

A Semantics for Complex Objects and Approximate Queries *

Peter Buneman, Susan Davidson, Aaron Watters
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Appeared in *7th ACM Principles of Database Systems*.

November 15, 1992

Abstract

A new definition of complex objects is introduced which provides a denotation for incomplete tuples as well as partially described sets. Set values are “sandwiched” between “complete” and “consistent” descriptions (representing the Smyth and Hoare powerdomains respectively), allowing the maximal values to be arbitrary subsets of maximal elements in the domain of the set. We also examine the use of rules in defining queries over such objects.

1 Introduction

A characteristic of “complex-object” [1, 2] databases and “higher-order” relations [3, 4] is that the components of tuples are not restricted to taking only atomic values, but may be other tuples or even sets of tuples. A second property of complex objects and related information structures is that there is a natural ordering on the domain of values with an associated algebra [5, 6, 7]. For example, in Bancilhon and Khoshafian’s ordering on tuples [1]

$$[Name \Rightarrow 'J.Doe'] \sqsubseteq [Name \Rightarrow 'J.Doe'; Age \Rightarrow 21]$$

*This research was supported in part by NSF IRI86-10617, NSF MCS 8219196-CER, ARO DAA6-29-84-k-0061, DAA29-84-9-0027, and a grant from AT&T’s Telecommunications Program at the University of Pennsylvania.

since the first tuple is not defined on Age . More generally, if \mathcal{V} is a partially ordered domain of values and \mathcal{L} is a set of labels, then a tuple is a function in $\mathcal{L} \rightarrow \mathcal{V}$, and the ordering on tuples is the ordering on the function space:

$$t_1 \sqsupseteq t_2 \equiv \forall l \in \mathcal{L}. t_1(l) \sqsupseteq t_2(l).$$

Since the domain of values can also contain sets, we need to extend this ordering to sets of values. In [1], two subsets A and B of the same domain are ordered by

$$A \sqsubseteq^b B \equiv \forall a \in A. \exists b \in B. a \sqsubseteq b.$$

and this ordering is inductively extended to order “complex objects” which are hierarchical structures containing both tuples and sets. In contrast, in an attempt to find a data type for natural join, [8] uses the ordering

$$A \sqsubseteq^{\sharp} B \equiv \forall b \in B. \exists a \in A. a \sqsubseteq b,$$

Both of these orderings are well-known in the study of the semantics of concurrency and non-determinism; \sqsubseteq^{\sharp} and \sqsubseteq^b are respectively called the Smyth and Hoare powerdomains [9]. To make \sqsubseteq^b and \sqsubseteq^{\sharp} orderings as opposed to a pre-ordering, A and B must be restricted to be *co-chains*. S is a co-chain in a domain \mathcal{D} if any pair of elements in S is incomparable, i.e. if $x, y \in S$ and $x \sqsubseteq y$ then $x = y$. Observe that the maximal elements of \sqsubseteq^{\sharp} and \sqsubseteq^b are respectively the empty set ($\{\}$) and the set of all maximal elements of \mathcal{D} .

Unfortunately, if one wants to assign a reasonable semantics to sets of values neither of these orderings in isolation is satisfactory since their maximal elements are uninteresting. [10], in formalizing incomplete information, describes the semantics of a tuple such as $[Name \Rightarrow 'J.Doe'; Age \Rightarrow _]$ as $\{[Name \Rightarrow 'J.Doe'; Age \Rightarrow i] \mid i \in I\}$ where I is the set of all possible (total) values for *Age*. More generally, if \mathcal{D} is a partially ordered space, we can define the denotation of a tuple x , $\llbracket x \rrbracket$, as $\{y \mid y \text{ is maximal in } \mathcal{D} \text{ and } y \geq x\}$. However, if we extend this simple-minded notion of semantics to tuples involving sets of values, the denotation of the tuple $[Name \Rightarrow 'J.Doe'; Children \Rightarrow \{'John', 'Mike'\}]$ would, using the Hoare powerdomain, be a tuple with all possible children, and, using the Smyth powerdomain, be a tuple with no children. Clearly, neither of these is acceptable; we need a domain in which the maximal elements are *subsets* of the maximal elements of \mathcal{D} .

In this paper, we use the Hoare and Smyth powerdomains *together* to place outer and inner bounds on – or to “sandwich” – subsets of the maximal elements of \mathcal{D} . A sandwich is then a pair of co-chains in \mathcal{D} . Sandwiches are ordered by how well they describe subsets of the maximal elements of \mathcal{D} , i.e., a better sandwich will describe fewer subsets. Using sandwiches, we will be able to develop a fixed-point semantics for queries in a richer domain: one which allows us to describe both approximate queries and incomplete information.

