

DEDUCTION TREES AND THE VIEW UPDATE PROBLEM IN INDEFINITE DEDUCTIVE DATABASES

C A JOHNSON
COMPUTER SCIENCE DEPT.
UNIVERSITY OF KEELE
STAFFS. ST5 5BG, ENGLAND
email : chrisj@cs.keele.ac.uk

Key words: Deductive databases, indefinite data, null values, view updates.

Abstract: The view update problem is concerned with indirectly modifying those tuples that satisfy a view (or derived table) by an appropriate update against the corresponding base tables. The notion of a deduction tree is defined and the relationship between such trees and the view update problem for indefinite deductive databases is considered. It is shown that a traversal of an appropriate deduction tree yields sufficient information to perform view updates at the propositional level. To obtain a similar result at the first order level, it is necessary (for theoretical and computational reasons) to impose some weak stratification and definiteness constraints on the database.

§1 INTRODUCTION.

In relational database terminology, a table is referred to as a base table if the database explicitly lists those tuples that hold in the table. By contrast, a table is a view if those tuples that hold in the view are *derived* by means of some defining formulae, that define the view in terms of base tables (and possibly previously defined views). In deductive database terminology, the equivalent of a base table is an *extensional* predicate, and that of a view, an *intensional* predicate. Thus a deductive database consists of an *extension* that contains formulae listing those tuples which satisfy the extensional predicates, and an *intension* that contains the defining formulae for intensional predicates.

Suppose that we wish to modify the tuples that satisfy an intensional predicate. Since there is no explicit representation of these tuples, the only way in which we can achieve this is by making an appropriate update against the extensional part of the database. Finding such an update is known as the view update problem.

Suppose for example that P is extensional, and the defining formula for Q is

$$Q(x) \leftarrow P(x, x)$$

then in order to insert $Q(a)$, it is clear that the appropriate modification of the extension is to add $P(a, a)$. $Q(a)$ could also be inserted by the addition of any formula of the form $P(a, a) \wedge \phi$, but clearly the insertion of ϕ is unnecessary, and in general we wish to change the extension in as small a manner as is possible.

Suppose now that Q is defined via the formulae

$$\begin{aligned} Q(x) &\leftarrow P(x, x) \\ Q(x) &\leftarrow R(x) \end{aligned}$$

then in order to add $Q(a)$, we might add either $P(a, a)$ or $R(a)$, but either of these insertions is stronger than is necessary. In an indefinite deductive database, we can add the disjunctive fact $P(a, a) \vee R(a)$, this being intuitively the “appropriate” modification of the extension [Gr93, p249]. This example shows that indefinite databases can handle the view update problem more effectively than their definite counterparts.

If the defining formulae for Q had instead been

$$\begin{aligned} Q(x) &\leftarrow P(x, y) \\ Q(x) &\leftarrow R(x) \end{aligned}$$

then the appropriate modification of the extension would seem to be to add the formula $R(a) \vee \exists y P(a, y)$. Various methods have been suggested in the literature for handling the existential quantifier, for instance using nulls [Ro89, Ka90] (i.e., by adding $R(a) \vee P(a, \omega)$,

where ω is a (new) null value) or by prompting the user to enter the value of the unknown variable y [At91].

Deletions can be more troublesome. If

$$Q(x) \leftarrow P(x, x)$$

then in order to delete $Q(b)$, we need to delete $P(b, b)$ (and again we wish to delete no more than is necessary). If however we have

$$Q(x) \leftarrow P(x, x) \wedge R(x)$$

then in order to delete $Q(b)$, we can either delete $P(b, b)$ or $R(b)$, i.e., there is no unique update against the extension that has the desired effect.

Paper overview. For the most part we follow [Fa83, Gr93], regarding the view update problem as that of amending the existential database so that the required update holds, but also ensuring that the database is amended no more than is necessary. For insertions, this always yields a unique update, although as indicated above, for deletions there can be ambiguity.

We attack the view update problem for indefinite deductive databases using the notion of a *deduction tree*. In Section 2 we define the notion of a deduction tree at the propositional level, and show that the traversal of a “maximal” deduction tree gives precisely the information required to insert or delete into/from any view. Such information can be used either (i) directly to update the extension (and moreover the tree traversal delivers the required update immediately in the required conjunctive normal form), or (ii) indirectly, for instance to prompt the user for additional information in order to simplify or disambiguate the update against the extension.

The results of Section 2 form the basis for our study of the first order level which is presented in Sections 3 and 4. Section 3 contains some preliminaries for the first order level, together with some motivating examples illustrating the construction of first order deduction trees.

We also illustrate various ways in which the information provided by a deduction tree traversal can be used to update the extension. At the first order level we employ nulls to handle existentially quantified variables, although we also indicate and emphasise that other disambiguation methods [At91, Br90, De90, Ka90, To88] (such as a dialogue with the user) could also be adopted. The output from the tree traversal provides the input into such a method, and the results of this paper are independent of the particular disambiguation policy employed.

At the first order level, two other problems present themselves. Firstly we would like our updated database to contain no redundancy. However, the removal of such redundancy

may involve a substantial amount of extra computation, or necessitate an unreasonable increase in the size of the extension. The other problem comes from the use of formulae of the form

$$Q(x) \leftarrow P(x, y) \quad \dots\dots (*)$$

in which the antecedent $P(x, y)$ contains the existentially quantified variable y , and where P is intensional. Intensional formulae are used to generate deduction trees, but for rules such as the one shown in (*), it may be very difficult to know whether the rule should be employed once or several times. We define the notion of a semi-definite predicate, and show that if P is semi-definite, then such rules need only be applied once. An appropriate maximal deduction tree can then be generated via a terminating construction (Section 4), and such a tree again yields the information required to perform view updates.

As mentioned above, in the case of deletions there may be many possible updates against the extension having the desired effect. The traversal of the maximal deduction tree also provides, as a by-product, a test for uniqueness. In the case of uniqueness, additional information, say from the user, might be desirable in order to both simplify and disambiguate the update; when uniqueness is not the case, such information would be required.

Section 5 contains some conclusions and open questions, and an appendix (Section 6) contains some of the more complex proofs.

Related Work. The view update problem is a well established topic of research in the context of databases [Ab85, Ab88, Ab93, Ba81, Da82, Go88, Ke91, Sc91], deductive databases [At91, At92, Br90, De90, Fa83, Gr93, Gu91, Kr92, Ma88, Ro89, To88], and more general logical databases [Wi90].

Our study of the view update problem was inspired by [Gr93], where the notion of a restricted SLD - tree is used to attack the view update problem at the propositional level in the case when the intensional database contains only stratified Horn rules. Restricted SLD - trees deliver the required update in disjunctive (rather than conjunctive) normal form, and thus a further transformation is necessary before insertion into the extension. Another disadvantageous feature of restricted SLD - trees is that they may contain infinite branches at the first order level.

The notion of a deduction tree was first introduced in [Jo93], where such trees are used to provide a query answering method for indefinite deductive databases. Deduction trees are closely related to SLO - resolution [Ra89].

The use of nulls as a means of handling existentially quantified variables was suggested in [Ro89, Ka90]. Various methods for disambiguation have been suggested. For example [At91] suggests the idea of a user dialogue, [Br90] uses knowledge about the domain of variables, [De90] uses ground terms that appear elsewhere in the “derivation” and [Ka90,

p656] suggests the possibility of using a type of minimality criteria. [Ke86] discusses the idea of specifying an (unambiguous) view update translator at view definition time, and [To88] employs a similar idea.

§2 THE PROPOSITIONAL LEVEL.

