & \text{if } \hat{t}(A) = m, \\ R(\hat{t}) \cdot [\hat{t}(A) \ \mathsf{op} \ x] & \text{otherwise.} \end{cases}$$

Observe that if $\hat{t}(A) \neq m$, we can immediately evaluate the condition by checking whether the expression $(\hat{t}(A) \ \mathsf{op} \ x)$ is true or not. If it is true, then $(\sigma_\theta R)(\hat{t}) = R(\hat{t})$; otherwise $(\sigma_\theta R)(\hat{t}) = 0$. These semantics coincide with the algebra on K-relations. When $\hat{t}(A) = m$, the attribute value is unknown and the extended tuple $\hat{t}$ may potentially satisfy the selection predicate. Thus, we need to keep the expression uninterpreted as part of the annotation. The case where the condition is of the form $(A \ \mathsf{op} \ B)$ is similar and thus omitted.

Projection. Let $R : \hat{\mathbb{D}}^U \to \hat{K}$ and $U' \subseteq U$. The projection $\pi_{U'} R : \hat{\mathbb{D}}^{U'} \to \hat{K}$ is defined as $(\pi_{U'} R)(\hat{t}) = \sum_{t' : R(t') \neq 0 \wedge \hat{t} = t' \text{ on } U'} R(t')$.

Union. Let $R_1, R_2 : \hat{\mathbb{D}}^U \to \hat{K}$. Then the union $R_1 \cup R_2 : \hat{\mathbb{D}}^U \to \hat{K}$ is defined as $(R_1 \cup R_2)(\hat{t}) = R_1(\hat{t}) + R_2(\hat{t})$.

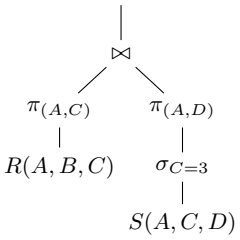
Renaming. Let $R : \hat{\mathbb{D}}^U \to \hat{K}$ and let $\beta : U \to U'$ be a bijection. To define the semantics for the renaming operator ρ_β, we need to rename the attributes in the annotation as well. For this, we define $\beta(\alpha_i(A)) = \alpha_i(\beta(A))$ and also $\beta([x \ \mathsf{op} \ y]) = [\beta(x) \ \mathsf{op} \ \beta(y)]$. (The function β

A	B	C	
a_1	b_1	50	1
a_2	b_2	400	1
a_3	b_3	30	1

(a) Table R.

A	C	D	
a_1	3	450	1
a_2	3	m	$\alpha_1(D)$
a_3	3	100	$\alpha_2()$
m	m	m	$\alpha_3(A, C, D) \cdot [D > 100]$

(b) Table S.



(c) Query Plan.

Figure 2 The initial m-tables R, S and the query plan for the running example.

behaves as the identity function for constants and attributes not in U.) For an annotation $R(\hat{t})$, we now define $\beta(R(\hat{t}))$ as the result of applying β to each element of the polynomial. We can now define $\rho_\beta R$ to be a $\hat{K}$-relation over U' such that: $(\rho_\beta R)(\hat{t}) = \beta(R(\hat{t} \circ \beta))$.

Cartesian Product. Let $R_i : \hat{\mathbb{D}}^{U_i} \to \hat{K}$ for $i = 1, 2$ and let $\hat{t}|U_i$ represent the restriction of the tuple $\hat{t}$ to the attributes of U_i. Then $R_1 \times R_2 : \hat{\mathbb{D}}^{U_1 \cup U_2} \to \hat{K}$ is defined as $(R_1 \times R_2)(\hat{t}) = R_1(\hat{t}_1) \cdot R_2(\hat{t}_2)$ where, $\hat{t}_i = \hat{t}|U_i, i = 1, 2$. An important point is that we assume w.l.o.g. that R_1, R_2 do not share any attributes in the annotations that are not in $U_1 \cup U_2$. If this happens, it is easy to rename these attributes such that there is no conflict.

We should note here that we defined the cartesian product instead of a natural join operator for simplicity of presentation: the natural join can be easily defined as a sequence of renaming, cartesian product, selection and projection.