2 The Sandwich ordering

We would like the spaces we are using to be rich enough to describe recursive record structures. For this purpose, the appropriate definition of a domain is an ω -algebraic consistently complete partial order [11], and most of the results we will give apply to such domains. However, since all our examples will be of finite, non-recursive structures, the only important property of a domain is that it is a partially ordered space \mathcal{D} such that the greatest lower bound of any nonempty subset X of \mathcal{D} , $\sqcap X$, exists. When two or more members of \mathcal{D} have an upper bound, we call them *consistent*.

Given a domain \mathcal{D} , let $\mathcal{C}(\mathcal{D})$ denote the co-chains on \mathcal{D} . A *sandwich* in \mathcal{D} is a pair (A, B) with $A, B \in \mathcal{C}(\mathcal{D})$ such that $\exists S \subseteq \mathcal{D}. A \sqsubseteq^{\sharp} S$ and $B \sqsubseteq^b S$.

Let $\mathcal{S}(\mathcal{D})$ denote the sandwiches on \mathcal{D} . We can define an ordering on $\mathcal{S}(\mathcal{D})$ by $(A_1, B_1) \sqsubseteq^S (A_2, B_2) \equiv A_1 \sqsubseteq^{\sharp} A_2$ and $B_1 \sqsubseteq^b B_2$.

Prop 1 $(\mathcal{S}(\mathcal{D}), \sqsubseteq^S)$ is a domain.

The most important property of $\mathcal{S}(\mathcal{D})$ for our purposes is that the maximal elements correspond to subsets of maximal elements of \mathcal{D} ; more formally, using $Tot(\mathcal{D})$ for the set of maximal or *total* elements in \mathcal{D} ,

Prop 2 The maximal elements of $\mathcal{S}(\mathcal{D})$ are pairs (T, T) where $T \subseteq Tot(\mathcal{D})$.

From our previous definition of meaning, the denotation of a sandwich is given by $\llbracket(A, B)\rrbracket \equiv \{(T, T) | T \in \text{Tot}(\mathcal{D}), A \sqsubseteq^{\#} T \text{ and } B \sqsubseteq^b T\}$. We can think of A and B as being *complete* and *consistent* information about some $T \in \text{Tot}(\mathcal{D})$. When $(T, T) \in \llbracket(A, B)\rrbracket$, we shall say that (A, B) *approximates* T .

For example, suppose we are seeking to approximate the set of students T who are interested in the study of databases. If we know they all took Database 1, then the list of all last names of students who registered for Database 1 would be a *complete* approximation A for T :

<i>Last Name</i>
Johnson
Pierce
Taylor
Cooper
Emerson
Billings

Note that this is an over-approximation of T since these students are not necessarily interested in databases. However, any student who *is* interested in databases must be described by this set (albeit incompletely).

We may additionally happen to know the names of a few interested students. Thus, from memory we can construct a *consistent* description B for T :

<i>First Name</i>	<i>Last Name</i>
Ella	Taylor
Burt	-
-	Pierce

Note that this is an under-approximation since it does not necessarily describe everything in T .

These two approximations together form a sandwich approximation (A, B) to the set of total descriptions T . Two examples of totally defined sets of names that are approximated by (A, B) are:

<i>First Name</i>	<i>Last Name</i>
Ella	Taylor
Burt	Cooper
Liza	Pierce
Burt	Pierce
Elvira	Johnson

<i>First Name</i>	<i>Last Name</i>
Ella	Taylor
Burt	Johnson
Fred	Pierce
Wayne	Cooper

However, the following would not be described by (A, B) since it is not completely described by A , even though it is consistent with B :

<i>First Name</i>	<i>Last Name</i>
Ella	Taylor
Burt	Elliot
Larry	Pierce

Unfortunately, the domain of sandwiches lacks a property that is enjoyed by the domain of partial tuples: Two elements of $\mathcal{S}(\mathcal{D})$ may denote the same subset of $\text{Tot}(\mathcal{D})$. A domain \mathcal{D} is *descriptive* if, for any $x \in \mathcal{D}$, $x = \sqcap \llbracket x \rrbracket$. If \mathcal{V} is descriptive, the domain $\mathcal{L} \rightarrow \mathcal{V}$ is descriptive, but any domain with a top element fails to be descriptive. Moreover, our domain of sandwiches also fails to be descriptive even though its maximal elements are the structures we want to approximate. Fortunately, there is a canonical element that denotes $\llbracket S \rrbracket$, which we now show how to compute.