In this section we consider the view update problem at the propositional level. In Sections 2.2 and 2.3 we introduce the notion of a deduction tree. Such trees are crucial to our methods of solving the view update problem, these being presented in Sections 2.4 (for insertions) and 2.5/2.6 (for deletions). In both cases, the required modification(s) of the extension are to be found via a traversal of an appropriate tree. In the case of insertions, there is at most one such modification of the extension. For deletions, the tree traversal also provides a simple test to check whether there is a unique modification against the extension, or several.

2.1 Terminology.

Throughout Section 2, \mathcal{L} will denote a propositional language $\mathcal{L} = \{P_1, P_2, \dots, P_n\}$ where each P_i is a predicate symbol. Predicates in \mathcal{L} are either *extensional* or *intensional* (but not both). $\text{EXT}(\mathcal{L})$ denotes the set of extensional predicates, and $\text{INT}(\mathcal{L})$ the set of intensional predicates.

A *rule* in \mathcal{L} is a formula C of the form

$$B_1 \vee B_2 \vee \dots \vee B_m \leftarrow A_1 \wedge A_2 \wedge \dots \wedge A_n$$

where

- (i) each A_i and each B_j is a predicate, $m > 0$,
- (ii) if $n > 0$, then each B_i is intensional, and
- (iii) if $n = 0$, then each B_i is extensional.

The set $\{B_1, B_2, \dots, B_m\}$ is called $\text{conseq}(C)$ (the consequents of C), and $\{A_1, A_2, \dots, A_n\}$, the set of antecedants, $\text{antec}(C)$.

In case (iii), C is usually referred to as a (disjunctive) *fact*, and we may identify C with the set $\text{conseq}(C)$.

We have written rules from right to left, since this more closely matches our tree notation to be given later. Where necessary, predicates will be superscripted to indicate which rule they come from. At various points it will simplify our presentation if the antecedants of a rule are ordered, in which case we shall assume that $\text{antec}(C)$ is a list.

A *model* for \mathcal{L} is a set $M \subseteq \mathcal{L}$. We say that M *models* C (and write $M \models C$) iff

$$\text{antec}(C) \subseteq M \implies \text{conseq}(C) \cap M \neq \emptyset.$$

A *deductive database* is a set of rules, and throughout T will denote a deductive database. $\text{EXT}(T)$, the *extension* of T , consists of those rules (facts) that have an empty antecedant. The *intension* of T is given by $\text{INT}(T) = T - \text{EXT}(T)$.

M is a model of T (written $M \models T$) iff $M \models C$ for each $C \in T$.

2.1.1 Definition. Let T and T' be deductive databases, then $T \models T'$ iff every model of T is a model of T' .

2.1.2 Definition. A model M of T is *minimal* iff M does not properly contain any model of T .

In the view update problem we aim to amend T by modifying $\text{EXT}(T)$, thus it is useful to know how the models of T are related to those of $\text{EXT}(T)$. The following two theorems show that minimal models of T are formed by closing minimal models of $\text{EXT}(T)$ under $\text{INT}(T)$. In both cases the proofs are trivial.

2.1.3 Theorem. If M is a minimal model of T , then $M \cap \text{EXT}(\mathcal{L})$ is a minimal model of $\text{EXT}(T)$.

2.1.4 Theorem. If M_{EXT} is a minimal model of $\text{EXT}(T)$, then there is a minimal model M of T such that $M \cap \text{EXT}(\mathcal{L}) = M_{\text{EXT}}$.

Recall that a rule C is *definite* iff its consequent contains a single predicate. Similarly we may (loosely) define a predicate P to be definite iff whenever P appears in the consequent of some rule C , then C is definite, and each predicate that appears in $\text{antec}(C)$ is definite. In our study of the view update problem we shall be focusing particularly on the intensional part of the database, thus we assume the existence of a set $SD(\mathcal{L}) \subseteq \text{INT}(\mathcal{L})$ whose predicates satisfy the following variant of definiteness.

2.1.5 Definition. If $P \in SD(\mathcal{L})$ and P appears in the consequent of some rule C , then

- (i) C is definite, and
- (ii) Each intensional predicate appearing in $\text{antec}(C)$ is in $SD(\mathcal{L})$.

A predicate appearing in $SD(\mathcal{L})$ is said to be *semi-definite*.

Define $SD(T) = \{C \in \text{INT}(T) \mid \text{conseq}(C) = \{P\}, P \in SD(\mathcal{L})\}$.

2.2 Inferring Disjunctions and Deduction Trees.

Deduction trees were introduced in [Jo93] as a means of deciding whether a given disjunct could be inferred from a deductive database. In this section we recall some of the definitions and results from [Jo93] necessary for an understanding of later sections.

2.2.1 Definition. Let $\mathcal{P} \subseteq \mathcal{L}$ be a set of predicate symbols, then $T \models \bigvee \mathcal{P}$ iff every model of T contains a predicate from \mathcal{P} .

Notice that \mathcal{P} contains only predicates. In the case when \mathcal{P} contains negated predicates we could alternatively use some closed world assumption (for instance using minimal models) to decide whether $\bigvee \mathcal{P}$ may be inferred from T . Such an alternative is not discussed in this paper. The following result is straightforward.

2.2.2 Theorem [Jo93]. Let $\mathcal{P} \subseteq \mathcal{L}$, then $T \models \bigvee \mathcal{P}$ iff either

- (a) $(\exists C \in \text{EXT}(T)) C \subseteq \mathcal{P}$, or
- (b) $(\exists C \in \text{INT}(T)) (\text{conseq}(C) \subseteq \mathcal{P}, \text{antec}(C) \cap \mathcal{P} = \emptyset, \text{ and } T \models A \vee \bigvee \mathcal{P} \text{ for each } A \in \text{antec}(C)).$

The following example (amended from [Jo93]) indicates how this theorem can be used to check whether a given disjunction may be inferred from T .

2.2.3 Example. Suppose that T contains the following rules:

- | | |
|--|---------------|
| 1. $A \vee B \leftarrow D \wedge E \wedge F$ | 4. $G \vee B$ |
| 2. $B \vee C \vee E \leftarrow F$ | 5. $C \vee D$ |
| 3. $F \leftarrow G$ | |

with $\mathcal{P} = \{A, B, C\}$. The consequent of rule 1 is contained in \mathcal{P} , thus by Theorem 2.2.2(b), $T \models \bigvee \mathcal{P}$ iff $T \models \bigvee \mathcal{P} \cup \{D\}$, $T \models \bigvee \mathcal{P} \cup \{E\}$ and $T \models \bigvee \mathcal{P} \cup \{F\}$. Let us denote this by the tree structure depicted in Figure 2.2.3(i).

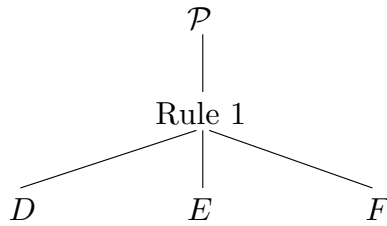


Figure 2.2.3(i).

The top node is the disjunction (or goal) to be proven, the middle node indicates which rule has been applied, and the lower nodes indicate the “subgoals” which result

from the application of the rule. For instance the node D indicates that we need to show that $T \models \bigvee \mathcal{P} \cup \{D\}$.

It is immediately the case that $T \models \bigvee \mathcal{P} \cup \{D\}$ since T contains $C \vee D$ (rule 5). In order to show that $T \models \bigvee \mathcal{P} \cup \{E\}$ we apply rule 2. By Theorem 2.2.2(b), $T \models \bigvee \mathcal{P} \cup \{E\}$ iff $T \models \bigvee \mathcal{P} \cup \{E, F\}$. We continue to depict this using our tree notation (see Figure 2.2.3 (ii)).

