A Semantics for Complex Objects and Approximate Answers

Citation for published version:
Buneman, P, Davidson, S & Watters, A 1991, 'A Semantics for Complex Objects and Approximate Answers' Journal of Computer and System Sciences, vol. 43, no. 1, pp. 170-218. DOI: 10.1016/0022-0000(91)90035-4

Digital Object Identifier (DOI):
10.1016/0022-0000(91)90035-4

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Journal of Computer and System Sciences

JOURNAL OF COMPUTER AND SYSTEM SCIENCES 43, 170–218 (1991)

A Semantics for Complex Objects and Approximate Answers*

O. P. BUNEMAN, S. B. DAVIDSON, AND A. WATTERS

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104

Received November 29, 1988; revised February 16, 1990

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 (respectively represented in the Smyth and Hoare powerdomains), allowing the maximal values to be arbitrary subsets of maximal elements in the domain of the space of descriptions. We then restrict our attention to complex objects which are in some sense "natural," i.e., those which represent "views" of entity–relationship databases, and define rules over these objects. The rules can be used not only as an integrity check on the information in the database, but can be used constructively to infer consistent instances of conclusions and to refine complete instances of the hypothesis. The system is shown to extend the power of datalog (without negation) and the relational algebra (with set difference), and to have an efficient implementation. © 1991 Academic Press, Inc.

1. INTRODUCTION

The distinguishing property of complex-object [1, 2] databases and higher-order relations [3–5] 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 [6–8]. For example, in Bancilhon and Khoshafian's ordering on tuples [2]

$$[\mathit{Name} \Rightarrow \text{'}J.Doe\text{'}] \sqsubseteq [\mathit{Name} \Rightarrow \text{'}J.Doe\text{'}; \mathit{Age} \Rightarrow 21]$$

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

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

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

Since the domain of values can also contain sets, we need to extend this ordering to sets of values. In [2], two subsets A and B of the same domain are ordered by

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

and this ordering is inductively extended to order complex objects, which are hierarchical structures containing both tuples and sets. For example, if

$$A = \{[\mathit{Name} \Rightarrow \text{'}\mathit{J.Doe}\text{'}]\}$$

and

$$B = \{[\mathit{Name} \Rightarrow \text{'}\mathit{J.Doe}\text{'}; \mathit{Age} \Rightarrow 21], [\mathit{Name} \Rightarrow \text{'}\mathit{A.Putnam}\text{'}]\}$$

then $A \sqsubseteq^{\flat} B$. Not only is every tuple in A no worse than some tuple in B, but B contains an additional tuple not appearing in A.

In contrast, in an attempt to find a data type for natural join, [9] uses the ordering

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

For example, if

$$A = \{[\mathit{Name} \Rightarrow \text{'}\mathit{J.Doe}\text{'}], [\mathit{Name} \Rightarrow \text{'}\mathit{A.Putnam}\text{'}]\}$$

and

$$B = \{[\mathit{Name} \Rightarrow \text{'}\mathit{J.Doe}\text{'}; \mathit{Age} \Rightarrow 21]\}$$

then $A \sqsubseteq^{\sharp} B$. Not only is every tuple in B no worse than some tuple in A (or remains the same), but a tuple in A has been eliminated in B.

Both of these orderings are well known in the study of the semantics of concurrency and non-determinism; $\sqsubseteq^{\sharp}$ and $\sqsubseteq^{\flat}$ are respectively called the Smyth and Hoare orderings [10]. Observe that the maximal elements of $\sqsubseteq^{\sharp}$ and $\sqsubseteq^{\flat}$ are respectively the empty set ($\{\ \}$) and the set of all maximal elements of $\mathscr{D}$.

However, if we want to approximate *sets*, neither the Hoare nor the Smyth ordering is, in isolation, satisfactory since their maximal elements are uninteresting. Reference [11], in formalizing incomplete information, describes the semantics of a tuple such as $[\mathit{Name} \Rightarrow \text{'}\mathit{J.Doe}\text{'}; \mathit{Age} \Rightarrow \_]$ as $\{[\mathit{Name} \Rightarrow \text{'}\mathit{J.Doe}\text{'}; \mathit{Age} \Rightarrow i] \mid i \in I\}$, where I is the set of all possible (total) values of *Age*. More generally, if $\mathscr{D}$ is a partially ordered space, we can define the denotation of a tuple x, $[\![x]\!]$, as $\{y \mid y \text{ is maximal in } \mathscr{D} \text{ and } y \geqslant x\}$. If we extend this simple-minded notion of semantics to tuples involving sets of values, the denotation of the tuple $[\mathit{Name} \Rightarrow \text{'}\mathit{J.Doe}\text{'}; \mathit{Children} \Rightarrow \{\text{'}\mathit{John}\text{'}, \text{'}\mathit{Mike}\text{'}\}]$ would, using the Hoare ordering, be a tuple with all possible children since adding entries improves the set, and, using the Smyth ordering, be a tuple with no children since deleting entries improves the set. Ideally, we would like to be able to say that this is *exactly* the set of children for *J.Doe*, and that it cannot be improved by either adding or deleting entries.

We therefore use the Hoare and Smyth orderings *together* to represent complete and consistent information about—or to "sandwich"—subsets of the maximal elements of $\mathscr{D}$. Sandwiches are then pairs of co-chains in $\mathscr{D}$, and are ordered by how well they describe subsets of the maximal elements of $\mathscr{D}$; i.e., a better sandwich describes fewer subsets. It is shown that this forms a domain, in which the maximal elements are exactly what we need: *subsets* of the maximal elements of $\mathscr{D}$. A similar use of complete information appears in [12, 13].

Although this space of records and sandwiches is rich enough to describe arbitrary sets and recursive record structures, we limit our attention to a "natural" subset in which a notion of rules can be precisely defined. That is, we believe that complex objects are frequently used as "views" of much simpler implementation structures, such as entity–relationship databases. For example, while an organization might view "departments" as having a name (*Dname*) and an associated set of employees (*Emps*):

$$\textit{Department}: \{[\textit{DName}: \textit{string}; \textit{Emps}: \{[\textit{FName}: \textit{string}; \textit{LName}: \textit{string}]\}]\},$$

this is just a convenient representation of a project-join view of the traditional database,

$$\textit{Employee}: \{[\textit{Id}: \textit{e-obj-id}; \textit{FName}: \textit{string}; \textit{LName}: \textit{string}]\};$$

$$\textit{Department}: \{[\textit{Code}: \textit{d-obj-id}; \textit{DName}: \textit{string}]\};$$

$$\textit{Emp\_Dept}: \{[\textit{Emp}: \textit{e-obj-id}; \textit{Dept}: \textit{d-obj-id}]\}$$

in which all members of the same department are collected into a set. Using this convention, we define rules of the form

$$p \text{ implies } q_1 \text{ or } q_2 \text{ or} \cdots \text{or } q_k,$$

where p and the q_i's are restricted to be complex objects which are views of the same entity–relationship database. The rules can be used in several ways: as an integrity check on the information in the database; as a means of improving the existing descriptions stored in the database; and as a means of appending new "views" on the database, i.e., for defining queries. That is, the interpretation given to our rules over sandwiches will constructively infer consistent instances of conclusions, q_i, as well as refine complete instances of the hypothesis, p. The system is shown to extend the power of Datalog (without negation) and the relational algebra (with set difference), and to have an efficient implementation.

The rest of this paper is organized as follows. Section 2 formally defines the domain of sandwiches. In Section 3, we give a semantics of complex objects as partial descriptions, and show how they can be used to approximate entity–relationship databases. Section 4 presents the general form of a rule and defines the interpretations applied: integrity check, consistent inference, and complete inference. We conclude in Section 5 with a discussion of the expressive power of our language and the complexity of its implementation.

2. The Sandwich Ordering

We would like the spaces we are using to be rich enough to describe recursive record structures. One choice, adopted in programming language theory, is to take our domains to be Scott domains [8]. For our purposes, the important property is that they are *bounded complete* partial orders. Any non-empty subset S of $\mathscr{D}$ has a greatest lower bound, $\bigsqcap S$. A more conservative approach [14] is to model recursive structures built from records and sets as regular trees [15]. In this paper, our examples are drawn from finite, non-recursive record structures.

Given a domain $\mathscr{D}$, let $\mathscr{C}(\mathscr{D})$ denote the finite co-chains on $\mathscr{D}$. The Smyth ($\sqsubseteq^{\sharp}$) and Hoare ($\sqsubseteq^{\flat}$) orderings, when applied to arbitrary subsets of $\mathscr{D}$, are only pre-orders. However, if we restrict our attention to the finite co-chains on $\mathscr{D}$ they are both partial orders with computable meets and joins. A co-chain is a subset of D with the property that any two elements are incomparable; i.e., S is a cochain if $x, y \in S$ and $x \sqsupseteq y$ imply $x = y$. The meets and joins in the two orderings are defined by

$$A_1 \sqcup^{\sharp} A_2 = \min\{a_1 \sqcup a_2 \mid a_1 \in A_1, a_2 \in A_2\}$$

$$A_1 \sqcap^{\sharp} A_2 = \min(A_1 \cup A_2)$$

$$A_1 \sqcap^{\flat} A_2 = \max\{a_1 \sqcap a_2 \mid a_1 \in A_1, a_2 \in A_2\}$$

$$A_1 \sqcup^{\flat} A_2 = \max(A_1 \cup A_2).$$

As an example, suppose A_1 and A_2 are the following relations:

$A_1 =$

Name	*Course*
Jones	CS100
Jones	Phil5
Smith	Phil5

$A_2 =$

Course	*Instructor*
Phil5	Dewey
Phil5	Cheatham
Math2	Howe

If we take these to be sets of records, and use the ordering on records described in the introduction, we have

$A_1 \sqcup^{\sharp} A_2 =$

Name	*Course*	*Instructor*
Jones	Phil5	Dewey
Smith	Phil5	Dewey
Jones	Phil5	Cheatham
Smith	Phil5	Cheatham

$A_1 \sqcup' A_2 =$

Name	*Course*	*Instructor*
Jones	CS100	—
Jones	Phil5	—
Smith	Phil5	—
—	Phil5	Dewey
—	Phil5	Cheatham
—	Math2	Howe

Note that $\sqcup^{\sharp}$ when applied to these relations gives the natural join. For a fuller discussion of this, see [16], where the following simple result is given.

PROPOSITION 1. *$(\mathscr{C}(\mathscr{D}), \sqsubseteq^{\sharp})$ and $(\mathscr{C}(\mathscr{D}), \sqsubseteq^{\flat})$ are distributive lattices (with top and bottom elements).*

To use these two orderings together, we define a *sandwich* in $\mathscr{D}$ to be a pair (A, B) with $A, B \in \mathscr{C}(\mathscr{D})$ such that $\exists S \subseteq \mathscr{D}. A \sqsubseteq^{\sharp} S$ and $B \sqsubseteq^{\flat} S$.

Let $\mathscr{S}(\mathscr{D})$ denote the sandwiches on $\mathscr{D}$. We can define an ordering on $\mathscr{S}(\mathscr{D})$ by

$$(A_1, B_1) \sqsubseteq^S (A_2, B_2) \qquad \text{iff} \quad A_1 \sqsubseteq^{\sharp} A_2 \text{ and } B_1 \sqsubseteq^{\flat} B_2.$$

PROPOSITION 2. *$(\mathscr{S}(\mathscr{D}), \sqsubseteq^S)$ is a distributive semilattice with a bottom element and pairwise bounded joins.*

Proof. The meet of (A_1, B_1) and (A_2, B_2) is $(A_1 \sqcap^{\sharp} A_2, B_1 \sqcap^{\flat} B_2)$. The bottom element is $(\{\bot\}, \varnothing\}$. To show that $(\mathscr{S}(\mathscr{D}), \sqsubseteq^S)$ has pairwise bounded joins, suppose that the sandwiches (A_1, B_1) and (A_2, B_2) are bounded above by a sandwich (A', B'). By definition there must be $S \in \mathscr{C}(\mathscr{D})$ such that $A' \sqsubseteq^{\sharp} S$ and $B' \sqsubseteq^{\flat} S'$. By transitivity, (S', S') must be a bound for $(A_1 \sqcup^{\sharp} A_2, B_1 \sqcup^{\flat} B_2)$. ▌

Thus the sandwich join of (A_1, B_1) and (A_2, B_2) is, when it exists, $(A_1 \sqcup^{\sharp} A_2, B_1 \sqcup^{\flat} B_2)$. However, note that if (C_1, C_1) and (C_2, C_2) are both "exact" sandwiches (i.e., each of C_1 and C_2 provides complete and consistent information about some set), their sandwich join (if it exists) will not, in general, be exact.

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

PROPOSITION 3. *The maximal elements of $\mathscr{S}(\mathscr{D})$ are pairs (T, T), where $T \subseteq \text{Tot}(\mathscr{D})$.*

Proof. If there is a sandwich (A, B) that dominates (T, T), where $T \subseteq \text{Tot}(\mathscr{D})$ then there must be a sandwich (T', T') that dominates (A, B) and hence (T, T). Since T contains only maximal elements we must have $T' \subseteq T$ and $T' \supseteq T$. Hence (T, T) is maximal in $\mathscr{S}(\mathscr{D})$. Conversely, any maximal sandwich of $\mathscr{S}(\mathscr{D})$ must be a pair of the form (T, T) and if T contains some non-maximal element of $\mathscr{D}$ we can replace it by a larger element to construct a pair (T', T') that is strictly greater than (T, T). Hence if (T, T) is maximal in $\mathscr{S}(\mathscr{D})$, T can only contain maximal elements of $\mathscr{D}$. ▌

From our previous definition of meaning, the denotation of a sandwich is given by $[\![(A, B)]\!] \equiv \{(T, T) \mid T \in \text{Tot}(\mathscr{D}), A \sqsubseteq^{\sharp} T \text{ and } B \sqsubseteq^{\flat} T\}$. We can think of A and B as being *complete* and *consistent* information about some $T \in \text{Tot}(\mathscr{D})$. When $(T, T) \in [\![(A, B)]\!]$, we say that (A, B) *approximates* T.

