선형논리에 기반한 불확실성 데이터베이스 의미론

(Semantics of Uncertain Databases based on Linear Logic)

박 성 우 †

(Sungwoo Park)

요 약 불확실성 데이터베이스의 의미론 정의는 보통 주어진 불확실성 데이터베이스를 여러 개의 관계형데이터베이스로 변환하는 산술적 접근방법을 취한다. 이 논문에서는 불확실성데이터베이스를 논리이론으로 변환하는 논리적 접근방법을 통해서 불확실성 데이터베이스의 의미론을 정의하고자 한다. 본 논문에서 제안하는 의미론의 가장 특징적인 면은 기존의 논리적 접근방법에서 사용해온 명제논리 대신에 선형논리를 논리적 근간으로 이용한다는 점이다. 선형논리는 논리식을 불변진리가 아닌 소비가능한 자원으로 해석하기 때문에 불확실성 데이터베이스의 의미론을 정의하는데 적합하다. 본 논문의 핵심 결과는 선형논리에 기반한 불확실성 데이터베이스의 의미론이 산술적 접근방식에서 설명하는 불확실성 데이터베이스의 의미론과 동등하다는 것이다.

키워드 : 불확실성데이터베이스, 의미론, 선형논리

Abstract In the study of the formal semantics of uncertain databases, we typically take an algebraic approach by mapping an uncertain database to possible worlds which are a set of relational databases. In this paper, 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 is suitable for a logical interpretation of uncertain information because unlike propositional logic, it treats logical formulae not as persistent facts but as consumable resources. As the main result, we show that our semantics is faithful to the operational account of uncertain databases in the algebraic approach.

Key words : Uncertain database, semantics, linear logic

1. Introduction

An important problem in database research 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 Imielin'ski and Lipski [1] and the logical approach proposed by Reiter [2]. 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 research was supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2006-331-D00521) and National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies and Experts Education.

† 정 회 원 : POSTECH 컴퓨터공학과 교수
gla@postech.ac.kr

논문접수 : 2009년 11월 3일
심사완료 : 2009년 11월 11일

정보과학회논문지: 소프트웨어 및 응용 제37권 제2호(2010.2)

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 [2-4].) 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 [5,6].

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 [7]. 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. 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.

The rest of this paper is organized as follows. Section 2 clarifies the goal of our work. 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 ::= $\cdot \mid X, U$

uncertain database R ::= $\cdot \mid T, R$

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

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

$$\frac{Z \hookrightarrow R}{Z? \hookrightarrow R}\,!?_1 \qquad \frac{}{Z? \hookrightarrow \cdot}\,!?_2 \qquad \frac{}{\cdot \hookrightarrow \cdot}\,!U_1 \qquad \frac{X \hookrightarrow R \quad U \hookrightarrow R'}{X, U \hookrightarrow R, R'}\,!U_2$$

Fig. 1 Rules for $U \hookrightarrow R$

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*(**Tom, Pizza**)$\odot$*Eat*(**Tom, Soup**) is well-formed whereas *Eat*(**Tom, Pizza**)$\odot$*Where*(**Pizza, R1**) 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.

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

U = *Eat*(**Tom, Pizza**)$\odot$*Eat*(**Tom, Soup**),
Eat(**John, Salad**)$\odot$*Eat*(**Jane, Steak**)

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 [8] (where we regard U as a program, R as a value, and $\hookrightarrow$ as an evaluation). For example, the rule $|\odot_L$ says that $Z \odot Z' \hookrightarrow R$ holds whenever $Z \hookrightarrow R$ holds, and the rule $|\odot_R$ 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 $|\odot_L$ and $|\odot_R$ suggest that $\odot$ is associative and commutative.

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

R_1 = *Eat*(**Tom, Pizza**), *Eat*(**John, Salad**), *Eat*(**Jane, Steak**)

R_2 = *Eat*(**Tom, Soup**), *Eat*(**John, Salad**), *Eat*(**Jane, Steak**)

R_3 = *Eat*(**Tom, Pizza**), *Eat*(**Jane, Steak**)

R_4 = *Eat*(**Tom, Soup**), *Eat*(**Jane, Steak**)

2.3 Goal

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

• $F(U)$ logically implies $F(R)$ if and only if $U \hookrightarrow R$.

It says that 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 $F(U)$ and $F(R)$.