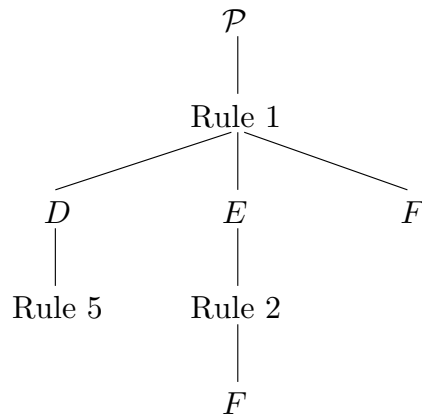


Figure 2.2.3(ii).

Again the nodes labelled Rule 5 and Rule 2 indicate applications of these rules. The node labelled Rule 5 has no child nodes precisely because rule 5 has no antecedents.

The lower node labelled F indicates that we need to prove $T \models \bigvee \mathcal{P} \cup \{E, F\}$. However note that this subgoal is subsumed by $\mathcal{P} \cup \{F\}$. We can thus ignore $\mathcal{P} \cup \{E, F\}$, treating it as proven. This type of search space pruning is known as *factorisation*.

The subgoal $\mathcal{P} \cup \{F\}$ is solved by applying rule 3 and then rule 4, yielding the final tree depicted in Figure 2.2.3(iii).

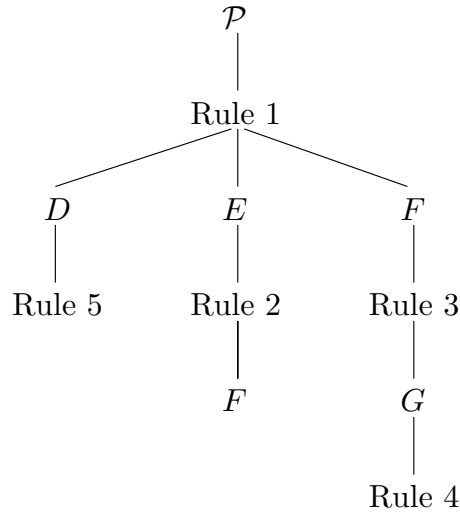


Figure 2.2.3(iii).

The tree structures depicted in the above example were introduced in [Jo93] as a means of providing a query answering method for indefinite deductive databases, and are referred to as deduction trees.

2.2.4 Definition. A *deduction tree* for \mathcal{P} in T is a tree \mathcal{T} containing three types of nodes: a goal node, rule nodes and predicate nodes.

- (1) The root node (at the top of the tree) is a goal node, this being labelled with \mathcal{P} . This is the only goal node in the tree. The root has exactly one child node, this being a rule node (satisfying condition (5) below).
- (2) Each rule node is labelled with a rule from T , and each predicate node is labelled with a predicate in \mathcal{L} . The label of a node N will be denoted by $lab(N)$. (Predicate nodes will be denoted by their label alone, or in the form N_R where R is the label of node N . A rule node whose label is rule i will be denoted by RN_i .)
- (3) If RN is a rule node labelled with rule C , then for each predicate P in $antec(C)$, RN has a (single) child node labelled with P . The child nodes are depicted from left to right using the ordering of $antec(C)$ (cf. Section 2.1). RN has no other child nodes. If N is a predicate node with $lab(N) = P$, then its parent node is a rule node M with $P \in antec(lab(M))$. A predicate node can have at most one child node. If the child exists, then it must be a rule node satisfying condition (5) below.
(If M is the parent of N , then we write $M > N$, the $>$ relationship being transitive.)
- (4) If N is a predicate node with label P , then $P \notin \mathcal{P}$, and for each predicate node $M > N$, M is not labelled with P , i.e., along any branch, a given predicate cannot label two or more nodes.

(5) If RN is a rule node labelled with C , then

$$\text{conseq}(C) \subseteq \mathcal{P} \cup \{P \in \mathcal{L} \mid P \text{ labels some predicate node above } RN\}$$

2.2.5 Example. Let T be the following database:

- | | | |
|--|--|-------------------------|
| 1. $Q_1 \leftarrow R \wedge S \wedge W \wedge W_1$ | 2. $Q \vee S \leftarrow U \wedge V \wedge V_1$ | 3. U |
| 4. $R \vee Q$ | 5. $W \leftarrow W_1$ | 6. $W_1 \leftarrow W_2$ |
| 7. W_2 | 8. $S \vee V \leftarrow W_1$ | 9. V_1 |

then a deduction tree for $\{Q, Q_1\}$ in T is shown in Figure 2.2.5.

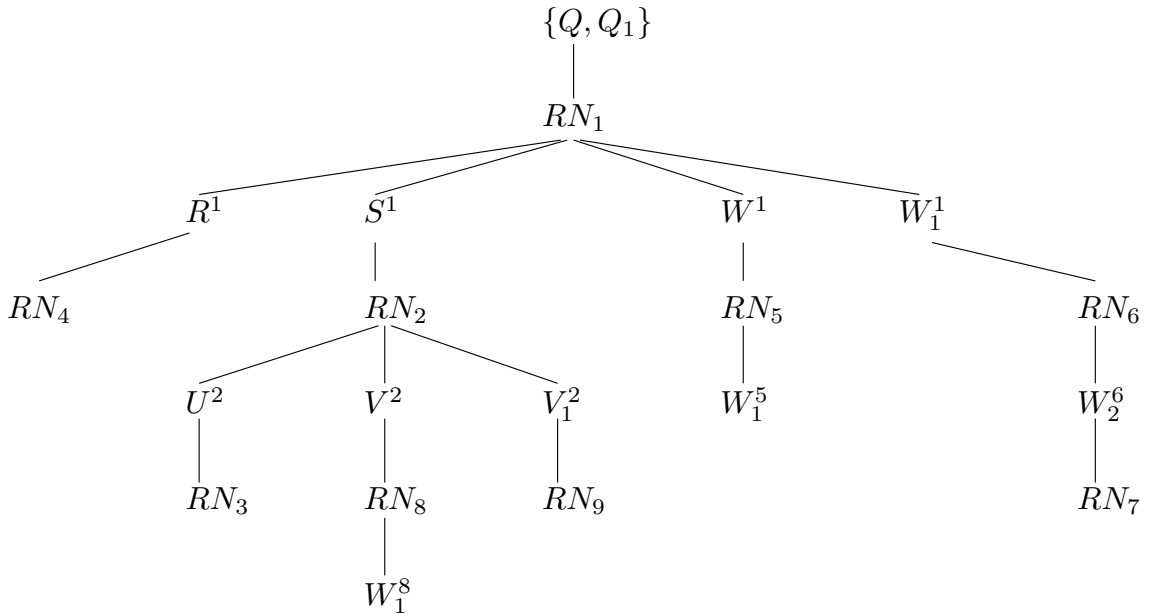


Figure 2.2.5

2.2.6 Definition. Let \mathcal{T} be a deduction tree. Given a predicate node N we may define $\text{ACT}(N)$ to be the sequence of predicates (or predicate nodes) from the root to (and including) N . $\text{WAIT}(N)$ is formed by taking each predicate in $\text{ACT}(N)$ in turn and placing its right siblings on WAIT .

For instance if \mathcal{T} is as shown in Figure 2.2.5, then $\text{ACT}(W_1^8) = (Q_1, Q, S^1, V^2, W_1^8)$ and $\text{WAIT}(W_1^8) = (W_1^1, W^1, V_1^2) = \text{WAIT}(V^2)$.

As mentioned in the introduction, our solution to the view update problem will necessitate a (depth first, left to right) traversal of an appropriate deduction tree. In the context of such a traversal, if ACT is the current (active) path, then WAIT specifies (or identifies) those paths that remain to be examined.

Also we would clearly like the tree traversal to be efficient, and thus methods of pruning deduction trees are of interest. In [Jo93] we presented two such methods, one of which, factorisation, is easily applicable in the present context.