For any domain \mathcal{D} and $x \in \mathcal{D}$, define the “promotion” of x $\text{Prm}(x) \equiv \sqcap \llbracket x \rrbracket$. Prm is a monotone increasing idempotent function $\mathcal{D} \rightarrow \mathcal{D}$ (a closure). The fixpoints of such a function must form a \sqcap closed subset of \mathcal{D} . An important property of Prm is given by

Prop 3 $\text{Prm}((\text{Prm}(x) \sqcup y) = \text{Prm}(x \sqcup y)$

which will be useful in computing the result of an “approximate query”.

In the domain of sandwiches $\mathcal{S}(\mathcal{D})$, it is readily shown that

$$Prm(A, B) = (A, \sqcap\{a \sqcup b \mid a \in A \text{ and } a \sqcup b \text{ exists } \mid b \in B\})$$

Note that since (A, B) is a sandwich, we know that for each $b \in B$ there is at least one $a \in A$ such that a and b are consistent.

For example, if the following two relations form a sandwich that approximates T ,

<i>First Name</i>	<i>Last Name</i>	<i>First Name</i>	<i>Last Name</i>
Ella	Taylor	–	Taylor
Burt	Cooper	Burt	Johnson
–	Pierce	Fred	–
–	Johnson	–	Cooper

then the following “promoted” sandwich will also approximate T :

<i>First Name</i>	<i>Last Name</i>	<i>First Name</i>	<i>Last Name</i>
Ella	Taylor	Ella	Taylor
Burt	Cooper	Burt	Johnson
–	Pierce	Fred	–
–	Johnson	Burt	Cooper

When the underlying domain \mathcal{D} on which we construct sandwiches is the domain of tuples ordered as suggested in the introduction, the least upper bound \sqcup^{Ob} of two sandwiches $(A_1, B_1), (A_2, B_2)$ is (if it exists) $(A_1 \bowtie A_2, B_1 \bowtie_{Null} B_2)$, where \bowtie is the natural join and \bowtie_{Null} is the null join (the maximal elements of $B_1 \cup B_2$) [3]. Furthermore, if (C_1, C_1) and (C_2, C_2) are both “exact” sandwiches (i.e., each of C_1 and C_2 provide complete and consistent information about some set), the sandwiches themselves are consistent when C_1 and C_2 have a lossless join.

3 Extended Complex Objects

We first introduce a notation which is, roughly speaking, an extension of the notation in [1] that describes sandwiches as opposed to sets. We assume the existence of a set \mathcal{L} of labels.

- (a) Atomic values such as 1,2,3,... (integers); ‘John’, ‘Jane’,... (strings); *true*, *false* (booleans) are complex objects.
- (b) If O_1, O_2, \dots, O_n are complex objects and $l_1, l_2, \dots, l_n \in \mathcal{L}$ are distinct labels then $[l_1 \Rightarrow O_1; l_2 \Rightarrow O_2; \dots; l_n \Rightarrow O_n]$ is a complex object. Objects of this form are *tuples*.
- (c) If $A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m$ are complex objects then, subject to the consistency restrictions described below, $(\{A_1, A_2, \dots, A_n\}, \{B_1, B_2, \dots, B_m\})$ is a complex object. Objects of this form are *sandwiches*.
- (d) If B_1, B_2, \dots, B_m are complex objects, $(_, \{B_1, B_2, \dots, B_m\})$ is a complex object.

Rule (d) is necessary because we need a method of describing sandwiches in which there are no complete constraints.

When the two components of a sandwich are identical it is said to be an “exact” approximation, and we will use one component to describe it. Thus instead of $(\{A_1, A_2, \dots, A_n\}, \{A_1, A_2, \dots, A_n\})$ we will use $\{A_1, A_2, \dots, A_n\}$. Using this notation, a database is a tuple of relations which are exact approximations. For example, the following is a database with one relation *People*, the tuples of which have approximate information for *Children*:

$$[\text{People} \Rightarrow \{ [\text{Name} \Rightarrow 'J.Doe'; \text{Children} \Rightarrow (\{ 'Jane', 'Paul', 'Ann' \}, \{ 'Ann' \})], \\ [\text{Name} \Rightarrow 'S.Dee'; \text{Children} \Rightarrow (_, \{ 'Paul' \})] \}]$$

An ordering on complex objects, \sqsubseteq^{Ob} , is obtained from the following rules:

- (a) $O \sqsubseteq^{Ob} O$ if O is atomic.
- (b) If $O = [l_1 \Rightarrow O_1; l_2 \Rightarrow O_2; \dots; l_n \Rightarrow O_n]$ and $O' = [l'_1 \Rightarrow O'_1; l'_2 \Rightarrow O'_2; \dots; l'_n \Rightarrow O'_n]$, then $O \sqsubseteq^{Ob} O'$ if for all $l_k (1 \sqsubseteq k \sqsubseteq n)$, there is a $l'_i (1 \sqsubseteq i \sqsubseteq n)$ such that $l'_i = l_k$ and $O_k \sqsubseteq^{Ob} O'_i$.
- (c) If $O = (A, B)$ and $O' = (A', B')$ then $O \sqsubseteq^{Ob} O'$ if for all $a \in A$ there exists a $a' \in A'$ such that $a \sqsubseteq^{Ob} a'$; and for all $b' \in B'$ there exists a $b \in B$ such that $b \sqsubseteq^{Ob} b'$.
- (d) If $O = (_, B)$ and $O' = (_, B')$, then $O \sqsubseteq^{Ob} O'$ if for all $b' \in B'$ there exists a $b \in B$ such that $b \sqsubseteq^{Ob} b'$. Also if $O = (_, B)$ and $O' = (A, B)$ then $O \sqsubseteq^{Ob} O'$ if for all $b' \in B'$ there exists a $b \in B$ such that $b \sqsubseteq^{Ob} B'$.

To make this definition of a sandwich consistent with the one given in the previous section, we introduce the following constraint: (A, B) is a legal sandwich iff A and B are co-chains and (with respect to \sqsubseteq^{Ob}) there is a set C such that for all $c \in C$ there is an $a \in A$ for which $a \sqsubseteq^{Ob} c$, and for all $b \in B$ there is a $c \in C$ for which $b \sqsubseteq^{Ob} c$. Because of this constraint, the definition of an object and of the ordering \sqsubseteq^{Ob} are interdependent. To give absolutely precise definitions would require induction on the objects (or a fixpoint construction for recursive records).

Under this ordering

$$\begin{aligned} & (_, \{ [Fn \Rightarrow 'John'], [Fn \Rightarrow 'Jane'; Ln \Rightarrow 'Doe'], [Ln \Rightarrow 'Jones'] \}) \\ & \quad \sqsubseteq^{Ob} \\ & (\{ [Ln \Rightarrow 'Doe'], [Ln \Rightarrow 'Jones'], [Fn \Rightarrow 'Mary'; Ln \Rightarrow 'Jones'] \}, \\ & \quad \{ [Fn \Rightarrow 'John'; Ln \Rightarrow 'Doe'], [Fn \Rightarrow 'Jane'; Ln \Rightarrow 'Doe'], [Fn \Rightarrow 'Mary'; Ln \Rightarrow 'Jones'] \}) \end{aligned}$$

Furthermore, if \mathcal{O} denotes the set of all objects,

Prop 4 $(\mathcal{O}, \sqsubseteq^{Ob})$ is a domain

In an object in which all sandwiches are “exact” approximations, the notation here – but *not* the semantics – agrees with that in [1]. For example,

$$\begin{aligned} & (_, \{ [Fn \Rightarrow 'John'], [Fn \Rightarrow 'Jane'; Ln \Rightarrow 'Doe'], [Ln \Rightarrow 'Jones'] \}) \\ & \quad \sqsubseteq^{Ob} \\ & (\{ [Fn \Rightarrow 'John'; Ln \Rightarrow 'Doe'], [Fn \Rightarrow 'Jane'; Ln \Rightarrow 'Doe'], [Fn \Rightarrow 'Mary'; Ln \Rightarrow 'Jones'] \}) \end{aligned}$$

In order to constrain the space of objects so that maximal elements are meaningful structures, we introduce the notion of a *type* whose syntax is given by the following rules:

- (a) Base types such as *int* (integer), *string* (character string), and *bool* (boolean) are types.
- (b) If $\tau_1, \tau_2, \dots, \tau_n$ are types and $l_1, l_2, \dots, l_n \in \mathcal{L}$ then $l_1 : \tau_1; l_2 : \tau_2; \dots; l_n : \tau_n$ is a type. Such types are *tuple* types.
- (c) If τ is a type, $\{\tau\}$ is a type. These are *sandwich* types.