Given a query $Q \in \mathcal{RA}^+$, and an m-multitable $M = (\{R_1, \ldots, R_\ell\}, \Sigma, \mathbf{min}, \mathbf{max})$, let us define by $\bar{Q}(M)$ the tuple $(\{Q(R_1, \ldots, R_\ell)\}, \Sigma, \mathbf{min}, \mathbf{max})$. Here we should note that we have not yet shown that $\bar{Q}(M)$ is a valid m-table; for this, we need to prove that $Q(R_1, \ldots, R_\ell)$ is an m-proper $\hat{K}$-relation. Before we do this, we first give a detailed example of applying the algebraic operations we defined to m-tables.

▶ **Example 11.** We illustrate querying over m-tables through an example. Consider the m-multitable $M = (\{R, S\}, \Sigma, \mathbf{min}, \mathbf{max})$. R is a complete relation (i.e. all annotations are 1), S has missing data, $\Sigma = \{T_1(D), T_2(), T_3(A, C, D)\}$, $\mathbf{min} = (1, 0, 0)$ and $\mathbf{max} = (1, 1, \infty)$. We present tables R and S in Figures 2a and 2b, respectively, with the initial annotations for the extended tuples. Observe that we have appended multiple extended tuples to relation S to represent its missing data. The relational query to be executed on the database is given in Figure 2c and the results obtained after applying the $\hat{K}$-relational algebra operators are presented in Figure 3.

▶ **Lemma 12.** *Let $M = (\{R_1, \ldots, R_\ell\}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-multitable, and Q be a query in $\mathcal{RA}^+$. Denote $R' = Q(R_1, \ldots, R_\ell)$. Then*

1. *R' is an m-proper $\hat{K}$-relation*
2. *$\mathcal{Q}_{R'} = Q(\mathcal{Q}_{R_1}, \ldots, \mathcal{Q}_{R_\ell})$ under both set and bag semantics, i.e., the incomplete database represented by R' and the resulting incomplete database after applying query Q are the same.*

▶ **Corollary 13.** *The $\mathcal{RA}^+$ operations defined for m-multitables map m-multitables to m-multitables, and thus form a well-defined algebra over m-multitables.*

A	C	D	
a_1	3	450	1
a_2	3	m	$\alpha_1(D)$
a_3	3	100	$\alpha_2()$
m	m	m	$\alpha_3(A,C,D)\cdot[D>100]\cdot[C=3]$

(a) Step 1: $S' = \sigma_{C=3}(S)$.

A	D	
a_1	450	1
a_2	m	$\alpha_1(D)$
a_3	100	$\alpha_2()$
m	m	$\alpha_3(A,C,D)\cdot[D>100]\cdot[C=3]$

(b) Step 2: $S'' = \pi_{A,D}(S')$.

A	C	
a_1	50	1
a_2	400	1
a_3	30	1

(c) Step 3: $R' = \pi_{A,C}(R)$.

A'	D	
a_1	450	1
a_2	m	$\alpha_1(D)$
a_3	100	$\alpha_2()$
m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]$

(d) Step 4.1: $S'' = \rho_{\{A\to A',C\to C'\}}(S'')$.

A	C	A'	D	
a_1	50	a_1	450	1
a_2	400	a_2	m	$\alpha_1(D)$
a_3	30	a_3	100	$\alpha_2()$
a_1	50	m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_1]$
a_2	400	m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_2]$
a_3	30	m	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_3]$

(e) Steps 4.2, 4.3: Result $= \sigma_{A=A'}(R'\times S'')$.

A	C	D		label	ϕ
a_1	50	450	1	certain	T
a_2	400	m	$\alpha_1(D)$	certain	T
a_3	30	100	$\alpha_2()$	possible	T
a_1	50	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_1]$	possible	$(D>100)\wedge(C'=3)\wedge(A'=a_1)$
a_2	400	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_2]$	possible	$(D>100)\wedge(C'=3)\wedge(A'=a_2)$
a_3	30	m	$\alpha_3(A',C',D)\cdot[D>100]\cdot[C'=3]\cdot[A'=a_3]$	possible	$(D>100)\wedge(C'=3)\wedge(A'=a_3)$

(f) Step 4.4: Projection along with application of SimpleLabel algorithm.

