AN APPROACH TO THE ORGANIZATION OF TAXONOMIES

by

CRAIG DAVID BISHOP

B.Sc. University of British Columbia

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department of Computer Science

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

March, 1981

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my department or his representative. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Craig Bishop

Abstract

The ISA hierarchy used by present day inferential database systems is deficient in that it does not represent a variety of domain relationships (type relationships). Such hierarchies associate explicitly defined sets with the leaves of a tree. Non-leaf nodes in the tree inherit members from their child nodes. In keeping with the use of sets, this paper gives motivation for having type relationships other than subset. Included are union, intersection and a disjointness condition.

A formalism for the typed database (TDB) is given, using the monadic first order predicate calculus as its theoretical basis. Non-unit formulae represent intensional (general) information about the world being represented and unit ground clauses represent extensional (specific) information. Predicates represent types, and constant symbols represent set members. This is connected to the set concept via predicate extensions. The extension is that set of constant symbols which are provable as arguments to the given predicate.

Given the so-called concreteness condition and consistency of the TDB, the desired set theoretic relationships (union, intersection and disjointness) of predicate extensions follow. This strengthens the link between the formalism of the predicate calculus and the more natural set representation.

A canonical form of a TDB is shown that admits an appropriate machine representation. Using this is can be determined if a constant symbol as an

argument to a given predicate is provable (domain membership) in constant time. An update algorithm is developed and is shown to be correct in that it maintains concreteness and consistency. Thus the TDB is shown to be a practical generalization of the ISA hierarchy but is considerably more expressive.

Table of Contents

Table of Contents

Acknowledgements

I wish to thank Dr. Raymond Reiter of the University of British Columbia for his perceptive and critical analysis of the ideas that went into this paper. His dedication and insight into inferential database theory has been a continuing source of enlightenment.

I would also like to thank Bell Northern Research of Ottawa for the use of their word processing facilities in the preparation of this manuscript.

Section A Introduction

In recent works on relational database systems, especially as they arise in artificial intelligence, the need for a database of domains (using Codd's definition of Domain[3]) has become apparent. In simple data processing applications, the acceptance or rejection of data in a domain is based on a decision procedure for that domain such as correct alpha/numeric form or membership in a predefined set. This makes the primary assumption that each domain is independant of all others. This may be realistic for domains in which each entry of one domain is more or less unrelated to other domains (like numeric ones). However, work in artificial intelligence and inferential database systems acknowledges the dependancies amoung domains. Winograd led the way by using an ISA hierarchy to relate domains of objects and their properties. eg. a MOVEABLE-OBJECT ISA PHYSICAL-OBJECT[15].

McSkimin and Minker formalize the ISA hierarchy by defining a semantic graph as a labelled tree in which the root node is the universal category (domain) and the leaves are primitive categories[7,8,9]. In their formalism the elements, which are associated with the primitive categories, are inherited by the parent categories. As can be seen in their examples these trees tend to be "leafy" with many categories, some with unnatural names (such as non/human/male/mam). There is much duplication of category names but no mechanism is given for insuring that duplicate categories contain the same elements.

Also, McSkimin and Minker define a partitioning (disjointness condition between categories) without defining an update algorithm that will maintain the partition. In short, though they define the ISA heirarchy, they do not include a mechanism for enforcing data consistency.

Reiter has adopted the idea of a typed database formalized in the monadic predicate calculus[10,11]. He replaces the idea of an element being a member of a domain with the proof theoretic notion of provability. Here the domain is a monadic predicate in which the element is its argument. The structure, mathematics and algorithms for this typed database are beyond the scope of his paper.

It is the purpose of this paper to propose a structure for Reiter's typed database and develop the necessary mathematics and algorithms.

A domain in Codd's terminology will hereafter be referred to as a TYPE. An element will be referred to as an INSTANCE. For example, HUMAN is a TYPE for which JOHN is an INSTANCE. JOHN is said to have the property HUMAN. A type database (TDB) is then a logical theory in which groups of TYPEs (and hence INSTANCES) satisfy certain mathematical relationships called type relationships.

In order for a TDB to be practical it must satisfy certain conditions:

1) It must be descriptively rich, capturing type relationships other than simple set containment.

2) It must be structured so as to encourage only the most natural information content.

3) It must admit algorithms that are computationally efficient in both time and space.

4) It must admit updates in such a way as to prevent a user from violating a type relationship. (ie. enforce integrity constraints)

Section B motivates the inclusion of type relationships other than set containment. Section C then defines a TDB. Condition 2 is addressed in section C which defines an acyclic TDB and introduces a graphic notation. Structuring is completed in section D by imposing the so-called concreteness condition on TDBs. With this and consistency all the desired set theoretic properties of the TDB follow. Section E is devoted to an algorithm for reducing a given TDB to one which is efficient in both time and space. The final section details an update algorithm which ensures that concreteness and consistency of the TDB are maintained during updates.

Section B Motivation

To see why a more robust set of type relationships than simple set containment is important consider the following statements:

1) JANE is a WOMAN; hence JANE is a FEMALE (ISA)

2) JANE is a FEMALE; hence JANE is a WOMAN
   or JANE is a GIRL (but not both)

3) a) JOHN is a BACHELOR; hence JOHN is a MAN (ISA)
   b) JOHN is a BACHELOR; hence JOHN is SINGLE (ISA)

4) JOHN is a MAN and JOHN is SINGLE; hence JOHN is a BACHELOR

Statements 3a and 3b bear a resemblance to statement 1 in that both display set containment. However, in view of statements 2 and 4 we see that somehow FEMALE is composed of WOMAN and GIRL - WOMAN and GIRL being disjoint TYPEs. In contrast MAN and SINGLE intersect to give BACHELOR. The standard ISA hierarchy does not provide the formal equivalent of statements 2 and 4.

An inference of type 2 is important in that we would not want JANE to have the property FEMALE without her having one of the properties WOMAN or GIRL. Similarly we would not want JANE to have both the properties WOMAN and GIRL. As seen in the following sections, "concreteness" satisfies the former condition, consistency the latter.

The type relationship of type 4 is important when we want to define a TYPE as having INSTANCES which are common to a set of TYPEs. In an airline reservation system JOHN might have the properties NON-SMOKING, COACH,

MOVIE-WATCHING and 747 from which the system could infer that JOHN is a member of CABIN-B.

The way the above statements pair suggests the need for only two classes of type relationships:

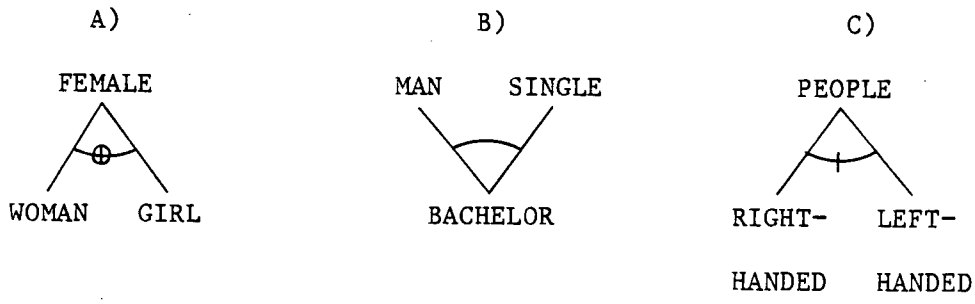
A) The TYPE FEMALE is equivalent to the disjoint union of the TYPEs WOMAN and GIRL

B) The TYPE BACHELOR is equivalent to the intersection of the TYPEs MAN and SINGLE

The union in A need not be disjoint, for example:

C) The TYPE PEOPLE is equivalent to the union of the TYPEs LEFT--HANDED and RIGHT-HANDED (allowing for the ambidextrous)

Graphically these are represented as:



Set theoretically these can be represented as:

A) FEMALE = WOMAN $\oplus$ GIRL

B) BACHELOR = MAN $\cap$ SINGLE

C) PEOPLE = RIGHT-HANDED U LEFT-HANDED

In the graphical representation, the more restricted TYPE is drawn below the TYPE of which it is a subset. The four original statements can now be written in terms of sets:

1) JANE ε WOMAN implies JANE ε FEMALE

2) JANE ε FEMALE implies JANE ε WOMAN $\oplus$ GIRL

3) JOHN ε BACHELOR implies JOHN ε MAN $\cap$ SINGLE

4) JOHN ε MAN $\cap$ SINGLE implies JOHN ε BACHELOR

Section C Formal Preliminaries

The set notation of the previous section is convenient for visualizing the TYPE relationships. The first order predicate calculus, however, offers a well developed proof theory[2,13]. For this reason we shall develop our results within a first order framework. Refer to appendix A for the interpretation given to special symbols used in what follows.

