A Logical Account of Uncertain Databases Based on Linear Logic*

Sungwoo Park and Seung-won Hwang
Pohang University of Science and Technology
Republic of Korea, 790-784
{gla,swhwang}@postech.ac.kr

ABSTRACT

A formal semantics of uncertain databases typically takes an algebraic approach by mapping an uncertain database to a set of relational databases, or possible worlds. We present a new semantics for uncertain databases which takes a logical approach by translating uncertain databases into logical theories. A characteristic feature of our semantics is that it uses linear logic, instead of propositional logic, as its logical foundation. Linear logic lends itself well to a logical interpretation of uncertain information because unlike propositional logic, it treats logical formulae not as persistent facts but as consumable resources.

We motivate our development by arguing that propositional logic is inadequate as a logical foundation for uncertain databases. As the main result, we show that our semantics is faithful to the operational account of uncertain databases in the algebraic approach.

1. INTRODUCTION

In efforts to overcome the limitation of relational databases in dealing with uncertain information, a number of approaches to formulating uncertain databases (*i.e.*, "how do we build an uncertain database?") have been proposed. One approach is to allow tuples to contain attribute variables whose contents are left unspecified (like null values) or subject to additional logical constraints, as in *Codd tables* [7], *v-tables* and *c-tables* [11], *g-tables* [1], and *Horn tables* [10]. An alternative approach is to use extended tuples whose syntax directly accommodates disjunctive information ('either A or B'), maybe information ('maybe A and maybe not'), or their combinations, as in *partial values* and *maybe tuples* [8], *I-tables* [17], *M-tables* [18], *or-sets* [12, 15], and *x-relations* [2].

*This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST)/Korea Science and Engineering Foundation(KOSEF), grant number 11-2008-007-03001-0.

ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.

An orthogonal problem to formulating uncertain databases is to study the semantics of uncertain databases (*i.e.*, "what is the meaning of a given uncertain database?"). Prior work on the semantics of relational databases (without uncertainty) is categorized into the algebraic approach advocated by Imieliński and Lipski [11] and the logical approach proposed by Reiter [20]. In our context of uncertain databases, the algebraic approach maps an uncertain database to a unique set of relational databases, or possible worlds, whereas the logical approach specifies how to translate an uncertain database to logical theories. The algebraic approach is useful when analyzing the efficiency of a specific implementation of database operations, and the logical approach is useful when proving the correctness of various database operations.

This paper develops a semantics for uncertain databases based on the logical approach. We restrict ourselves to uncertain databases that use two kinds of extended tuples, *disjunctive tuples* and *maybe tuples*, but no attribute variables. (Logical accounts of uncertain databases using attribute variables have been studied previously, for example, in [20, 26, 21].) A disjunctive tuple $T_1 \odot \cdots \odot T_n$ denotes an exclusive disjunction between ordinary tuples T_1 through T_n. Given that Z is a disjunctive tuple, a maybe tuple $Z?$ states that Z may be the case, without ruling out the possibility that Z is not the case. The use of disjunctive tuples and maybe tuples is adopted in recent work on uncertain databases [2, 23].

The main difficulty in developing such a semantics is to translate a maybe tuple $Z?$. Translating $Z?$ to a logical disjunction $Z \vee \neg Z$ (where $\neg$ stands for logical negation) is meaningless in classical propositional logic because $A \vee \neg A$ is true for any logical formula A. Liu and Sunderraman [17, 18] and Zimanyi [27] give a semantics based on propositional logic, but it requires Z in $Z?$ to be an ordinary tuple and is not compositional in the sense that the translation of an uncertain database combining U_1 and U_2 is not a direct sum of the individual translations of U_1 and U_2.

In order to achieve a semantics that is both general (permitting Z in $Z?$ to be a disjunctive tuple) and compositional (translating individual extended tuples independently of each other), we take a radical departure from the traditional logical approach by choosing as the logical foundation not propositional logic but *linear logic* [9]. Unlike propositional logic which interprets logical formulae as persistent facts, linear logic treats logical formulae as resources which can be consumed to produce new resources, or equivalently, as descriptions of transient states. As such, linear logic offers considerably simpler solutions than propositional logic

to those problems that require resource interpretations or need to model state transitions. For example, the *situation calculus* [22], the standard solution to the frame problem in artificial intelligence, is considerably more complex than those solutions based on linear logic [3, 19]. Based on linear logic, our semantics also interprets extended tuples as descriptions of consumable resources. As the main result, we show that the semantics is faithful to the operational account of uncertain databases in the algebraic approach.

We are not the first to use linear logic in database research. Lee and Tsang [14] present deductive databases based on linear logic. Bidoit *et al.* [4] use linear logic as a formal system to reason about database updates. Considering the fact that linear logic provides an elegant solution to the frame problem, however, we find it surprising that linear logic is little in use for database research where the frame problem is a recurring theme (see [5] for an example). We believe that database research has plenty of opportunities to draw on the vast literature on linear logic.

The rest of this paper is organized as follows. Section 2 clarifies the goal of our work and gives an informal account of why propositional logic is not a good choice as the logical foundation for uncertain databases. Section 3 gives an introduction to linear logic. Section 4 develops a semantics based on linear logic and proves the main result. Section 5 discusses related work and Section 6 concludes.

2. PRELIMINARIES

We use the following predicates in examples where x, y, and z are term variables:

- $Eat(x, y)$ means that person x wants to eat meal y.
- $Where(y, z)$ means that meal y is served at restaurant z.
- $Go(x, z)$ means that person x visits restaurant z.

2.1 Syntax for uncertain databases

We formulate an uncertain database U as a set of extended tuples, or *x-tuples*, $X_1, \cdots, X_n$. A relational database R is a special case of an uncertain database which is a set of ordinary tuples $T_1, \cdots, T_n$.

uncertain database	U	::=	$\cdot \mid X, U$
relational database	R	::=	$\cdot \mid T, R$

Note that both uncertain databases and relational databases can be empty.