Figure 3 Execution of the query plan over m-tables and obtaining m-labeled tuples.

▶ **Theorem 14.** *The m-multitables form a strong representation system for positive relational algebra for both set and bag semantics. Moreover, evaluating a positive relational algebra query on m-multitables has polynomial data complexity.*

We conclude this section by observing that one can apply all the known optimizations on relational algebra plans when querying m-tables, since the standard algebraic identities under bag semantics are preserved (see also Appendix, for more details).

5 The Expressive Power of m-tables

In this section, we discuss the expressive power of m-tables. We first compare the expressiveness of m-tables to c-tables under both the closed and open world assumption. Then, we characterize the set of incomplete databases that can be expressed through m-tables. Our results in this section hold only for the case of set semantics.

m-tables versus c-tables. Our first result shows that the class of $m^{\not=}$-tables in $\mathcal{M}^c$ can capture precisely the expressiveness of c-tables under CWA.

▶ **Theorem 15.** *The c-tables under CWA and the $m^{\not=}$-tables in $\mathcal{M}^c$ have the same expressive power, that is:*
1. *For every c-table C, there exists an $m^{\not=}$-table $M \in \mathcal{M}^c$ such that $\mathsf{Mod}_S(M) = \mathsf{Mod}_C(C)$;*
2. *For every $m^{\not=}$-table $M \in \mathcal{M}^c$, there exists a c-table C such that $\mathsf{Mod}_C(C) = \mathsf{Mod}_S(M)$.*

The detailed proof of this statement is given in the Appendix. As we discussed earlier, m-tables are *strictly* more expressive than c-tables under closed world semantics, since c-tables under CWA cannot express incomplete databases with arbitrarily large instances.

Under the open world assumption, c-tables can express incomplete databases with arbitrarily large instances.

▶ **Proposition 16.** *For every c-table C, there exists an $m^{\not=}$-table M s.t. $\mathsf{Mod}_S(M) = \mathsf{Mod}_O(M)$.*

It turns out that general m-tables are strictly more expressive than c-tables under OWA, in the sense that there exists an m-table M such that $\mathsf{Mod}_S(M)$ is not expressible through a c-table under OWA. Indeed, consider the following example.

▶ **Example 17.** Let $M = (R, \{T_1(A)\}, (0), (\infty))$, where $R(A, B)$ is a binary $\hat{K}$-relation that consists of a single extended tuple $(m, 1)$ with annotation $\alpha_1(A)$. Suppose there exists a c-table C such that $\mathsf{Mod}_S(M) = \mathsf{Mod}_O(C)$. Since the instance $\{(1, 1)\}$ belongs in $\mathsf{Mod}_S(M)$, it must also belong in $\mathsf{Mod}_O(C)$. But then $\{(1, 1), (1, 2)\}$ must also belong in $\mathsf{Mod}_O(C)$; however, this contradicts that C expresses $\mathsf{Mod}_S(M)$, since $(1, 2)$ cannot belong in any instance of $\mathsf{Mod}_S(M)$.

To summarize, m-tables can express a strictly larger class of incomplete databases in comparison to c-tables under both the closed and open world assumption.

Characterizing the Expressiveness. Following [14], we define $\mathcal{N} = \{I \mid I \subseteq \mathbb{D}^U, I \text{ finite}\}$ as the *zero-information* incomplete database. Each subset of $\mathcal{N}$ forms an incomplete database; our goal is to characterize the subsets of $\mathcal{N}$ which are representable by m-tables. We also define $\mathcal{Z}_U = \{\{t\} \mid t \in \mathbb{D}^U\}$ as the incomplete database that represents the set of all relations with exactly one tuple.

▶ **Definition 18** ([14])**.** Let $\mathcal{L}$ be a query language. An incomplete database $\mathcal{I}$ is *$\mathcal{L}$-definable* if there exists a query $Q \in \mathcal{L}$ such that $\mathcal{I} = Q(\mathcal{Z}_U)$. We further say that a representation system is *$\mathcal{L}$-complete* if it can represent any $\mathcal{L}$-definable incomplete database.