To illustrate the use of these ideas, 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;

Last Name
Johnson
Pierce
Taylor
Cooper
Emerson
Billings

where the absence of a column (in this case, the First name column) indicates that the values are null. This convention is used throughout the rest of the paper. Note that this is an over-approximation of T, because we cannot assume that people who register for Database 1 are 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:

First Name	*Last Name*
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:

First Name	*Last Name*
Ella	Taylor
Burt	Cooper
Liza	Pierce
Burt	Pierce
Elvira	Johnson

and

First Name	*Last Name*
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:

First Name	*Last Name*
Ella	Taylor
Burt	Elliot
Larry	Pierce

Turning back to our technical study of sandwiches, the domain of sandwiches lacks a property that is enjoyed by the domain of partial tuples: two elements of $\mathscr{S}(\mathscr{D})$ may denote the same subset of $\mathrm{Tot}(\mathscr{S}(\mathscr{D}))$. We would like to be able to "promote" a sandwich to the largest element in the domain that describes the same set of maximal elements. Unfortunately, it is both difficult to define and to compute a promoted sandwich with this property. Consider the following sandwich (complete information is on the left)

First Name	*Last Name*	*Last Name*
Ella	Taylor	Taylor
—	Pierce	Pierce

A promoted sandwich should contain, on the complete side, the tuple (n, Pierce) for every n in the domain of first names. If this domain is infinite, the complete side will contain an infinite cochain—violating our definition of a sandwich. If the domain is finite, we still need knowledge of the entire domain, which is not in general available. We therefore consider a weaker promotion operator P, which we show to be the best that one can obtain *on the available evidence*. For a sandwich (A, B), this is defined [17] by

$$P(A, B) = \left(A, \left\{\bigsqcap \{a \sqcup b \mid a \in A \text{ and } a \sqcup b \text{ exists}\} \mid b \in B\right\}\right).$$

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. Thus $P(A, B)$ is always well-defined.

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

First Name	*Last Name*	*First Name*	*Last Name*
Ella	Taylor	—	Taylor
Burt	Cooper	Burt	Johnson
—	Pierce	Fred	—
—	Johnson	—	Cooper

then the following sandwich, promoted by P, will also approximate T:

First Name	*Last Name*	*First Name*	*Last Name*
Ella	Taylor	Ella	Taylor
Burt	Cooper	Burt	Johnson
—	Pierce	Fred	—
—	Johnson	Burt	Cooper

Note that P only required us to take meets and joins of tuples in the given sandwich; it did not require further knowledge of the domain. To show that P is the best one can do, given a sandwich (A, B) in $\mathscr{S}(\mathscr{D})$ consider the subset of $\mathscr{D}$ that one can compute by joining together partial information in $A \cup B$. This is a downward closed subset of $\mathscr{D}$ that is closed under joins that exist in $\mathscr{D}$. Such a subset is called a *strong ideal* of $\mathscr{D}$; and we define $\mathscr{I}(A, B)$ to be the smallest strong ideal containing $A \cup B$. Given some other domain $\mathscr{D}'$ we say $\mathscr{I}(A, B)$ can be *embedded* in $\mathscr{D}'$ if it can be embedded as a strong ideal. What we now show is that (A, B) and $P(A, B)$ denote the same sets of maximal elements for any domain in which $\mathscr{I}(A, B)$ is embedded, but that no sandwich greater than $P(A, B)$ can have the same denotation in all such domains.

PROPOSITION 4. *For any domain $\mathscr{D}$ in which $\mathscr{I}(A, B)$ is embedded, $[\![(P(A, B))]\!]_{\mathscr{D}} = [\![(A, B)]\!]_{\mathscr{D}}$. Moreover if s is a sandwich in $\mathscr{I}(A, B)$ such that $s \sqsupset^{S} P(A, B)$ then there is a domain $\mathscr{D}'$ in which $\mathscr{I}(A, B)$ can be embedded such that $[\![(P(A, B))]\!]_{\mathscr{D}'} \neq [\![s]\!]_{\mathscr{D}'}$.*

Proof. The first part of this result is immediate from the definition of P. For the second part, suppose $s = (A', B')$ is above the promoted sandwich $(A, B^P) = P(A, B)$. For each point in $\mathscr{I}(A, B)$ of the form $a \sqcup b$, where $a \in A$, $b \in B^P$ and the join is defined, augment $\mathscr{I}(A, B)$ with a point $t_{a \sqcup b}$. Extend the ordering such that $t_{a \sqcup b}$ dominates only $a \sqcup b$ and the elements below it.

Suppose $B' \sqsupset^{\flat} B^P$. Choose any $b' \in B'$ where for each $b \in B^P$, $b \not\sqsupseteq b'$. In this case for every $b \in B^P$ there must be some $a \in A$ where $a \sqcup b \not\sqsupseteq b'$. Letting $T' = \{t_{a \sqcup b} \mid a \in A, b \in B^P, a \sqcup b \not\sqsupseteq b'\}$ we have $(T', T') \sqsupseteq^{S} (A, B^P)$ but that $T' \not\sqsupseteq^{\flat} B'$ as required.

On the other hand, suppose that $A' \sqsupseteq^{\sharp} A$. In this case there must be some $a \in A$ not dominating any $a' \in A'$. As before, we introduce t_a dominating only a and the elements below it, and take T as before. It follows that $T \cup t_a$ is a set of maximal elements approximated by (A, B) but not by (A', B'). ∎

We complete this section with two results that are used later to show that the computations we perform by applying "rules" to a database have well-defined outcomes. A *closure* c is a monotone $(x \sqsupseteq y \Rightarrow c(x) \sqsupseteq c(y))$, inflationary $(c(x) \sqsupseteq x)$ and idempotent $(c(c(x)) = c(x))$ function on a lattice. An important property of a closure defined on a semilattice with pairwise bounded joins is

PROPOSITION 5. *If c is a closure then $x \sqcup y$ exists iff $c(x) \sqcup y$ exists, and $c(c(x) \sqcup y) = c(x \sqcup y)$ if $x \sqcup y$ exists.*

Proof. The first part is proved by noting that $c(x \sqcup y)$ is a bound for $c(x)$ and y, provided $x \sqcup y$ exists. For the second part, since c is inflationary we have

$c(x) \sqsupseteq x$, and by monotonicity of c we have $c(c(x) \sqcup y) \sqsupseteq c(x \sqcup y)$. To establish the reverse inequality we have

$$\begin{aligned} c(c(x) \sqcup y) &\sqsubseteq c(c(x \sqcup y) \sqcup y) && \text{by monotonicity of } c \\ &= c(c(x \sqcup y)) && \text{because } c \text{ is inflationary} \\ &= c(x \sqcup y) && \text{by idempotence.} \end{aligned}$$ ∎

Now it is readily established that P is a closure on the lattice of sandwiches. This property of closures tells us that, in performing a sequence of joins, we need only apply the promotion to the final result (though intermediate promotions may improve the efficiency).

A second result concerning closures is crucial to understanding the action of rules on sandwiches. We need to be sure that, regardless of the order in which the rules are applied, we get a well-defined result (provided they are applied often enough). The same idea is used [18] to provide a semantics for dataflow computation.

PROPOSITION 6. *If $c_1, c_2, ..., c_n$ are closures on a finite lattice, and if there is a common fixed point of these closures, i.e., a point x such that $c_i(x) = x$ for $i \in 1 \cdots n$, then there is a unique minimal point with this property.*

Proof. To obtain this result, we only need to assume that the c_i are monotone and inflationary. Let S_i be any set of fixed points of c_i. By monotonicity, $c_i(\sqcap S_i) \sqsubseteq \{\sqcap c_i(x) \mid x \in S_i\} = \sqcap S_i$ and, since c_i is inflationary, $c_i(\sqcap S_i) \sqsupseteq \sqcap S_i$. Thus the fixed points of each c_i form a meet-closed subset of $\mathscr{D}$. The intersection of the fixed points of c_i is therefore meet-closed and has a minimal element. ∎

We are, in practice, able to guarantee that our rules operate within finite lattices. This follows from the fact that for record structures, the ideals $\mathscr{I}(A, B)$ generated by a sandwich are always finite.

3. Data Models and Complex Objects

In order to give a formal account of the semantics of complex objects as partial descriptions, we need to produce a "real world" which is the space of things described by complex objects. Our strategy is to use a version of the entity–relationship (E-R) model [19] as the real world and complex objects as the space of descriptions. Other approaches are possible, such as those presented in [20, 21]. In addition to making our notion of semantics precise, this approach allows us to formulate rules over complex objects. Moreover, certain types of complex objects, which we believe to be "unnatural" even though they are allowed by the syntax (e.g., sets of sets of integers), are excluded because there is no E-R database that they represent.

After giving a syntax for complex objects and restricting them to be "views" of E-R databases, we show how they represent E-R instances. To lay the groundwork for defining rules over complex objects, we close this section by discussing in what sense a term of a rule can be said to be satisfied by a complex object.

3.1. *Complex-Object Types and Instances*

We first describe a syntax for "exact" complex objects; these do not contain missing values (within a given type), and they contain only "exact sandwiches." They thus conform to the normal definition of complex objects and the syntax for their types and instances follows roughly that given in [1, 4].

A complex object type is defined as follows.

(a) Base types such as *int* (integer), *string* (character string), and *bool* (boolean) are types.

(b) If $\tau_1, \tau_2, ..., \tau_n$ are types and $l_1, l_2, ..., l_n \in \mathscr{L}$ then $[l_1:\tau_1; l_2:\tau_2; ...; l_n:\tau_n]$ is a type. Such types are *tuple* types.

(c) If τ is a type, $\{\tau\}$ is a type. These are *set* types.

The syntax for typed values follows that of types:

(a) Atomic values such as 1, 2, 3, ... are values of type *int*; *'cat'*, *'dog'*, ... are values of type *string*; etc.

(b) If $v_1:\tau_1, v_2:\tau_2, ..., v_n:\tau_n$ and $l_1, l_2, ..., l_n \in \mathscr{L}$ then $[l_1 \Rightarrow v_1, l_2 \Rightarrow v_2, ..., l_n \Rightarrow v_n]$ is a value of type $[l_1:\tau_1; l_2:\tau_2; ...; l_n:\tau_n]$.

(c) If $v_1, v_2, ..., v_n$ are all of type τ then $\{v_1, v_2, ..., v_n\}$ is of type $\{\tau\}$.

For example,

$$[\textit{Persons}\colon \{[\textit{Fn}\colon \textit{string};\ \textit{Ln}\colon \textit{string};\ \textit{Children}\colon \{\textit{string}\}]\}];$$

is a type and

$$\begin{aligned}[&\textit{Persons} \Rightarrow \{[\textit{Fn} \Rightarrow \textit{'John'};\ \textit{Ln} \Rightarrow \textit{'Doe'};\ \textit{Children} \Rightarrow \{\textit{'Sally'}, \textit{'Sue'}\}];\\ &\qquad [\textit{Fn} \Rightarrow \textit{'Mary'};\ \textit{Ln} \Rightarrow \textit{'Brown'};\ \textit{Children} \Rightarrow \{\textit{'Peter'}, \textit{'James'}\}]\}]\end{aligned}$$

is an object of that type.

3.2. *The Entity–Relationship Model*

The authors are unaware of any especially clean formulation of the E-R model, though the process of mapping this model to the relational model [22] is reasonably well understood. We treat an E-R schema as a collection of named relations partitioned into two sets: *entities*, $\mathscr{E}$ and *relationships*, $\mathscr{R}$. We use $\mathscr{L}(Q)$ for the attribute names of an entity or relationship $Q \in \mathscr{E} \cup \mathscr{R}$, and we assume that for each $Q \in \mathscr{E} \cup \mathscr{R}$ there is a function Θ_Q which maps $L(Q)$ into the set of base types. The base types are the domains of relational database terminology and are usually left out of E-R diagrams. For example, in Fig. 1, typical values would be $\Theta_{Employee}(FName) = string$, and $\Theta_{Emp\text{-}Dept}(Contract) = int$.

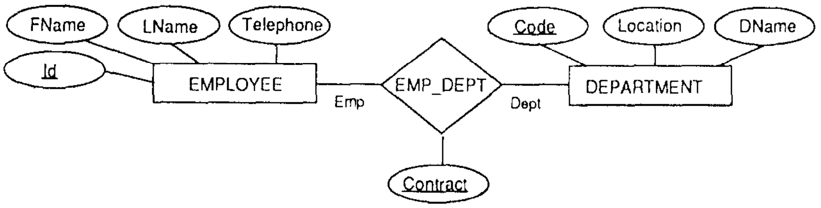

FIG. 1. E-R diagram for employee–department database.

So far this is just a formulation of the relational model. We now add some information that defines the graphical structure of an E-R diagram.

- For each entity $E \in \mathscr{E}$ there is a distinguished attribute $I_E \in \mathscr{L}(E)$ called its *identity*. It is assumed that I_E is a key for E.
- For each relationship $R \in \mathscr{R}$ there is a partial function $\rho_R : \mathscr{L}(R) \rightarrow \mathscr{E}$ such that $\Theta_R(I_{\rho_R(l)}) = \Theta_R(l)$ whenever $\rho_R(l)$ is defined.

The function ρ specifies the connection between relationships and entities. To illustrate the identity and function ρ using Fig. 1, $I_{Employee} = Id$, $\rho_{Emp\_Dept}(Emp) = Employee$, and $\Theta_{Emp\_Dept}(Emp) = int$, which is the same as $\Theta_{Employee}(Id)$. This definition excludes the possibility of entities being attributes of other entities, or relationships being attributes of other relationships. We have made this restriction for simplicity, although in general we see no problem in allowing these possibilities. Also for simplicity, we have limited the identity of an entity to being a single attribute and, whether or not it is generated by the system or by the user, we insist that exactly one such attribute is present for each entity. We have no explicit mention of "isa" relationships; these are represented when we deal with rules.

An *instance* of an E-R schema is simply a relational instance of $\mathscr{E} \cup \mathscr{R}$ that satisfies inclusion dependencies specified by the function ρ; namely if r is a tuple in $R \in \mathscr{R}$ and $\rho_R(l) = E$ then there is a tuple e in $\mathscr{E}$ such that $r.l = e.I_E$.