If O is an object and τ is a type, O is a *total object* of type τ (written $O : \tau$) if one of the following holds:

- (a) O is an atomic value, τ is a base type and $O \in \tau$ (strictly $\llbracket O \rrbracket \in \llbracket \tau \rrbracket$). Examples: $3 : \text{int}$, $'J.Doe' : \text{string}$.
- (b) $O = [l_1 \Rightarrow O_1; l_2 \Rightarrow O_2, \dots, l_n \Rightarrow O_n]$, $\tau = [l_1 \Rightarrow \tau_1; l_2 \Rightarrow \tau_2, \dots, l_n \Rightarrow \tau_n]$, and $O_i : \tau_i (1 \sqsubseteq i \sqsubseteq n)$

- (c) $O = (\{O_1, O_2, \dots, O_n\}, \{O_1, O_2, \dots, O_n\})$ (i.e., O is an exact approximation), $\tau = \{\tau'\}$, and $O_i : \tau' (1 \sqsubseteq i \sqsubseteq n)$,

For example,

$$[Persons \Rightarrow \{[Fn: string; Ln: string; Children: \{string\}]\}];$$

is a type and

$$[Persons \Rightarrow \{[Fn \Rightarrow 'John'; Ln \Rightarrow 'Doe'; Children \Rightarrow \{'Sally', 'Sue'\}]; \\ [Fn \Rightarrow 'Mary'; Ln \Rightarrow 'Brown'; Children \Rightarrow \{'Peter', 'James'\}] \}]$$

is an object of that type. Note that sandwiches of an object of some type are necessarily exact.

If $O : \tau$ and $O' \sqsubseteq^{Ob} O$ then we say O' is a *partial object* of type τ (written $O' <: \tau$).

Prop 5 *If $O_1 <: \tau$ and $O_2 <: \tau$ and O_1 and O_2 are consistent, then $(O_1 \sqcup^{Ob} O_2) <: \tau$*

The notion of a type has been introduced because we do not want tuples to be able to grow in an unbounded fashion, i.e., we do not want every attribute to be applicable to every tuple.

4 Rules and monotone functions

We now look at some applications of these more general domains and how we may use rules to define both consistent and complete approximations for some query. The syntax for rules follows the syntax for complex objects except that we shall later want to place some restrictions on their form to guarantee monotonicity.

We shall also divide rules into two classes: *necessary* rules, and *sufficient* rules that correspond to the two halves of the sandwich ordering \sqsubseteq^\sharp (the *complete side*), and \sqsubseteq^b (the *consistent side*). We will not discuss here the use of negation to derive complete information from consistent information, or *vice versa*. For another discussion of the connection between powerdomains and modalities see [12].

In the following we use a slightly abused notation for convenience and readability, writing

$$A \sqsupseteq^\sharp O \text{ and } B \sqsubseteq^b O$$

for an sandwich $O = (A', B')$ and sets of objects A and B, meaning respectively

$$(A, \{\}) \sqsupseteq^{Ob} (A',) \text{ and } (_, B) \sqsubseteq^{Ob} (_, B')$$

We will also use a containment notation to extract field values from records, for example if

$$[A \Rightarrow X; B \Rightarrow Y] \sqsubseteq [B \Rightarrow 1; C \Rightarrow John]$$

then we must have $X = _$ and $Y = 1$. Additionally, we will only consider ‘canonical sandwiches’ in the following discussion – those for which $(A, B) = Prm((A, B))$.

Suppose we have a database O of type

$$[CS4: \{[Name : string; Phone : string; Section : int]\}; \\ CS5: \{[Name : string; Room : string; Section : int]\} \\ GS: \{[Name : string; Degree : string]\} UE: \{[Name : string; Sal : int]\}]$$

containing information on all university employees (UE), all graduate students (GS) and all the teachers of courses CS5 and CS4. All of the values for CS5, CS4, UE, and GS will be sandwiches, and need not be maximal descriptions.

Further suppose that we wish to derive from this information the best possible description for the set of teaching fellows (TF) – in particular we would like to know their names, phone numbers and salaries.

Although we do not explicitly have information on the target set TF we can use our knowledge of its relationship to the other information to derive an approximation for TF.

Before deriving the description we must first adjoin to the database a type for the TF sandwich to the existing database:

$$[TF : \{[Name : string; Phone : string; Sal : int]\}]$$

Once the type is established we may use necessary and sufficient rules to determine an approximation for the TF set. It is interesting to note that the rules for this particular example corresponds to an *is-a* hierarchy among the datasets:

$$(TF \text{ is-a } GS) \text{ and } (TF \text{ is-a } UE) \text{ and } (CS4 \text{ is-a } TF) \text{ and } (CS5 \text{ is-a } TF).$$

The following paragraphs shows expressions for these rules and explain their semantics.