We are primarily interested in $\mathcal{RA}$-definable and $\mathcal{RA}^+$-definable incomplete databases. We start by applying a result of [14], which shows that an incomplete database $\mathcal{I}$ is $\mathcal{RA}$-definable if and only if $\mathcal{I}$ is representable by a c-table under CWA. Combining this with Theorem 15, we obtain that $m^{\not=}$-tables in $\mathcal{M}^c$ (as c-tables) capture exactly the incomplete databases that are expressed through an $\mathcal{RA}$ query over $\mathcal{Z}_U$:

▶ **Corollary 19.** *$m^{\not=}$-tables are $\mathcal{RA}$-complete; every $m^{\not=}$-table in $\mathcal{M}^c$ is $\mathcal{RA}$-definable.*

This result characterizes the expressivity of m-tables using $\mathcal{Z}_U$ as the starting point; it turns out that a small fragment of m-tables is enough to capture all of $\mathcal{RA}$ over $\mathcal{Z}_U$. To understand the true expressive power of m-tables, we need to use $\mathcal{N}$ as the starting point.

▶ **Proposition 20.** *For every m-table M, there exists a query $Q \in \mathcal{RA}$ s.t.* $\mathsf{Mod}_S(M) = Q(\mathcal{N})$.

The above proposition tells us that every m-table can be expressed as a relational query over $\mathcal{N}$. For the construction in the proof, which is presented in the Appendix, we need the difference operator to define the appropriate query Q.

▶ **Theorem 21.** *The set of incomplete databases expressed by free m-tables and by $Q(\mathcal{N})$, for $Q \in \mathcal{RA}^+$, are the same, that is:*
1. *For every free m-table M, there exists a query $Q \in \mathcal{RA}^+$ such that* $\mathsf{Mod}_S(M) = Q(\mathcal{N})$.
2. *For every $Q \in \mathcal{RA}^+$, there exists a free m-table M such that* $\mathsf{Mod}_S(M) = Q(\mathcal{N})$.

In other words, free m-tables capture exactly the incomplete databases that can be constructed by computing a positive relational algebra query over $\mathcal{N}$. As for general m-tables, we have shown that they describe a subset of the incomplete databases that can be computed through a relational algebra query over $\mathcal{N}$. It is not clear whether the converse holds, that is, if every incomplete database $\mathcal{I} = Q(\mathcal{N})$, where $Q \in \mathcal{RA}$, can be represented by m-tables. We leave this as part of future work.

6 Labeling Schemes

Interpreting the semantics of an m-table, and in particular the annotation $R(\hat{t})$ for each extended tuple $\hat{t}$ can be non-trivial. Additionally, given an m-table it is not immediately clear whether a tuple is certain or possible in the corresponding incomplete database. To address this issue, we describe a way to interpret an m-table, under set semantics, such that it tells the user which tuples are certain, and which tuples are possible (and under which conditions). We would like to emphasize that the labeling in this section is done only under set semantics.

6.1 Semantics of Labels

We first propose a labeling scheme for missing data. Each tuple $\hat{t}$, following the structure of m-tables, will be an extended tuple that takes values from $\hat{\mathbb{D}} = \mathbb{D} \cup \{m\}$ and every extended tuple will be associated with one of certain and possible labels. Formally, we define:

▶ **Definition 22** (Labeling)**.** An *m-labeled tuple* is a triple of the form $(\hat{t}, \lambda, \phi)$ such that:
- $\hat{t} : U \to \hat{\mathbb{D}}$ is an extended tuple,
- $\lambda \in \{\mathsf{certain}, \ \mathsf{possible}\}$,
- ϕ is a conjunction of expressions $(x \ \mathsf{op} \ y)$, where $x, y \in \mathbf{U} \cup \mathbb{D}$ and $\mathsf{op} \in \{=, <, >, \neq, \geq, \leq\}$.

An *m-labeling scheme* is a finite set of m-labeled tuples.