For example, the schema S for the E-R diagram shown in Fig. 1 is given by

$\{$*Employee*: $\{[$*Id*: *e-id-type*; *FName*: *string*; *LName*: *string*; *Telephone*: *integer*$]\}$;
Department: $\{[$*Code*: *d-id-type*; *DName*: *string*; *Location*: *string*$]\}$;
Emp_Dept: $\{[$*Emp*: *e-obj-id*; *Dept*: *d-obj-id*; *Contract*: *ed-id-type*$]\}\}$.

Here the entities are $\{$*Employee*, *Department*$\}$ where

$$I_{Employee} = Id \quad \text{and} \quad I_{Department} = Code.$$

The relationship is *Emp_Dept* where

$$\mathrm{dom}(\rho_{Emp\_Dept}) = \{Emp, Dept\},$$

$$\rho_{Emp\_Dept}(Emp) = Employee,$$

and

$$\rho_{Emp\_Dept}(Dept) = Department.$$

Note that *Emp_Dept* also has an identity *Contract*, since it is not ruled out by the definition.

A relational instance of this example could be

Employee

Id	*FName*	*LName*	*Telephone*
1	Ella	Taylor	8986570
2	Burt	Cooper	3876593

Department

Code	*DName*	*Location*
52	Systems	To3
53	Physics	Whl

Emp_Dept

Emp	*Dept*	*Contract*
1	52	998
1	53	75
2	53	77

3.3. *Complex Objects as Views of E-R Databases*

We can think of a complex object as view of an E-R instance in which we may describe more about a tuple by describing its relationships to other tuples in the database; in particular, we may lose the identities of entities. In this sense, we can regard complex objects as *views* of an E-R schema. The way we build up such views is to treat each entity as a record type, some of whose attributes are set types; we may also treat relationships as record types whose entity attributes are record types. For example, given an *Employee* we can construct an attribute *Emp_Dept @ Emp* as a set type that contains records derived from the *Emp_Dept* relationship.

Given an E-R schema, we generate a complex-object view of the database by first specifying a *type* that is consistent with that schema. We therefore define a relationship *con* as follows:

- Any base type (e.g., *int*, *string*, *bool*) in a complex object type is consistent with the same type in an E-R schema; i.e., τ *con* τ if τ is base.

- For an entity E, $[l_1:\tau_1; l_2:\tau_2; \ldots; l_n:\tau_n]$ *con* E iff for each l_i, one of the following holds

  — $l_i \in \mathscr{L}(E)$ and τ_i *con* $\Theta_E(l_i)$. The associated types (which must be base) agree.

  — l_i is of the form $R@l$ where E is an entity such that $\rho_R(l) = E$ and τ_i is of the form $\{\sigma\}$ where σ *con* R.

- For a relationship R, $[l_1:\tau_1; l_2:\tau_2; ...; l_n:\tau_n]$ *con* R iff for each l_i one of the following holds

— $l_i \in (\mathscr{L}(R) - \text{dom}(\rho_R))$ and τ_i *con* $\Theta_R(l_i)$. I.e. l_i is an ordinary attribute of R so that τ_i is base.

— $\rho_R(l_i) = E$ and τ_i *con* E.

Finally, given an E-R schema $S = \mathscr{E} \cup \mathscr{R}$ and a complex object type $\tau = [l_1:\tau_1; l_2:\tau_2; ...; l_n:\tau_n]$, τ *con* S iff, for each l_i, $l_i = Q$, where $Q \in S$ and $\tau_i = \{\sigma\}$, where σ *con* Q. Note that we are again deliberately confusing the names of entities with the entities themselves and similarly for relationships.

To illustrate these ideas, we give three examples of complex object types that are consistent with the previous database. V_1 defines a view in which the *Emp_Dept* relation contains attributes from the *Employee* and *Department* relations:

V_1: [*Emp_Dept*: {[*Emp*: [*FName*: *string*; *LName*: *string*; *Telephone*: *int*];
Dept: [*DName*: *string*; *Location*: *string*]]}].

V_2 defines a view in which the *Employee* records contain information about the set of departments the employee is in:

V_2: [*Employee*: {[*FName*: *string*; *LName*: *string*;
Emp_Dept@*Emp*: {[*Dept*: [*Dname*: *string*]]}]}].

V_3 defines a view in which the *Employee* record contains not only information about the set of departments the employee is in, but also information about the set of co-workers in each department:

V_3: [*Employee*: {[*FName*: *string*; *LName*: *string*; *Emp_Dept*@*Emp*:
{[*Dept*: [*Dname*: *string*; *Emp_Dept*@*Dept*:
{[*Emp*: [*FName*: *string*; *LName*: *string*]]}]]}]}].

Note that a complex object type is always, at the top level, a record of sets and that the only way sets can appear at lower levels is through relationships in the E-R schema. For example, the complex object type [*Persons*: {[*Fn*: *string*; *Ln*: *string*; *Children*: {*string*}]}] is not consistent with any E-R diagram, because {*string*} could not be an attribute of any entity or relationship.

PROPOSITION 7. *If S is an E-R schema (or a component of an E-R schema) and τ_1, τ_2 are complex-object types such that τ_1 con S and τ_2 con S, then $\tau_1 \sqcup \tau_2$ con S.*

The definition of $\bigsqcup$ for types is essentially the same as the definition of the join for nested records, and is therefore omitted.

Given an E-R schema S, an instance I of S, and a complex-object type τ consistent with S, we can now define the complex object view associated with that type. The construction of the view follows the definition of consistency.

If $Q \in \mathscr{E} \cup \mathscr{R}$ is an entity or relationship, t is a tuple in Q and $\tau = [l_1 : \tau_1 ; l_2 : \tau_2 ; \ldots ; l_n : \tau_n]$ is a complex object type such that τ *con* Q, the complex object view $V(Q, t, \tau)$ is a complex object $[l_1 \Rightarrow v_1 ; l_2 \Rightarrow v_2 ; \ldots ; l_n \Rightarrow v_n]$ defined as follows:

- $Q \in \mathscr{E}$: If l_i is of the form $R@l$, then $v_i = \{ V(R, t', \sigma) \mid t'.l = t.I_Q \}$; otherwise $v_i = t.l_i$.
- $Q \in \mathscr{R}$: If $l_i \in \mathrm{dom}(\rho_R)$ and $\rho_R(l_i) = E$, then $v_i = V(E, t', \tau_i)$ where t' is the (unique) tuple in E such that $t.l_i = t'.I_E$; otherwise $v_i = t.l_i$.

Finally, given an instance of an E-R schema $S = \mathscr{E} \cup \mathscr{R}$ and a complex-object type $[l_1 : \{\sigma_1\} ; l_2 : \{\sigma_2\} ; \ldots ; l_n : \{\sigma_n\}]$, the view generated by that type is the complex object $[l_1 \Rightarrow v_1 ; l_2 \Rightarrow v_2 ; \ldots ; l_n \Rightarrow v_n]$, where $v_i = \{ V(l_i, t, \tau_i) \mid t \in \mathit{Inst}(l_i) \}$. Here $\mathit{Inst}(l_i)$ is the relational instance of the entity or relationship named l_i.

For example, using the relational instance of *Employee*, *Department and Emp_Dept* given earlier, the view generated by V_1 would be:

Emp_Dept

Emp			*Dept*	
FName	*LName*	*Telephone*	*DName*	*Location*
Ella	Taylor	8986570	Systems	To3
Ella	Taylor	8986570	Physics	Wh1
Burt	Cooper	3876593	Physics	Wh1

which could also be represented as

$$\begin{aligned} V_1 \Rightarrow [&\mathit{Emp\_Dept} \Rightarrow \{[\mathit{Emp} \Rightarrow [\mathit{FName} \Rightarrow \text{'Ella'}; \mathit{LName} \Rightarrow \text{'Taylor'};\\ &\mathit{Telephone} \Rightarrow \text{'8986570'}]; \mathit{Dept} \Rightarrow [\mathit{DName} \Rightarrow \text{'Systems'}; \mathit{Location} \Rightarrow \text{'To3'}]],\\ &[\mathit{Emp} \Rightarrow [\mathit{FName} \Rightarrow \text{'Ella'}; \mathit{LName} \Rightarrow \text{'Taylor'}; \mathit{Telephone} \Rightarrow \text{'8986570'}];\\ &\mathit{Dept} \Rightarrow [\mathit{DName} \Rightarrow \text{'Physics'}; \mathit{Location} \Rightarrow \text{'Wh1'}]],\\ &[\mathit{Emp} \Rightarrow [\mathit{FName} \Rightarrow \text{'Burt'}; \mathit{LName} \Rightarrow \text{'Cooper'}; \mathit{Telephone} \Rightarrow \text{'3876593'}];\\ &\mathit{Dept} \Rightarrow [\mathit{DName} \Rightarrow \text{'Physics'}; \mathit{Location} \Rightarrow \text{'Wh1'}]]\}]. \end{aligned}$$

The relational instance of the view generated by V_2 would be:

Employee

FName	*LName*	*Emp_Dept @ Emp Dept DName*
Ella	Taylor	Systems Physics
Burt	Cooper	Physics

Finally, the relational instance of the view generated by V_3 would be:

Employee				
		Emp_Dept @ Emp Dept		
			Emp_Dept @ Dept Emp	
FName	*LName*	*DName*	*FName*	*LName*
Ella	Taylor	Systems	Ella	Taylor
		Physics	Ella	Taylor
			Burt	Cooper
Burt	Cooper	Physics	Ella	Taylor
			Burt	Cooper

Returning to our technical discussion, given two such views with joinable types, it is unclear whether there exists a well-defined join of the views which represent the same E-R instance. It is the need to find a domain in which we can define joinable objects (and hence an algebra) that motivates our definition of *partial objects.*

Partial objects are complex objects in which sets are replaced by sandwiches. Given a complex object type τ, partial objects of type τ are constructed as follows:

(a) Atomic values are defined as before.

(b) Let $\tau = [l_1 : \tau_1; l_2 : \tau_2; ...; l_n : \tau_n]$, and v_i be a partial object of type τ_i for $1 \leqslant i \leqslant m$, $m \leqslant n$. Then $[l_1 \Rightarrow v_1, l_2 \Rightarrow v_2, ..., l_m \Rightarrow v_m]$ is a partial object of type τ.

(c) If $\tau = \{\sigma\}$ and $v_1, v_2, ..., v_n, w_1, w_2, ..., w_m$ are all partial objects of type σ, then $(\{v_1, v_2, ..., v_n\}, \{w_1, w_2, ..., w_m\})$ and $(\_, \{w_1, w_2, ..., w_m\})$ are partial objects of type τ.

Rule (b) says that we can drop fields from complex objects to obtain partial objects; in rule (c) we replace sets by sandwiches. The notation $\_$ is the least element in the Smyth ordering, indicating the absence of any consistent information.

An ordering $\sqsubseteq^{\text{Ob}}$ on partial objects is readily obtained by using the partial record ordering to order record values, and the sandwich ordering to order sandwich values (extended to include $\_$). Both of these orderings were defined in the previous section, and from these definitions it is clear that

PROPOSITION 8. *$(\mathcal{O}, \sqsubseteq^{\text{Ob}})$ is a semilattice with bottom and pairwise bounded joins.*

This gives us a space in which joins are well-defined; in particular, using the notation $v <: \tau$ to indicate that v is a partial object of type τ.

PROPOSITION 9. *If $v_1 <: \tau_1$ and $v_2 <: \tau_2$ and $v_1 \sqcup^{\mathrm{Ob}} v_2$ exists, then $v_1 \sqcup^{\mathrm{Ob}} v_2 <: \tau_1 \sqcup \tau_2$.*

The purpose of imposing types on complex objects is simply to construct a space in which our partial objects are bounded by maximal elements. Without such a restriction it is quite easy to describe unbounded complex objects that represent an E-R instance. The fact that the maximal objects (under $\sqsubseteq^{\mathrm{Ob}}$) of a given type are isomorphic to the complete complex objects of that type is useful to us in preventing rules from "joining" objects on missing or partial information.

Our final task in this section is to describe partial E-R instances and their relationship to partial objects.

Given an E-R schema, an *E-R sandwich* is obtained by replacing the entity and relationship instances by sandwiches, i.e., pairs of instances, and by allowing null values. More formally, with each entity of type $[l_1 : \tau_1; l_2 : \tau_2; ...; l_n : \tau_n]$ associate a partial object (a sandwich) of type $\{[l_1 : \tau_1; l_2 : \tau_2; ...; l_n : \tau_n]\}$, and similarly associate a sandwich with each relationship. Note that in this case, the types $\tau_1, \tau_2, ..., \tau_n$ are all base. Each sandwich is a pair of relational instances of the entity or relationship with null values. E-R sandwiches therefore give rise to complex objects, but the reverse is not, in general, true.

3.4. *Terms and Satisfiability*

The topic of Section 4 is rules for complex objects: their general form and inferences. However, before we can delve into this topic, we need to know what a term is and in what sense a term can be said to be satisfied by a complex object.

We start this discussion with a syntax for *terms* which is obtained by adding variables to our syntax for (untyped) complex objects. By convention, variables are letters X, Y, Z from the end of the alphabet. To define a *typed* term, we assume the existence of a set of typed variables X^τ, Y^σ, ..., where $\sigma, \tau, ...$ are record or base types. The rules for the formation of typed terms are then

- For any typed variable X^τ for some record or base type τ, X^τ is a term of type τ.
- Atomic values are terms (of the appropriate base type).
- If $t_1, t_2, ..., t_m$ are terms of type $\tau_1, \tau_2, ..., \tau_m$, respectively, then $[l_1 \Rightarrow \tau_1; l_2 \Rightarrow \tau_2; ...; l_n \Rightarrow \tau_m]$ is of type $[l_1 : \tau_1; l_2 : \tau_2; ...; l_n : \tau_n]$, where $m \leqslant n$.
- If $t_1, t_2, ..., t_n$ are terms of type τ then $\{t_1; t_2; ...; t_n\}$ is of type $\{\tau\}$.