For instance we might know that all the teachers of CS4 are known to be teaching fellows, which we represent by the following *sufficient* rule:

$$[TF \Rightarrow \{[Name \Rightarrow X; Phone \Rightarrow Y]\}] : - [CS4 \Rightarrow \{[Name \Rightarrow X; Phone \Rightarrow Y]\}]$$

Intuitively this rule should be read as follows: “a person is known to be a TF *if* that person is known to be a teacher of CS4.” Formally this rule can be translated into an inference function f which generates a consistent description of TF’s from the consistent description of CS4. This f can be given by the following definition:

$$\lambda O. [TF \Rightarrow \max \{ [Name \Rightarrow X; Phone \Rightarrow Y] \mid [CS4 \Rightarrow C] \subseteq O \text{ and } \{E\} \sqsubseteq^b C \text{ and } [Name \Rightarrow X; Phone \Rightarrow Y] \subseteq E \}]$$

This function has the important property of *monotonicity* – it produces better inferences given better evidence. We consider this property to be a basic requirement for any form of inference system over partial information.

We then use the function f to find a least point O_f satisfying:

$$O_f \sqsupseteq^{Ob} f(O_f) \text{ and } O_f \sqsupseteq^{Ob} O.$$

The resulting O_f is the *effect* of applying the above rule to the database O . O_f is guaranteed to be unique since f is monotone.

In a similar fashion we might know that all teachers of CS5 are known to be teaching fellows, which would be expressed in the following rule:

$$[TF \Rightarrow \{[Name \Rightarrow X]\}] : - [CS5 \Rightarrow \{[Name \Rightarrow X]\}].$$

This rule in turn should be read: “A person is known to be a teaching fellow *if* that person is known to be a teacher of CS5.” And the monotone inference function f' for this rule would be expressed as

$$\lambda O. [TF \Rightarrow \max \{ [Name \Rightarrow X] \mid [CS5 \Rightarrow C] \subseteq O \text{ and } \{E\} \sqsubseteq^b C \text{ and } [Name \Rightarrow X] \subseteq E \}].$$

Note that the sufficient rules may *introduce* new elements to the consistent description of the target set TF, but cannot eliminate existing elements.

The necessary rules have a similar syntax, but a radically different meaning and effect. If, for example, we know that all teaching fellows are graduate students we could express this fact in the following *necessary* rule:

$$[TF \Rightarrow \{[Name \Rightarrow X]\}] - : [GS \Rightarrow \{[Name \Rightarrow X]\}]$$

which should be read: “a person is possibly a teaching fellow *only if* that person is possibly a graduate student.” This inference rule would also give rise to a function g , monotone in \sqsubseteq^{Ob} , that produces a complete description for the TF set given a complete description for GS. Using lambda notation we express g as follows:

$$\lambda O. [TF \Rightarrow \min \{ [Name \Rightarrow X] \mid [GS \Rightarrow G] \subseteq O \text{ and } \{E\} \sqsupseteq^{\sharp} G \text{ and } [Name \Rightarrow X] \subseteq E \}]$$

As in the sufficient rules we can now define O_g to be the least object satisfying

$$O_g \sqsupseteq^{Ob} g(O_g) \text{ and } O_g \sqsupseteq^{Ob} O.$$

And we consider O_g to be the effect of applying the above rule to the database O .

Similarly if we know that all teaching fellows are university employees then we can express this information in the following rule

$$[TF \Rightarrow \{[Name \Rightarrow X; Sal \Rightarrow Y]\}] - : [UE \Rightarrow \{[Name \Rightarrow X; Sal \Rightarrow Y]\}]$$

which in turn would generate a monotone inference function g' defined as

$$\lambda O. [TF \Rightarrow \min \{ [Name \Rightarrow X; Sal \Rightarrow Y] \mid [UE \Rightarrow U] \subseteq O \text{ and } \{E\} \sqsupseteq^\sharp U \text{ and } [Name \Rightarrow X; Sal \Rightarrow Y] \subseteq E \}].$$

Notice that necessary rules can *eliminate* existing possibilities but cannot introduce anything that was not possible before.

The effect of these rules taken together will be the least O_R that is an upper bound in \sqsubseteq^{Ob} for

$$\{f(O_R), f'(O_R), g(O_R), g'(O_R), O\}$$

if it exists. Note that the evidence in the database may contradict the rules – for instance a teacher of CS5 who is not a graduate student would be anomalous. If such an *anomaly* exists any attempt to compute O_R will yield an inadmissible sandwich. We regard the ability to detect anomalies to be major advantage of our system.

However, if no anomalies arise the TF component of this O_R will be the best correct estimate we can find for the target set of teaching fellows with the information given. Note that a partial description for the TF sets may provide an exact answer for some queries – for instance the set of TF's with low grades may be exactly delineated by the estimate.