Definition TYPE Database (TDB)

A TDB is a 4-tuple (**C**, **P**, **I**, **E**) satisfying:

1) **C** is a finite set of constants.

2) **P** is a finite set of unary predicate signs, called TYPES.

3) **E** is a finite set of atomic formulae of the form Pc where

$$P \in \mathbf{P} \quad \text{and} \quad c \in \mathbf{C}.$$

E is called the extension of the TDB.

4) **I** is a finite set of well formed formulae of the form:

i) $(x)\ Px \equiv P_1x \wedge P_2x \wedge \ldots P_nx$ or

ii) $(x)\ Px \equiv P_1x \vee P_2x \vee \ldots P_nx$

where $[P, P_1, P_2, \ldots P_n] \subset \mathbf{P}$.

In the event that **I** contains a formula of the form (ii), it may also contain all of the formulae:

iii) $(x) \sim (P_ix \wedge P_jx)$ for each $1 <= i < j <= n$.

In which case the formula (ii) is said to be exclusive.

I is called the intension of the TDB.

For Example:

(**C**, **P**, **I**, **E**) is a TDB where:

C = [john, mary] (INSTANCES)

P = [FEMALE, WOMAN, GIRL, MAN, SINGLE, BACHELOR] (TYPES)

I = [(x) BACHELOR x $\equiv$ MAN x $\wedge$ SINGLE x,

(x) FEMALE x $\equiv$ WOMAN x v GIRL x,

(x) $\sim$ (WOMAN x $\wedge$ GIRL x)] (intension)

E = [WOMAN mary, MAN john, SINGLE john] (extension)

Notation:

1) For brevity, the variable 'x' will be dropped where understood.
2) (**C**, **P**, **I**, **E**) will be referred to as T's constants, predicates, intension and extension where clarification is required.
3) Members of **C** will be written in lower case, members of **P** in upper case.
4) For reasons of notational convenience we sometimes write W ε **I** where W is actually a clause[13].
5) A TDB will be said to be consistent whenever **I** U **E** is.
6) When we write T $\vdash$ W we will mean **I** U **E** $\vdash$ W for any formula W.
7) R (U, V) will refer to the resolvents[13] of the formulae U and V.

With this definition of a TDB we can replace the notion of an INSTANCE being a member of a TYPE (john ε BACHELOR) with the predicate calculus notion of provability (**I** U **E** $\vdash$ BACHELOR john). However, we would still like to view a type as a set with certain set theoretic relationships.

Definition Extension

The extension of P ε P written $\|P\|_T$ is defined:

$$\|P\|_T \equiv [c \mid c \varepsilon C, \ I \cup E \vdash Pc]$$

The subscript on $\|P\|_T$ will be dropped where implicit. Hence we can write $c \varepsilon \|P\|$ for $T \vdash Pc$. The set theoretic relationships between extensions of the last section do not automatically follow. We will give conditions under which each of the relationships given do follow.

Acyclicity of a TDB

The idea of a TYPE viewed as an ISA hierarchy is firmly related to the ideas of generalization and restriction. Logically we would like to think of one type containing another (ie. $\|P\| \subset \|Q\|$). This leads to the notion of a hierarchy of TYPES, motivating the following acyclicity definition[6].

Definitions ALPHA, BETA and TAU

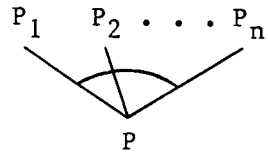
The relations $ALPHA_T$, $BETA_T$ and TAU_T are defined on a TDB, T, over P x P:

1) P $ALPHA_T$ Q iff $Q \equiv P_1 \vee P_2 \vee \ldots P_n \varepsilon I$ such that
   $P = P_i$ for some i, $1 <= i <= n$
2) P $BETA_T$ Q iff $P \equiv Q_1 \wedge Q_2 \wedge \ldots Q_n \varepsilon I$ such that
   $Q = Q_i$ for some i, $1 <= i <= n$
3) $TAU_T \equiv ALPHA_T \cup BETA_T$

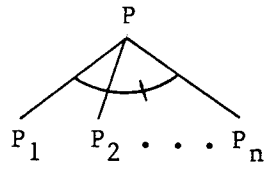
The subscripts on $ALPHA_T$, $BETA_T$ and TAU_T will be dropped where implicit. A TDB is said to be acyclic iff TAU^+ is irreflexive. ie.

$$P_0 \text{ TAU } P_1 \text{ and } P_1 \text{ TAU } P_2 \text{ and } \ldots P_{n-1} \text{ TAU } P_n$$
$$\text{implies not } P_n \text{ TAU } P_0 \quad (n > 0)$$

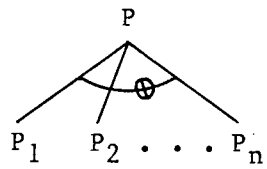
The graphical representation of Section B can be adopted with



representing (x) $Px \equiv P_1x \wedge P_2x \wedge \ldots P_nx$ (4i) and with



representing (x) $Px \equiv P_1x \text{ v } P_2x \text{ v } \ldots P_nx$ (4ii) and finally



representing (x) $Px \equiv P_1x \text{ v } P_2x \text{ v } \ldots P_nx$, (x) $\sim (P_ix \wedge P_jx)$ (4ii and 4iii). Diagram C.1 gives an example of this graphical representation of I.

By convention, if P TAU Q then P is directly below Q in the graph. Thus a TDB is acyclic when and only when its graph is acyclic. Throughout this paper the acyclicity of all TDBs is assumed.

The following lemma gives us the property of set containment desired:

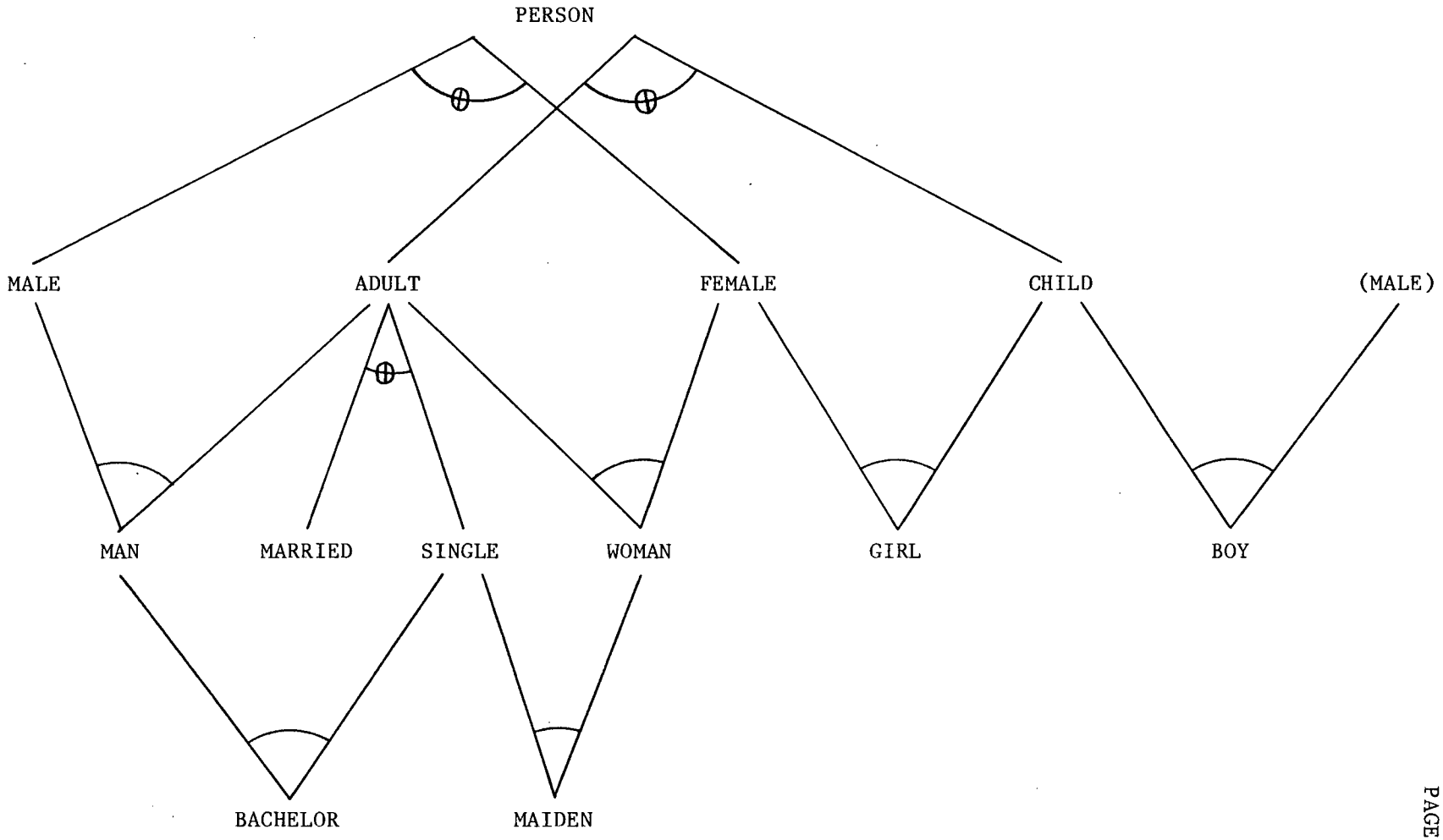


Diagram C.1, A Sample I

Lemma C.1 Generalization

If T is a TDB, $[P, Q] \subset \mathbf{P}$ and $P\ TAU^{+}\ Q$

then $T \vdash (x)\ Px \supset Qx$

Proof: By induction on n it is sufficient to prove

$P\ TAU\ Q$ implies $T \vdash (x)\ Px \supset Qx$

Case n = 1 $P\ TAU\ Q$

Subcase a $P\ ALPHA\ Q$ then $Q \equiv P_1 \vee P_2 \vee \ldots P_n \in I$

such that $P = P_i$, $1 <= i <= n$

In clausal form $\overline{P_i} \vee Q \in I$ or $(x)\ Px \supset Qx \in I$

Subcase b $P\ BETA\ Q$ then $P \equiv Q_1 \wedge Q_2 \wedge \ldots Q_n \in I$

such that $Q = Q_i$, $1 <= i <= n$

In clausal form $\overline{P} \vee Q_i \in I$ or $(x)\ Px \supset Qx \in I$

Case n > 1 $P\ TAU^{n}\ Q$

Induction assumption:

$P\ TAU^{n-1}\ Q_1$ implies $T \vdash (x)\ Px \supset Q_1x$ for any $Q_1 \in \mathbf{P}$

Since $P\ TAU^{n}\ Q$ then for some $Q_1 \in \mathbf{P}$,

$P\ TAU^{n-1}\ Q_1$ and $Q_1\ TAU\ Q$

Then $T \vdash (x)\ Px \supset Q_1x$ by induction assumption and

$T \vdash (x)\ Q_1x \supset Qx$ by case n = 1

Therefore $T \vdash (x)\ Px \supset Qx$ QED

Corollary C.1 Containment

If T is a TDB, $[P, Q] \subset P$ and P TAU$^+$ Q

then $\|P\| \subset \|Q\|$

In terms of the graph of I, we are assured that if a path exists between P and Q in a 'vertical' direction then every INSTANCE of the TYPE P is an INSTANCE of the TYPE Q. We can see now why acyclicity is the only practical choice for if P TAU$^+$ Q and Q TAU P then $\|P\| \subset \|Q\| \subset \|P\|$ or $\|P\| = \|Q\|$. This would certainly be of no use in a practical TDB.

The following lemma establishes the relationship between set intersection and part 4i in the definition of a TDB.

Lemma C.2 Intersection

For a TDB, T, if $P \equiv P_1 \wedge P_2 \wedge \ldots P_n \in I$

then $\|P\| = \|P_1\| \cap \|P_2\| \cap \ldots \|P_n\|$

Proof:

Case 1 $c \in \|P\|$ (or $T \vdash Pc$)

In clausal form $\overline{Px} \vee P_i x \in I$

for each i, $1 <= i <= n$

By resolution $T \vdash P_i c$ implying $c \in \|P_i\|$ or

$c \in \|P_1\| \cap \|P_2\| \cap \ldots \|P_n\|$

Case 2 $c \in \|P_1\| \cap \|P_2\| \cap \ldots \|P_n\|$

(or $T \vdash P_i c$ for each i, $1 <= i <= n$)

In clausal form $Px \vee \overline{P_1 x} \vee \overline{P_2 x} \vee \ldots \overline{P_n x} \in I$

By resolution $T \vdash Pc$ implying $c \in \|P\|$

Section D develops a parallel result relating the union of extensions to the formula $P \equiv P_1 \vee P_2 \vee \ldots P_n$.

Section D The Definite Component of a TDB

It has been recognized by Reiter[11] and Minker[9] that restricting intensional information to Horn clauses (clauses containing at most one positive literal) effectively controls the size of the proof search space without overly constraining the expressiveness of the formalism. This can be seen by viewing Horn clauses as procedures[5] in a non-deterministic programming language similar to MICRO-PLANNER[14]. Ie., the formula,

$$P_1 \wedge P_2 \wedge \ldots P_n \supset P,$$

could be programmed,

THE_CONSEQUENT_OF (P_1, P_2, ... P_n) IS P;

and the clause,

$$\overline{P_1} \vee \overline{P_2} \vee \ldots \overline{P_n},$$

could be programmed,

THE_CONSEQUENT_OF (P_1, P_2, ... P_n) IS INCONSISTENCY.

Proofs done in this way are reasonably efficient as only unit consequents are ever generated. Thus the maximum number of procedure firings is bounded by the number of elements in **P**.

In this section we will consider clauses of the former form (definite clauses) to develop the desired result on union of extensions.