Given an E-R schema S, a term $[l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n]$ is consistent with S if there is a view $[l_1 : \tau_1, ..., l_n : \tau_n]$ such that t_i is of type τ_i, $1 \leqslant i \leqslant n$. Note that because this is an E-R schema the τ_i are all set types and therefore none of the t_i are themselves variables; variables can only appear as components of these terms. As an example, consider the following term whose type is consistent with the E-R diagram of Fig. 1.

$$\begin{aligned}&[\mathit{Employee} \Rightarrow \{[\mathit{FName} \Rightarrow X; \mathit{LName} \Rightarrow Y; \mathit{Emp\_Dept@Emp}\\ &\Rightarrow \{[\mathit{Dept} \Rightarrow [\mathit{Dname} \Rightarrow \mathit{'Sales'} \mathit{Emp\_Dept@Dept}\\ &\Rightarrow \{[\mathit{Emp} \Rightarrow [\mathit{FName} \Rightarrow X]]\}]]\}]\}].\end{aligned} \tag{1}$$

We now ask what it means for a term of type τ to be satisfied by a complex object of type τ' when τ is consistent with τ'. Let us digress to examine the notion of satisfaction in [2] where a similar syntax is used for *untyped terms*, and satisfaction is defined with respect to the Hoare ordering. That is, a term t is satisfied by a complex object C if there is a substitution σ for the variables in t that places it below C in the Hoare ordering, i.e., $\sigma t \sqsubseteq^{\flat} C$. Rules are then defined as pairs of terms of the form $t_1 :- t_2$ where all the variables in t_1 occur in t_2. An example of a rule is

$$\{[A \Rightarrow X; C \Rightarrow Z]\} :- [R_1 \Rightarrow \{[A \Rightarrow X; B_1 \Rightarrow Y]\}; R_2 \Rightarrow \{[B_2 \Rightarrow Y; C \Rightarrow Z]\}]. \quad (2)$$

If C is a complex object (which we can think of here as being untyped, i.e., a term without variables) the result of applying a rule $t_1 :- t_2$ to C is defined to be the complex object

$$\bigsqcup^{\flat} \{(\sigma t_1) \mid (\sigma t_2) \sqsubseteq^{\flat} C\}.$$

While this is a monotone function (in $\sqsubseteq^{\flat}$) on complex objects, it does not have the desired result. In rule (2) above, the apparent intention is to compute the join of the relations R_1 and R_2 on $B_1 = B_2$, i.e., $\pi_{AC}(\sigma_{B_1 = B_2}(R_1 \bowtie R_2))$. However, since $\perp$ is a possible substitution for Y, what is actually given by this expression is $\pi_{AC}(R_1 \bowtie R_2)$, which is the cross product of R_1 and R_2 since they have no fields named in common. Thus without some restrictions, this formulation of rules on complex objects does not give us the power of Datalog.

A fix to this problem is to work within some type and to restrict the substitutions so that they are, in some sense, maximal for that type. However, as we remarked at the beginning, the maximal complex objects in the Hoare ordering are uninteresting. We prefer to work in the sandwich ordering in which we can actually approximate sets of values. We should therefore extend our syntax for partial objects to include variables. We would then have a precise notion of satisfiability. This is what the authors attempted to do in [17] but the syntactic structures are, to be frank, horrendous. As an alternative, we can use an E-R type to "flatten" our complex object terms (defined as above) into conjunctions of flat literals and then work within a sandwich interpretation for rules that provides a natural extension of Datalog.

The flattening process is straightforward and is best illustrated through an example. Each record sub-term of the given term is either an entity or relationship type in the E-R diagram to which it conforms, so we create a conjunction of literals, one for each record, in the given term. Using the example (1) above, we get

$$\begin{aligned} &\mathit{Employee}(-, -, -, -) \wedge \mathit{Emp\_Dept}(-, -, -) \wedge \mathit{Dept}(-, -, -) \\ &\qquad \wedge \mathit{Emp\_Dept}(-, -, -) \wedge \mathit{Employee}(-, -, -, -), \end{aligned}$$

where the literals are listed in the same order that the corresponding record terms are introduced in the given term (depth-first order).

For each constant in the given term, place that constant in the appropriate position in the corresponding literal.

$$\mathit{Employee}(-,-,-,-) \wedge \mathit{Emp\_Dept}(-,-,-) \wedge \mathit{Dept}(-,\mathit{'Sales'},-)$$
$$\wedge\ \mathit{Emp\_Dept}(-,-,-) \wedge \mathit{Employee}(-,-,-,-).$$

For each variable, if that variable has a base type (i.e., it is an attribute), place it in the appropriate attribute position of the corresponding literal. If it is a record term, place it in the identity position for that record.

$$\mathit{Employee}(-,X,Y,-) \wedge \mathit{Emp\_Dept}(-,-,-) \wedge \mathit{Dept}(-,\mathit{'Sales'},-)$$
$$\wedge\ \mathit{Emp\_Dept}(-,-,-) \wedge \mathit{Employee}(-,X,-,-).$$

Whenever a term t_1 is of the form $[..., R@l \Rightarrow t_2, ...]$, which happens when t_1 is an entity variable and t_2 is a relationship, create a fresh variable (if necessary) and place it in the identity position of t_1 and the attribute of t_2 that corresponds to that entity. Also, if t_1 is of the form $[..., l \Rightarrow t_2, ...]$ where t_1 is a relationship and t_2 is an entity, create a fresh variable (if necessary) and place it in the identity position in the literal corresponding to t_2, and the attribute of t_1 that corresponds to that entity. Note that in this process we may identify two existing variables.

$$\mathit{Employee}(U,X,Y,-) \wedge \mathit{Emp\_Dept}(-,U,V) \wedge \mathit{Dept}(V,\mathit{'Sales'},-)$$
$$\wedge\ \mathit{Emp\_Dept}(-,W,V) \wedge \mathit{Employee}(W,X,-,-).$$

Although it might appear that we have simply reduced complex-object rules to Datalog, recall that we are no longer working in a normal relational domain. We therefore think of a rule as something that, in the presence of complete information, generates a constraint on the database rather than a new "facts." This is the subject of the next section.

4. Rules and Interpretation

The rules our system accepts have the form

$$\underset{\text{(hypothesis)}}{p} \quad \text{implies } \underset{\text{(conclusions)}}{q_1 \text{ or } q_2 \text{ or} \cdots \text{or } q_k},$$

where p and the q_i's are terms. As shown at the end of Section 3, these terms can be translated to conjunctions of literals from some E-R database schema. We call such a conjunction of literals a *pattern*, and assume that all terms have been translated into patterns for the remainder of this section. We refer to p as the "hypothesis," and "q_1 or q_2 or $\cdots$ or q_k" as the conclusion. When the structure of p is important we write

$$\underbrace{L_1 \text{ and } L_2 \text{ and } \cdots L_m}_{\text{(hypothesis)}} \text{ implies } \underbrace{q_1 \text{ or } q_2 \text{ or } \cdots \text{ or } q_k}_{\text{(conclusions)}}.$$

Note that this form is more general than that of [2] (discussed in Section 3) since it allows a *disjunction* of terms in the conclusion.

Although the rules may contain disjunctions, the inferences derived using these rules are *constructive*. That is, the rules are interpreted as a group of possible actions that the user instructs the database to perform; the database will not use non-constructive reasoning in order to make clever inferences. For example, suppose our database has information indicating that there is a direct flight leaving Philadelphia which is either going to Columbus or Chicago. Furthermore, suppose that the database knows there are direct flights from both Columbus and Chicago to San Francisco. Given this information, we can correctly conclude that there must be an indirect flight from Philadelphia to San Francisco—but our system will not do so because the intermediate stop cannot be explicitly presented; the inference that there is an indirect flight is not constructive.

There are many situations where non-constructive reasoning can be misleading (game playing, for example). However, our chief motivation for preferring constructive reasoning is to allow a tractable implementation that behaves in a predictable way. If a classical interpretation is applied to our rules, it becomes quite easy to specify problems that are thought to be intractable (NP-complete problems) with a short sequence of simple rules. Consequently, a complete implementation of such a system would be capable of transforming a small and innocent looking sequence of rules into an exponential program—quite likely to the surprise and dismay of the user.

With constructive reasoning, in contrast, it is clear how the system will behave and what inferences it will make. Furthermore, we later show that the constructive interpretation admits an implementation that is always polynomial in the size of the database.

In this section, we first give an intuitive overview of the constructive reasoning performed by our system. We then present some extended relational operators that are necessary to give precise meanings to rules. Finally, we give precise meanings to the various interpretations of explicit rules (integrity check, consistent inference, and complete inference) and define implicit inferences and constraints that are derived from the semantics of an E-R schema. To conclude the section we explain the meaning of a set of rules.

4.1. *Intuitive Meanings*

In this section we motivate the interpretations we apply to each rule the user asserts.

When there is no missing information in the database the meaning of a rule reduces to an integrity assertion which insists that whenever you see some instance of the pattern on the left side in the database you must also see a compatible instance for one of the patterns occurring on the right side—where "instance" means a substitution for the variables in the patterns. Diagrammatically

p	implies	q_1 or q_2 or $\cdots$ or q_k
if you see an instance of this		
	then	you must see a compatible instance of one of these.

If the database does not satisfy this test the system must flag an integrity error. This is the "maximal interpretation" we use to justify three more general interpretations.

When the database has approximate information the system performs three types of actions derived from each rule.

Integrity check. Check that the (partial) information in the database does not constructively contradict the entailment relationship specified by the rule.

Consistent inference. Constructively infer instances of the conclusions which must be true by combining the entailment specified by the rule with the information in the database.

Complete inference. Constructively refine the realm of possibility (i.e., complete sets) for the hypothesis by combining the entailment specified by the rule with information in the database.

The combination of the three interpretations given above is the meaning our system ascribes to rules.

We illustrate the actions of the rule with an example. Recall that we motivated the notion of the sandwich with the sandwich approximation to the set of students interested in database research (A, B) given by entity set approximation DBS-Enthusiast:

Last Name	*First Name*	*Last Name*
Johnson	Ella	Taylor
Pierce	Burt	—
Taylor	—	Pierce
Cooper		
Emerson		
Billings		

Suppose in addition we were able to obtain a sandwich approximation (A', B') for the students interested in programming languages given by PL-Enthusiast entity set:

First Name	*Last Name*	*First Name*	*Last Name*
Ella	Taylor		
Josiah	Taylor		
—	Pierce		
—	Wayne		

(note that B' is empty—we are not yet certain there is any such person). Note that since both of the above sandwich approximations are not maximal, we can concievably improve the information they contain through some inference mechanism.

Furthermore, suppose we can determine a "total" approximation (T, T) for the set of students who are interested in operating systems—the OS-Enthusiast entity set.

First Name	*Last Name*	*First Name*	*Last Name*
Burt	Charles	Burt	Charles
Sal	Emerson	Sal	Emerson
Burt	Emerson	Burt	Emerson
Ray	Vito	Ray	Vito

Since this is a maximal sandwich, where all information is known exactly, we cannot infer any additional information about this set without contradiction.

Finally, suppose we assert the following rule with the intended meaning that "all students interested in databases are either interested in operating systems or programming languages."

$$\begin{gathered}\textit{DBS-Enthusiast} \Rightarrow \{[\textit{Firstname} \Rightarrow X; \textit{Lastname} \Rightarrow Y]\} \text{ implies}\\ \textit{OS-Enthusiast} \Rightarrow \{[\textit{Firstname} \Rightarrow X; \textit{Lastname} \Rightarrow Y]\} \text{ or}\\ \textit{PL-Enthusiast} \Rightarrow \{[\textit{Firstname} \Rightarrow X; \textit{Lastname} \Rightarrow Y]\}.\end{gathered}$$

In the rest of this section we explain how we derive a consistency constraint and inference interpretations from this rule, and we describe the changes to the above sandwich approximations that the system infers.

The more general form of the *integrity check* verifies that if an instance of the hypothesis is *known* then at least one of the conclusions must be *conceivable* (i.e., within the realm of possibility). Diagrammatically we require

p	implies	q_1 or q_2 or $\cdots$ or q_k
If an instance p' of p is known to be true		
	then	one or more compatible q_i is in the realm of possibility.

Again, if this check fails the system must flag an integrity error. Note that if the database contains exact information this interpretation reduces to the maximal interpretation.

In our example, we must check that each consistent entry in the approximation to DBS-Enthusiasts corresponds to a compatible entry in one of the complete approximations for OS-Enthusiasts or PL-Enthusiasts. In fact they do: "Ella Taylor" in B corresponds to "Ella Taylor" in A'; "Burt —" in B corresponds to either "— Wayne" in A' or "Burt Emerson" in T; "— Pierce" in B corresponds to "— Pierce" in A'. Thus the system detects no explicit contradiction and does not raise an integrity error.

However, we are not satisfied with simply checking that the database does not contradict the rules—we also use each rule to make inferences that either increase the number of facts known to be true (the consistent information) or decrease the realm of possibility of facts which the database considers to be possible (the complete information).

To use a rule to infer facts which must be true, we apply a *consistent interpretation* to the rule—one such interpretation for each conclusion. The consistent interpretation for conclusion q_1 insists that whenever an instance of p is *known* to be true and no other conclusion q_j is possible, we can correctly infer that the simplest instance of q_1 that is consistent with the hypothesis must be true. Diagrammatically

p	implies	q_1 or q_2 or $\cdots$ or q_k
If an instance p' of p is known to be true	*and*	no compatible instance of any $q_2 \cdots q_k$ is within the realm of possibility
	then	conclude that the weakest compatible instance of q_1 must be true!

A similar interpretation is provided for each q_i.

In the example rule, since "— Pierce" is in B and no entry in T can possibly have the last name "Pierce," we are forced to conclude that there must be a PL-Enthusiast named "— Pierce" and hence we introduce this entry into B'. Similarly since we cannot resolve the entry "Ella Taylor" in B with any entry in T, we introduce the entry into B'. However, since the entry "Burt —" may or may not correspond to the entry "Burt Charles" in T we cannot immediately conclude that someone named Burt is a PL-Enthusiast, and thus we do not introduce this entry into B'. The improved approximation (A', B') for PL-Enthusiasts is given by

First Name	*Last Name*	*First Name*	*Last Name*
Ella	Taylor	Ella	Taylor
Josiah	Taylor	—	Pierce
—	Pierce		
—	Wayne		

Here, the complete estimate A' remains unchanged while the consistent estimate B' is improved. Although we define a similar inference for the OS-Enthusiast entity set there can be no meaningful improvement to the approximation representing the OS-Enthusiasts since that approximation is maximal—any introduction of additional elements would constitute a contradiction.