Clearly the examples given above are quite simple. We will need additional constructs for more interesting examples. For instance suppose we wanted to say “we know that Y loves X *if* we know that X is a child of Y,” within a database O of type

$$[\textit{Persons}: \{ [\textit{Name}: \textit{string}, \textit{Children}: \{ \textit{string} \}, \textit{Loves}: \{ \textit{string} \}] \}].$$

In this situation we need some way to carry over additional information about the parent Y that is not explicitly mentioned in the statement of the rule. To solve this problem we introduce the *tag* notation \rightarrow and express the rule as follows:

$$[\textit{persons} \Rightarrow \{Y \rightarrow [\textit{loves} \Rightarrow \{X\}]\}] - : [\textit{persons} \Rightarrow \{Y \rightarrow [\textit{children} \Rightarrow \{X\}]\}]$$

This can be translated into the following monotone function k over objects

$$\lambda O. [\textit{persons} \Rightarrow \max \{ Y \sqcup^{Ob} [\textit{loves} \Rightarrow L] \mid [\textit{persons} \Rightarrow P] \subseteq O \text{ and } \{Y\} \sqsubseteq^b P \text{ and } [\textit{children} \Rightarrow C] \subseteq Y \text{ and } L = \max \{X \mid \{X\} \sqsubseteq^b C\} \}]$$

Notice that this is an example of a rule for which $k(O) \sqcup^{Ob} O$ does not necessarily yield the fixed point O_k . The analogous necessary rule would be

$$[\textit{persons} \Rightarrow \{Y \rightarrow [\textit{loves} \Rightarrow \{X\}]\}] - : [\textit{persons} \Rightarrow \{Y \rightarrow [\textit{children} \Rightarrow \{X\}]\}]$$

This rule states the implausible assumption that the only things persons might love are their children. The associated monotone function h can be expressed as

$$\lambda O. [\textit{persons} \Rightarrow \min \{ Y \sqcup^{Ob} [\textit{loves} \Rightarrow L] \mid [\textit{persons} \Rightarrow P] \subseteq O \text{ and } \{Y\} \sqsupseteq^\sharp P \text{ and } [\textit{children} \Rightarrow C] \subseteq Y \text{ and } L = \min \{X \mid \{X\} \sqsupseteq^\sharp C\} \}]$$

It may be possible to define recursive types and self referencing complex objects using such a notation (see [13]).

Another important type of rule we have not yet shown is one where a variable occurs twice on the right hand side. For example we may want to insist that a possible grandparent-child relationship must be linked by a possible parent. In an appropriately typed database this rule could be expressed by:

$$[GC \Rightarrow \{ [G \Rightarrow X; C \Rightarrow Y] \}] - : [PC \Rightarrow \{ [P \Rightarrow X; C \Rightarrow Z], [P \Rightarrow Z; C \Rightarrow Y] \}]$$

For necessary rules this notation provides no difficulties and the above example translates into the monotone inference function:

$$\lambda O. [GC \Rightarrow \min \{ [G \Rightarrow X; C \Rightarrow Y] \mid [PC \Rightarrow R] \subseteq O \text{ and } \{E_1, E_2\} \sqsupseteq^{\sharp} R \text{ and} \\ [P \Rightarrow X; C \Rightarrow Z] \subseteq E_1 \text{ and } [P \Rightarrow Z; C \Rightarrow Y] \subseteq E_2 \}]$$

However, consider the corresponding sufficient rule which insists that a known grandparent-child relationship must have a known parent linking them:

$$[GC \Rightarrow \{ [G \Rightarrow X; C \Rightarrow Y] \}] : - [PC \Rightarrow \{ [P \Rightarrow X; C \Rightarrow Z], [P \Rightarrow Z; C \Rightarrow Y] \}]$$

Along with its seemingly meaningful inference function $h'[B]$.

$$\lambda O. [GC \Rightarrow \max \{ [G \Rightarrow X; C \Rightarrow Y] \mid [PC \Rightarrow R] \subseteq O \text{ and } \{E_1, E_2\} \sqsubseteq^b R \text{ and} \\ [P \Rightarrow X; C \Rightarrow Z] \subseteq E_1 \text{ and } [P \Rightarrow Z; C \Rightarrow Y] \subseteq E_2 \}]$$

Although this characterization seems reasonable it turns out that h' produces unacceptable inferences. For example, consider the following databases O and O' , where both contain an exact description for the parent child relationship PC:

$$O = [PC \Rightarrow \{ [P \Rightarrow [First \Rightarrow Fred] ; C \Rightarrow [First \Rightarrow John]], [P \Rightarrow [First \Rightarrow John] ; C \Rightarrow [First \Rightarrow Ella]] \}] \\ O' = [PC \Rightarrow \{ [P \Rightarrow [First \Rightarrow Fred] ; C \Rightarrow [First \Rightarrow John; Last \Rightarrow Smith]], \\ [P \Rightarrow [First \Rightarrow John; Last \Rightarrow Wayne] ; C \Rightarrow [First \Rightarrow Ella]] \}]$$