By viewing the attributes in ϕ as variables, we can view ϕ as a logical formula over $\mathbf{U}$. Given an assignment $v : \mathbf{U} \to \mathbb{D}$ that satisfies ϕ, we obtain an *instantiation* $v \circ \hat{t}$ of the tuple $\hat{t}$, where $v(\hat{t}[A]) = v(A)$ if $A \in \mathsf{m}(\hat{t})$, otherwise $v(\hat{t}[A]) = \hat{t}[A]$. We say that the set of all such instantiations is the *expansion* of $(\hat{t}, \phi)$ and we denote it as $\mathcal{D}(\hat{t}, \phi)$. Note that if ϕ is not satisfiable, then $\mathcal{D}(\hat{t}, \phi) = \emptyset$. Also, if $\mathsf{m}(\hat{t}) = \emptyset$ (so $\hat{t}$ has only constants) and $\phi = \mathsf{T}$ (i.e. the boolean formula ϕ is always true), we simply have $\mathcal{D}(\hat{t}, \phi) = \{\hat{t}\}$.

Let $\mathcal{I}$ be an incomplete database, and let $\mathsf{pos}(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} I$ denote the set of all possible tuples in $\mathcal{I}$. Recall that for $t \in \mathbb{D}^U$, we say that t is certain in $\mathcal{I}$ if for every $I \in \mathcal{I}$ we have $t \in I$; and that t is possible in $\mathcal{I}$ if there exists $I \in \mathcal{I}$ such that $t \in I$. We next generalize the definitions of certainty and possibility for extended tuples. We say that $(\hat{t}, \phi)$ is *certain*

in $\mathcal{I}$ if for every $I \in \mathcal{I}$ we have $\mathcal{D}(\hat{t}, \phi) \cap I \neq \emptyset$. We also say that $(\hat{t}, \phi)$ is *possible* in $\mathcal{I}$ if $\mathcal{D}(\hat{t}, \phi) \subseteq \mathsf{pos}(\mathcal{I})$. Observe that if $\hat{t}$ takes only values from $\mathbb{D}$ and $\phi = \mathsf{T}$, then the above definitions collapse to the standard definitions of possible and certain tuples.

We will focus on m-labeling schemes with the following property: for any $(\hat{t}, \mathsf{certain}, \phi)$, we have that $\phi = \mathsf{T}$. In other words, every extended tuple with label certain has no constraints on its possible values. In this case we simplify the notation of an m-labeled tuple to $(\hat{t}, \mathsf{certain})$ and its expansion to $\mathcal{D}(\hat{t})$: we call this a *simple m-labeling scheme*.

▶ **Definition 23** (Soundness)**.** Let $\mathcal{I}$ be an incomplete database, and $\mathcal{S}$ a simple m-labeling scheme. $\mathcal{S}$ is *c-sound* w.r.t. $\mathcal{I}$ if for every $(\hat{t}, \mathsf{certain}, \phi) \in \mathcal{S}$, $(\hat{t}, \phi)$ is certain in $\mathcal{I}$. $\mathcal{S}$ is *p-sound* w.r.t. $\mathcal{I}$ if for every $(\hat{t}, \mathsf{possible}, \phi) \in \mathcal{S}$, $(\hat{t}, \phi)$ is possible in $\mathcal{I}$.

If $\mathcal{S}$ is both c-sound and p-sound, we simply say that $\mathcal{S}$ is sound. A sound labeling scheme is a conservative under-approximation of an incomplete database.

▶ **Definition 24** (Completeness)**.** Let $\mathcal{I}$ be an incomplete database, and $\mathcal{S}$ a simple m-labeling scheme. $\mathcal{S}$ is *c-complete* w.r.t. $\mathcal{I}$ if for every $(\hat{t}, \mathsf{T})$ that is certain in $\mathcal{I}$, there exists $(\hat{t}', \mathsf{certain}) \in \mathcal{S}$ such that $\mathcal{D}(\hat{t}') \subseteq \mathcal{D}(\hat{t})$. $\mathcal{S}$ is *p-complete* w.r.t. $\mathcal{I}$ if for every $t \in \mathsf{pos}(\mathcal{I})$, there exists $(\hat{t}, \lambda, \phi) \in \mathcal{S}$ such that $t \in \mathcal{D}(\hat{t}, \phi)$.

Analogous to the definition of soundness, a complete labeling scheme is a conservative over-approximation of an incomplete database. A sound and complete labeling scheme captures exactly both the generalized possible and certain tuples.