An x-tuple is either a disjunctive tuple Z or a maybe tuple $Z?$. A disjunctive tuple is either an ordinary tuple T or an exclusive disjunction $Z_1 \odot Z_2$ between two disjunctive tuples Z_1 and Z_2. A tuple has the form $P(t_1, \cdots, t_n)$ which applies a predicate P to a sequence of terms $t_1, \cdots, t_n$. We abbreviate $P(t_1, \cdots, t_n)$ as $P(\vec{t})$.

tuple	T	::=	$P(\vec{t})$
disjunctive tuple	Z	::=	$T \mid Z \odot Z$
x-tuple	X	::=	$Z \mid Z?$

An x-tuple is well-formed if all tuples in it share the same predicate. For example, $Eat(\mathsf{Tom}, \mathsf{Pizza}) \odot Eat(\mathsf{Tom}, \mathsf{Soup})$ is well-formed whereas $Eat(\mathsf{Tom}, \mathsf{Pizza}) \odot Where(\mathsf{Pizza}, \mathsf{R}_1)$ is not. An uncertain database is well-formed if all x-tuples in it are well-formed and share the same predicate. We consider only well-formed uncertain databases.

$$\frac{}{P(\vec{t}) \hookrightarrow P(\vec{t})}\ \mathsf{I}P \qquad \frac{Z \hookrightarrow R}{Z \odot Z' \hookrightarrow R}\ \mathsf{I}\odot_L \qquad \frac{Z' \hookrightarrow R}{Z \odot Z' \hookrightarrow R}\ \mathsf{I}\odot_R$$

$$\frac{Z \hookrightarrow R}{Z? \hookrightarrow R}\ \mathsf{I}?_1 \qquad \frac{}{Z? \hookrightarrow .}\ \mathsf{I}?_2$$

$$\frac{}{. \hookrightarrow .}\ \mathsf{I}U_1 \qquad \frac{X \hookrightarrow R \quad U \hookrightarrow R'}{X, U \hookrightarrow R, R'}\ \mathsf{I}U_2$$

Figure 1: Rules for $U \hookrightarrow R$

Here is an example of an uncertain database using the predicate $Eat(x, y)$:

$$\begin{array}{rcl} U & = & Eat(\mathsf{Tom}, \mathsf{Pizza}) \odot Eat(\mathsf{Tom}, \mathsf{Soup}), \\ & & Eat(\mathsf{John}, \mathsf{Salad})?, Eat(\mathsf{Jane}, \mathsf{Steak}) \end{array}$$

U says: 1) Tom wants to eat either Pizza or Soup; 2) John may want to eat Salad; 3) Jane wants to eat Steak.

2.2 Instantiating uncertain databases

We write $U \hookrightarrow R$ to mean that relational database R is an instance of uncertain database U. Figure 1 shows the rules for deducing $U \hookrightarrow R$ which follow the style of the big-step operational semantics for programming languages [13] (where we regard U as a program, R as a value, and $\hookrightarrow$ as an evaluation). For example, the rule $\mathsf{I}\odot_L$ says that $Z \odot Z' \hookrightarrow R$ holds whenever $Z \hookrightarrow R$ holds, and the rule $\mathsf{I}\odot_R$ says that $Z \odot Z' \hookrightarrow R$ holds whenever $Z' \hookrightarrow R$ holds. Note that there can be multiple relational databases to which the same uncertain database instantiates. We may think of the system in Figure 1 as the operational account of uncertain databases in the algebraic approach. The rules $\mathsf{I}\odot_L$ and $\mathsf{I}\odot_R$ suggest that $\odot$ is associative and commutative.

As an example, U in Section 2.1 instantiates to one of the following relational databases:

$$\begin{array}{rcl} R_1 & = & Eat(\mathsf{Tom}, \mathsf{Pizza}), Eat(\mathsf{John}, \mathsf{Salad}), \\ & & Eat(\mathsf{Jane}, \mathsf{Steak}) \\ R_2 & = & Eat(\mathsf{Tom}, \mathsf{Soup}), Eat(\mathsf{John}, \mathsf{Salad}), \\ & & Eat(\mathsf{Jane}, \mathsf{Steak}) \\ R_3 & = & Eat(\mathsf{Tom}, \mathsf{Pizza}), Eat(\mathsf{Jane}, \mathsf{Steak}) \\ R_4 & = & Eat(\mathsf{Tom}, \mathsf{Soup}), Eat(\mathsf{Jane}, \mathsf{Steak}) \end{array}$$

2.3 Goal

The goal of our work is to specify a semantics for uncertain databases with a semantic function $\mathcal{F}$ from uncertain databases to logical formulae. The semantic function $\mathcal{F}$ needs to satisfy the following requirement:

- $\mathcal{F}(U)$ logically implies $\mathcal{F}(R)$ if and only if $U \hookrightarrow R$.

It says that $\mathcal{F}$ is faithful to the operational account of uncertain databases given in Figure 1. Then the problem of testing $U \hookrightarrow R$ reduces to the problem of checking the relation between logical formulae $\mathcal{F}(U)$ and $\mathcal{F}(R)$.

It is important that we do not stipulate a specific definition of 'logical implication' because it is to be determined by the logical foundation for the semantics and the definition of $\mathcal{F}$. For example, a semantics based on propositional logic may deduce that A 'logically implies' B when $A \supset B$ is true (where $\supset$ stands for propositional implication), but depending on the definition of $\mathcal{F}$, it may also be reasonable

to deduce the same judgment when the converse $B \supset A$ is true.

2.4 Inadequacy of propositional logic for uncertain databases

This subsection gives an informal argument that propositional logic is not a good choice as the logical foundation for uncertain databases. We set out to define the semantic function $\mathcal{F}$ with propositional logic as its logical foundation, and illustrate that it is far from obvious to obtain a definition of $\mathcal{F}$ satisfying the requirement. In order to obtain a compositional definition of $\mathcal{F}$, we assume an inductive definition

$$\mathcal{F}(X, U) = \mathcal{F}(X) \wedge \mathcal{F}(U)$$

where $\wedge$ stands for logical conjunction.