It is important to observe that the introduction of these entries into B' are all monotone operations in the Hoare powerdomain. Also, note that we make use of negative information about the realm of possible OS-Enthusiasts to infer positive information about the known PL-Enthusiasts.

More interestingly since the implication relationship asserted by a rule also constrains the realm of possibility, we introduce a *complete interpretation* for the rule. The primary intuition here is that a substitution for variables that is not possible for at least one of the conclusions cannot be possible for the hypothesis. We restate this in a more useful (but more opaque) form: the set of possible instances for a literal in the hypothesis, say L_1, that *must* match p can be restricted by the set of possible instances for the conclusions. Diagrammatically

L_1 and L_2 and $\cdots$ and L_m	implies	q_1 or q_2 or $\cdots$ or q_k
Letting X be possibilities for L_1 which must match hypothesis p	*and*	Letting Y be the complete set of possibilities that may match any q_i
	conclude	
"$X \cap Y$" is an improved estimate of all possibilities for L_1 which must match hypothesis p!		

Here the intersection is in quotes because set intersection is not appropriate in the context of partial information—what we need is the algebra of the Smyth powerdomain.

In our example database we must check each hypothetical PL-Enthusiast listed in A against both the conceivable OS-Enthusiasts listed in T and the conceivable PL-Enthusiasts listed in A'. We immediately note that entries "Johnson," "Cooper," and "Billings" in A cannot be resolved with any entries in T or A', and thus these possibilities can be eliminated. Note that this elimination is a monotone action in the Smyth powerdomain and hence constitutes a form of monotone inference of negative information.

Moreover we note that since the entry "Taylor" in A can only be resolved against "Ella Taylor" or "Josiah Taylor" in A', we can replace "Taylor" with these two improved possibilities. Similarly the entry "Emerson" in A can be refined to the more restrictive descriptions "Sal Emerson" and "Burt Emerson" from T. Again, these replacements constitute monotone operations in the Smyth powerdomain, and since they restrict the number of possible first names they constitute a monotone inference of negative information. Finally the improved approximation for the set of PL-Enthusiasts is given by

First Name	*Last Name*
Sal	Emerson
Burt	Emerson
Ella	Taylor
Josiah	Taylor
—	Pierce

First Name	*Last Name*
Ella	Taylor
Burt	—
—	Pierce

where the realm of possible entries in A has been refined but the known entries in B remain the same.

These interpretations are made rigorous in the following sections.

4.2. *Some Extended Relational Operators*

Before the translation of the rules into their respective interpretations can be given, we must introduce some algebraic operators. For notational convenience, these operators are expressions involving extensions of the relational algebra (see [23]). These extensions deal with null values in a manner which reflects their various meanings within differing contexts.

In the presence of null values, the operations of cross-product and projection do not require any revision. However, since the operations of difference and selection both make explicit reference to the values of attributes, it is not immediately clear how they should behave in the presence of nulls. We therefore introduce an extension of the difference operator, the *incompatibility operator*, as well as two extensions of the selection operator, *pessimistic selection* and *unifying selection*.

The incompatibility operator. There are several meaningful extensions of the relational subtraction operator in the presence of null values, but the only one needed by our system is the incompatibility operator **inc** defined by

$$X \textbf{ inc } Y = \{x \mid x \in X \text{ and } x \sqcup y \text{ is not defined for each } y \in Y\}$$

whenever X and Y are finite sets of values within some domain $\mathscr{D}$. Note that if X and Y are both conventional relations (sets of tuples containing no null values) over the same schema, we have

$$X \textbf{ inc } Y = X - Y,$$

where the right-hand side is the conventional relational subtraction operation. In the presence of null values, the intuition is that X **inc** Y gives the set of tuples of X which could not possibly match any tuple of Y under any substitution for the missing information.

More generally, if X is a set of (partial) substitutions that extend to make a predicate q true, and Y is a complete set of (partial) substitutions outside of which predicate p must be false, then X **inc** Y gives the set of substitutions where q must be true and p must be false. This last interpretation is the one our system requires.

For example, suppose X is the relation representing all managers

First Name	*Last Name*
Lena	Gomez
William	Daley
Pam	—
Ann	Barnacle
—	Phillips
Sal	Priestly

and Y is the relation representing full time employees (under the same schema as X)

First Name	*Last Name*
—	Daley
—	Priestly
Jay	Percy

In this case X **inc** Y, the set of managers who cannot be full time employees, would be

First Name	*Last Name*
Ann	Barnacle
—	Phillips
Lena	Gomez

The following easily proven monotonicity property is important in later sections.

PROPOSITION 10. *If* $X \sqsubseteq^{\flat} X'$ *and* $Y \sqsubseteq^{\sharp} Y'$ *then* X **inc** $Y \sqsubseteq^{\flat} X'$ **inc** Y'.

The pessimistic selection operator. In defining the rules, it is useful to have a selection operator which assumes that if anything can go wrong it will—such an operator is the pessimistic selection operator. This operator has two "simple" forms.

If c is any non-null constant, then we define

$$\sigma_{A=c} X = \{x \mid x \in X \text{ and } x.A = c\}.$$

Thus if X is a set of employee records containing a *FirstName* field, then

$$\sigma_{FirstName = 'John'} X$$

will return the descriptions of employees whose *FirstName* is known to be "John." Any tuple with a null *FirstName* will not be returned.

More interestingly, define

$$\sigma_{A=A'}X = \{x \mid x \in X \text{ and } x.A \neq \bot \text{ and } x.A' \neq \bot \text{ and } x.A = x.A'\}.$$

Thus with X as given above,

$$\sigma_{FirstName = LastName} X$$

will give the descriptions of all employees whose *FirstNames* are known to be the same as their last names.

The general form of the pessimistic selection operator is given as follows: If $E = \{e_1, ..., e_n\}$ is a set of equality constraints of form $A = A'$ or $A = c$ then

$$\sigma_E X = \sigma_{e_1} \cdots \sigma_{e_n} X.$$

It is an easy exercise to show that this selection operator is uniquely defined regardless of the ordering on the e_i's.

Clearly, $\sigma_E X$ will return the set of elements of X which must match the selection criteria in E no matter what substitutions are made for null values. Furthermore, it is a monotone operator in the Hoare ordering with the following easily verified property:

PROPOSITION 11. *If $X = \sigma_E X$ and $X \sqsubseteq^\sharp X'$ then $X' = \sigma_E X'$.*

The unifying selection operator. An alternative extension to the selection operator is one that assumes that everything will go right—this operator is the unifying selection operator υ defined below.

If c is any non-null constant then we define

$$\upsilon_{A=c} X = \{x \sqcup [A \Rightarrow c] \mid x \in X \text{ and the join is defined}\}.$$

Thus if X is a set of employee records containing a *FirstName* field then

$$\upsilon_{FirstName = 'John'} X$$

will return the descriptions of employees whose *FirstName might* be "John"—and the operator will optimistically fill in "John" in any first name which is unknown.

In a similar fashion, we define

$$\upsilon_{A=A'} X = \{x \sqcup [A \Rightarrow x.A'; A' \Rightarrow x.A] \mid x \in X \text{ and the join is defined}\}.$$

Thus, in parallel to the previous example

$$\upsilon_{FirstName = LastName} X$$

gives the employees whose *FirstNames* might be the same as their last names—and the operator optimistically forces them to be equal whenever one name is defined and the other is not. If neither the first name nor the last name is defined, the employee is retrieved, but the names remain undefined.

The intuition behind these atomic forms is that $\sigma_{e_1} X$ returns a complete list of descriptions from X which *might* satisfy e_1 and the values they would necessarily have *if* they did.

The general form of the unifying selection operator can now be defined as follows. Suppose $E = \{e_1, ..., e_n\}$ is a set of equality constraints of form $A = A'$ or $A = c$ then

$$\sigma_E X$$

is the least fixed point Y of a non-deterministic selection program with initial value

$$Y := X$$

and iterated selection operations

$$Y := v_{e_i} Y$$

for each $e_i \in E$. It is not difficult to see that this least fixed point is defined and unique, independent of the order in which the selection operations are performed. Furthermore, v_E operator is obviously monotone in the Smyth ordering with the following easily proven property:

PROPOSITION 12. *If $X \sqsubseteq^\flat v_E X$ and $X' \sqsubseteq^\flat v_E X'$ and $X \sqsubseteq^\flat X'$ then $v_E X \sqsubseteq^\flat v_E X'$.*

Finally we introduce a mild abuse of notation. Suppose two sets of records r and s have non-null values in disjoint attribute sets X and Y, respectively. We use the notation

$$r \times s$$

to denote all possible combinations of tuples from r with tuples from s, in place of

$$\pi_X r \times \pi_Y s$$

or any more complicated notation.

Using these extensions of the relational algebra, we can now give the precise meaning of rules.

4.3. *Syntactic Simplification*

As it stands, a flat rule is not amenable to immediate interpretation. In order to simplify the notational difficulties, we transform such rules into a *compiled form.* We then assign meanings to the compilation, and define the meaning of an arbitrary rule to be the meaning of its compiled form.

To illustrate the transformation of a rule into its compiled form we use a running example interspersed with the description of the translation process. Consider the following rule, R_0, defined over the *Employee–Department* schema introduced in the previous section, which insists that "Everyone in the sales department has phone extension 2265 or 8798":

$$\begin{array}{lll}
\text{Employee}(X, -, -) \text{ and} & & \\
\text{EmpDept}(-, X, Y) \text{ and} & & \\
\text{Dept}(Y, \text{'Sales'}, -) & & \\
& \text{implies} & \\
& & \text{Employee}(X, -, -, 2265) \text{ or} \\
& & \text{Employee}(X, -, -, 8792).
\end{array}$$

For the purposes of discussion we refer to the individual literals of R_0 as

$$\begin{aligned}
L_1 &= \text{Employee}(X, -, -, -)\\
L_2 &= \text{EmpDept}(-, X, Y)\\
L_3 &= \text{Dept}(Y, \text{'Sales'}, -)\\
L_4 &= \text{Employee}(X, -, -, 2265)\\
L_5 &= \text{Employee}(X, -, -, 8792).
\end{aligned}$$

Furthermore we distinguish the patterns of R_0 by

$$\begin{aligned}
p &= L_1 \text{ and } L_2 \text{ and } L_3\\
q_1 &= L_4\\
q_2 &= L_5
\end{aligned}$$

and thus

$$R_0 = p \text{ implies } q_1 \text{ or } q_2$$

provides a more compact representation for our example rule.

Following the above model we parse any flat rule as follows. Identify the set of literals that occur in the rule by $\{L_1, ..., L_n\}$, the hypothesis of the rule by p, and the conclusions for the rule by $\{q_1, ..., q_k\}$. Thus

$$R = p \text{ implies } q_1 \text{ or} \cdots \text{or } q_k$$

is an abbreviated representation for any such rule R.

The first step in the translation process is to create a surrogate literal for each literal occurring in the rule, introducing fresh variables $\{V_i\}$ in the place of every argument. In the case of R_0 we have the following surrogate literals.

$$L_1' = \text{Employee}(V_0, V_1, V_2, V_3)$$
$$L_2' = \text{EmpDept}(V_4, V_5, V_6)$$
$$L_3' = \text{Dept}(V_7, V_8, V_9)$$
$$L_4' = \text{Employee}(V_{10}, V_{11}, V_{12}, V_{13})$$
$$L_5' = \text{Employee}(V_{14}, V_{15}, V_{16}, V_{17}).$$

The variables introduced in this step serve as "column names" in extended relational algebra expressions to be presented later. We call the V_i's the *new variables* and the variables occurring in the initial form of the rule the *old variables.*

We immediately notice that the surrogate literals do not express the structural information given by old variables and constants occurring in the initial rule. This information is introduced via constraints as defined below.

As a second step in the compilation process, we associate to each pattern ψ in R a set of *local constraints* through the following two rules:

- Wherever a constant c occurs in the pattern which corresponds to a new variable V_i introduce the constraint $(V_i = c)$.
- Wherever an old variable X occurs twice in the pattern corresponding to new variables V_i and V_j introduce the constraint $(V_i = V_j)$.

To represent the set of local constraints for ψ we write E_ψ.

Using these rules in the case of R_0 we obtain the following local constraints.

$$E_p = \{(V_0 = V_5), (V_6 = V_7), (V_8 = \text{'Sales'})\}$$
$$E_{q_1} = \{(V_{13} = 2265)\}$$
$$E_{q_2} = \{(V_{17} = 8792)\}.$$

Later, these local constraints are used in selection expressions to define what it means for a piece of the database to "match" or "not match" a pattern—under varying interpretations for the word "match."

As a final step in the translation process we add global constraints E_R which relate the structure of the hypothesis p to the structures of the conclusions q_i. In a manner similar to the previous procedure we have the obvious rule

- Wherever an old variable X occurs in the hypothesis p corresponding to new variable V_i and in a conclusion q_l corresponding to new variable V_j introduce the constraint $(V_i = V_j)$.

Using this rule for R_0 we get the constraints

$$\{(V_0 = V_{10}), (V_0 = V_{14}), (V_5 = V_{10}), (V_5 = V_{14})\}$$

representing the variable matching between the hypothesis and the conclusions.

But there is also a more subtle consideration which must be represented by the global constraints—object identity. Since the first column of an employee literal represents an object identity, the occurrence of the X in the hypothesis literal L_1 and in the conclusion literals L_4 and L_5 not only requires that the identity columns must match, but that all other columns within the literal must match also. Hence we need the additional step:

Whenever the following are simultaneously satisfied

- Old variable X occurs in hypothesis literal L_i;
- old variable X occurs in conclusion literal L_j;
- L_i and L_j are of the same class (have the same head); and
- in both occurrences X is in the role of an object identity

introduce constraints equating the corresponding columns of surrogate literals L'_i and L'_j.

Constraints generated by the above procedure require that objects with the same identities that are in the same contexts must share the same descriptions.