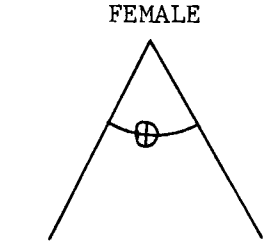
Definition Definite Component

Given a TDB, T = (**C, P, I, E**), the definite component of T, $\Delta(T)$, is defined as the TDB, (**C, P,** [W $\in$ **I** | W is a definite clause], **E**)

Note that E contains only definite clauses. We can now see exactly what parts of a TDB are definite:

$W \in I$	Clausal Form	Definite?
$P \equiv P_1 \wedge P_2 \wedge \ldots P_n$	$\overline{P} \vee P_1,\ \overline{P} \vee P_2,\ \ldots \overline{P} \vee P_n$	yes
	$\overline{P_1} \vee \overline{P_2} \vee \ldots \overline{P_n} \vee P$	yes
$P \equiv P_1 \vee P_2 \vee \ldots P_n$	$\overline{P_1} \vee P,\ \overline{P_2} \vee P,\ \ldots \overline{P_n} \vee P$	yes
	$\overline{P} \vee P_1 \vee P_2 \vee \ldots P_n$	no
$\sim (P \wedge Q)$	$\overline{P} \vee \overline{Q}$	no

We would like a result parallel to lemma C.2 but relating unions of extensions to the formula $P \equiv P_1 \vee P_2 \vee \ldots P_n$. However, as the following example shows, it would not necessarily hold:



C = [jane]

P = [FEMALE, GIRL, WOMAN]

I = [FEMALE $\equiv$ WOMAN v GIRL,
$\sim$ (WOMAN $\wedge$ GIRL)]

E = [FEMALE jane]

$\|\text{FEMALE}\| = [\text{jane}]$ and $\|\text{WOMAN}\| = \|\text{GIRL}\| = [\ ]$

since $T \not\vdash \text{WOMAN jane}$ and $T \not\vdash \text{GIRL jane}$.

So not only do we not have $\|\text{FEMALE}\| = \|\text{WOMAN}\| \cup \|\text{GIRL}\|$ but we have a TDB in which unnatural information is contained. It is hard to conceptualize jane as a "WOMAN or GIRL" without her being either a WOMAN or a GIRL. Actually,

jane is one of the two, we just can't determine which. For these reasons we insist on a TDB being "concrete".

Definition Concreteness

1) The formula $P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$ is concrete iff for every $c \in C$, $\Delta(T) \vdash Pc$ implies there is an i such that $\Delta(T) \vdash P_i c$.

2) A TDB is concrete iff every formula of the above form is concrete.

We could have defined concreteness in terms of provability from T rather than $\Delta(T)$. Unfortunately as we shall see, it turns out that at update time it would then be excessively difficult to maintain concreteness.
The disadvantage in the approach taken is that equivalence must be established between T and $\Delta(T)$ before the result on union of extensions can be attained. Theorem D.1 accomplished this.

The Davis and Putnam result[4] used in the proof of this theorem applies only for the propositional calculus. The following definition and lemma rectify this.

Definition Propositional Component

Given a TDB, T, and $c \in C$ we define the propositional component of T with respect to c, $PROP_T c$ as:

$$PROP_T c \equiv [P \mid Pc \in E] \cup [W \mid (x)Wx \in I]$$

((x)Wx represents any formula in I with free variable x)

Using the last example:

$$PROP_T jane = [FEMALE, \quad FEMALE \equiv WOMAN \vee GIRL, \quad \sim (WOMAN \wedge GIRL)\ ]$$

Lemma D.1 Propositional Equivalence

Given a consistent TDB, T, $c \in \mathbf{C}$ and $P \in \mathbf{P}$

then $T \vdash Pc$ iff $PROP_T c \vdash P$

Proof:

Part 1) Assume $\mathbf{E} \cup \mathbf{I} \cup [\overline{Pc}]$ is unsatisfiable. By Herbrand's theorem, if we replace each clause in **I** with the set of clauses obtained by replacing the free variable 'x' with each member of **C** then the remaining set is unsatisfiable. Call this the ground form of **I**. Since $\mathbf{I} \cup \mathbf{E}$ was originally consistent, the ground form of $\mathbf{I} \cup \mathbf{E}$ is. $[\overline{Pc}]$ cannot resolve against any clause which does not have c's as its ground constant. Therefore all the constants in **C** except c are irrelevant to the inconsistency. If we construct the clause set

$$[Q \mid Qc \in \mathbf{E}] \cup [W \mid Wc \in \text{ground form of } \mathbf{I}] \cup [\overline{P}]$$

then we have a propositionally unsatisfiable set. Therefore,

$PROP_T c \vdash P$.

Part 2) Assume $PROP_T c \cup [\overline{P}]$ is unsatisfiable. Clearly $[Wc \mid W \in PROP_T c \cup [\overline{P}]]$ is unsatisfiable. This is a ground substitution of $\mathbf{E} \cup \mathbf{I} \cup [\overline{Pc}]$. Therefore $T \vdash Pc$.

Theorem D.1 also requires that one factor out certain non-definite clauses. These are the ones that allow jane to be a "WOMAN or GIRL" without the necessity of her being one of WOMAN or GIRL.

Definition Dead Clauses

Given a TDB, T and c ε C then the dead clauses of T with respect to c, $DEAD_Tc$ is defined:

$$DEAD_Tc \equiv [P \supset P_1 \vee P_2 \vee \ldots P_n \varepsilon PROP_Tc \mid PROP_{\Delta(T)}c \vdash P, \quad n > 1]$$

Lemma D.2 Dropping of Dead Clauses

Given a concrete TDB, T, c ε C and W ε $DEAD_Tc$

then $PROP_Tc \sim DEAD_Tc \vdash W$.

Proof:

$W = P \supset P_1 \vee P_2 \vee \ldots P_n \varepsilon DEAD_Tc$

and $PROP_{\Delta(T)}c \vdash P$.

By concreteness for some i, $1 <= i <= n$,

$PROP_{\Delta(T)}c \vdash P_i$.

P_i subsumes W so $PROP_{\Delta(T)}c \vdash W$.

Now $DEAD_Tc$ contains no definite clauses so

$PROP_{\Delta(T)}c \subset PROP_Tc \sim DEAD_Tc$ giving

$PROP_Tc \sim DEAD_Tc \vdash W$. QED

We will now prove equivalence of T and $\Delta(T)$. The strategy will be to categorize the set of clauses produced by resolution within $PROP_T c \sim DEAD_T c$ and show that if a unit Qc is provable from T then Q will be in that set.

Theorem D.1 $T - \Delta(T)$ Equivalence

Given a consistent concrete TDB, T, $c \in C$ and $Q \in P$

then $T \vdash Qc$ iff $\Delta(T) \vdash Qc$.

Proof:

Assume $PROP_T c \vdash Q$.

Then $PROP_T c \sim DEAD_T c \vdash Q$ by Lemma D.2.

So $PROP_T c \sim DEAD_T c \cup [\overline{Q}]$ is unsatisfiable.

Define the following sets iteratively:

$T_0 = PROP_T c \sim DEAD_T c \cup [\overline{Q}]$

$T_{i+1} = T_i$ when $Q \in T_i$ or

when T_i contains no positive units, otherwise

$= T_i \sim [X \vee P \vee Y \in T_i] \sim [X \vee \overline{P} \vee Y \in T_i] \cup$

$[X \vee Y \mid X \vee \overline{P} \vee Y \in T_i]$

where P is the lexicographically first positive unit in T_i and X and Y are negative predicate and positive predicate disjunctions. (possibly null)

The following is an invariant statement on T_i:

If $W \in T_i$ then

A) $W = P$ such that $PROP_{\Delta(T)} c \vdash P$ for some $P \in P$ or

B) $W = \overline{P_j}$ such that $P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$

which is exclusive and

$PROP_{\Delta(T)} c \vdash P_i$ $(i \neq j)$ or

C) $W = \overline{Q_1} \vee \overline{Q_2} \vee \ldots \overline{Q_m} \vee P$ $(m >= 1)$ such that

$P \equiv P_1 \wedge P_2 \wedge \ldots P_n \in I$ and

$[Q_1, Q_2, \ldots Q_m] \subset [P_1, P_2, \ldots P_n]$ and

$PROP_{\Delta(T)^c} \vdash W$ or

D) $W \in T_0$

The invariant statement is proved by the following subproof:

Subproof: By induction on i

Case i = 0 Clear by condition D

Case i > 0 If $W \in T_{i+1}$ then $W \in T_i$ or

$W = R(U, V)$ for $[U, V] \subset T_i$ with U as the lexicographically first unit in T_i.