Consider a disjunctive tuple $T_1 \odot T_2$. Since $T_1 \odot T_2$ denotes an exclusive disjunction between T_1 and T_2, it is reasonable to define $\mathcal{F}(T_1 \odot T_2)$ as

$$(T_1 \wedge \neg T_2) \vee (\neg T_1 \wedge T_2).$$

This definition of $\mathcal{F}(T_1 \odot T_2)$, however, is unsatisfactory because of its use of logical negation. For example, $T \odot T$ and T are operationally equivalent in that both instantiate only to T, but $\mathcal{F}(T \odot T)$ denotes logical falsehood which cannot be logically equivalent to $\mathcal{F}(T)$. As another example, consider the following databases:

$$\begin{array}{rcl} U & = & \mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza}) \odot \mathit{Eat}(\mathsf{Tom}, \mathsf{Soup}) \\ R & = & \mathit{Where}(\mathsf{Pizza}, \mathsf{R}_1),\ \mathit{Where}(\mathsf{Soup}, \mathsf{R}_2) \\ U' & = & \mathit{Go}(\mathsf{Tom}, \mathsf{R}_1) \odot \mathit{Go}(\mathsf{Tom}, \mathsf{R}_2) \end{array}$$

With the assumption that $\mathit{Eat}(x, y)$ and $\mathit{Where}(y, z)$ produce $\mathit{Go}(x, z)$, a join operation between U and R produces U'. Similarly, with the assumption that $\mathit{Eat}(x, y) \wedge \mathit{Where}(y, z)$ proves $\mathit{Go}(x, z)$, we would expect $\mathcal{F}(U)$ and $\mathcal{F}(R)$ to prove $\mathcal{F}(U')$. This is not the case, however, because of logical negation: $\neg\mathit{Eat}(x, y) \wedge \mathit{Where}(y, z)$ does not prove $\neg\mathit{Go}(x, z)$.

A more serious problem arises when translating maybe tuples. It is reasonable to define $\mathcal{F}(Z?)$ as $Z \vee \neg Z$ because the intuition behind $Z?$ is that Z may or may not be true. Unfortunately $A \vee \neg A$ is a tautology and conveys no information in classical propositional logic. That is, $A \vee \neg A$ is true for any logical formula A by the law of excluded middle, and defining $\mathcal{F}(Z?)$ as $Z \vee \neg Z$ makes it impossible to distinguish between an uncertain database U and its extension $U, Z?$. That is, we have

$$\mathcal{F}(U) \ \equiv\ \mathcal{F}(U) \wedge (Z \vee \neg Z) \ =\ \mathcal{F}(U, Z?)$$

where logical equivalence $A \equiv B$ means both $A \supset B$ and $B \supset A$.

A quick fix is to base the semantics on *intuitionistic propositional logic* [24] in which the law of excluded middle does not hold and $A \vee \neg A$ actually conveys some information. In our context, for example,

$$\mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza}) \vee \neg\mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza})$$

declares that $\mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza})$ is one of the tuples under consideration, thereby admitting Tom and Pizza as potential arguments to the predicate Eat.

Even in intuitionistic propositional logic, however, this definition of $\mathcal{F}(Z?)$ is unsatisfactory. As an example, consider the following databases where a join operation between U and R produces U':

$$\begin{array}{rcl} U & = & \mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza})? \\ R & = & \mathit{Where}(\mathsf{Pizza}, \mathsf{R}_1) \\ U' & = & \mathit{Go}(\mathsf{Tom}, \mathsf{R}_1)? \end{array}$$

With the assumption that $\mathit{Eat}(x, y) \wedge \mathit{Where}(y, z)$ proves $\mathit{Go}(x, z)$, we would expect $\mathcal{F}(U)$ and $\mathcal{F}(R)$ to prove $\mathcal{F}(U')$ because a join operation between U and R produces U'. Again this is not the case because of logical negation: $\neg\mathit{Eat}(x, y) \wedge \mathit{Where}(y, z)$ does not prove $\neg\mathit{Go}(x, z)$.

These observations lead to the conclusion that propositional logic, whether classical or intuitionistic, may not be the right choice as the logical foundation for uncertain databases. In contrast, our semantics uses linear logic as its logical foundation and does not suffer from those problems due to logical negation. In addition, linear logic provides a more elegant semantics for uncertain databases than propositional logic, just like linear logic provides a solution to the frame problem in artificial intelligence that is considerably simpler than the situation calculus based on propositional logic.

$$\frac{}{A \Rightarrow A}\ \mathit{Init} \qquad \frac{\Delta, A, B \Rightarrow C}{\Delta, A \otimes B \Rightarrow C}\ \otimes L \qquad \frac{\Delta \Rightarrow A \quad \Delta' \Rightarrow B}{\Delta, \Delta' \Rightarrow A \otimes B}\ \otimes R$$

$$\frac{\Delta, A \Rightarrow C}{\Delta, A \,\&\, B \Rightarrow C}\ \&L_1 \qquad \frac{\Delta, B \Rightarrow C}{\Delta, A \,\&\, B \Rightarrow C}\ \&L_2$$

$$\frac{\Delta \Rightarrow A \quad \Delta \Rightarrow B}{\Delta \Rightarrow A \,\&\, B}\ \&R$$

$$\frac{\Delta \Rightarrow A \quad \Delta', B \Rightarrow C}{\Delta, \Delta', A \multimap B \Rightarrow C}\ \multimap L \qquad \frac{\Delta, A \Rightarrow B}{\Delta \Rightarrow A \multimap B}\ \multimap R$$

$$\frac{\Delta \Rightarrow C}{\Delta, \mathbf{1} \Rightarrow C}\ \mathbf{1}L \qquad \frac{}{\cdot \Rightarrow \mathbf{1}}\ \mathbf{1}R \qquad \frac{}{\Delta \Rightarrow \top}\ \top R$$

Figure 2: Inference rules in the sequent calculus for linear logic

3. LINEAR LOGIC

This section presents a decidable fragment of linear logic that our semantics uses. Instead of a model-theoretic approach, we take a proof-theoretic approach which relies on inference rules to deduce new logical theories from existing logical theories. As we will see, the proof-theoretic approach is a natural choice because inference rules easily express the relationship between uncertain databases and their corresponding relational databases (which are equivalent to possible worlds in the algebraic approach).