TABLE I

Relational Approximations of Employee, Department, and Emp_Dept

Employee$\sharp$				Employee$\flat$			
Id	*FName*	*LName*	*Telephone*	*Id*	*FName*	*LName*	*Telephone*
12	Joe	Smith	2265	12	Joe	Smith	$\perp$
12	Joe	Smith	5910	13	Sally	Jones	$\perp$
13	Sally	Jones	$\perp$	14	John	Smith	$\perp$
14	John	Smith	$\perp$				
$\perp$	Peter	Walters	$\perp$				

Department$\sharp$			Department$\flat$		
Code	*DName*	*Location*	*Code*	*DName*	*Location*
2	Sales	RB3	2	Sales	RB3
3	Marketing	HD2	3	Marketing	HD2
4	Sales	RD7	4	Sales	RD7

Emp_Dept$\sharp$			Emp_Dept$\flat$		
Contract	*Emp*	*Dept*	*Contract*	*Emp*	*Dept*
98	12	2	98	12	2
75	13	2	77	14	3
77	14	3			
78	$\perp$	4			

Using this last rule on R_0 we derive the additional constraints

$$\{(V_1 = V_{11}), (V_2 = V_{12}), (V_3 = V_{13}), (V_1 = V_{15}), (V_2 = V_{16}), (V_3 = V_{17}),\}.$$

Hence the global constraints for R_0 are

$$E_{R_0} = \{(V_0 = V_{10}), (V_0 = V_{14}), (V_5 = V_{10}), (V_5 = V_{14}), (V_1 = V_{11}),$$
$$(V_2 = V_{12}), (V_3 = V_{13}), (V_1 = V_{15}), (V_2 = V_{16}), (V_3 = V_{17}),\}$$

These completely characterize the relationship between the structure of the hypothesis and the structures of the conclusions.

Finally we call the collection of all derived patterns, surrogate literals, and equality constraints (both local and global) the *compiled form* of a rule R. In particular R_0 has a compiled form consisting of p, $\{q_1, q_2\}$, $\{L'_1, ..., L'_5\}$, E_p, $\{E_{q_1}, E_{q_2}\}$, and E_{R_0}.

In relational algebra expressions involving the compiled information, the subscripts would get out of hand unless we use abbreviations. Hence when we project over the new variables associated with a literal L we write π_L in place of any more complex notation. Similarly when we wish to project over the new variables associated with a pattern ψ we write π_ψ. Furthermore for selection expressions we write σ_ψ or v_ψ for selections using the local constraints E_ψ for a pattern ψ. In the same manner we write σ_R or v_R for selections using the global constraints E_R of a rule R.

Thus armed, we are prepared to define the interpretation of a rule. We use R_0 as a running example, and clarify the interpretations by showing their effect on an instance of the *Employee–Department* database given in Table I.

4.4. *The Integrity Check Interpretation of a Rule*

In this section, we mechanize the interpretation of a rule as an integrity check on the information in the database. We derive a rigorous expression of the assertion "wherever the hypothesis is true one of the conclusions must be possible," with diagrammatic representation:

p	implies	q_1 or q_2 or $\cdots$ or q_k
If an instance p' of p is known to be true		
	then	one or more compatible q_i is in the realm of possibility.

Thus we must rigorously identify the "versions of p" which are known to be true and the "realm of possible versions of q." Furthermore, we need to define under what circumstances a version of p is "compatible" with a version of q.

The incarnations of a literal. Recall that each literal L of a rule R is associated with some "component" F of a database, where the component is either a class of objects or a relationship among objects in the database. For instance, in the example rule R_0 the literal

$$L_5 = \text{Employee}(X, -, -, 8792)$$

is associated with the class of Employees in the database. A given database instance will have a complete approximation and a consistent approximation for F, and these approximations will be used in the interpretation of R. In order to translate the approximations into forms which are amenable to algebraic calculations, we create a relational *incarnation* for each approximation.

If the sandwich approximation for the class or relationship F is given by $\langle A, B\rangle$, the complete incarnation for L is generated by naming each column of A with the new variable that corresponds to each argument of L. In the example of L_5 above, we have new variables V_{14} through V_{17}

$$L_5' = \text{Employee}(V_{14}, V_{15}, V_{16}, V_{17})$$

and hence the complete incarnation for L_5 (using the instance given in Table I) is:

V_{14}	V_{15}	V_{16}	V_{17}
12	Joe	Smith	2265
12	Joe	Smith	5910
13	Sally	Jones	$\perp$
14	John	Smith	$\perp$
$\perp$	Peter	Walters	$\perp$

We use the notation $L^\sharp$ to identify the complete incarnation for L.

Applying the same renaming procedure to the consistent approximation B, we obtain the consistent incarnation of any literal L, written $L^\flat$.

The true instances of a pattern. To identify the portion of the database which constructively satisfies a pattern ψ, form a cross product of the appropriate consistent information in the database, and exclude every combination which might not satisfy the pattern. What remains are combinations of descriptions which necessarily satisfy the pattern ψ. More precisely, if ψ is constructed from literals L_0 through L_l define:

$$\tau_\psi = \sigma_\psi(L_0^\flat \times L_1^\flat \times \cdots \times L_l^\flat).$$

Note that we use the pessimistic selection operator to ensure that comparisons involving null values are considered to fail—with the rationale that if at some later stage the nulls are filled in with appropriate values the comparisons may fail.

In our running example, τ_p would be

τ_p:

V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	$V_{10} \cdots V_{17}$
12	Joe	Smith	$\perp$	98	12	2	2	Sales	RB3	$\perp \cdots \perp$

This calculation defines a monotone function that takes a database instance and returns an element τ_ψ of the Hoare powerdomain over tuples.

All conceivable instances of a pattern. In a similar manner, the complete portion of the database which *might* satisfy the pattern ψ is identified by forming a cross product from the appropriate complete approximations of the database and attempting to unify the combinations of descriptions to force them to satisfy the pattern. More precisely, define

$$\kappa_\psi = v_\psi(L_0^\sharp \times L_1^\sharp \times \cdots \times L_n^\sharp).$$

In our example, κ_{q_1} and κ_{q_2} are:

	$V_0 \cdots V_9$	V_{10}	V_{11}	V_{12}	V_{13}	$V_{14} \cdots V_{17}$
	$\perp \cdots \perp$	12	Joe	Smith	2265	$\perp \cdots \perp$
κ_{q_1}:	$\cdot$	13	Sally	Jones	2265	$\cdot$
	$\cdot$	14	John	Smith	2265	$\cdot$
	$\perp \cdots \perp$	$\perp$	Peter	Walters	2265	$\perp \cdots \perp$

	$V_0 \cdots V_{13}$	V_{14}	V_{15}	V_{16}	V_{16}
κ_{q_2}:	$\perp \cdots \perp$	13	Sally	Jones	8792
	$\cdot$	14	John	Smith	8792
	$\perp \cdots \perp$	$\perp$	Peter	Walters	8792

This calculation is also a monotone operation taking a database instance to an element of the Smyth powerdomain on tuples, κ_ψ.

The integrity check. Once τ_p and each κ_{q_i} are calculated, it is a simple matter to rigorously express the integrity constraint represented by the rule R. Intuitively, every true instance for p must be compatible with some conceivable instance for some q_i. Formally, we require

$$\tau_p \ \mathbf{inc}\ v_R(\kappa_{q_1} \sqcap^\sharp \kappa_{q_2} \sqcap^\sharp \cdots \sqcap^\sharp \kappa_{q_k}) = \varnothing.$$

If this equality is not satisfied then there is an instance of the hypothesis which the database maintains to be true but which suggests conclusions that the database insists are not possible. When such a contradiction occurs, the database explicitly violates the rule R, and the system will flag an integrity violation.

In our example, $\pi_p v_R(\kappa_{q_1} \sqcap^\sharp \kappa_{q_2})$ would be given by

V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
12	Joe	Smith	2265	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
13	Sally	Jones	2265	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
14	John	Smith	2265	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	Peter	Walters	2265	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
13	Sally	Jones	8792	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
14	John	Smith	8792	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	Peter	Walters	8792	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

(where we projected out irrelevant columns). Since the equality is satisfied, the database instance given in Table I does not violate R_0.

4.5. *The Consistent Interpretations for a Rule*

Using methods similar to the above, we now define the consistent inferences that a rule provides. Informally, when an instance of p is true and no compatible instance of q_2 through q_k are possible, the system will infer the weakest compatible instance of q_1 to be true. Diagrammatically

p	implies	q_1 or q_2 or $\cdots$ or q_k
If an instance p' of p is known to be true	*and*	no compatible instance of any $q_2 \cdots q_k$ is within the realm of possibility
	then	conclude that the weakest compatible instance of q_1 must be true!

A similar rule is defined for each conclusion q_i.

A relatively straightforward translation of the above intuition is given by

$$\tau'_{q1} = v_{q1} \pi_{q1}((v_R \tau_p) \,\mathbf{inc}\, (\kappa_{q2} \sqcap^\sharp \kappa_{q3} \sqcap^\sharp \cdots \sqcap^\sharp \kappa_{qk})).$$

Here, $v_R \tau_p$ extends the instances of p which are known to be true to include the consequential restrictions for each of the possible conclusions. From this, the incompatibility operator removes those elements where a conclusion other than q_1 is possible. What remains are instances of the hypothesis where q_1 is the only conceivable conclusion. Since q_1 must also satisfy its local constraints, we then apply v_{q1} to the result, strengthening the inference.

In our running example:

	$V_0 \cdots V_9$	$V_{10}V_{11}$	V_{12}	$V_{13}V_{14}V_{15}$	V_{16}	V_{17}
$v_R \tau_p$:	(as in τ_p)	12 Joe	Smith	$\perp$ 12 Joe	Smith	$\perp$

	$V_{10}V_{11}$	V_{12}	V_{13}
τ'_{q1}:	12 Joe	Smith	2265

This computation is not obviously monotone on an instance of the database since v is not always monotone over the Hoare powerdomain. However, we note that for any database instance $v_R \tau_p$ only fills in null values in τ_p and never eliminates elements. Thus we have

$$\tau_p \sqsubseteq^\flat v_R \tau_p$$

and conclude that this calculation defines a monotone function mapping a database instance to the Hoare powerdomain over records. However, v_{q_1} may eliminate some tuple, and therefore fail to be monotone over the Hoare powerdomain. In this case, there must be an instance of the hypothesis which cannot match any instance of any of the conclusions, which is an anomaly. Therefore, in the absence of anomalies, this computation defines a monotone function from database instances to the Hoare powerdomain over records. If an anomaly is present, it will be detected by the integrity constraint interpretation of the rule.

After this computation, τ'_{q_1} represents instances of q_1 which must be true given the evidence of the database and the rule R. In order to translate this information into facts to insert in the database, we must project out the information pertaining to each individual literal. Thus if L_i is a literal occurring in q_1, we compute the inferences for L_i by

$$\tau'_{L_i} = \pi_{L_i} \tau'_{q_1}.$$

If L_i is associated with entity (or relationship) set F with current approximation (A, B) in the database instance then we insert τ'_{L_i} into the database representation by

$$B := B \sqcup^{\flat} \text{unname}(\tau_{L_i})$$

(where "unname" forgets the artificial column names V_i).

The collection of all such effects for all literals L_i occurring in q_1 defines the monotone consistent inferences for q_1. We define similar consistent inferences for each q_i, and the collection of all such inferences defines the consistent interpretation for the rule R.

Proposition 13. *Each consistent interpretation of a rule is an inflationary operator on database instances.*

In our example, no inferences can be made for q_2, and the effect on the database instance of Table I is the following improvement to *Employee*:

Employee$\flat$

Id	*FName*	*LName*	*Telephone*
12	Joe	Smith	2265
13	Sally	Jones	$\perp$
14	John	Smith	$\perp$

4.6. *The Complete Interpretation for a Rule*

The dual interpretation to the consistent interpretation restricts the possibilities for the hypothesis by the possibilities for the conclusions. Diagrammatically

L_1 and L_2 and $\cdots$ and L_m	implies	q_1 or q_2 or $\cdots$ or q_k
Letting X be possibilities for L_1 which must match hypothesis p	*and*	Letting Y be the complete set of possibilities that may match any q_i
	conclude	
"$X \cap Y$" is an improved estimate of all possibilities for L_1 which must match hypothesis p!		

Here we need to take care *not* to alter any possibilities for L_1 that may *not* match p.

In order to differentiate possible combinations of information that are known to match p and possible combinations which may not match p, we partition the realm of possible values associated with L_1 as follows. First identify those situations which necessarily match p (those where all other components of p are known to be true)

$$\kappa^0_{L_1} = \sigma_p(L_1^\sharp \times L_2^\flat \times \cdots \times L_m^\flat).$$

Then identify those situations which might not match p

$$\bar{\kappa}_{L_1} = L_1^\sharp - \pi_{L_1}\kappa^0_{L_1}$$

(here set subtraction is meaningful since $\pi_{L_1}\kappa^0_{L_1}$ is a subset of $L_1^\sharp$). The action of the rule will improve the estimate of $\kappa^0_{L_1}$ but leave the other possibilities in $\bar{\kappa}_{L_1}$ unchanged.