$PROP_{\Delta(T)} \vdash U$ by A and D above.

V is of the form B, C or D above:

B) $V = \overline{P_j}$ such that $P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$ which is exclusive and

$PROP_{\Delta(T)^c} \vdash P_i$ $(i \neq j)$

$U = P_j$ and $PROP_{\Delta(T)} \vdash P_j$.

But $\overline{P_i} \vee \overline{P_j} \in I$ by exclusiveness.

This violates the consistency of T, so $V \neq \overline{P_j}$.

C) $V = \overline{Q_1} \vee \overline{Q_2} \vee \ldots \overline{Q_m} \vee P$ $(m >= 1)$

$P \equiv P_1 \wedge P_2 \wedge \ldots P_n \in I$ and

$[Q_1, Q_2, \ldots Q_m] \subset [P_1, P_2, \ldots P_n]$ and

$PROP_{\Delta(T)^c} \vdash V$.

$U = Q_i$ for some i, $1 <= i <= m$.

Clearly W has the same form as C with $PROP_{\Delta(T)} \vdash W$ unless m = 1 in which case $W = P$ with $PROP_{\Delta(T)} \vdash P$ which is in the form A.

D) $V \in PROP_T c \sim DEAD_T c \cup [\overline{Q}]$

Case 1 $V = \overline{P_1} \vee P_2 \in I$, trivially $PROP_{\Delta(T)} c \vdash V$.

$U = P_1$ so $PROP_{\Delta(T)} \vdash P_2$ which is of form A.

Case 2 $V = \overline{P_1} \vee \overline{P_2} \vee \ldots \overline{P_n} \vee P \in I$

See C above.

Case 3 $V = \overline{P_i} \vee \overline{P_j}$ $(i < j)$

where $P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$ which is exclusive.

Assume without loss of generality that $U = P_i$.

Then $W = \overline{P_j}$ which is in the form B.

Case 4 $V = \overline{P} \vee P_1 \vee P_2 \vee \ldots P_n$, $U = P$.

Since $PROP_{\Delta(T)} c \vdash P$ then $V \in DEAD_T c$.

But then $V \notin T_0$ which is a contradiction.

Case 5 $V = \overline{Q}$

But then $U = Q$ was the lexicographically first unit in T_i so by construction $T_{i+1} = T_i$.

This completes the subproof of the invariant statement on T_i.

As a corollary, T_i contains no non-unit positive clauses.

By the results of Davis and Putnam[4], T_{i+1} is unsatisfiable whenever T_i is. Since T_0 is unsatisfiable, by induction for all i, T_i is unsatisfiable.

Also, T_i contains a positive unit for every i, for if not then the interpretation in which every predicate is taken as false is a model for T_i.

Note that T_{i+1} has at least one less predicate symbol than T_i (the lexicographically first unit in T_i) unless $T_{i+1} = T_i$. Since T_i always contains a positive unit and since P is finite, then $T_{i+1} = T_i$ for some i. At this point $Q \in T_i$ since T_i contains a positive unit. By the invariant statement on T_i,

$$PROP_{\Delta(T)}c \vdash Q \quad \text{QED.}$$

We can now relate part 4ii in the definition of a TDB to the union of extensions.

Lemma D.3 Union

Given a consistent concrete TDB, T and a formula

$$P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$$

then $\|P\| = \|P_1\| \cup \|P_2\| \cup \ldots \|P_n\|$.

Proof:

Part 1 $c \in \|P\|$ or $T \vdash Pc$.

By theorem D.1, $\Delta(T) \vdash Pc$.

By concreteness, for some i, $\Delta(T) \vdash P_ic$

giving $T \vdash P_ic$ or $c \in \|P_i\|$.

Part 2) $c \in \|P_i\|$ or $T \vdash P_ic$ for some i.

In clausal form $\overline{P_i} \vee P \in I$.

By resolution $T \vdash Pc$ or $c \in \|P\|$.

Until now we have not differentiated between disjoint and non-disjoint union of extensions of the form $P \equiv P_1 \text{ v } P_2 \text{ v } \ldots P_n \in I$. However, in most (but not all) applications we would like disjointness strictly enforced.

Lemma D.4 Disjointness

Given a consistent TDB, T, and an exclusive formula $P \equiv P_1 \text{ v } P_2 \text{ v } \ldots P_n \in I$

then for each i, j, $1 <= i < j <= n$, $\|P_i\| \cap \|P_j\| = [\ ]$.

Proof: Let $c \in \|P_i\| \cap \|P_j\|$.

Then $T \vdash P_i c$ and $T \vdash P_j c$.

But $\overline{P_i} \text{ v } \overline{P_j} \in I$.

This violates consistency, so no such c exists.

Corollary D.1 Disjoint Union

Given a consistent concrete TDB, T, and an exclusive formula

$P \equiv P_1 \text{ v } P_2 \text{ v } \ldots P_n \in I$

then $\|P\| = \|P_1\| \oplus \|P_2\| \oplus \ldots \|P_n\|$.

In summary:

1) For conjunctions $\|P\| = \|P_1\| \cap \|P_2\| \cap \ldots \|P_n\|$

Furthermore if T is consistent and concrete:

2) For disjunctions $\|P\| = \|P_1\| \text{ U } \|P_2\| \text{ U } \ldots \|P_n\|$

3) For exclusive disjunctions $\|P\| = \|P_1\| \oplus \|P_2\| \oplus \ldots \|P_n\|$

Section E Transformation of a TDB to its Reduced Form

One use of a TDB is to answer the question "Is $T \vdash Pc$". Reiter for one requires this for his "typed unification algorithm" as described in [10,11]. Algorithm E.1 below computes $[P \mid T \vdash Pc]$ given $c \in \mathbf{C}$, for a concrete TDB, and detects any inconsistency in $PROP_T c$.

As $[P \mid T \vdash Pc]$ may contain a large number of P for any given c, we define a reduced form of T, R(T), which contains only the Pc's necessary to span **E**. This corresponds to associating INSTANCES with primitive semantic categories in [7,8,9]. Thus, we can determine if $c \in \|P\|$ by looking at the primitive categories below P.

We have shown that formulae of the form $\overline{P} \vee P_1 \vee P_2 \vee \ldots P_n \in \mathbf{I}$ can be removed with no effect when T is both concrete and consistent. This motivates computing on the Horn component of concrete TDBs.

Definition Horn Component

Given a TDB, $T = (\mathbf{C}, \mathbf{P}, \mathbf{I}, \mathbf{E})$, define the Horn component of T, H(T), as

$$H(T) = (\mathbf{C}, \mathbf{P}, [W \in \mathbf{I} \mid W \text{ is a Horn clause}], \mathbf{E})$$

Note that **E** contains only Horn clauses. Looking at this case by case:

$W \in \mathbf{I}$	Clausal Form	Definite?	Horn?
$P \equiv P_1 \wedge P_2 \wedge \ldots P_n$	$\overline{P} \vee P_1,\ \overline{P} \vee P_2,\ \ldots \overline{P} \vee P_n$	yes	yes
	$\overline{P_1} \vee \overline{P_2} \vee \ldots \overline{P_n} \vee P$	yes	yes
$P \equiv P_1 \vee P_2 \vee \ldots P_n$	$\overline{P_1} \vee P,\ \overline{P_2} \vee P,\ \ldots \overline{P_n} \vee P$	yes	yes
	$\overline{P} \vee P_1 \vee P_2 \vee \ldots P_n$	no	no
$\sim(P \wedge Q)$	$\overline{P} \vee \overline{Q}$	no	yes

The following algorithm computes $[P \mid T \vdash Pc]$ given $c \in \mathbf{C}$

Algorithm E.1 For computing $DOMAIN_T c$ given $c \in C$

$T_0 \leftarrow PROP_{H(T)}c$

$S_0 \leftarrow [\ ]$

FOR i = 0 to infinity DO

BEGIN

IF T_i contains no positive unit

THEN $DOMAIN_T c \equiv S_i$,

$k \leftarrow i$, EXIT

P <-- the lexicographically first positive unit in T_i

IF $\overline{P} \in T_i$

THEN $PROP_T c$ is inconsistent,

$DOMAIN_T c$ is undefined, EXIT

$S_{i+1} \leftarrow S_i \cup [P]$

$T_{i+1} \leftarrow T_i \sim [X \vee P \vee Y \in T_i] \sim [X \vee \overline{P} \vee Y \in T_i]$

$\cup [X \vee Y \mid X \vee \overline{P} \vee Y \in T_i]$

where X and Y are disjunctions of negative and positive predicates respectively (possibly null).