3.1 Linear logic with linear hypotheses

In linear logic, every formula denotes a resource that can be consumed to produce new resources. It uses the following inductive definition of formulae; we use metavariables A, B, C for formulae:

$$\text{formula} \quad A \quad ::= \quad P(\vec{t}) \mid A \otimes A \mid A \,\&\, A \mid A \multimap A \mid \mathbf{1} \mid \top$$

For predicates, we use the same notation $P(\vec{t})$ that we use for tuples in uncertain databases; hence tuples in uncertain databases can be thought of as predicates in linear logic. A

simultaneous conjunction $A \otimes B$ denotes a pair of resources A and B; hence consuming $A \otimes B$ produces both A and B. An *alternative conjunction* $A \,\&\, B$ denotes a resource that produces one of A and B as requested; hence we can choose to produce from $A \,\&\, B$ either A or B, but not both. A *linear implication* $A \multimap B$ is a resource that produces B when combined with A; hence consuming both A and $A \multimap B$ produces B. The *unit* $\mathbf{1}$ denotes no resource, or "nothing." The *top* $\top$ denotes an unspecified resource, or "something."

We formulate linear logic as a *sequent calculus* which is equivalent to other formulations such as the natural deduction system, but simplifies proofs of metatheorems in Section 4 (because it generates only *normal proofs*). The basic judgment in the sequent calculus is a *linear sequent* $\Delta \Rightarrow A$ where a *linear context* Δ is a set of formulae:

$$\text{linear context} \quad \Delta \quad ::= \quad \cdot \mid A, \Delta$$

$\Delta \Rightarrow A$ means that we have to produce a new resource A by consuming every existing resource in Δ exactly once, *i.e.*, *linearly.* (Hence we use such terms as "linear" sequents and "linear" contexts.) Note that the definition of $\Delta \Rightarrow A$ implies that we have to consume *all* resources in Δ. That is, $\Delta \Rightarrow A$ does not hold if some resources in Δ remain unconsumed. We say that two formulae A and B are logically equivalent, written as $A \equiv B$, if both $A \Rightarrow B$ and $B \Rightarrow A$ hold.

Figure 2 shows inferences rules for linear sequents which should be read *not top-down but bottom-up.* The system explains the meaning of logical connectives with left rules and right rules. A left rule specifies how to exploit an existing formula involving a particular connective. For example, the left rule for $\otimes$, namely the rule $\otimes L$, decides to exploit an existing formula $A \otimes B$ and splits it into A and B; hence the problem of proving $\Delta, A \otimes B \Rightarrow C$ reduces to the problem of proving $\Delta, A, B \Rightarrow C$. A right rule specifies how to produce a new formula involving a particular connective. For example, the right rule for $\otimes$, namely the rule $\otimes R$, decides to produce a new formula $A \otimes B$ by attempting to produce A from Δ and B from Δ'; hence the problem of proving $\Delta, \Delta' \Rightarrow A \otimes B$ reduces to the problem of proving $\Delta \Rightarrow A$ and $\Delta' \Rightarrow B$. The rule *Init* has no premise because consuming A immediately produces A; it has an implication that a linear sequent must consume all resources in its linear context. Note that there is no left rule for $\top$.

Now it is easy to show that $\mathbf{1}$ is the identity for $\otimes$, *i.e.*, $\mathbf{1} \otimes A \equiv A \otimes \mathbf{1} \equiv A$, and that $\top$ is the identity for $\&$, *i.e.*, $\top \,\&\, A \equiv A \,\&\, \top \equiv A$. Both $\&$ and $\otimes$ are associative and commutative, and $\multimap$ is right associative.

The system in Figure 2 is decidable [16]: there is a decision procedure for testing whether $\Delta \Rightarrow A$ holds or not.

3.2 Linear logic with unrestricted hypotheses

Linear logic also provides the "of course" modality ! which allows a resource to be consumed in an unrestricted way. A formula $!A$ denotes an infinite supply of resource A which we may use any number of times, including zero times:

$$\text{formula} \quad A \quad ::= \quad \cdots \mid\, !A$$

In order to incorporate the modality !, we follow the style in [6] and use an extended linear sequent $\Gamma; \Delta \Rightarrow A$ where an *unrestricted context* Γ is a set of formulae:

$$\text{unrestricted context} \quad \Gamma \quad ::= \quad \cdot \mid A, \Gamma$$

$\Gamma; \Delta \Rightarrow A$ means that we have to produce a new resource A by consuming every resource in Δ exactly once but any resource in Γ as many times as necessary, *i.e.*, in an unrestricted way. A linear sequent $\Delta \Rightarrow A$ is now an abbreviation of $\cdot; \Delta \Rightarrow A$.

$$\frac{\Gamma, A; \Delta, A \Rightarrow C}{\Gamma, A; \Delta \Rightarrow C}\ \mathit{Copy} \qquad \frac{\Gamma, A; \Delta \Rightarrow C}{\Gamma; \Delta, !A \Rightarrow C}\ !L \qquad \frac{\Gamma; \cdot \Rightarrow A}{\Gamma; \cdot \Rightarrow !A}\ !R$$

Figure 3: Inference rules for the modality !

The rules for extended linear sequents are derived from the previous rules in Figure 2 by rewriting every linear sequent $\Delta \Rightarrow A$ as $\Gamma; \Delta \Rightarrow A$. In addition, we need three new rules in Figure 3. The rule *Copy* enables us to use resources in unrestricted contexts. The left rule $!L$ decides to exploit an existing formula $!A$ by incorporating A into the unrestricted context. The premise of the right rule $!R$ proves that A is a resource that can be produced any number of times because it does not require additional resources except those in Γ.

Although linear logic with the modality ! is undecidable in general [16], restricting A in $!A$ to atomic formulae recovers its decidability. Intuitively we can always get rid of the modality ! from an extended linear sequent by repeatedly applying the rules $!L$ and $!R$, and for each atomic formula A, we need to apply the rule *Copy* no more times than it occurs in the original sequent. Our semantics for uncertain databases restricts A in $!A$ to atomic formulae and thus uses a decidable fragment of linear logic.

4. SEMANTICS BASED ON LINEAR LOGIC

This section presents our semantics for uncertain databases. We define the semantic function $\mathcal{F}$ in such a way that the following invariant holds:

- $\mathcal{F}(U) \Rightarrow \mathcal{F}(R)$ if and only if $U \hookrightarrow R$.

Since $\mathcal{F}(U) \Rightarrow \mathcal{F}(R)$ proves $\mathcal{F}(U) \multimap \mathcal{F}(R)$ and vice versa, 'logical implication' in our semantics is in fact 'linear implication.' Then $\mathcal{F}$ satisfies the requirement given in Section 2.3.