2.2.7 Definition. Let \mathcal{T} be a deduction tree.

- (a) If N is a predicate node, then N (or $\text{ACT}(N)$) is said to be *factored* iff there is some predicate node M (labelled R) on $\text{ACT}(N)$ such that $\text{WAIT}(M)$ contains R .
- (b) A branch B (through \mathcal{T}) is a sequence of nodes (N_0, N_1, \dots, N_k) such that N_0 is the root node, N_k is a leaf node, and for each $0 < i \leq k$, N_i is a child node of N_{i-1} . If N_k is a predicate node, then we may write $\text{ACT}(B)$ for $\text{ACT}(N_k)$.
- (c) B is said to be *factored* iff some factored predicate node lies on B .
- (d) \mathcal{T} is said to be *factored* iff every predicate leaf node is factored.

For instance in Figure 2.2.5, the predicate nodes W_1^8 and W_1^5 are factored (by the presence of W_1^1) and thus the entire tree is factored. As indicated in Example 2.2.3, factored branches may be discarded, since they are subsumed by some branch that will be examined at a later stage in the traversal.

2.2.8 Theorem [Jo93]. Suppose that $\mathcal{P} \subseteq \mathcal{L}$, then the following are equivalent:

- (a) $T \models \bigvee \mathcal{P}$.
- (b) There is a deduction tree for \mathcal{P} in T in which each leaf node is a rule node.
- (c) There is a factored deduction tree for \mathcal{P} in T .

2.2.9 Theorem. Let \mathcal{T} be a deduction tree for \mathcal{P} in T , then the following are equivalent:

- (a) $T \models \bigvee \mathcal{P}$.
- (b) For each predicate leaf node N , $T \models \bigvee \text{ACT}(N)$.
- (c) For each unfactored predicate leaf node N , $T \models \bigvee \text{ACT}(N)$.

Proof. (a) \rightarrow (b) is trivial since $\mathcal{P} \subseteq \text{ACT}(N)$. (b) \rightarrow (c) is trivial. The proof of (c) \rightarrow (a) is similar to that of Theorem 2.2.8, thus we sketch the details.

Assume that (c) holds and that $T \not\models \bigvee \mathcal{P}$. Using Theorem 2.2.2 we may construct a branch $N_0 > RN_{C_0} > N_1 > RN_{C_1} > N_2 \dots$ through \mathcal{T} such that for each j , (i) $T \not\models \bigvee \text{ACT}(N_j)$, and (ii) for each right sibling N' of N_j , $T \models \bigvee \text{ACT}(N')$.

By condition (i) we can see that B must terminate in a predicate node N_k , and by condition (ii), N_k cannot be factored, thus contradicting condition (c). ■

2.2.10 View updates.

How does the above theorem relate to view updates? Suppose that $T \not\models \bigvee \mathcal{P}$. Condition (c) of Theorem 2.2.9 gives us some insight into how we can modify T , so as to infer $\bigvee \mathcal{P}$: If \mathcal{T} is a deduction tree for \mathcal{P} in T , and $T' = T \cup \{\bigvee \text{ACT}(N) \mid N \text{ is an unfactored predicate leaf node in } \mathcal{T}\}$, then $T' \models \bigvee \mathcal{P}$.

Recall two points however. Firstly, the required modification to T should involve the extension only, thus suggesting that we should instead add $\bigvee (\text{EXT}(\mathcal{L}) \cap \text{ACT}(N))$ to T .

Secondly, we do not wish to strengthen T any more than is necessary. The following (trivial) lemma indicates that in order to weaken T' we should extend \mathcal{T} .

2.2.11 Lemma. Let \mathcal{T} be a deduction tree for \mathcal{P} in T , N an unfactored predicate leaf node in \mathcal{T} and C a rule in T such that $\text{conseq}(C) \subseteq \text{ACT}(N)$ and $\text{antec}(C) \cap \text{ACT}(N) = \emptyset$. Suppose \mathcal{T}' is formed from \mathcal{T} by appending RN_C as a child of N (and appending the appropriate child nodes to RN_C from $\text{antec}(C)$).

Then $T \cup \{\bigvee \text{ACT}(M) \mid M \text{ is an unfactored predicate leaf node in } \mathcal{T}\} \models T \cup \{\bigvee \text{ACT}(M') \mid M' \text{ is an unfactored predicate leaf node in } \mathcal{T}'\}$.

Thus we might expect that in order to make T' as weak as possible, we should extend \mathcal{T} until each branch which terminates in an unfactored predicate node cannot be extended further (i.e. is maximal). This idea is formalised in Definition 2.3.1 below.

2.3 Maximal Deduction Trees.

2.3.1 Definition. A deduction tree \mathcal{T} for \mathcal{P} in T is said to be *maximal* iff whenever N is an unfactored predicate leaf node in \mathcal{T} , then

- (a) there is no rule C in $\text{INT}(T)$ such that $\text{conseq}(C) \subseteq \text{ACT}(N)$ and $\text{antec}(C) \cap \text{ACT}(N) = \emptyset$, and
- (b) there is no rule C in $\text{EXT}(T)$ such that $\text{conseq}(C) \subseteq \text{ACT}(N)$.

Maximal trees are central to our view update methods. Specifically, if we wish to modify T so as to infer $\bigvee \mathcal{P}$, then generating a maximal tree \mathcal{T} for \mathcal{P} in T tells us whether (or not) $T \models \bigvee \mathcal{P}$ (see Theorem 2.3.7 below), and if $T \not\models \bigvee \mathcal{P}$, then (see Section 2.4) the unfactored predicate leaf nodes in \mathcal{T} indicate precisely which disjunctive facts we need to add to $\text{EXT}(T)$ so as to infer $\bigvee \mathcal{P}$ (without strengthening T any more than is necessary).

On the other hand, if we wish to prevent the inference of \mathcal{P} , then generating a maximal tree in $\text{INT}(T)$ tells us whether (or not) $T \models \bigvee \mathcal{P}$ (see Theorem 2.3.8), and if $T \models \bigvee \mathcal{P}$, then the unfactored branches provide a means of generating appropriate modifications to $\text{EXT}(T)$ (see Sections 2.5 and 2.6).

2.3.2 Example. Consider the following database:

- | | | |
|--|--|--------|
| 1. $Q_1 \leftarrow R \wedge S \wedge W \wedge W_1$ | 2. $Q \vee S \leftarrow U \wedge V \wedge V_1$ | 3. U |
| 4. $W \leftarrow W_1$ | 5. $W_1 \leftarrow W_2$ | |
| 6. W_2 | 7. $S \vee V \leftarrow W_1$ | |

then a maximal deduction tree for $\{Q, Q_1\}$ in T is shown in Figure 2.3.2.

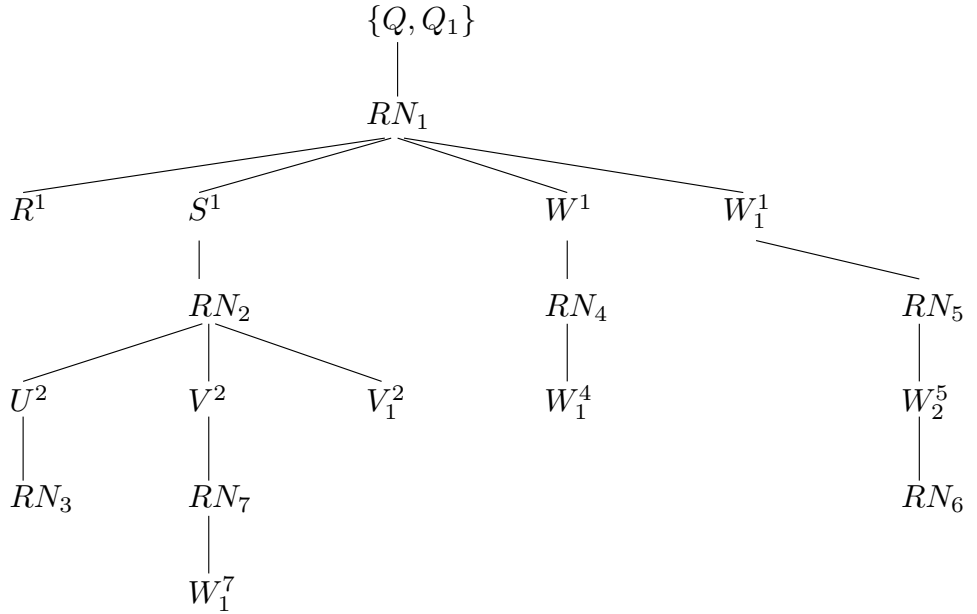


Figure 2.3.2

It is useful to isolate the property given in Definition 2.3.1(a).