END

Example: Using the I of diagram C.1 and letting:

E = [SINGLE john, MALE john]

and c = john, the algorithm will produce the following values:

i	P	$T_i \sim T_{i+1}$ (dropped)	$T_{i+1} \sim T_i$ (added)
0	MALE	MALE	
		MAN $\supset$ MALE	
		BOY $\supset$ MALE	
		MALE $\supset$ PERSON	PERSON
		$\sim$(MALE $\wedge$ FEMALE)	$\overline{\text{FEMALE}}$
		MALE $\wedge$ ADULT $\supset$ MAN	ADULT $\supset$ MAN
		MALE $\wedge$ CHILD $\supset$ BOY	CHILD $\supset$ BOY
1	PERSON	PERSON	
		FEMALE $\supset$ PERSON	
		ADULT $\supset$ PERSON	
		CHILD $\supset$ PERSON	
2	SINGLE	SINGLE	
		BACHELOR $\supset$ SINGLE	
		MAIDEN $\supset$ SINGLE	
		SINGLE $\supset$ ADULT	ADULT
		$\sim$(SINGLE $\wedge$ MARRIED)	$\overline{\text{MARRIED}}$
		MAN $\wedge$ SINGLE $\supset$ BACHELOR	MAN $\supset$ BACHELOR
		SINGLE $\wedge$ WOMAN $\supset$ MAIDEN	WOMAN $\supset$ MAIDEN
3	ADULT	ADULT	
		MAN $\supset$ ADULT	
		WOMAN $\supset$ ADULT	
		MARRIED $\supset$ ADULT	
		$\sim$(ADULT $\wedge$ CHILD)	$\overline{\text{CHILD}}$
		ADULT $\supset$ MAN	MAN
		FEMALE $\wedge$ ADULT $\supset$ WOMAN	FEMALE $\supset$ WOMAN
4	MAN	MAN	
		BACHELOR $\supset$ MAN	
		MAN $\supset$ BACHELOR	BACHELOR
5	BACHELOR	BACHELOR	

T_6 contains no positive unit. Thus:

$DOMAIN_T$john = [MALE, PERSON, SINGLE, ADULT, MAN, BACHELOR]

The algorithm can be implemented by hash indexing predicate symbols into a copy of the formulae in **I** in which they appear (only one copy is made for each formula). For efficiency, positive units are kept in a separate list. When a formula of the form $X \vee P \vee Y \in \mathbf{I}$ is to be dropped it is tagged as deleted. When a formula of the form $X \vee \overline{P} \vee Y$ is to be replaced by $X \vee Y$ the predicate symbol $\overline{P}$ is deleted from the copy. If a positive unit results it is added to the positive unit list. If the null clause results, $DOMAIN_T c$ is declared to be undefined and the original clause $\overline{P} \vee \overline{Q}$ that caused the contradiction can be printed for inspection.

If we assume that the time taken to delete the symbol $\overline{P}$ from $X \vee \overline{P} \vee Y$ is constant, the time taken to add a symbol to the positive unit list is constant and the hash indexing is performed in constant time (by design) then the following is a bound on the time complexity[1] of the algorithm:

$$O\left(\sum_{W \in PROP_{H(T)}c} \#\text{ predicate symbols in } W\right)$$

Theorem E.1 Algorithm E.1 Correctness

Given a concrete TDB, T, and $c \in \mathbf{C}$

If $PROP_T c$ is consistent

then $DOMAIN_T c = [P \mid T \vdash Pc]$

otherwise $DOMAIN_T c$ is undefined

Proof:

First, note that the class of Horn clauses is closed under resolution

(ie. $R(W_1, W_2)$ is Horn whenever W_1 and W_2 are). It follows that the only positive clauses in T_i for all i are units. This is needed to apply the Davis and Putnam[4] theorem below.

Case A $PROP_T c$ is consistent

Subcase 1 $P \in DOMAIN_T c$.

For some i, $P \in T_i$ giving $PROP_{H(T)} c \vdash P$

and hence $PROP_T c \vdash P$

Subcase 2 $T \vdash Pc$

Assume $P \notin T_i$ for all i

Define $H'(T) = H(T) \cup [\overline{P}]$

Starting the algorithm with $H'(T)$ rather than $H(T)$ we produce T_i' rather than T_i.

Note that when defined,

$$T_i' = T_i \cup [\overline{P}]$$

The algorithm will terminate in one of two ways:

1) $[P, \overline{P}] \subset T_i'$ for some i

($[Q, \overline{Q}] \not\subset T_i'$ when $Q \in P$ and $Q \neq P$

since T_0 is consistent)

But then $P \in T_i'$ contradiction

2) T_k' contains no positive unit; but then the interpretation in which every predicate symbol is false is a model for T_k'. Since $T \vdash Pc$ by theorem D.1, $\Delta(T) \vdash Pc$ giving $H(T) \vdash Pc$ or $T_0 \cup [\overline{P}] = T_0'$ is unsatisfiable. By Davis and Putnam[4], T_k' is unsatisfiable which is a contradiction. Therefore $P \in T_i$ for some i.

The algorithm must terminate since T_{i+1} contains at least one fewer predicate symbol then T_i.

Case B $PROP_T c$ is unsatisfiable

Statement: $PROP_{H(T)} c$ is unsatisfiable

Subproof: By lemma D.2 $PROP_T c \sim DEAD_T c$ is unsatisfiable. If

we define T_0 as $PROP_T c \sim DEAD_T c$ and let the algorithm define T_i for each i, then the invariant statements of theorem D.1 hold with one addition: $W \in T_i$ may be the null clause.

The algorithm will terminate in one of two ways:

1) $[P, \overline{P}] \subset T_i$ for some i where P is the lexicographically first unit in T_i. By statements A and B of theorem D.1,

   $PROP_{\Delta(T)} c \vdash P$ and

   for some $Q \in P$, $\overline{P} \vee \overline{Q} \in I$ and $PROP_{\Delta(T)} c \vdash Q$.

   Since $\overline{P} \vee \overline{Q} \in H(T)$ and $\Delta(T) \subset H(T)$, $PROP_{H(T)} c$ is inconsistent.

2) By the results of Davis and Putnam[4], for each i, T_i is unsatisfiable. Since T_{i+1} contains at least one less literal than T_i, for some k, T_k contains no positive literals. The interpretation in which every predicate symbol is false is a model for T_k unless T_k contains the null clause. Since T_k is unsatisfiable it must contain the null clause. T_0 doesn't contain the null clause so for some i, $[P, \overline{P}] \subset T_i$ where P is the lexicographically first unit T_i. In this case though, the algorithm will have terminated as in case A.

This ends the subproof.

As in the subproof, the algorithm will not terminate with T_k containing no units (Starting with $T_0 = PROP_{H(T)} c$ unsatisfiable). Therefore, $[P, \overline{P}] \subset T_i$ for some i. The algorithm will stop with $DOMAIN_T c$ undefined. QED

This ends the correctness proof of algorithm E.1

The following corollary is used in the proof of the update algorithm.