▶ **Example 25.** Suppose we are given any incomplete database $\mathcal{I}$ for the relation $R(A, B, C)$. Consider the m-labeling scheme $\mathcal{S}$ that consists only of a single tuple: $((m, m, m), \mathsf{possible}, \mathsf{T})$. We first claim that this is a c-sound labeling. This trivially holds, since $\mathcal{S}$ contains no extended tuples with a certain tuple. We also claim that $\mathcal{S}$ is p-complete. Indeed, the expansion of $((m, m, m), \mathsf{T})$ is $\mathbb{D}^{(A,B,C)}$, and thus any tuple in $\mathsf{pos}(\mathcal{I})$ will also belong in $\mathcal{D}((m, m, m), \mathsf{T})$. This construction implies that we can always construct a trivial c-sound and p-complete labeling scheme for any incomplete database.

As we will see shortly, it is computationally hard to construct a sound and complete m-labeling for every incomplete database and $\mathcal{RA}^+$ query. However, we will show that a sound and complete simple m-labeling is possible for a particular case of incomplete databases that are defined through m-tables. We should emphasize here that an m-labeling scheme is not a representation system of $\mathcal{I}$, since we cannot reconstruct $\mathcal{I}$ from $\mathcal{S}$.

6.2 A Simple Label Inference Algorithm

We describe a simple procedure that, given an m-table M, constructs a c-sound and p-complete simple m-labeling scheme for $\mathsf{Mod}_S(M)$. We can use this procedure, together with the completeness of m-tables for $\mathcal{RA}^+$, to construct a c-sound and p-complete simple m-labeling for $Q(\mathcal{I})$ over an incomplete database $\mathcal{I}$ that is represented by an m-table.

The algorithm SimpleLabel is given in Algorithm 1. The intuition for the inference procedure is that, an extended tuple $\hat{t}$ is certain only in the case where there exists a monomial that has no constraints in $\mathcal{E}$, and further, the lower bounds on the cardinalities of α_i's (min_i's) are at least 1 (line 5).

If the tuple is labeled as possible, we compute the formula ϕ in lines 10-13 by taking the conjunction of the expressions in $\mathcal{E}$. Figure 3 shows the resulting labels after applying SimpleLabel on the final result of the running example of Section 4.

Algorithm 1 SimpleLabel.

$\mathcal{S} \leftarrow \emptyset$
for each $\hat{t} \in R$ s.t. $R(\hat{t}) \neq 0$ **do**
  Let $R(\hat{t}) = \sum_{k=1}^{n} \prod_{i_k} \alpha_{i_k}(A_{i_k}) \cdot \prod_{j=1}^{m_k} \theta_j$
  for $k = 1, \ldots, n$ **do**
    if $\exists i_k$ s.t. $min_{i_k} = 0$ or $m_k > 0$ **then**
      $\lambda \leftarrow$ possible
    else
      $\lambda \leftarrow$ certain
    end if
    $\phi \leftarrow \mathsf{T}$
    for all $\theta_j = [x \text{ op } y]$ **do**
      $\phi \leftarrow \phi \wedge (x \text{ op } y)$
    end for
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\hat{t}, \lambda, \phi)\}$
  end for
end for
return $\mathcal{S}$

▶ **Proposition 26.** *Given an m-table M,* SimpleLabel *computes in polynomial time (data complexity) a c-sound and p-complete simple m-labeling scheme $\mathcal{S}$ w.r.t. to* $\mathsf{Mod}_S(M)$.

From the previous proposition, we see that SimpleLabel provides conservative labeling, i.e., if a tuple is marked as certain, then it is definitely so, but all certain tuples may not be identified. Even though SimpleLabel does not produce sound and complete labelings, we can prove several interesting properties if we restrict the expressive power of m-tables.

▶ **Lemma 27.** *Let $M = (\{R\}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-table such that, for every $i = 1, \ldots, N$ we have $max_i = \infty$. Then,* SimpleLabel *produces a p-sound m-labeling scheme.*

This lemma tells us that whenever there is no upper bound on the size of the relations in Σ, we can efficiently construct a p-sound and p-complete labeling scheme, and thus capture exactly the possible tuples.