4.1 Definition of the semantic function $\mathcal{F}$

Informally our semantics interprets x-tuples as descriptions of resources in the following way.

- Consuming a disjunctive tuple $T_1 \odot \cdots \odot T_n$ produces one of T_1, $\cdots$, T_n. Note that once a new tuple is produced, the original disjunctive tuple disappears.
- Consuming a maybe tuple $Z?$ produces either Z or nothing. Similarly to disjunctive tuples, the original maybe tuple disappears when either Z or nothing is produced.
- Separate x-tuples represent independent resources. That is, consuming an x-tuple does not affect other x-tuples.
- An ordinary tuple T, which is a special case of an x-tuple, represents a resource that permits an unrestricted use. Hence T does *not* disappear even after it is consumed. Intuitively T contains no element of uncertainty and thus denotes a persistent fact.

This resource interpretation of x-tuples exhibits a pleasant correspondence with logical connectives in linear logic.

- Consuming an x-tuple X to produce another x-tuple X' is encoded as a linear implication $X \multimap X'$.
- A disjunctive tuple $T_1 \odot \cdots \odot T_n$ is encoded as an alternative conjunction $T_1 \,\&\, \cdots \,\&\, T_n$.
- A maybe tuple $Z?$ is encoded as an alternative conjunction $Z \,\&\, \mathbf{1}$ where $\mathbf{1}$ means 'nothing.'
- A set of x-tuples X_1 through X_n is encoded as a simultaneous conjunction $X_1 \otimes \cdots \otimes X_n$.
- An ordinary tuple T is encoded as $!T$ where T is assumed to have the form of a predicate.

Then an uncertain database consisting of x-tuples $X_1, \cdots, X_n$ corresponds to a simultaneous conjunction $X_1 \otimes \cdots \otimes X_n$, and instantiating an uncertain database U to a relational database R corresponds to a linear implication $U \multimap R$.

Formally the semantic function $\mathcal{F}$ uses $\otimes$, $\&$, $\mathbf{1}$, and $!$ in linear logic:

$$\begin{array}{rcl} \mathcal{F}(P(\vec{t})) & = & !P(\vec{t}) \\ \mathcal{F}(Z \odot Z') & = & \mathcal{F}(Z) \,\&\, \mathcal{F}(Z') \\ \mathcal{F}(Z?) & = & \mathcal{F}(Z) \,\&\, \mathbf{1} \\ \mathcal{F}(\cdot) & = & \mathbf{1} \\ \mathcal{F}(X, U) & = & \mathcal{F}(X) \otimes \mathcal{F}(U) \end{array}$$

The definition of $\mathcal{F}$ is based on the following observations:

- A tuple $P(\vec{t})$ itself contains no element of uncertainty and thus denotes a persistent fact. For example, we may not use $\mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza}) \odot \mathit{Eat}(\mathsf{Tom}, \mathsf{Soup})$ twice to obtain both $\mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza})$ and $\mathit{Eat}(\mathsf{Tom}, \mathsf{Soup})$, but once we decide to choose $\mathit{Eat}(\mathsf{Tom}, \mathsf{Pizza})$, it becomes a persistent fact which may be used any number of times afterwards. Hence we translate $P(\vec{t})$ to $!P(\vec{t})$.
- A disjunctive tuple $Z \odot Z'$ allows us to choose either Z or Z'. Hence we translate it to $\mathcal{F}(Z) \,\&\, \mathcal{F}(Z')$.
- A maybe tuple $Z?$ states that Z may or may not be the case. If Z is not the case, we choose to ignore it, obtaining no information, instead of refuting it with logical negation. Hence we can choose to produce Z or nothing from $Z?$, and translate $Z?$ to $\mathcal{F}(Z) \,\&\, \mathbf{1}$.
- An empty uncertain database gives no information. Hence we translate it to $\mathbf{1}$.
- Given an uncertain database consisting of X and U, we may use X and U independently of each other. Hence we translate X, U to $\mathcal{F}(X) \otimes \mathcal{F}(U)$, which means that $\mathcal{F}$ is compositional.

The definition of $\mathcal{F}$ does not use the top $\top$, but we will use it in Section 4.3 where we discuss membership problems in uncertain databases.

4.2 Soundness and completeness of $\mathcal{F}$

We now show that the invariant holds on the semantic function $\mathcal{F}$. We need to prove the soundness and completeness of $\mathcal{F}$ in the following sense:

THEOREM 4.1 (SOUNDNESS OF $\mathcal{F}$).
If $U \hookrightarrow R$, then $\mathcal{F}(U) \Rightarrow \mathcal{F}(R)$.

THEOREM 4.2 (COMPLETENESS OF $\mathcal{F}$).
If $\mathcal{F}(U) \Rightarrow \mathcal{F}(R)$ where the proof does not use the rules for the modality $!$ given in Figure 3, then $U \hookrightarrow R$.

The assumption on the proof of $\mathcal{F}(U) \Rightarrow \mathcal{F}(R)$ in Theorem 4.2 implies that we regard $!P(\vec{t})$ as an atomic formula and never decompose it to add $P(\vec{t})$ to an unrestricted context. (Hence we do not need extended linear sequents.) The rationale is that the proof of $\mathcal{F}(U) \Rightarrow \mathcal{F}(R)$ concerns itself only with the relationship between U and R, and not with deducing new logical theories from U and R. Without this assumption, the completeness of $\mathcal{F}$ fails. For example, $\mathcal{F}(U) \Rightarrow \mathcal{F}(\cdot)$ holds for any uncertain database U by repeatedly applying the rule $!L$ while $U \hookrightarrow \cdot$ holds only if U is empty or consists of maybe tuples.

The soundness of $\mathcal{F}$ is easy to prove because of the compositionality of $\mathcal{F}$:

PROOF THEOREM 4.1. By induction on the structure of the proof of $U \hookrightarrow R$. □

The proof of the completeness of $\mathcal{F}$ is also straightforward (mainly because of the use of the sequent calculus in formulating linear logic), but requires a series of lemmas. We write $\otimes_{i=1}^{n} !P(\vec{t_i})$ for $!P(\vec{t_1}) \otimes \cdots \otimes !P(\vec{t_n})$, and $\cup_{i=1}^{n} P(\vec{t_i})$ for $P(\vec{t_1}), \cdots, P(\vec{t_n})$. Theorem 4.2 follows immediately from Lemma 4.10.