Corollary E.1 $\Delta(T)$ - $DOMAIN_T$ Equivalence

Given a TDB, T (not necessarily concrete or consistent) and $c \in C$

1) If $P \in DOMAIN_T c$ then $\Delta(T) \vdash Pc$

2) If $\Delta(T) \vdash Pc$ then $P \in DOMAIN_T c$ or $DOMAIN_T c$ is undefined

Proof:

Part 1 $P \in DOMAIN_T c$

The invariant statements of theorem D.1 hold with the addition that W may be the null clause.

Therefore $\Delta(T) \vdash Pc$

Part 2 $\Delta(T) \vdash Pc$

Hence $H(T) \vdash Pc$.

If T is consistent then by theorem E.1, Case A, subcase 2, $P \in T_i$ for some i (we do not need theorem D.1 here) or $P \in DOMAIN_T c$.

If T is unsatisfiable then by theorem E.1, Case B, H(T) is unsatisfiable and $DOMAIN_T c$ is undefined. QED

As a TDB, T, becomes large, $[DOMAIN_T c \mid c \in C]$ may become very large. The following allows us to save only the Pc that span **E**. Ie., store only the 'lowest' extensions in the graph.

Definition The Reduced or Canonical TDB

Given a consistent concrete TDB,

$T = (C, P, I, E)$, the reduced TDB, R(T), is defined:

$R(T) = (C, P, I, [Pc \mid P \in DOMAIN_T c$ and

if $Q \in DOMAIN_T c$ then not $Q\ TAU^+\ P])$

Using the example of diagram C.1:

$[P \mid Pc \in E]$ = [SINGLE, MALE]

$DOMAIN_T c$ = [MALE, PERSON, SINGLE, ADULT, MAN, BACHELOR]

but $[P \mid Pc \in R(T)$'s extension$]$ = [BACHELOR]

This is a good space reduction. In fact, in no case is R(T)'s extension larger than T's. Note that we cannot assume that the concreteness or consistency of R(T) follows from that of T.

Lemma E.1 Spanning

Given a consistent concrete TDB, T,

if $P \in DOMAIN_T c$ then for some $Q \in P$ with $Qc \in R(T)$'s extension,

$Q\ TAU^*\ P$

Proof: Assume for each $n >= 0$ with $Qc \in R(T)$'s extension, that

not $Q\ TAU^n\ P$

By definition of R(T)'s extension, $Pc \in R(T)$'s extension

Choosing $n = 0$, $Q = P$ we have a contradiction for $Q\ TAU^0\ P$.

R(T)'s extension is said to span $[Pc \mid T \vdash Pc]$.

Getting back to the original problem, "Is $T \vdash Pc$?" we see that we need only scan R(T)'s extension for Qc such that $Q\ TAU^*\ P$.

Theorem E.2 T - R(T) Equivalence

Given a consistent concrete TDB, T,

then if $Pc \in R(T)$'s extension then $T \vdash Pc$

and if $Pc \in E$ then $R(T) \vdash Pc$

Proof:

Part 1) Assume $Pc \in R(T)$'s extension.

$P \in DOMAIN_T c$ so by theorem E.1, $T \vdash Pc$

Part 2) Assume $Pc \in E$

or $P \in DOMAIN_T c$.

By the spanning lemma (E.1) for some $Q \in P$,

such that $Qc \in$ R(T)'s extension, Q TAU* P.

By the generalization lemma (C.1), $R(T) \vdash Pc$ QED

Corollary E.2

Given a consistent concrete TDB, T, $c \in C$, $P \in P$,

$T \vdash Pc$ iff $R(T) \vdash Pc$

Corollary E.3 Concreteness of R(T)

Given a consistent concrete TDB, T, R(T) is concrete

Proof:

Let $P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$

Assume for some $c \in C$, $\Delta(R(T)) \vdash Pc$

Then $T \vdash Pc$ by theorem E.2

and $\Delta(T) \vdash Pc$ by theorem D.1

and $\Delta(T) \vdash P_i c$ by T's concreteness (for some i, $1 <= i <= n$)

This gives $P_i c \in DOMAIN_T c$ by theorem D.1

By the spanning lemma (E.1), for some Qc $\in$ R(T)'s extension,

Q TAU* P_i

By the generalization lemma (C.1) (which resolves only definite clauses), $\Delta(R(T)) \vdash P_i c$ QED

Corollary E.4 Consistency of R(T)

Given a consistent concrete TDB, T, R(T) is consistent

Proof:

Let $S = [Pc \mid P \in DOMAIN_T c] \cup$

$[\overline{Pc} \mid P \in P$ and $P \notin DOMAIN_T c]$

S is a model for H(T)[12].

If it is not a model for T then for some

$$P \equiv P_1 \text{ v } P_2 \text{ v } \ldots P_n \in I,$$

$P \in DOMAIN_T c$ and each $P_i \notin DOMAIN_T c$, $1 <= i <= n$

This contradicts concreteness, so S is a model for T.

R(T) is concrete so similarly S is a model for R(T). QED

Thus R(T) is a suitable replacement for a consistent TDB, T.

Section F Updating a TDB

Thus far we have shown that a TDB will have the desired set theoretic properties if concreteness and consistency are maintained. We have given a canonical (reduced) form of the TDB that can be efficiently computed while conserving both space and time in answering "Is $T \vdash Pc$?". We must now specify an algorithm that will update the canonical TDB while guaranteeing concreteness and consistency.

Given a concrete and consistent TDB, T, and $c \in C$ the algorithm accepts additions to and deletions from $DOMAIN_T c$ so as to leave the resulting TDB concrete and consistent. It is assumed that originally $T = R(T)$. The algorithm will produce $T' = R(T')$.

Algorithm F.1 For updating a canonical TDB T, given $c \in C$

$UPDATE_T c$:

```
        BEGIN
        WRITE (c, "has the root properties:", [P | Pc ∈ E])
    S:  INPUT (A ⊂ P (additions)  and  D ⊂ P (deletions) )
        IF  [Pc | P ∈ D] ⊄ E  THEN
            WRITE (D ~ [P | Pc ∈ E],  "cannot be deleted,  try again")
            GOTO S
    L:  T' <-- (C, P, I, E U [Pc | P ∈ A] ~ [Pc | P ∈ D])
        Calculate DOMAIN_T'c
        CONCRETE <-- TRUE
    E:  IF DOMAIN_T'c is undefined (because P̄ v Q̄ ∈ I) THEN
            WRITE (c, "cannot be both a", P, "and a", Q, "try again")
            GOTO S
```

C: FOR EACH $P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$ such that

$P \in DOMAIN_T{}'c$ but no $P_i \in DOMAIN_T{}'c$

WRITE ("A", P, "must also be one of", P_1, P_2, ... P_n, "add one of them")

CONCRETE <-- FALSE

IF NOT CONCRETE THEN

INPUT ($A' \subset P$ (additions))

A <-- A U A'

GOTO L

Calculate R(T') given $DOMAIN_T{}'c$ and T' *

T' <-- R(T').

WRITE ("update accepted")

END

Note that in the step marked *, $DOMAIN_T{}'c_1$ for $c_1 \neq c$ need not be recalculated as $DOMAIN_T{}'c_1 = DOMAIN_T c_1$. The following theorem gives the update properties desired.

Theorem F.1 Algorithm F.1 Correctness

Given a consistent concrete and canonical TDB, T and $c \in C$ then the resultant canonical TDB, T', is consistent and concrete.

Proof:

Part 1 Concreteness

Let $P \equiv P_1 \vee P_2 \vee \ldots P_n \in I$ and $\Delta(T') \vdash Pc$

By corollary E.1, since $DOMAIN_T{}'c$ is defined in order for algorithm termination, $P \in DOMAIN_T{}'c$. Since the algorithm terminated (CONCRETE is TRUE), for some i, $P_i \in DOMAIN_T{}'c$. Corollary E.1 gives $\Delta(T') \vdash P_i c$.