▶ **Lemma 28.** *Let $M = (\{R\}, \Sigma, \mathbf{min}, \mathbf{max})$ be an m-table such that, for every $i = 1, \ldots, N$ we have $min_i = 0$. Then,* SimpleLabel *produces a c-certain m-labeling scheme.*

Lemma 28 is the analogue of Lemma 27: if the size of each relation in Σ is lower bounded by 0, then we obtain a c-certain and c-sound m-labeling and thus compute exactly the certain answers. Combining the two lemmas:

▶ **Theorem 29.** *If $M = (\{R\}, \Sigma)$ is a free m-table, then* SimpleLabel *produces a sound and complete m-labeling scheme w.r.t.* $\mathsf{Mod}_S(M)$.

6.3 Certainty and Possibility in m-tables

▶ **Proposition 30.** *Let M be a free $m^{\not=}$-table. Then, the following tasks can be completed in polynomial time data complexity: (1) tuple certainty, (2) tuple possibility, and (3) tuple q-possibility and q-certainty for $q \in \mathcal{RA}^+$.*

Regarding arbitrary $m^{/=}$-tables, in the Appendix, we show why SimpleLabel fails to produce a p-sound labeling scheme when the upper cardinality constraints are different than ∞, and why it fails to obtain a c-certain labeling when the lower cardinality constraints are different than 0. We conclude with two results on the complexity of tuple certainty and possibility for general $m^{/=}$ tables.

▶ **Proposition 31.** *Tuple possibility in $m^{/=}$-tables is NP-complete (data complexity).*

▶ **Proposition 32.** *Tuple certainty in $m^{/=}$-tables is coNP-complete (data complexity).*

7 Related Work

Incomplete databases have been extensively studied in various contexts; we refer the reader to [1, 25] for a survey and to [22] for a broad perspective on this area.

Numerous models for uncertain information are discussed and compared in [14]. *Conditional tables* (c-tables) [16, 17] are considered one of the most expressive representation system for representing incomplete databases. The $\mathcal{R}^A_{prop}$ model [24] has also been shown to be *closed*, *complete* and as expressive as c-tables. We have provided a detailed comparison of m-tables with c-tables in Section 3 and we have shown that m-tables are strictly more expressive than c-tables (and thus than the $\mathcal{R}^A_{prop}$ as well). In [2, 3], the focus is on providing complexity and decidability results for querying over incomplete databases and we have utilized results from [2] to show complexity results for obtaining certain answers with m-tables.

The use of many-valued logic to handle missing information has been proposed in [6, 9, 26]; this is complementary to our work, and adding multi-valued logic into m-tables should be interesting future work. Our definitions of certain and possible answers are similar to the certain answers defined in [6, 23]; however, the notion of certainty and possibility in our work is defined for extended tuples that represent a set of tuples and not just for single tuples. In this context, our work is also related to the partial results work of [19], where the idea is to execute a given query on an incomplete database in the usual way, and then provide insights into the possible anomalies of the result tuples by labeling the tuples and attribute values with potential errors. They do not focus on developing a formal framework to provide all possible results and they lack a systematic approach to obtain labels for 'partial' results.

Querying over incomplete databases under the open world assumption has been explored in [21] and [8]. In both these pieces of work, the focus is on decidability results on whether *complete* queries can be obtained over possibly incomplete data, with constraints on the missing data. However, our focus is on obtaining a representation system that provides all possible results, while trying to label certain answers.

An extended tuple with all 'm' values is similar to the idea proposed in [10], where they introduce a tuple with all attribute values as 'open' to represent an unknown number of missing tuples. However, their work does not extend beyond the 'open' value; their focus is not on obtaining a representation system or to identify certain answers.

Queries over data integrated (DI) sources have a similar scenario where, frequently, some subset of the information will be uncertain or not available [7, 11, 20, 21]. Solution approaches proposed in this context use schema information of the sources to improve the answers. Our work can be seen as complementary to the work done in this area and our work is applicable in DI scenarios as well.