2.3.3 Definition. Given $\mathcal{P} \subseteq \mathcal{L}$, then a set \mathcal{C} is a *cover* of \mathcal{P} in T iff $\mathcal{P} \subseteq \mathcal{C} \subseteq \mathcal{L}$ and

$$\forall C \in \text{INT}(T) (\text{conseq}(C) \subseteq \mathcal{C} \implies \text{antec}(C) \cap \mathcal{C} \neq \emptyset).$$

A cover \mathcal{C} of \mathcal{P} is *minimal* iff it does not properly contain a cover of \mathcal{P} .

Notice that $\text{EXT}(T)$ plays no part in the definition of a cover, thus a cover in T is a cover in $\text{INT}(T)$ and vice versa. Part (b) of the following lemma follows immediately from Theorem 2.2.2.

2.3.4 Lemma. If \mathcal{C} is a cover in T , then

- (a) $\mathcal{L} - \mathcal{C} \models \text{INT}(T)$.
- (b) $T \models \bigvee \mathcal{C}$ iff there is an $E \in \text{EXT}(T)$ such that $E \subseteq \mathcal{C}$.

The following (trivial) results detail the relationship between covers and branches through a maximal tree.

2.3.5 Theorem. Let $\mathcal{P} \subseteq \mathcal{L}$, and \mathcal{T} be a maximal deduction tree for \mathcal{P} in T .

- (a) For each unfactored predicate leaf node N , $\text{ACT}(N)$ is a cover of \mathcal{P} in T .
- (b) If \mathcal{C} is a minimal cover of \mathcal{P} in T , then either there is an $E \in \text{EXT}(T)$ such that $E \subseteq \mathcal{C}$ or there is an unfactored predicate leaf node N such that $\text{ACT}(N) = \mathcal{C}$.
- (c) For each unfactored predicate leaf node N , $\text{ACT}(N)$ is a minimal cover of \mathcal{P} iff there is no unfactored predicate leaf node N' such that $\text{ACT}(N') \subset \text{ACT}(N)$.

Proof. Parts (a) and (c) are straightforward.

(b). We may construct a branch B through \mathcal{T} such that for each predicate node M on B , $\text{lab}(M) \in \mathcal{C}$, and for each right sibling M' of M , $\text{lab}(M') \notin \mathcal{C}$. If B terminates with a rule node RN_E , then $E \in \text{EXT}(T)$ and $E \subseteq \mathcal{C}$. If B terminates in a predicate node, then B cannot be factored and $\text{ACT}(B) \subseteq \mathcal{C}$. Thus by part (a) and the minimality of \mathcal{C} we must have that $\text{ACT}(B) = \mathcal{C}$. ■

Note that if \mathcal{T} is a deduction tree in $\text{INT}(T)$, then each leaf node is a predicate node.

2.3.6 Theorem. Let $\mathcal{P} \subseteq \mathcal{L}$, and \mathcal{T} be a maximal deduction tree for \mathcal{P} in $\text{INT}(T)$.

- (a) For each unfactored branch B through \mathcal{T} , $\text{ACT}(B)$ is a cover of \mathcal{P} .
- (b) If \mathcal{C} is a minimal cover of \mathcal{P} in $\text{INT}(T)$, then there is an unfactored branch B through \mathcal{T} such that $\text{ACT}(B) = \mathcal{C}$.
- (c) For each unfactored branch B through \mathcal{T} , $\text{ACT}(B)$ is a minimal cover of \mathcal{P} iff there is no unfactored branch B' through \mathcal{T} such that $\text{ACT}(B') \subset \text{ACT}(B)$.

The following two results provide the basis for our view update methods.

2.3.7 Theorem. Let $\mathcal{P} \subseteq \mathcal{L}$ and \mathcal{T} be a maximal deduction tree for \mathcal{P} in T . Then $T \models \bigvee \mathcal{P}$ iff \mathcal{T} is factored.

Proof (\rightarrow). Let N be an unfactored predicate leaf node in \mathcal{T} . By Theorem 2.3.5(a), $\text{ACT}(N)$ is a cover of \mathcal{P} , and by Theorem 2.2.9, $T \models \bigvee \text{ACT}(N)$. But then by Lemma 2.3.4(b), there is an $E \in \text{EXT}(T)$ such that $E \subseteq \text{ACT}(N)$, thus contradicting condition (b) of Definition 2.3.1.

The implication (\leftarrow) follows immediately from Theorem 2.2.8. ■

2.3.8 Theorem. Let $\mathcal{P} \subseteq \mathcal{L}$, and \mathcal{T} be a maximal deduction tree for \mathcal{P} in $\text{INT}(T)$. Then $T \models \bigvee \mathcal{P}$ iff for each unfactored predicate leaf node N in \mathcal{T} there is an $E \in \text{EXT}(T)$ such that $E \subseteq \text{ACT}(N)$.

Proof. The proof of the implication from left to right is similar to that of Theorem 2.3.7. The converse follows immediately from Theorem 2.2.9. ■

2.3.9 Corollary. Let C be a rule, then $T \models C$ iff whenever \mathcal{C} is a cover of $\text{conseq}(C)$ in T , then either $\text{antec}(C) \cap \mathcal{C} \neq \emptyset$ or $(\exists E \in \text{EXT}(T))(E \subseteq \mathcal{C})$.

2.4 View Updates : Insertions.

Suppose that $\mathcal{P} \subseteq \mathcal{L}$ with $T \not\models \bigvee \mathcal{P}$. We wish to strengthen $\text{EXT}(T)$ to form a database $T(\bigvee \mathcal{P})$ such that $T(\bigvee \mathcal{P}) \models \bigvee \mathcal{P}$. Moreover we do not wish to strengthen $\text{EXT}(T)$ any more than is necessary, i.e., if T' is any database such that $\text{INT}(T') = \text{INT}(T)$ and $T' \models T \wedge \bigvee \mathcal{P}$, then we should have that $T' \models T(\bigvee \mathcal{P})$.

2.4.1 Definition. Suppose that \mathcal{T} is a deduction tree for \mathcal{P} in T and B is a branch through \mathcal{T} terminating in a predicate node. Let $\text{EXT}(B) = \text{EXT}(\mathcal{L}) \cap \text{ACT}(B)$.

2.4.2 Definition. Let \mathcal{T} be a maximal deduction tree for \mathcal{P} in T , and let $(B_i \mid 1 \leq i \leq j)$ enumerate those branches through \mathcal{T} that terminate in an unfactored predicate node. (Note that by Theorem 2.2.8, if $T \not\models \bigvee \mathcal{P}$, then $j \geq 1$.)

Suppose that there is a database T^* such that $\text{INT}(T^*) = \text{INT}(T)$ and $T^* \models \bigvee \mathcal{P}$. For each $i \leq j$, $\text{ACT}(B_i)$ is a cover of \mathcal{P} in $\text{INT}(T) = \text{INT}(T^*)$, and thus by Corollary 2.3.9 there is an $E \in \text{EXT}(T^*)$ such that $E \subseteq \text{ACT}(B_i)$. In particular, $\text{EXT}(B_i) \neq \emptyset$.