Part 2 Consistency

If T' is not consistent then since T is consistent, $PROP_{T'}$'c must be unsatisfiable. Theorem E.1 applies since T' is concrete saying $DOMAIN_{T'}$'c is undefined. This contradicts termination of the algorithm. Therefore T' is consistent.

By corollaries E.3 and E.4, R(T') is concrete and consistent. QED

The following is a sample update using the I of diagram C.1 and an empty **E**.

Input	Response	E or E'
- Update john	- john has the root properties [ ]	E = [ ]
- A=[ADULT]	- An ADULT must be one of MARRIED or SINGLE, add one.	E' = [ADULTjohn]
- A=[MALE,SINGLE]	- Update accepted	E' = [ADULTjohn, MALEjohn, SINGLEjohn] E' = [BACHELORjohn]
- Update john	- john has the root properties [BACHELOR]	E = [BACHELORjohn]
- A=[MARRIED] and D=[SINGLE]	- [SINGLE] cannot be deleted, try again	
- A=[MARRIED]	- john cannot be both MARRIED and SINGLE, try again	E' = [MARRIEDjohn, BACHELORjohn]
- A=[MARRIED] and D=[BACHELOR]	- An ADULT must be one of MALE or FEMALE, add one	E' = [MARRIEDjohn]
- A=[MALE]	- Update accepted	E' = [MARRIEDjohn, MALEjohn]

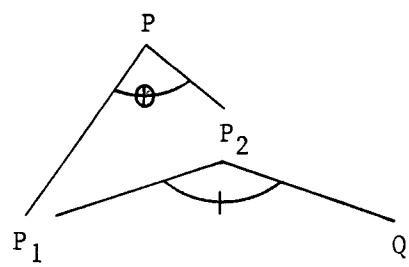
It may seem redundant that we must "Add MALE" when we did not explicitly delete MALE. However, when BACHELOR is deleted, the algorithm has no way in general of determining if MAN and/or SINGLE are to remain in the TDB. Rather than trying to second guess the user, the algorithm asks him to correct any dropped information.

Before the first update, when **E** = [ ], for the update theorem to apply, T must be concrete and consistent. It is certainly concrete. In order for it to be consistent, **I** must be consistent. The interpretation in which all predicate symbols are false under the interpretation is a model for I, so it is consistent. Thus by induction on the update theorem we are assured that if we start with **E** = [ ] then T will always be concrete and consistent after the update.

Although there are no inconsistencies in **I**, there may be constraints that prevent certain kinds of updates from taking place. By this we mean that given some $P \in \mathbf{P}$, for no $c \in \mathbf{C}$ is $Pc \in \mathbf{E}$ while T is consistent.
Ie. $\|P\| = [\ ]$ always.

For example, consider the following:

I =



Clearly, $\mathbf{I} \vdash (x)\ \overline{P_1}x$.

Therefore if the TDB is to be kept consistent, $\|P_1\| = [\ ]$ always. This is clearly a case of bad structuring of **I**. It can be avoided by ensuring that

for all P ϵ **P**, **I** U [Px] is consistent. This can be guaranteed by letting E = [Pc] for any c ϵ **C** and calculating $DOMAIN_I c$. If undefined, then **I** is badly structured above P.

Summary and Conclusions

A generalized structure has been proposed for Reiter's typed database [10] to replace the more restricted notion of the ISA hierarchy. It provides a rich structuring facility and imposes only the natural requirements of so-called concreteness and consistency. Results have been given to ensure that the database has the desired set theoretic properties. An update algorithm has been specified that guarantees concreteness and consistency along with a technique for testing if a specific element is a member of a given domain. A means for testing a non-atomic formula against the database was not included and is an open topic.

While traditional relational databases suggest no need for a typed database, work in artificial intelligence and inferential database systems often are deficient because of the need for a more interdependant domain structure. Since the TDB provides the most rudimentary level of inferencing it must be well structured and mathematically complete in order that inferencing mechanisms built on it stand on firm ground. It must not limit the descriptive power of a host inferential database system, thereby reducing its applicability. We suggest that the proposed structure and algorithms for a typed database are adequate for managing the domain structure of inferential database systems.

Bibliography

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D. (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass., 1974.

2. Chang, C.L., and Lee, R.C.T. (1973). Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York, 1973.

3. Date, C.J. (1975). An Introduction to Data Base Systems. Addison-Wesley, Reading, Mass., 1975.

4. Davis, M., and Putnam, H. (1959). A Computational Proof Procedure. Rensselaer Polytechnical Institution, Troy, New York, AFOSR TR 59-124.

5. Gallaire, H., Minker, J., and Nicolas, J.M. (1978). An Overview and Introduction to Logic and Data Bases. Logic and Data Bases, Eds. H. Gallaire and J. Minker, Plenum Press, New York, 1978, P.12.

6. Gries, D. (1971). Compiler Construction for Digital Computers. John Wiley and Sons, Inc., 1971.

7. McSkimin, J.R. (1976). The Use of Semantic Information in Deductive Question-Answering Systems. Ph.D. Thesis, Dept. of Computer Science, Univ. of Maryland, College Park, Md., 1976.

8. McSkimin, J.R., and Minker, J. (1977). The Use of a Semantic Network in a Deductive Question-Answering System. Dept. of Computer Science, Univ. of Maryland Tech. Report TR-506, 1977.

9. Minker, J. (1978). An Experimental Relational Data Base System Based on Logic. Logic and Data Bases, Eds. H. Gallaire and J. Minker, Plenum Press, New York, 1978.

10. Reiter, R. (1977). An Approach to Deductive Question-Answering. Technical report 3649, Bolt Beranek and Newman Inc., Cambridge, Mass., Sept. 1977.

11. Reiter, R. (1978a). Deductive Question-Answering on Relational Data Bases. Logic and Data Bases, Eds. H. Gallaire and J. Minker, Plenum Press, New York, 1978.

12. Reiter, R. (1978b). On closed world data bases. Logic and data bases, Eds. H. Gallaire and J. Minker, Plenum Press, New York, 1978.

13. Robinson, J.A. (1965). A Machine Oriented Logic Based on the Resolution Principle. J. ACM 12 (Jan. 1965), 25-41.

14. Sussman, G., Winograd, T., and Charniak, E. (1970). MICRO-PLANNER Reference Manual. A.I. MEMO no. 203, M.I.T., Cambridge, Mass., 1970.

15. Winograd, T. (1972). Understanding Natural Language. Academic Press, 1972.

Appendix A Dictionary of Symbols

Sets:

[ ]	-	Used in set definitions as a replacement for { }
ϵ , $\notin$	-	Set membership and non-membership
$\subset$, $\not\subset$	-	Set inclusion and non-inclusion
$\cap$, U	-	Set intersection and union
$\oplus$	-	Disjoint union
$\sim$	-	Set difference
|	-	Set qualification (read "such that")
$\|\|P\|\|$	-	Extension of a predicate, (see section C)

Predicate Calculus Formulae:

$\wedge$	-	Conjunction symbol
v	-	Disjunction symbol
$\supset$	-	Implication
$\equiv$	-	Equivalence
$\sim$	-	Negation
$\overline{P}$	-	Also negation
(x)	-	Universal quantificaion
$\vdash$, $\nvdash$	-	Symbols for provabity and non-provability
$R(W_1, W_2)$	-	The resolvents of formulae W_1 and W_2 [13]

Mathematical Relations:[6]

ALPHA, BETA and TAU - Relations defined in section C

R^n - Composition of relation R, n times

R^+ - Transitive closure of relation R

R^* - Reflexive transitive closure of relation R

General Usage:

$\equiv$ - read "is defined as", in definitions and algorithms

O - computational complexity of an algorithm (order complexity)[1]