In our running example,

	V_0	V_2	V_3	V_4	V_4	V_5	V_6	V_7	V_8	V_9	$V_{10}\cdots V_{17}$
$\kappa^0_{L_1}$:	12	Joe	Smith	2265	98	12	2	2	Sales	RB3	$\perp\cdots\perp$
	12	Joe	Smith	5910	98	12	2	2	Sales	RB3	$\perp\cdots\perp$

	V_0	V_1	V_2	V_3
$\bar{\kappa}_{L_1}$:	13	Sally	Jones	$\perp$
	14	John	Smith	$\perp$
	$\perp$	Peter	Walters	$\perp$

To compute the complete set of possibilities that match any q_i we take a "union" (Smyth meet) of the complete sets of possibilities for each individual q_i

$$\kappa_{q_1} \sqcap^\sharp \kappa_{q_2} \sqcap^\sharp \cdots \sqcap^\sharp \kappa_{q_k}.$$

We then combine these possibilities with the set of possibilities known to match p for L_1 and attempt to resolve the global constraints via unifying selection

$$v_R(\kappa^0_{L_1} \times (\kappa_{q_1} \sqcap^\sharp \kappa_{q_2} \sqcap^\sharp \cdots \sqcap^\sharp \kappa_{q_k})).$$

What remains is a correct improved estimate for the possibilities known to match the pattern p, once we have removed extraneous information. Hence we set

$$\kappa'_{L_1} = \pi_{L_1} \nu_R(\kappa^0_{L_1} \times (\kappa_{q_1} \sqcap^\sharp \kappa_{q_2} \sqcap^\sharp \cdots \sqcap^\sharp \kappa_{q_n})).$$

For our example,

$\kappa_{q_1} \sqcap^\sharp \kappa_{q_2}$:

$V_0 \cdots V_9$	$V_{10}V_{11}$	V_{12}	V_{13}	$V_{14}V_{15}$	V_{16}	V_{17}
$\perp \cdots \perp$	12 Joe	Smith	2265	$\perp$ $\perp$	$\perp$	$\perp$
·	13 Sally	Jones	2265	$\perp$ $\perp$	$\perp$	$\perp$
·	14 John	Smith	2265	$\perp$ $\perp$	$\perp$	$\perp$
·	$\perp$ Peter	Walters	2265	$\perp$ $\perp$	$\perp$	$\perp$
·	$\perp$ $\perp$	$\perp$	$\perp$	13 Sally	Jones	8792
·	$\perp$ $\perp$	$\perp$	$\perp$	14 John	Smith	8792
$\perp \cdots \perp$	$\perp$ $\perp$	$\perp$	$\perp$	$\perp$ Peter	Walters	8792

κ'_{L_1}:

V_0	V_1	V_2	V_3
12	Joe	Smith	2265

We now add back all possibilities that are not affected by the rule

$$L_1^{\sharp'} := \bar{\kappa}_{L_1} \sqcap^\sharp \kappa'_{L_1}$$

to obtain a refined approximation $L_1^{\sharp'}$ for the realm of possible combinations for the entities that are associated with the subhypothesis L_1.

To show that this inference is monotone observe that

$$\kappa^0_{L_1} \sqsubseteq^\sharp \kappa'_{L_1}$$

and thus we have

$$L_1^\sharp = \bar{\kappa}_{L_1} \sqcap^\sharp \kappa^0_{L_1} \sqsubseteq^\sharp \bar{\kappa}_{L_1} \sqcap^\sharp \kappa'_{L_1} = L_1^{\sharp'}$$

which demonstrates that the new set of possibilities $L_1^{\sharp'}$ is a monotone improvement to the initial set of possibilities $L_1^\sharp$ in the Smyth ordering.

In parallel to the consistent interpretation, we put the derived information back into the database by "unnaming" $L_1^{\sharp'}$ and combining it into the database. Thus if L_1 is associated with the entity or relationship set F which is described by the sandwich (A, B) in the current database instance we assign

$$A := A \sqcup^\sharp \text{unname}(L_1^{\sharp'})$$

which is obviously the same as

$$A := \text{unname}(L_1^{\sharp'})$$

It is important to note that this inference defines an inflation on database instances.

As before, we define a similar inference for each L_i in p. The collection of all such inferences is the complete interpretation for the rule R.

Since in our example L_2 and L_3 do not produce any useful inferences, the complete interpretation of R_0 on the database of Table I is:

Employee♯

Id	*FName*	*LName*	*Telephone*
12	Joe	Smith	2265
13	Sally	Jones	$\perp$
14	John	Smith	$\perp$
$\perp$	Peter	Walters	$\perp$

4.7. *Structural Inferences and Constraints*

In addition to the inferences derived from the user-specified rules we can also make use of the E-R database schema to derive inferences and constraints.

The structure and semantics of an E-R database by itself entails the correctness of several integrity constraints and constructive inferences, even if no explicit rules are declared. We therefore define *monotone meta-rules* and meta-constraints which make use of the following semantic knowledge:

1. Any *sandwich* datastructure (A, B) approximates a *set*, where A is a complete approximation that is possibly "too large" but never "too small," and B is a consistent approximation that is possibly "too small" but never "too large." This gives rise to the *promotion inference.*

2. An *object identity* always corresponds to a unique individual in a given set. This gives rise to the *identity enforcement inferences.*

3. Relationship sets always relate individuals in the participating entity sets. This gives rise to *domain restrictions* and *existence inferences.*

The meta-constraints defined in this section automatically apply to each set approximation and relationship.

The *promotion inferences* assert that there is a connection between the consistent information and the complete information in a sandwich. As described in Section 2, every sandwich S is equivalent in meaning to the more explicitly informative sandwich $P(S)$, if the latter is defined. Furthermore if $P(S)$ is not defined then S cannot approximate any set. Hence, for every sandwich S in the database we introduce the following implicit monotone inference:

$$\frac{SetLabel \Rightarrow S}{SetLabel \Rightarrow P(S)}.$$

An integrity error is detected whenever $P(S)$ is not defined.

We can further improve the consistent side of a sandwich by considering the intended meaning of the *object identities.* Suppose that we have an approximation

$S=(A, B)$ to the set *Employees* in which the consistent approximation B contains the following two descriptions:

$$x=[Id \Rightarrow 12;\ Firstname \Rightarrow \text{'Jon'}]$$

and

$$y=[Id \Rightarrow 12;\ Lastname \Rightarrow \text{'Simon'}].$$

Since an object identity always corresponds to a unique individual and x and y are in the consistent approximation of *Employee*, it follows that x and y can be replaced by the combined description

$$x \sqcup y=[Id \Rightarrow 12;\ Firstname \Rightarrow \text{'Jon'};\ Lastname \Rightarrow \text{'Simon'}].$$

Note that in other situations there may be two descriptions in B that share the same object identity but do not have a join. For instance, if we had

$$x=[Id \Rightarrow 12;\ Firstname \Rightarrow \text{'Jon'}]$$

and

$$y=[Id \Rightarrow 12;\ Firstname \Rightarrow \text{'Simon'}]$$

there is no way to combine them into a single record. If such a situation occurs, the database must be incorrect since every element of B is considered to represent *known* individuals, and no individual can have two contradictory descriptions. Since the semantics for the database does not tell us how to resolve the conflict, the only sensible response is to raise an error.

We abstract the above reasoning into the following identity enforcement constraint

$$\frac{SetLabel \Rightarrow (A, B) \quad x, y \in B \quad x.id=y.id}{SetLabel \Rightarrow (A, B \sqcup^{b} \{x \sqcup y\})}$$

which results in an error if the join does not exist.

In a similar manner, suppose that there is a $z \in B$ which asserts the existence of an employee 3:

$$z=[Id \Rightarrow 3;\ Firstname \Rightarrow \text{'Ann'}]$$

and that there is a $w \in A$ hypothesizing that it is within the realm of possibility that employee 3 has phone extension 6483

$$w=[Id \Rightarrow 3;\ Phone \Rightarrow 6483].$$

Since employee 3 is known to be represented by z, we can correctly replace $w \in A$ with the improved approximation

$$w \sqcup z=[Id \Rightarrow 3;\ Firstname \Rightarrow \text{'Ann'};\ Phone \Rightarrow 6483].$$

In cases where no such join exists, the *possibility* w contradicts *known information* z, and hence should be eliminated. Note that we cannot eliminate w since that would be contrary to the semantics of the consistent approximation B.

More generally, we have the following identity enforcement inference

$$\frac{SetLabel \Rightarrow (A, B) \quad z \in B \quad w \in A \quad w \sqcup z \text{ is defined} \quad w.id = z.id}{SetLabel \Rightarrow ((A - \{w\}) \cup \{w \sqcup z\}, B)}$$

or alternatively

$$\frac{SetLabel \Rightarrow (A, B) \quad z \in B \quad w \in A \quad w \sqcup z \text{ is not defined} \quad w.id = z.id}{SetLabel \Rightarrow ((A - \{w\}), B)}$$

thus improving or eliminating possible representations for individuals for whom some information is known.

In addition to these local constraints, we can derive other constraints by considering the connection between an entity set and a relationship involving that set. The first inference hinges on the observation that any known relationship requires the existence of the objects declared to be related (within the appropriate entity sets).

For example, using the E-R database of Section 3.3, let the sandwich that approximates the *Employee* set be $S = (A, B)$ as before, and the sandwich that approximates the *Emp_Dept* set be $S' = (A', B')$. To illustrate the *existence inferences*, suppose that B' (the set of known *Emp_Dept* relationships) contains an entry asserting that employee 26 works for department 34:

$$[Emp \Rightarrow 26; Dept \Rightarrow 34].$$

Further, suppose that there is no employee 26 described in B (the set of known employees). Since any meaningful known relationship must always relate known entities, we can infer that employee 26 must be a known employee, even though no other information about employee 26 can be immediately inferred. Mirroring this reasoning, we can correctly replace B with

$$B \sqcup^{\flat} \{[Id \Rightarrow 26]\}.$$

More generally, we have the following monotone inference:

$$\frac{EntityName \Rightarrow (A, B) \quad RelName \Rightarrow (A', B') \quad \Phi(\text{EntityName, RelName}, L)}{EntityName \Rightarrow (A, (B \sqcup^{\flat} rename_{L \mapsto \text{id}} \pi_L B'))}.$$

Where $\Phi(\text{EntityName, RelName}, L)$ indicates that the entity EntityName participates in relationship RelName in the "role" labeled by L, and $rename_{L \mapsto \text{id}}$ changes the name of L to *id*.

Dual to the existence inferences are the *domain restrictions*, which improve the complete approximations of a relationship based on the complete approximations for the entity sets it relates. For example, suppose that the complete approximation for the *Emp_Dept* relationship, A', postulates that employee 10 might work for department 6:

$$[Emp \Rightarrow 10;\ Dept \Rightarrow 6].$$

If the complete approximation for *Employee* contains no employee with identity 10 and no null identities, we can eliminate the above possibility. Mirroring this reasoning, we can correctly infer

$$A' := A' - \{y\}.$$

Generalizing the above reasoning, we get

$$\frac{EntityName \Rightarrow (A, B) \quad RelName \Rightarrow (A', B') \quad \Phi(\text{EntityName, RelName}, L)}{RelName \Rightarrow ((A' \sqcup^{\sharp} \text{rename}_{\text{id} \mapsto L} \pi_{\text{id}} A), B)}.$$

This inference will, among other things, eliminate hypothetical relationships relating identities which cannot exist. If E has more than one role in R, the roles must be considered separately, each role generating a domain restriction. Note that if the null identity occurs in A the above action will cause no change.

The collection of all promotion inferences, identity enforcement inferences, existence inferences, and domain restrictions form a monotone inference system that reflects the constraints inherent in an entity–relationship schema. Observant readers may note that other *non-constructive* inferences could be justified, if we apply the appropriate interpretation to the database. However, as mentioned in the beginning of this section, we do not consider these inferences.

4.8. *The Meaning of a Program*

All of these inferences and constraints contribute to the meaning of a deductive program under our system.

A deductive program for our system is an E-R schema augmented with a set of rules

$$P = \{R_0, ..., R_l\}.$$

To each of these rules we attribute an integrity interpretation, and both consistent and complete interpretations. From the E-R schema we derive implicit constraints and inferences. Given a database approximation Δ we say that Δ is consistent with the program if it satisfies all the integrity constraints and the implicit constraints.

To actively infer better information from Δ, we non-deterministically apply the implicit inferences, as well as the complete and consistent inferences generated by the rules until we either reach a fixed point Δ' or detect an anomaly. If no anomaly is detected, we say Δ' is the model of P derived from Δ.

PROPOSITION 14. *For any fixed initial database Δ, either the program will identify an anomaly under any possible sequence of inferences, or the program will derive a unique minimal fixed point Δ'.*

Proof. This is a consequence of the fact that all of the inferences define inflationary operators on the underlying database if no anomalies occur. To make these partial operators totally defined, we introduce an artificial database instance T which denotes database inconsistency such that $\mathrm{T} \sqsupseteq X$ for all database instances X, and whenever a database instance is inconsistent we set it to T. In this case all inferences become totally defined inflations and it follows from Proposition 6 in Section 2 that the least fixed point is unique. ▌

Given a program P, we generate a set of inferences and integrity interpretations from P. To perform a query, the user provides the system with a pattern. On a given database Δ, we define the meaning of such a pattern to be the matches for the pattern within the database Δ'—but only in the case where the saturated database Δ' is consistent with the integrity constraints.

This completes the semantic characterization of our system.

5. SUMMARY AND ANALYSIS

The interpretation of rules in our system yields an intuitively appealing, constructive inference mechanism. While the system is not complete for a classical interpretation of the rules, in this section we show that our system can emulate Datalog (without negation) and the relational algebra, in a natural way. Furthermore, while there is no *explicit* negation (literals cannot be negated), a form of monotone negation is captured in the semantics of the rules.

After discussing our treatment of negation, we show how our system can emulate Datalog and the relational algebra, and derive a bound on the time complexity for an implementation of the system.

5.1. *Monotone Negation*

A form of monotone negation is hidden in both the consistent and complete interpretations of a rule, such as

$$\underbrace{L_1 \text{ and } L_2 \text{ and} \cdots L_m}_{\text{(hypothesis)}} \text{ implies } \underbrace{q_1 \text{ or } q_2 \text{ or} \cdots \text{or } q_k}_{\text{(conclusions)}}.$$

Recall that in the consistent interpretation of a rule no instance of any conclusion such as q_1 will be inferred to be true unless the remaining q_i's are necessarily false. This is an example of the use of negative information to infer positive information.

In a more subtle fashion, negative information is *inferred* from the complete interpretations; complete instances for a literal in the hypothesis which necessarily match the hypothesis but do not match any possible instance for the conclusions are eliminated—i.e., inferred to be false. More intuitively, the realm of possibility for the conclusions is used to refine the realm of possibility for the hypothesis.

To make this discussion clear, consider the example rule from Section 4.1

$$\textit{DBS-Enthusiast} \Rightarrow \{[\textit{Firstname} \Rightarrow X;\ \textit{Lastname} \Rightarrow Y]\} \text{ implies}$$

$$\textit{OS-Enthusiast} \Rightarrow \{[\textit{Firstname} \Rightarrow X;\ \textit{Lastname} \Rightarrow Y]\} \text{ or}$$

$$\textit{PL-Enthusiast} \Rightarrow \{[\textit{Firstname} \Rightarrow X;\ \textit{Lastname} \Rightarrow Y]\}.$$

In our consistent inferences, when we concluded that "Ella Taylor" was a PL-Enthusiast, we used the fact that she could not possibly be an OS-Enthusiast since there was no compatible entry in the complete approximation for OS-Enthusiast. We did not conclude that "Burt —" was a PL-Enthusiast since there was a compatible entry "Burt Charles" in the complete approximation for OS-Enthusiast. Thus we used implicit negative information from the complete descriptions to infer consistent information.