Thus:

(a) If some $\text{EXT}(B_i) = \emptyset$, then $T(\bigvee \mathcal{P})$ is undefined.

(b) If each $\text{EXT}(B_i) \neq \emptyset$, then let

$$T(\bigvee \mathcal{P}) = T \cup \{\bigvee \text{EXT}(B_i) \mid 1 \leq i \leq j\}$$

In practice we would also wish to remove from $T(\bigvee \mathcal{P})$ any redundant (i.e. subsumed) facts in the extension. This point relates to the difference between a maximal tree in T and a maximal tree in $\text{INT}(T)$. For the purposes of view insertions, the usage of a maximal tree in T (rather than $\text{INT}(T)$) has the effect of pruning the tree traversal and reducing redundancy (since if a fact $C \in \text{EXT}(T)$ can be applied to a branch B (i.e. $C \subseteq \text{EXT}(B)$), then $\text{EXT}(B)$ is redundant in $T(\bigvee \mathcal{P})$).

By Theorem 2.3.5, $T(\bigvee \mathcal{P})$ is equivalent to $T \cup \{\bigvee \mathcal{C} \cap \text{EXT}(\mathcal{L}) \mid \mathcal{C} \text{ is a cover of } \mathcal{P} \text{ in } T\}$. Thus $T(\bigvee \mathcal{P})$ is well defined, i.e., is independent (modulo equivalence) of the choice of \mathcal{T} , and Theorems 2.4.4 and 2.4.5 below follow trivially from this observation. Another point worth noting is that the above definition specifies precisely the relationship between the view \mathcal{P} and those extensional facts that need to be inserted. By contrast the use of restricted SLD trees [Gr93] gives no such insight.

2.4.3 Example. Consider the following database:

- | | | |
|--|--|--------|
| 1. $Q_1 \leftarrow R \wedge S \wedge W \wedge W_1$ | 2. $Q \vee S \leftarrow U \wedge V \wedge W_1$ | 3. U |
| 4. $W \leftarrow W_1$ | 5. $W_1 \leftarrow W_2$ | |
| 6. W_2 | 7. $S \vee Q_1 \leftarrow V_1$ | |

where $\text{EXT}(\mathcal{L}) = \{R, V, V_1, U, W_2\}$, and suppose that we wish to add $Q \vee Q_1$ to the database. The appropriate maximal deduction tree is depicted in Figure 2.4.3.

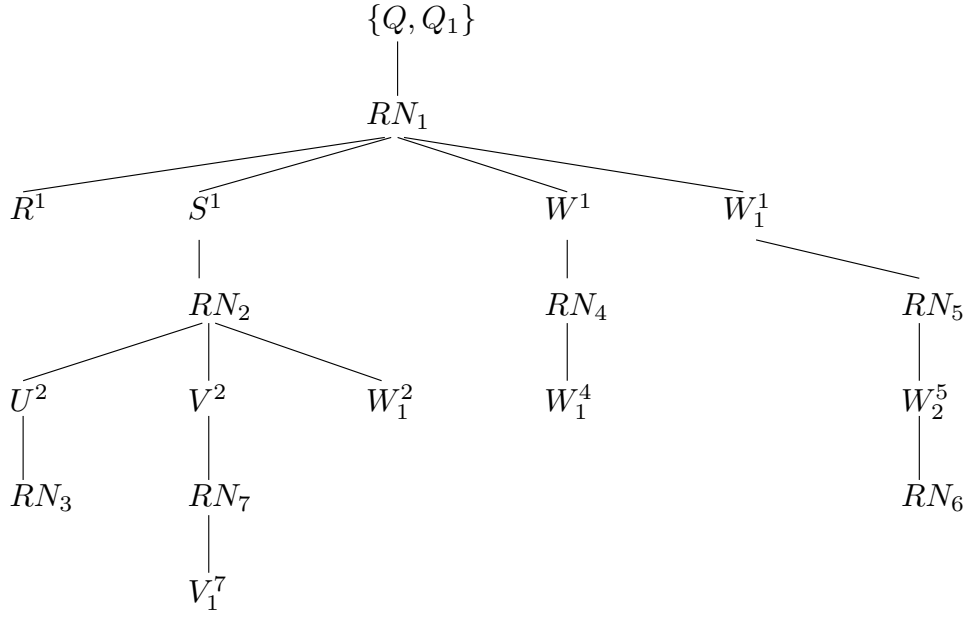


Figure 2.4.3

If \mathcal{C} is a cover of $\{Q, Q_1\}$ in $\text{INT}(T)$, then it is easy to check that there must be some unfactored branch through the tree whose predicates all lie in \mathcal{C} . Thus either $\text{ACT}(R^1) \subseteq \mathcal{C}$, $\text{ACT}(U^2) \subseteq \mathcal{C}$, $\text{ACT}(V_1^7) \subseteq \mathcal{C}$ or $\text{ACT}(W_2^5) \subseteq \mathcal{C}$.

Thus if $\text{INT}(T^*) = \text{INT}(T)$ and $T^* \models T \wedge (Q \vee Q_1)$, then we must have that $T^* \models R \wedge (V \vee V_1)$, whence

$$T(Q \vee Q_1) = T \cup \{R\} \cup \{V \vee V_1\}.$$

Alternatively, if we did not wish to add the disjunctive fact $V \vee V_1$ to the extension, then we could say prompt the user to enter truth values for V and V_1 .

2.4.4 Theorem. $T(\bigvee \mathcal{P})(\bigvee \mathcal{P}')$ and $T(\bigvee \mathcal{P}')(\bigvee \mathcal{P})$ are equivalent.

2.4.5 Theorem. M is a model of $T(\bigvee \mathcal{P})$ iff

- (a) $M \models T$, and
- (b) $M \cap \text{EXT}(\mathcal{L})$ intersects every cover of \mathcal{P} in T .

The following theorem shows that $T(\bigvee \mathcal{P})$ has the required properties.

2.4.6 Theorem (a) $T(\bigvee \mathcal{P}) \models T \wedge \bigvee \mathcal{P}$.

(b) If $T \models \bigvee \mathcal{P}$, then $T(\bigvee \mathcal{P}) = T$.

(c) If $T' \models T \wedge \bigvee \mathcal{P}$, with $\text{INT}(T') = \text{INT}(T)$, then $T' \models T(\bigvee \mathcal{P})$.

Proof. Let \mathcal{T} be the maximal deduction tree used in the construction of $T(\bigvee \mathcal{P})$.

(a) Since $T \subseteq T(\bigvee \mathcal{P})$, we have that $T(\bigvee \mathcal{P}) \models T$. Moreover \mathcal{T} is a deduction tree in $T(\bigvee \mathcal{P})$.

For each unfactored predicate leaf node N in \mathcal{T} , there is a fact C in $\text{EXT}(T(\bigvee \mathcal{P}))$ such that $C \subseteq \text{ACT}(N)$, thus $T(\bigvee \mathcal{P}) \models \bigvee \mathcal{P}$ by Theorem 2.2.9.

(b) If $T \models \bigvee \mathcal{P}$, then every predicate leaf node in \mathcal{T} must be factored (by Theorem 2.3.7).

(c) We need to show that $T' \models \bigvee \text{EXT}(B_i)$ for each B_i (in the construction of $T(\bigvee \mathcal{P})$).