Under these circumstances the given sufficient rule would derive the following consistent descriptions for GC:

$$h'(O') = h'(O) = [GC \Rightarrow \{ [G \Rightarrow [First \Rightarrow Fred] C \Rightarrow [First \Rightarrow Ella]] \}]$$

Thus to prevent the unification of incompatible values we might propose the function h'' (analogous to the approach taken in [1]) defined as follows:

$$\lambda O. [GC \Rightarrow \max \{ [G \Rightarrow X; C \Rightarrow Y] \mid [PC \Rightarrow R] \subseteq O \text{ and } \{E_1, E_2\} \sqsubseteq^b R \text{ and} \\ [P \Rightarrow X; C \Rightarrow Z_1] \subseteq E_1 \text{ and } [P \Rightarrow Z_2; C \Rightarrow Y] \subseteq E_2 \text{ and } Z = Z_1 \sqcup^{Ob} Z_2 \text{ is defined} \}]$$

But under this definition h'' is not an acceptable inference function since $h''(O) = h'(O)$ and

$$h''(O') = [GC \Rightarrow \{\}]$$

These calculations demonstrate that h' is *not monotone*, since

$$O \sqsubseteq^{Ob} O' \text{ but } h''(O) \not\sqsupseteq^{Ob} h''(O')$$

The way to correct this situation is to insist that sufficient rules only unify *maximal* values. Thus the following definition for h' will be monotone:

$$\lambda O. [GC \Rightarrow \max \{ [G \Rightarrow X; C \Rightarrow Y] \mid [PC \Rightarrow R] \subseteq O \text{ and } \{E_1, E_2\} \sqsubseteq^b R \text{ and} \\ [P \Rightarrow X; C \Rightarrow Z] \subseteq E_1 \text{ and } [P \Rightarrow Z; C \Rightarrow Y] \subseteq E_2 \text{ and } Z \text{ is maximal} \}]$$

And in the problematic example given above we now have $h'(O) = h'(O') = [GC \Rightarrow \{\}]$, which conforms to the monotonicity requirement.

References

- [1] F. Bancilhon and S. Khoshafian, “A Calculus for Complex Objects,” in *PODS*, Mar. 1986.
- [2] S. Abiteboul and R. Hull, “IFO: A formal semantic database model,” in *Proceedings third ACM SIGACT-SIGMOD symposium on Principles of Database systems, Waterloo, Canada*, Apr. 1984.
- [3] A. Roth, H. Korth, and A. Silberschatz, “Extended Algebra and Calculus for \rightarrow 1NF Relational Databases,” Technical Report TR-84-36, Department of Computer Sciences, The University of Texas at Austin, 1984. revised 1985.
- [4] Z. Ozsoyoglu and L. Yuan, “A New Normal Form for Nested Relations,” *ACM transactions on Database Systems*, vol. 12, pp. 111–136, Mar. 1987.
- [5] H. Ait-Kaci, *A Lattice Theoretic Approach to Computation based on Calculus of Partially Ordered Type Structures*. PhD thesis, University of Pennsylvania, 1985.
- [6] W. Rounds and R. Kasper, “A Complete Logical Calculus for Record Structures Representing Linguistic Information,” tech. rep., University of Michigan, Electrical Engineering and Computer Science Department, 1985.
- [7] D. Scott, “Domains for Denotational Semantics,” in *ICALP*, July 1982.
- [8] P. Buneman and A. Ohori, “Using Powerdomains to Generalize Relational Databases,” *Theoretical Computer Science*, To appear.
- [9] M. Smyth, “Power Domains,” *J. Computer and System Sciences*, vol. 16, no. 1, pp. 23–36, 1978.
- [10] T. Imielinski and W. Lipski, “Incomplete Information in Relational Databases,” *Journal of the ACM*, Oct. 1984.
- [11] J. Stoy, *Denotational Semantics: The Scott-Strachey approach to Programming Language Theory*. MIT Press, 1977.
- [12] Winskel, “On Powerdomains and Modalities,” *Theoretical Computer Science*, vol. 36, pp. 127–137, 1985.
- [13] H. Ait-Kaci, “An Algebraic Semantics Approach to the Effective Resolution on Type Equations,” *Theoretical Computer Science*, vol. 45, pp. 293–351, 1986.