8 Conclusion

In this paper, we proposed a new representation system called m-tables, which can represent various forms of missing data and generalizes many existing representation systems. We showed that m-tables form a *closed* and *strong* representation system for both set and bag semantics, and are strictly more expressive than c-tables. Further, we propose a simple labeling algorithm that labels tuples as certain or possible by interpreting the annotations of the m-table. One immediate line for future work is to extend m-table semantics to aggregate and group by operations. Another interesting direction is to use the annotations and labeling scheme to "repair" a result when missing data becomes available in the future.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

2 Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. On the representation and querying of sets of possible worlds. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, SIGMOD'87, pages 34–48, New York, NY, USA, 1987. ACM. doi:10.1145/38713.38724.

3 Antoine Amarilli and Michael Benedikt. Finite open-world query answering with number restrictions (extended version). *CoRR*, abs/1505.04216, 2015. URL: http://arxiv.org/abs/1505.04216.

4 Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'11, pages 153–164, New York, NY, USA, 2011. ACM. doi:10.1145/1989284.1989302.

5 James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends databases*, 1(4):379–474, April 2009. doi:10.1561/1900000006.

6 Marco Console, Paolo Guagliardo, and Leonid Libkin. Approximations and refinements of certain answers via many-valued logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 349–358, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12813.

7 AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

8 Wenfei Fan and Floris Geerts. Capturing missing tuples and missing values. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 169–178, 2010. doi:10.1145/1807085.1807109.

9 G. H. Gessert. Four valued logic for relational database systems. *SIGMOD Rec.*, 19(1):29–35, March 1990. doi:10.1145/382274.382401.

10 Georg Gottlob and Roberto Zicari. Closed world databases opened through null values. In *Proceedings of the 14th International Conference on Very Large Data Bases*, VLDB'88, pages 50–61, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc. URL: http://dl.acm.org/citation.cfm?id=645915.671794.

11 G. Grahne. Information integration and incomplete information. In *IEEE Data Eng. Bull.*, pages 46–52, 2002.

12 Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB'07, pages 675–686. VLDB Endowment, 2007. URL: http://dl.acm.org/citation.cfm?id=1325851.1325929.

13 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'07, pages 31–40, New York, NY, USA, 2007. ACM. doi:10.1145/1265530.1265535.

14 Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In *Proceedings of the 2006 International Conference on Current Trends in Database Technology*, ICDT'06, pages 278–296, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11896548_24.

15 Paolo Guagliardo and Leonid Libkin. Making sql queries correct on incomplete databases: A feasibility study. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'16, pages 211–223, New York, NY, USA, 2016. ACM. doi:10.1145/2902251.2902297.

16 Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984. doi:10.1145/1634.1886.

17 Tomasz Imielinski and Witold Lipski, Jr. On representing incomplete information in a relational data base. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB'81, pages 388–397. VLDB Endowment, 1981. URL: http://dl.acm.org/citation.cfm?id=1286831.1286869.

18 Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD'10, pages 951–962, New York, NY, USA, 2010. ACM. doi:10.1145/1807167.1807269.

19 Willis Lang, Rimma V. Nehme, Eric Robinson, and Jeffrey F. Naughton. Partial results in database systems. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD'14, pages 1275–1286, New York, NY, USA, 2014. ACM. doi:10.1145/2588555.2612176.

20 Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'02, pages 233–246, New York, NY, USA, 2002. ACM. doi:10.1145/543613.543644.

21 Alon Y. Levy. Obtaining complete answers from incomplete databases. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB'96, pages 402–412, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. URL: http://dl.acm.org/citation.cfm?id=645922.673332.

22 Leonid Libkin. Incomplete data: What went wrong, and how to fix it. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS'14, pages 1–13, New York, NY, USA, 2014. ACM. doi:10.1145/2594538.2594561.

23 Leonid Libkin. Sql's three-valued logic and certain answers. *ACM Trans. Database Syst.*, 41(1):1:1–1:28, March 2016. doi:10.1145/2877206.

24 Anish Das Sarma, Omar Benjelloun, Alon Halevy, and Jennifer Widom. Working models for uncertain data. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE'06, pages 7–, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/ICDE.2006.174.

25 Ron van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publishers, Norwell, MA, USA, 1998. doi:10.1007/978-1-4615-5643-5_10.

26 Kwok-bun Yue. A more general model for handling missing information in relational databases using a 3-valued logic. *SIGMOD Rec.*, 20(3):43–49, September 1991. doi:10.1145/126482.126487.