Since $\text{INT}(T) = \text{INT}(T')$, $\text{ACT}(B_i)$ is a cover of \mathcal{P} in T' . The result then follows from Corollary 2.3.9, since $T' \models \bigvee \mathcal{P}$. ■

2.4.7 Traversing \mathcal{T} . The following algorithm (amended from [Jo93]) constructs and traverses (in a depth first, left to right manner) a maximal deduction tree for \mathcal{P} in T and outputs those branches that terminate in an unfactored predicate node. It may thus be used to construct $T(\bigvee \mathcal{P})$. SOLVE is the traversal procedure, the terminology here being due to the fact that deduction tree traversal is also employed ([Jo93]) to prove that $T \models \bigvee \mathcal{P}$, i.e., to solve the goal \mathcal{P} . REDUCE is the backtracking mechanism, and EXTEND corresponds to the extension of ACT via a rule node (and the corresponding child predicate nodes).

```

SOLVE(ACT, WAIT)
{ if WAIT factors ACT
    { REDUCE(ACT, WAIT);
      SOLVE(ACT, WAIT)
    }
  else if ( $\exists C \in \text{EXT}(T)$ )( $C \subseteq \text{ACT}$ )
    { REDUCE(ACT, WAIT);
      SOLVE(ACT, WAIT)
    }
  else if ( $\exists C \in \text{INT}(T)$ )( $\text{conseq}(C) \subseteq \text{ACT}$  and  $\text{antec}(C) \cap \text{ACT} = \emptyset$ )
    { EXTEND(ACT, C, WAIT);
      SOLVE(ACT, WAIT)
    }
  else { output( $\text{EXT}(\mathcal{L}) \cap \text{ACT}$ ); REDUCE(ACT, WAIT) }
}
EXTEND(ACT, C, WAIT)
{ pick  $K_1 \in \text{antec}(C)$ ;
  push( $\{H^C \mid H \in \text{antec}(C) - \{K_1\}\}$ , WAIT);

```

```

    push( $K_1^C$ ,ACT);
  } /*push pushes the appropriate atoms onto the given stack */
  [ $K^A, H^B$ ] {return( $A = B$ )} /* tests whether two atoms come from the same rule */
  REDUCE(ACT,WAIT)
  { if WAIT== $\emptyset$  {STOP}
    else
      {  $K = pop$ (ACT); /* pop removes the head and returns the atom removed */
        if [ $K, head$ (WAIT)] {  $L = pop$ (WAIT); push( $L,ACT$ )}
        else REDUCE(ACT,WAIT)
      }
    }
  }
  main ( )
  { push ( { $P \mid P \in \mathcal{P}$ }, ACT); WAIT=  $\emptyset$ ; SOLVE(ACT,WAIT) }

```

2.5 View Updates : Deletions.

We now come to the problem of deleting (or preventing) the inference of $\bigvee \mathcal{P}$. Suppose that $T \models \bigvee \mathcal{P}$, then we wish to find databases T^* such that

- (i) $\text{INT}(T^*) = \text{INT}(T)$, $T \models T^*$ and $T^* \not\models \bigvee \mathcal{P}$, and
- (ii) whenever $\text{INT}(T') = \text{INT}(T)$, $T \models T'$, $T' \models T^*$ and $T' \not\models \bigvee \mathcal{P}$, then $T^* \models T'$.

Condition (ii) of course expresses the idea that we should not weaken T any more than is necessary. Let us first consider the case of extensional predicates. Suppose that $\mathcal{S} \subseteq \text{EXT}(\mathcal{L})$ with $T \models \bigvee \mathcal{S}$. In order to modify $\text{EXT}(T)$ in order to prevent the inference of $\bigvee \mathcal{S}$ we need to remove (or more precisely weaken) those facts in $\text{EXT}(T)$ that subsume \mathcal{S} . The following definition does exactly this.

2.5.1 Definition. Given T and $\mathcal{S} \subseteq \text{EXT}(\mathcal{L})$, let

$$T(\neg \bigvee \mathcal{S}) = \text{INT}(T) \cup \{C \in \text{EXT}(T) \mid C \not\subseteq \mathcal{S}\} \cup \{C \vee P \mid C \in \text{EXT}(T), C \subseteq \mathcal{S}, P \in \text{EXT}(\mathcal{L}) - \mathcal{S}\}.$$

Notice that if $\mathcal{S} = \text{EXT}(\mathcal{L})$, then $T(\neg \bigvee \mathcal{S}) = \text{INT}(T)$. Also, if $T \not\models \bigvee \mathcal{S}$, then $T(\neg \bigvee \mathcal{S}) = T$.

2.5.2 Lemma. Let $\mathcal{S}, \mathcal{S}' \subseteq \text{EXT}(\mathcal{L})$.

- (a) $T \models T(\neg \bigvee \mathcal{S})$ and $T(\neg \bigvee \mathcal{S}) \not\models \bigvee \mathcal{S}$.
- (b) If $T \models \bigvee \mathcal{S}'$ and $\mathcal{S}' \not\subseteq \mathcal{S}$, then $T(\neg \bigvee \mathcal{S}) \models \bigvee \mathcal{S}'$.

- (c) Suppose that $\text{INT}(T) = \text{INT}(T')$, $T \models T'$ and $T' \not\models \bigvee \mathcal{S}$. Then $T(\neg \bigvee \mathcal{S}) \models T'$.
- (d) Suppose that $T \models \bigvee \mathcal{S} \wedge \bigvee \mathcal{S}'$. Then $T(\neg \bigvee \mathcal{S})$ and $T(\neg \bigvee \mathcal{S}')$ are equivalent iff $\mathcal{S} = \mathcal{S}'$.

Proof. Part (a) is trivial.

(b). Pick $C \in \text{EXT}(T)$ such that $C \subseteq \mathcal{S}'$. If $C \subseteq \mathcal{S}$, then $C \cup \{P\} \in T(\neg \bigvee \mathcal{S})$ for any $P \in \mathcal{S}' - \mathcal{S}$, where $C \cup \{P\} \subseteq \mathcal{S}'$. If $C \not\subseteq \mathcal{S}$, then $C \in T(\neg \bigvee \mathcal{S})$. In either case it is clear that $T(\neg \bigvee \mathcal{S}) \models \bigvee \mathcal{S}'$.

(c). Suppose that $C \in \text{EXT}(T')$, then $C \not\subseteq \mathcal{S}$ (for otherwise $T' \models \bigvee \mathcal{S}$), and $T \models C$. But then by part (b), $T(\neg \bigvee \mathcal{S}) \models C$.

(d). Suppose that $\mathcal{S}' \not\subseteq \mathcal{S}$. By parts (a) and (b), $T(\neg \bigvee \mathcal{S}) \models \bigvee \mathcal{S}'$ and $T(\neg \bigvee \mathcal{S}') \not\models \bigvee \mathcal{S}'$, and hence $T(\neg \bigvee \mathcal{S})$ and $T(\neg \bigvee \mathcal{S}')$ are not equivalent. ■

2.5.3 Intensional predicates. We now move on to consider the case of intensional predicates. Suppose that $\mathcal{P} \subseteq \mathcal{L}$ with $T \models \bigvee \mathcal{P}$. For the remainder of this section and Section 2.6, let \mathcal{T} be a maximal deduction tree for \mathcal{P} in $\text{INT}(T)$.

Let B be an unrefuted branch through \mathcal{T} . Since $T \models \bigvee \mathcal{P}$, we have (by Theorem 2.3.8) that $\text{EXT}(T) \models \bigvee \text{EXT}(B)$. Moreover we can prevent the inference of $\bigvee \mathcal{P}$ by preventing the inference of $\bigvee \text{EXT}(B)$ as above. Thus let

$$T_B = T(\neg \bigvee \text{EXT}(B)).$$

Notice that in general there may be many branches through \mathcal{T} and hence no unique weakening of T that fails to infer $\bigvee \mathcal{P}$.