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:

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

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

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

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

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

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

Fig. 2 Inference rules in the sequent calculus for linear logic

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:

linear context Δ ::= $\cdot \mid A, \Delta$

$\Delta \Rightarrow A$ 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 1 is the identity for $\otimes$, *i.e.*, $1 \otimes A \equiv A \otimes 1 \equiv A$, and that $\top$ is the identity for &, *i.e.*, $\top \,\&\, A \equiv A \,\&\, \top \equiv A$. Both & and $\otimes$ are associative and

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

Fig. 3 Inference rules for the modality !

commutative, and $\multimap$ is right associative.

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:

formula A == $\cdots \mid !A$

In order to incorporate the modality !, we follow the style in [9] and use an extended linear sequent $\Gamma; \Delta \Rightarrow A$ where an *unrestricted context* Γ is a set of formulae:

unrestricted context Γ == $\cdot \mid A, \Gamma$

$\Gamma; \Delta \Rightarrow A$ 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 $\cdot; \Delta \Rightarrow A$.

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.

4. Semantics based on linear logic

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

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

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

4.1 Definition of the semantic function *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 $Z \,\&\, 1$ where 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 F uses $\otimes$, &, 1 and ! in linear logic:

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

The definition of 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 *Eat*(**Tom, Pizza**) $\odot$ *Eat*(**Tom, Soup**) twice to obtain both *Eat*(**Tom, Pizza**) and *Eat*(**Tom, Soup**), but once we decide to choose *Eat*(**Tom, 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 $F(Z) \,\&\, F(Z')$.
- 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 $F(Z) \,\&\, 1$.
- An empty uncertain database gives no information. Hence we translate it to 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 $F(X) \otimes F(U)$, which means that F is compositional.

4.2 Soundness and completeness of F

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

Theorem 4.1 (Soundness of F) If $F(U) \Rightarrow F(R)$ where the proof does not use the rules for the modality ! given in Figure 3, then $U \hookrightarrow R$.

Theorem 4.2 (Completeness of F) If $U \hookrightarrow R$, then $F(U) \Rightarrow F(R)$.

The assumption on the proof of $F(U) \Rightarrow F(R)$ in Theorem 4.1 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 $F(U) \Rightarrow F(R)$ 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 F fails. For example, $F(U) \Rightarrow F(\cdot)$ 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 completeness of F is easy to prove because of the compositionality of F. The proof of the soundness of F is also straightforward mainly because of the use of the sequent calculus in formulating linear logic. Theorem 4.1 follows immediately from Lemma 4.3.

Lemma 4.3 If $F(U) \Rightarrow \otimes_{i=1}^{n} !\,P(\vec{t_i})$,

then $U \hookrightarrow \cup_{i=1}^{n} P(\vec{t_i})$.

Proof. By induction on the size of U (not on its structure). □

5. Related work

While logical accounts of uncertain databases using attribute variables (null values in particular) have been studied thoroughly (see [10] for a survey), there is a distinct lack of research on logical accounts of uncertain databases with maybe infor-

mation, 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 [11] (and later adopted by Zimanyi [12]) 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 [2]:

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

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

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:

$U_1 = (Eat(\mathbf{Tom, Pizza}) \odot Eat(\mathbf{Tom, Soup}))?$

$U_2 = Eat(\mathbf{Tom, Pizza})? \odot Eat(\mathbf{Tom, Soup})?$

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 F for uncertain databases, such a semantic function will make the logical account of uncertain databases not only theoretically interesting but also practically important.

Reference

[1] T. Imieli'nski and W. Lipski, Jr. Incomplete information in relational databases. *Journal of ACM*, 31(4):761-791, 1984.

[2] 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, pp.191-233. Springer, 1984.

[3] M. Y. Vardi. Querying logical databases. In *Proceedings of the Symposium on Principles of Database Systems*, pp.57-65. ACM, 1985.

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

[5] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: databases with uncertainty and lineage. In *Proceedings of the International Conference on Very Large Data Bases*, pp.953-964, 2006.

[6] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proceedings of the International Conference on Data Engineering*, pp.7. IEEE, 2006.

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

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

[9] 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.

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

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

[12] E. Zim'anyi. Incomplete and Uncertain Information in Relational Databases. PhD thesis, Universit'e Libre de Bruxelles, Brussels, Belgium, October 1992.

박 성 우

1996년 KAIST 전산학 학사. 1998년 KAIST 전산학 석사. 2005년 Carnegie Mellon University 전산학 박사. 2006년~현재 POSTECH 컴퓨터공학과 조교수