LEMMA 4.3. *If $\mathcal{F}(Z) \Rightarrow !P(\vec{t})$, then $Z \hookrightarrow P(\vec{t})$.*

PROOF. By induction on the structure of Z.
The case $Z = T$ is trivial.
We let $Z = Z_1 \odot Z_2$ and $\mathcal{F}(Z) = \mathcal{F}(Z_1) \,\&\, \mathcal{F}(Z_2)$.
$\mathcal{F}(Z_1) \Rightarrow !P(\vec{t})$ or $\mathcal{F}(Z_2) \Rightarrow !P(\vec{t})$ by the rules $\&L_1$ and $\&L_2$.
$Z_1 \hookrightarrow P(\vec{t})$ or $Z_2 \hookrightarrow P(\vec{t})$ by induction hypothesis.
$Z \hookrightarrow P(\vec{t})$ by the rules $\mathsf{I} \odot_L$ and $\mathsf{I} \odot_R$. □

PROPOSITION 4.4. *If $\mathcal{F}(X) \Rightarrow !P(\vec{t})$, then $X \hookrightarrow P(\vec{t})$.*

PROOF. By induction on the structure of X.
The case $X = Z$ follows from Lemma 4.3.
We let $X = Z?$ and $\mathcal{F}(X) = \mathcal{F}(Z) \,\&\, \mathbf{1}$.
$\mathcal{F}(Z) \Rightarrow !P(\vec{t})$ or $\mathbf{1} \Rightarrow !P(\vec{t})$ by the rules $\&L_1$ and $\&L_2$.
$\mathcal{F}(Z) \Rightarrow !P(\vec{t})$ because $\mathbf{1} \Rightarrow !P(\vec{t})$ is not provable.
$Z \hookrightarrow P(\vec{t})$ by Lemma 4.3.
$X \hookrightarrow P(\vec{t})$ by the rule $\mathsf{I}?_1$. □

LEMMA 4.5. *$\mathcal{F}(Z) \Rightarrow \mathbf{1}$ never holds.*

PROOF. By induction on the structure of Z.
The case $Z = T$ is trivial (because the proof does not use the rules for $!$).
We let $Z = Z_1 \odot Z_2$ and $\mathcal{F}(Z) = \mathcal{F}(Z_1) \,\&\, \mathcal{F}(Z_2)$.
$\mathcal{F}(Z_1) \Rightarrow \mathbf{1}$ or $\mathcal{F}(Z_2) \Rightarrow \mathbf{1}$ by the rules $\&L_1$ and $\&L_2$.
Neither case holds by induction hypothesis. □

PROPOSITION 4.6. *If $\mathcal{F}(X) \Rightarrow \mathbf{1}$, then $X \hookrightarrow \cdot$.*

PROOF. By Lemma 4.5, we have $X = Z?$. Then we have $X \hookrightarrow \cdot$ by the rule $\mathsf{I}?_2$. □

LEMMA 4.7. *If $\mathcal{F}(Z) \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})$, then $n = 1$.*

PROOF. By induction on the structure of Z.
The case $Z = T$ is trivial.
We let $Z = Z_1 \odot Z_2$ and $\mathcal{F}(Z) = \mathcal{F}(Z_1) \,\&\, \mathcal{F}(Z_2)$.
The proof applies the rule $\&L_1$ or $\&L_2$ immediately because the right formula is a simultaneous conjunction.
In either case, we have $n = 1$ by induction hypothesis. □

PROPOSITION 4.8. *If* $\mathcal{F}(X) \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})$, *then* $n \leq 1$.

PROOF. By induction on the structure of X.
The case $X = Z$ follows from Lemma 4.7.
We let $X = Z?$ and $\mathcal{F}(X) = \mathcal{F}(Z)$ & $\mathbf{1}$.

The proof applies the rule $\&L_1$ or $\&L_2$ immediately because the right formula is a simultaneous conjunction.

If $\mathcal{F}(Z) \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})$, we have $n = 1$ by Lemma 4.7.

If $\mathbf{1} \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})$, we have $n = 0$ because there is no proof of $\mathbf{1} \Rightarrow !P(\vec{t})$. □

LEMMA 4.9. *Suppose* $\dfrac{\vdots}{\dfrac{\Delta \Rightarrow C}{\mathcal{F}(U_1), \cdots, \mathcal{F}(U_n) \Rightarrow C}\ \mathit{Rule}}$ *where Rule is a left rule* $\otimes L$, $\&L_1$, $\&L_2$, *or* $\mathbf{1}L$. *Then we have* $\Delta = \mathcal{F}(U'_1), \cdots, \mathcal{F}(U'_m)$ *such that* $U'_1, \cdots, U'_m \hookrightarrow R$ *implies* $U_1, \cdots, U_n \hookrightarrow R$ *for any relational database* R.

PROOF. By case analysis of the rule *Rule*. □

LEMMA 4.10. *If* $\mathcal{F}(U) \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})$, *then* $U \hookrightarrow \cup_{i=1}^{n} P(\vec{t_i})$.

PROOF. By induction on the size of U (not on its structure).
If $U = \cdot$, then we have $n = 0$ and $U \hookrightarrow \cdot$ by the rule $\mathsf{I}U_1$.
If $U = X$, then $U \hookrightarrow \cup_{i=1}^{n} P(\vec{t_i})$ holds by Propositions 4.4, 4.6, and 4.8.
If $U = X, U'$ where U' is not empty, we have $\mathcal{F}(U) = \mathcal{F}(X) \otimes \mathcal{F}(U')$. We consider a proof $\mathcal{D}$ of $\mathcal{F}(X) \otimes \mathcal{F}(U') \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})$.

Suppose that $\mathcal{D}$ does not use the rule $\otimes L$. Then we have $n > 1$ and $\mathcal{F}(X) \otimes \mathcal{F}(U') = \otimes_{i=1}^{n} !P(\vec{t_i})$, which implies $X, U' = \cup_{i=1}^{n} P(\vec{t_i})$ and $U \hookrightarrow \cup_{i=1}^{n} P(\vec{t_i})$.