Furthermore, in the complete inferences we inferred that the entry "Johnson" could be removed from the complete description for DBS-Enthusiasts because there was no possible compatible entry in the complete descriptions for either OS-Enthusiasts or PL-Enthusiasts. We also replaced the entry "Taylor" with the only possible compatible conclusion entries—"Josiah Taylor" and "Ella Taylor." Both these inferènces refined the realm of possible DBS-Enthusiasts—for instance "Jon Taylor" and "Josiah Johnson" were both implicitly excluded from the realm of possible DBS-Enthusiasts. Thus these inferences implicitly concluded false information.

In spite of this use and derivation of negative information, our system is *monotone* both in a technical and intuitive sense. Technically, the refined approximation to a set is always a monotone improvement in the sandwich power-domain. Intuitively, if program P_1 has more rules than program P_2 then in the absence of anomalies P_1 will infer a strictly stronger set of facts than P_2 from any given database—P_1 will infer more facts to be true and more facts to be false (not possible). It will never be the case that an inference made by P_2 will contradict inferences made by P_1. In a similar sense, better information in the initial database will always result in better conclusions by the inference system—and never contradictory conclusions (again in the absence of anomalies).

This form of monotone negation stands in contrast to various other non-monotonic inference strategies. Non-monotonic strategies postulate that "If the rules fail to show that a fact is true, then the intention of the program is that the fact is false." Although this approach is simple and has advantages, it is interesting to consider a situation where this is *not* the intention of the program. What if the user does not intend to assign any significance to the failure of a proof, but still wants to make meaningful reference to the false facts? By separating the consistent information from the complete information our system allows a notion of falseness that is separate from the notion of truth—but these notions are connected by the consistency constraints.

5.2. *Datalog*

It is easily seen that our system contains the power of datalog, since our rules are proper generalizations of datalog rules. A datalog rule corresponds to the consistent interpretation for the special subclass of rules of the form

$$p \text{ implies } L_1,$$

where p is a pattern not involving the need for object identities, and L_1 is a single conclusion literal.

Note that datalog without negation involves no negative information whatsoever—positive facts are used to infer more positive facts, and no fact is ever inferred to be false. Hence the complete sides of the sandwiches in this emulation (which implicitly represent negative information) must be initialized to $\{\bot\}$ (representing no information). The data for a datalog program (positive ground literals) are represented by initial consistent side information in the sandwiches of the emulation.

For example, consider the datalog program that has three initial ground literals each representing an hourly flight by a given airline

flight('Chicago', 'Denver').

flight('Denver', 'Oakland').

flight('Denver', 'Atlanta').

with the interpretation that there is a flight from Chicago to Denver, one from Denver to Oakland, and one from Denver to Atlanta. Furthermore suppose the program includes a single rule, insisting that flights that stop over in an intermediate city are also legitimate flights:

$$\text{flight}(X, Y) :- \text{flight}(X, Z), \text{flight}(Z, Y).$$

This program translates directly into our approach as

$$\mathit{flight} \Rightarrow \{[\mathit{Source} \Rightarrow X; \mathit{Destination} \Rightarrow Z], [\mathit{Source} \Rightarrow Z; \mathit{Destination} \Rightarrow Y]\}$$
$$\text{implies } \mathit{flight} \Rightarrow \{[\mathit{Source} \Rightarrow X; \mathit{Destination} \Rightarrow Y]\}$$

and improves the initial sandwich estimate for *flight*

Source	*Destination*	*Source*	*Destination*
—	—	Chicago	Denver
		Denver	Oakland
		Denver	Atlanta

to

Source	*Destination*	*Source*	*Destination*
—	—	Chicago	Denver
		Denver	Oakland
		Denver	Atlanta
		Chicago	Oakland
		Chicago	Atlanta

5.3. *Relational Algebra*

To demonstrate that the relational algebra can be expressed in our system, it suffices to show that each of the relational operators—projection, selection, union, subtraction, and join—can be translated into our rules. However, the demonstration is more complex than the demonstration for datalog since relational subtraction forces us to use the complete sides of the sandwiches and the complete interpretations for the rules.

Suppose we have two relations, r and s, and each defined over attributes *FName* and *LName*. To express the relational assignment

$$q := r \cup s$$

in terms of our rules, we start with an entity Q to which we assign the "totally undefined" sandwich value $(\{\bot\}, \{\ \})$, and we create the completely defined sandwich approximations $R \Rightarrow (r, r)$ and $S \Rightarrow (s, s)$ to represent r and s, respectively. Any tuple in either r or s must also an element of q, and thus we assert

$$[R \Rightarrow \{[\mathit{FName} \Rightarrow X;\ \mathit{LName} \Rightarrow Y]\}]$$
$$\text{implies } [Q \Rightarrow \{[\mathit{FName} \Rightarrow X;\ \mathit{LName} \Rightarrow Y]\}],$$

and

$$[S \Rightarrow \{[\mathit{FName} \Rightarrow X;\ \mathit{LName} \Rightarrow Y]\}]$$
$$\text{implies } [Q \Rightarrow \{[\mathit{FName} \Rightarrow X;\ \mathit{LName} \Rightarrow Y]\}].$$

Furthermore, since each element of q must be in either r or s, we assert

$$[Q \Rightarrow \{[\mathit{FName} \Rightarrow X;\ \mathit{LName} \Rightarrow Y]\}]$$
$$\text{implies } [R \Rightarrow \{[\mathit{FName} \Rightarrow X;\ \mathit{LName} \Rightarrow Y]\}] \text{ or}$$
$$[S \Rightarrow \{[\mathit{FName} \Rightarrow X;\ \mathit{LName} \Rightarrow Y]\}].$$

It is easy to see that whenever entities R and S are approximated by maximal sandwiches (r, r) and (s, s), Q will be inferred to be a maximal sandwich that corresponds to the union of R and S. It is not difficult to generalize this construction for arbitrary unions.

In a similar way we can emulate the other positive relational operators: selection, projection, and join. The only difficulty is relational subtraction, which we illustrate by an example.

To emulate the assignment

$$q' := r - s$$

we again start by establishing a new entity-set Q' to have a null approximation and R and S to have maximal approximations as before. We then note that after the assignment anything in Q' must be in R. Thus

$$[Q' \Rightarrow \{[FName \Rightarrow X; LName \Rightarrow Y]\}]$$
$$\text{implies } [R \Rightarrow \{[FName \Rightarrow X; LName \Rightarrow Y]\}].$$

Furthermore anything in r must be in q' or s:

$$[R \Rightarrow \{[FName \Rightarrow X; LName \Rightarrow Y]\}]$$
$$\text{implies } [Q' \Rightarrow \{[FName \Rightarrow X; LName \Rightarrow Y]\}] \text{ or}$$
$$[S \Rightarrow \{[FName \Rightarrow X; LName \Rightarrow Y]\}].$$

The trick is that we must now insist that s and q' are disjoint:

$$[S \Rightarrow \{[FName \Rightarrow X; LName \Rightarrow Y]\}] \text{ and } [Q' \Rightarrow \{[FName \Rightarrow X; LName \Rightarrow Y]\}]$$
$$\text{implies } \mathbf{False}.$$

False is a constant entity set which always denotes the empty set (an unmatchable pattern). Whenever R and S are approximated by maximal sandwiches, the inferred approximation for Q' will be a maximal sandwich corresponding to the relational expression $r - s$ (again, ignoring irrelevant object identities).

In this way, any relational algebra expression can be compiled into a sequence of rules that compute a maximal sandwich denoting the resulting relation from maximal sandwiches denoting the input relations.

5.4. *Time Complexity*

PROPOSITION 15. *For a fixed database schema and a fixed set of rules our system admits a polynomial implementation strategy in the size of the database.*

Proof. To justify this claim, we observe that our system has no form of invented values or function symbols. Thus our inference system navigates a search space Σ constructed from a fixed number of record types and constants occurring in the underlying database. It is clear that the total number of records occurring in Σ is a polynomial in the number of constants available in the initial database, and hence it is a polynomial in the size of the database.[1]

[1] We emphasize that this polynomial outer bound exists because we identify the database with a flat E-R representation—which, for example, prevents us from computing the powerset of a set. A less restrictive notion of complex objects might allow such computations—with a corresponding increase in complexity.

Now recall that each rule is interpreted as an integrity constraint and a fixed number of inferences. From our discussion, it is clear that each of these interpretations has an obvious (but inefficient) implementation as an operator of polynomial complexity. To complete the proof, we need to show that these operators need be applied only a polynomial number of times before we detect an anomaly or reach the fixed point where the rules have saturated. To see this, note that each inference need only be executed when one of its inputs has changed; hence if we can show that the database can only go through a polynomial number of changes, we are done.

Consider an arbitrary consistent set B within the initial database. Changes are made to B only by the insertion of a new record which is not dominated by any record already within B. Since no record is ever deleted from B until it is dominated by some other record, the total number of such introductions is bounded by the total number of records occurring in Σ.

Similarly, consider an arbitrary complete set A within the initial database. Changes are made to A only by replacing a non-empty set of existing elements of $\alpha \subseteq A$ by a co-chain of records $\alpha' \sqsupseteq^{\sharp} \alpha$ which may be empty. Since no record need ever be introduced twice in such a process, the total number of introductions is also bounded by the number of possible records.

Since the database is composed of a fixed number of complete and consistent sets that can only go through a polynomial number of changes, the number of applications of the inferences and integrity checks need only be applied a polynomial number of times; hence the system admits an implementation of polynomial complexity. ▮

It should be noted that the implementation suggested by the proof given above is much less efficient that what can be achieved, and that the search for optimization techniques for our system is an interesting research direction.

5.5. *Conclusions*

We have defined a database system that includes the notions of object identities, partial information, complex objects, integrity constraints, and inference rules. This system is built upon the sandwich powerdomain construction, and uses an entity–relationship conceptual modeling strategy as a unifying paradigm. The system has been shown to

- contain the power of datalog;
- contain the power of relational algebra;
- include a form of monotone negation; and
- admit an implementation strategy of polynomial time complexity.

Combined with the intuitively appealing interpretations that users can apply to both the rules and the data in the system, these properties argue in favor of our approach.

In future work, we plan to devise optimization strategies to improve the performance of the system, and develop a prototype. We also hope to study the relationship of our work to other extensions of Datalog that deal with sets such as those suggested in [24, 25].

ACKNOWLEDGMENTS

We acknowledge the many comments, corrections, and criticisms that have improved this paper. In particular we thank Serge Abiteboul, Tomasz Imielinski, Atsushi Ohori, Samson Abramsky, Elsa Gunter, Carl Gunter, and the referees.

REFERENCES

1. 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, 1984," pp. 119–132.
2. F. BANCILHON AND S. KHOSHAFIAN, Calculus for complex objects, *in* "Proceedings ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1986."
3. A. ROTH, H. KORTH, AND A. SILBERSCHATZ, "Extended Algebra and Calculus for $\neg$ 1NF Relational Databases," Technical Report TR-84-36, Department of Computer Sciences, The University of Texas at Austin, 1985.
4. Z. OZSOYOGLU AND L. YUAN, A new normal form for nested relations, *ACM Trans. Database Systems* **12** (1987), 111–136.
5. H. JAESCHKE AND J. SCHEK, Remarks on the Algebra on non first normal form relations, *in* "Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems," pp. 124–138, 1982.
6. H. AÏT-KACI, "A Lattice Theoretic Approach to Computation based on Calculus of Partially Ordered Type Structures," Ph.D. thesis, University of Pennsylvania, 1985.
7. 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.
8. D. SCOTT, Domains for Denotational Semantics, *in* "International Conference on Autonoma, Languages and Programming, July 1982."
9. P. BUNEMAN AND A. OHORI, Using powerdomains to generalize relational databases, *Theoret. Comput. Sci.*, in press.
10. M. SMYTH, Power domains, *J. Comput. System Sci.* **16** (1978), 23–36.
11. T. IMIELINSKI AND W. LIPSKI, Incomplete information in relational databases, *J. Assoc. Comput. Mach.* **31** (1984), 761–791.
12. A. MOTRO, Integrity = validity + completeness, *ACM Trans. Database Systems* **14** (1989), 480–502.
13. Z. PAWLAK, Rough sets, *Internat. J. Comput. Inform. Sci.* **11** (1982), 341–366.
14. A. OHORI, Semantics of types for database objects, *Theoret. Comput. Sci.*, in press. [Special issue dedicated to 2nd International Conference on Database Theory]
15. B. COURCELLE, Fundamental properties of infinite trees, *Theoret. Comput. Sci.* **25** (1983), 95–169.
16. P. BUNEMAN, A. JUNG, AND A. OHORI, Using powerdomains to generalize relational databases, *Theoret. Comput. Sci.*, to appear.
17. P. BUNEMAN, S. DAVIDSON, AND A. WATTERS, A semantics for complex objects and approximate queries, *in* "Proceedings ACM SIGACT-SIGMOD Symposium on Principles of Database Systems," pp. 305–314.

18. R. Jagadeesan, P. Panangaden, and K. Pingali, A fully abstract semantics for a functional language with logic variables, *in* "Proceedings IEEE Symposium on Logic in Computer Science 1989," pp. 294–303.
19. P. Chen, The entity–relationship Model: Towards a unified view of data, *ACM Trans. Database Systems* 1 (1976), 9–36.
20. G. Kuper and M. Vardi, A new approach to database logic, *in* "Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems," pp. 86–96, 1984.
21. D. Maier, A logic for objects, *in* "Proceedings of Workshop on Deductive Databases and Logic Programming," 1986.
22. V. Markowitz and A. Shoshani, On the correctness of representing extended entity–relationship structures in the relational model, *in* "Proceedings ACM Sigmod Conference, Portland Oregon, 1989."
23. D. Maier, "The Theory of Relational Databases," Computer Science Press, 1983.
24. S. Abiteboul and S. Grumbach, "Advances in Database Programming Languages," Addison–Wesley, Reading, MA, 1990.
25. S. Naqvi and S. Tsur, "A Logical Query Language for Data and Knowledge Bases," Freeman, San Francisco, 1989.