Now suppose that $\mathcal{D}$ uses the rule $\otimes L$. Then $\mathcal{D}$ has the following structure

$$\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\cup_i \mathcal{F}(U_i) \Rightarrow \otimes_{p=1}^{m} !P(\vec{t_{i_p}}) \quad \cup_j \mathcal{F}(U_j) \Rightarrow \otimes_{q=1}^{l} !P(\vec{t_{j_q}})}\otimes R}{\left.\begin{array}{c}[\mathcal{F}(X)], [\mathcal{F}(U')] \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i}) \\ \vdots \\ \mathcal{F}(X), \mathcal{F}(U') \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})\end{array}\right\}\mathcal{E}}}{\mathcal{F}(X) \otimes \mathcal{F}(U') \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})}\otimes L$$

where $\mathcal{E}$ consists only of applications of left rules and we have

$$\begin{array}{rcl} \otimes_{i=1}^{n} !P(\vec{t_i}) & = & (\otimes_{p=1}^{m} !P(\vec{t_{i_p}})) \otimes (\otimes_{q=1}^{l} !P(\vec{t_{j_q}})) \\ \cup_{i=1}^{n} P(\vec{t_i}) & = & \cup_{p=1}^{m} P(\vec{t_{i_p}}), \cup_{q=1}^{l} P(\vec{t_{j_q}}) \\ [\mathcal{F}(X)], [\mathcal{F}(U')] & = & \cup_i \mathcal{F}(U_i), \cup_j \mathcal{F}(U_j). \end{array}$$

Here $[\mathcal{F}(X)]$ is a linear context originating from $\mathcal{F}(X)$ in the course of applying those left rules in $\mathcal{E}$. Similarly $[\mathcal{F}(U')]$ is a linear context originating from $\mathcal{F}(U')$. Note that the right formula $\otimes_{i=1}^{n} !P(\vec{t_i})$ remains intact in $\mathcal{E}$.

From $\cup_i \mathcal{F}(U_i) \Rightarrow \otimes_{p=1}^{m} !P(\vec{t_{i_p}})$, we obtain

$$\mathcal{F}(\cup_i U_i) \Rightarrow \otimes_{p=1}^{m} !P(\vec{t_{i_p}}).$$

Similarly we obtain

$$\mathcal{F}(\cup_j U_j) \Rightarrow \otimes_{q=1}^{l} !P(\vec{t_{j_q}}).$$

Since both $\cup_i U_i$ and $\cup_j U_j$ are strictly smaller than U, we obtain $\cup_i U_i \hookrightarrow \cup_{p=1}^{m} P(\vec{t_{i_p}})$ and $\cup_j U_j \hookrightarrow \cup_{q=1}^{l} P(\vec{t_{j_q}})$ by induction hypothesis. Combining the two results, we obtain $\cup_i U_i, \cup_j U_j \hookrightarrow \cup_{i=1}^{n} P(\vec{t_i})$. By repeatedly applying Lemma 4.9 to $\mathcal{E}$ (starting from $[\mathcal{F}(X)], [\mathcal{F}(U')] \Rightarrow \otimes_{i=1}^{n} !P(\vec{t_i})$), we eventually obtain $U \hookrightarrow \cup_{i=1}^{n} P(\vec{t_i})$. □

4.3 Application: membership problems

In addition to logically interpreting uncertain databases, the semantic function $\mathcal{F}$ also enables us to logically interpret membership problems arising in uncertain databases. We consider four membership problems discussed in [23]:

- The *instance membership problem* tests whether a relational database R is an instance of an uncertain database U, *i.e.*, $U \hookrightarrow R$. By Theorems 4.1 and 4.2, the problem reduces to proving $\mathcal{F}(U) \Rightarrow \mathcal{F}(R)$.
- The *instance certainty problem* tests whether an uncertain database U is equivalent to a relational database R, *i.e.*, R is the only instance of U. A logical equivalence $\mathcal{F}(U) \equiv \mathcal{F}(R)$ checks whether U and R are equivalent or not.
- The *tuple membership problem* tests whether an uncertain database U instantiates to a relational database containing a specific tuple T, *i.e.*, $U \hookrightarrow T, R$ for some relational database R. By Theorems 4.1 and 4.2, the problem reduces to proving $\mathcal{F}(U) \Rightarrow \mathcal{F}(T, R)$, or $\mathcal{F}(U) \Rightarrow \mathcal{F}(T) \otimes \mathcal{F}(R)$. Since R is unknown, we may prove $\mathcal{F}(U) \Rightarrow \mathcal{F}(T) \otimes \top$ instead.
- The *tuple certainty problem* tests whether a tuple T appears in every instance of an uncertain database U. In order for U to produce T in all its instances, U must be equivalent to T, U' for some uncertain database U'. Hence the problem reduces to proving $\mathcal{F}(U) \equiv \mathcal{F}(T, U')$, or $\mathcal{F}(U) \equiv \mathcal{F}(T) \otimes \mathcal{F}(U')$.

4.4 Extension of $\mathcal{F}$

We now discuss how our definition of the semantic function $\mathcal{F}$ straightforwardly extends to a more general definition of x-tuples. Consider the following new definition of *x-tuples* which does not require maybe tuples to contain only disjunctive tuples:

$$\text{x-tuple} \qquad X \quad ::= \quad T \mid \ X? \mid X \odot X$$

We revise the definition of $\mathcal{F}$ as well as the rules $\mathsf{I} \odot_L$, $\mathsf{I} \odot_R$, $\mathsf{I}?_1$, and $\mathsf{I}?_2$ in Figure 1 so as to reflect the change in the definition of x-tuples:

$$\begin{array}{rcl} \mathcal{F}(P(\vec{t})) & = & !P(\vec{t}) \\ \mathcal{F}(X \odot X') & = & \mathcal{F}(X)\ \&\ \mathcal{F}(X') \\ \mathcal{F}(X?) & = & \mathcal{F}(X)\ \&\ \mathbf{1} \end{array}$$

$$\dfrac{X \hookrightarrow R}{X \odot X' \hookrightarrow R}\ \mathsf{I} \odot_L \qquad \dfrac{X' \hookrightarrow R}{X \odot X' \hookrightarrow R}\ \mathsf{I} \odot_R$$

$$\dfrac{X \hookrightarrow R}{X? \hookrightarrow R}\ \mathsf{I}?_1 \qquad \dfrac{}{X? \hookrightarrow \cdot}\ \mathsf{I}?_2$$

Then Theorems 4.1 and 4.2 continue to hold and $\mathcal{F}$ remains sound and complete. The proof of Theorem 4.1 proceeds by induction on the structure of the proof of $U \hookrightarrow R$, and the proof of Theorem 4.2 is similar to the previous case except that it proves Propositions 4.4, 4.6, and 4.8 directly by induction on the structure of X.

5. RELATED WORK

While logical accounts of uncertain databases using attribute variables (null values in particular) have been studied thoroughly (see [25] for a survey), there is a distinct lack of research on logical accounts of uncertain databases with maybe information, perhaps because of the difficulty of directly mapping maybe information to logical formulae. The only work that we are aware of is the semantics proposed by Liu and Sunderraman [17] (and later adopted by Zimanyi [27]) which is based on propositional logic. Their semantics circumvents the problem of translating maybe tuples by generating a first-order formula that amounts to declaring at once all predicates present in a given uncertain database. For example, the uncertain database U in Section 2.1 generates the following first-order formula which essentially states the completion axiom in the formulation of Reiter [20]:

$$\forall x.\forall y.\mathit{Eat}(x,y) \supset \\ \mathit{Eat}(\mathsf{Tom},\mathsf{Pizza}) \vee \mathit{Eat}(\mathsf{Tom},\mathsf{Soup}) \vee \\ \mathit{Eat}(\mathsf{John},\mathsf{Salad}) \vee \mathit{Eat}(\mathsf{Jane},\mathsf{Steak})$$

Since there is now a possibility that $\mathit{Eat}(\mathsf{John},\mathsf{Salad})$ is true, the semantics just ignores the maybe tuple $\mathit{Eat}(\mathsf{John},\mathsf{Salad})?$.

Unlike our semantics, the semantics of Liu and Sunderraman requires Z in $Z?$ to be an ordinary tuple. If Z is allowed to be a disjunctive tuple, the semantics may translate operationally different uncertain databases to the same logical formulae. For example, the following uncertain databases U_1 and U_2 are translated identically even though they are operationally different:

$$\begin{aligned} U_1 &= (\mathit{Eat}(\mathsf{Tom},\mathsf{Pizza}) \odot \mathit{Eat}(\mathsf{Tom},\mathsf{Soup}))? \\ U_2 &= \mathit{Eat}(\mathsf{Tom},\mathsf{Pizza})?, \mathit{Eat}(\mathsf{Tom},\mathsf{Soup})? \end{aligned}$$

Another difference is that their semantics is not compositional: the translation of an uncertain database U_1, U_2 is not a direct sum of the individual translations of U_1 and U_2. In summary, our semantics is more general and compositional, yet considerably simpler, thanks to the use of linear logic as its logical foundation.

6. CONCLUSION

We have studied a formal semantics of uncertain databases. We take a logical approach of translating uncertain databases to logical formulae. Our semantics distinguishes itself from prior efforts by using linear logic as its logical foundation. We show that our semantics is faithful to the operational account of uncertain databases in the algebraic approach.

As future work, we plan to investigate a logical interpretation of operations on uncertain databases. For example, we could define a semantic function from database operators to logical formulae so that the problem of testing the correctness of database operations reduces to checking the relation between logical formulae. In conjunction with the semantic function $\mathcal{F}$ for uncertain databases, such a semantic function will make the logical account of uncertain databases not only theoretically interesting but also practically important.

Acknowledgement

We are grateful to Hyeonseung Im and anonymous reviewers for their helpful comments.

7. REFERENCES

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 34–48, New York, NY, USA, 1987. ACM.

[2] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: databases with uncertainty and lineage. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 953–964. VLDB Endowment, 2006.

[3] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4(2):115–132, 1986.

[4] N. Bidoit, S. Cerrito, and C. Froidevaux. A linear logic approach to consistency preserving updates. *Journal of Logic and Computation*, 6(3):439–463, 1996.

[5] A. J. Bonner and M. Kifer. The state of change: A survey. *Lecture Notes in Computer Science*, 1472:1–36, 1998.

[6] B. Chang, K. Chaudhuri, and F. Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131, School of Computer Science, Carnegie Mellon University, 2003.

[7] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[8] L. G. DeMichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Transactions on Knowledge and Data Engineering*, 1(4):485–493, 1989.

[9] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[10] G. Grahne. Horn tables-an efficient tool for handling incomplete information in databases. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 75–82, New York, NY, USA, 1989. ACM.

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

[12] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete object—a data model for design and planning applications. In *SIGMOD '91: Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, pages 288–297, New York, NY, USA, 1991. ACM.

[13] G. Kahn. Natural semantics. In *4th Annual Symposium on Theoretical Aspects of Computer Sciences (STACS)*, pages 22–39. Springer-Verlag, 1987.

[14] D.-T. Lee and C. P. Tsang. Linear logic for deductive databases. *New Generation Computing*, 17(2):201–228, 1999.

[15] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. In *PODS '93: Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 37–48, New York, NY, USA, 1993. ACM.

[16] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, Apr. 1992.

[17] K.-C. Liu and R. Sunderraman. Indefinite and maybe information in relational databases. *ACM Transactions on Database Systems*, 15(1):1–39, 1990.

[18] K.-C. Liu and R. Sunderraman. A generalized relational model for indefinite and maybe information. *IEEE Transactions on Knowledge and Data Engineering*, 3(1):65–77, 1991.

[19] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic I: actions as proofs. *Theoretical Computer Science*, 113(2):349–370, 1993.

[20] R. Reiter. Towards a logical reconstruction of relational database theory. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pages 191–233. Springer, 1984.

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

[22] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380, 1991.

[23] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 7, Washington, DC, USA, 2006. IEEE Computer Society.

[24] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*. North-Holland, 1988.

[25] R. van der Meyden. Logical approaches to incomplete information: a survey. In *Logics for databases and information systems*, pages 307–356. Kluwer Academic Publishers, 1998.

[26] M. Y. Vardi. Querying logical databases. In *PODS '85: Proceedings of the fourth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 57–65, New York, NY, USA, 1985. ACM.

[27] E. Zimányi. *Incomplete and Uncertain Information in Relational Databases*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, October 1992.