2.5.4 Theorem. $T_B \not\models \bigvee \mathcal{P}$.

Proof. Let \mathcal{T} be the maximal deduction tree used in the definition of T_B . By the definition of T_B , \mathcal{T} is a maximal deduction tree for \mathcal{P} in $\text{INT}(T_B)$. Moreover, by construction there is no fact in $\text{EXT}(T_B)$ that subsumes $\text{EXT}(B)$, and the result then follows from Theorem 2.3.8. ■

Notice that the above theorem only says that from T_B we cannot infer $\bigvee \mathcal{P}$. It is not necessarily the case that $\neg \bigvee \mathcal{P}$ can be assumed (under say the GCWA [Mi82, Gr86]), i.e., there may well be minimal models of T_B that do satisfy \mathcal{P} . For instance if $T = \{P \leftarrow Q \wedge R, Q, R, S \vee T\}$, then $T(\neg \{Q\}) = (T - \{Q\}) \cup \{Q \vee R, Q \vee S, Q \vee T\}$. But then $\{Q, R, S, P\}$ is a minimal model of $T(\neg \{Q\})$ containing P .

Suppose that B and B' are unfactored branches through \mathcal{T} with $\text{EXT}(B') \subset \text{EXT}(B)$. It is then easy to check that $T_{B'} \models T_B$, but $T_B \not\models T_{B'}$. In particular, T_B cannot satisfy the conditions given at the beginning of this section.

2.5.5 Example. Consider the following database:

- | | | |
|--|--|-----------------|
| 1. $Q_1 \leftarrow R \wedge S \wedge W \wedge W_1$ | 2. $Q \vee S \leftarrow U \wedge V \wedge W_1$ | 3. U |
| 4. $W \leftarrow W_1$ | 5. $W_1 \leftarrow W_2$ | 6. W_2 |
| 7. $S \vee Q_1 \leftarrow V_1$ | 8. R | 9. $V \vee V_1$ |

where $\text{EXT}(\mathcal{L}) = \{R, U, V, V_1, W_2\}$, and suppose that we wish to delete $Q \vee Q_1$ from T . A maximal deduction tree in $\text{INT}(T)$ is depicted in Figure 2.5.5.

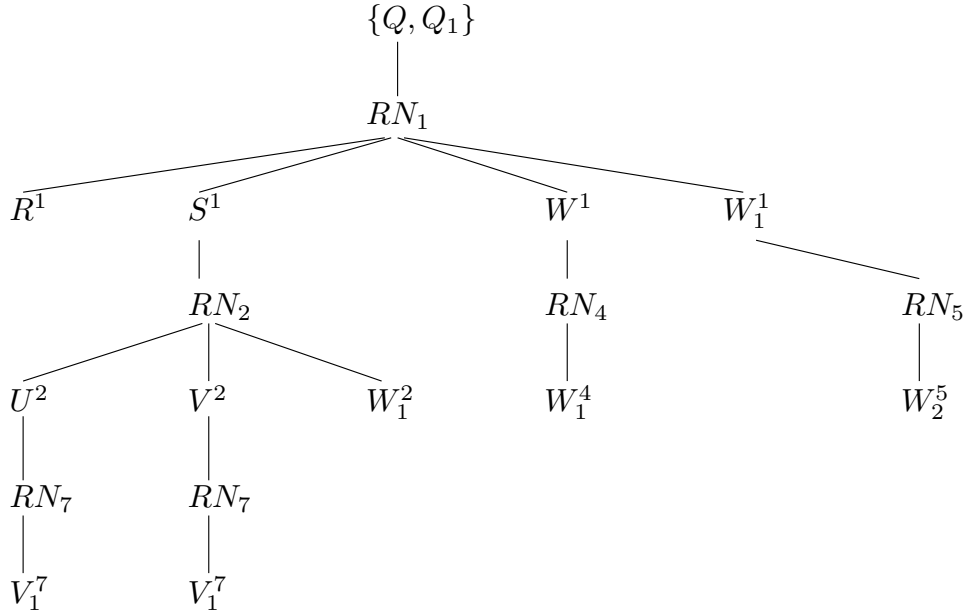


Figure 2.5.5

Again we can easily check that each unfactored branch through the tree defines a cover, and that for any cover there is an unfactored branch all of whose predicates are contained in the cover. Suppose that we take $B = \text{ACT}(V_1^7)$, then

$$T_B = (T - \{V \vee V_1\}) \cup \{V \vee V_1 \vee R, V \vee V_1 \vee U, V \vee V_1 \vee W_2\}$$

and clearly $\{U, R, W_2, W_1, W\}$ is a model of T_B omitting Q and Q_1 .

Again, we could alternatively ask the user how we might modify the fact $V \vee V_1$ to prevent it being inferred from the database.

2.5.6 Definition. (a) An unfactored branch B through \mathcal{T} is *ext-minimal* iff there is no unfactored branch B' through \mathcal{T} such that $\text{EXT}(B') \subset \text{EXT}(B)$.

(b) A cover \mathcal{C} of \mathcal{P} in T is *ext-minimal* iff there is no cover \mathcal{C}' of \mathcal{P} in T such that $\mathcal{C}' \cap \text{EXT}(\mathcal{L}) \subset \mathcal{C} \cap \text{EXT}(\mathcal{L})$.

We can show that if B is ext-minimal, then T_B satisfies the required conditions given at the beginning of this section.

2.5.7 Theorem. Let \mathcal{T} be the deduction tree used in Section 2.5.3 and B an ext-minimal unfactored branch through \mathcal{T} . Suppose that $\text{INT}(T') = \text{INT}(T)$, $T \models T'$, $T' \models T_B$, and $T' \not\models \bigvee \mathcal{P}$. Then $T_B \models T'$.

Proof. Since $\text{INT}(T') = \text{INT}(T)$, \mathcal{T} is a maximal deduction tree for \mathcal{P} in $\text{INT}(T')$. Thus, by Theorem 2.3.8 we may find an unfactored branch B' in \mathcal{T} such that $\text{EXT}(T') \not\models \bigvee \text{EXT}(B')$.

Case 1: $\text{EXT}(B') \subseteq \text{EXT}(B)$. Since B is ext-minimal we must have $\text{EXT}(B') = \text{EXT}(B)$. But then by Lemma 2.5.2(c), $T_B \models T'$.

Case 2: $\text{EXT}(B') \not\subseteq \text{EXT}(B)$. By Lemma 2.5.2(b), $T_B \models \bigvee \text{EXT}(B')$, and since $T' \models T_B$ we have that $T' \models \bigvee \text{EXT}(B')$. ■

Thus the ext-minimal unfactored branches B through \mathcal{T} generate appropriate modifications to $\text{EXT}(T)$ for the deletion of $\bigvee \mathcal{P}$. The following theorem shows that these branches generate all such modifications.

2.5.8 Theorem. Suppose that T^* is a database such that

- (a) $\text{INT}(T) = \text{INT}(T^*)$, $T \models T^*$, and $T^* \not\models \bigvee \mathcal{P}$, and
- (b) whenever $\text{INT}(T') = \text{INT}(T)$, $T \models T'$, $T' \models T^*$, and $T' \not\models \bigvee \mathcal{P}$, then $T^* \models T'$.

Let \mathcal{T} be a maximal deduction tree for \mathcal{P} in $\text{INT}(T)$. Then there is some ext-minimal unfactored branch B through \mathcal{T} such that T^* is equivalent to T_B .

Proof. Since $T^* \not\models \bigvee \mathcal{P}$ there is (by Theorem 2.3.8) an unfactored branch B' through \mathcal{T} such that $T^* \not\models \bigvee \text{EXT}(B')$. Pick an ext-minimal unfactored branch B such that $\text{EXT}(B) \subseteq \text{EXT}(B')$. Clearly $T^* \not\models \bigvee \text{EXT}(B)$ and thus by Lemma 2.5.2(c), $T_B \models T^*$.

By Theorem 2.5.4, $T_B \not\models \bigvee \mathcal{P}$, and hence by hypothesis (b), $T^* \models T_B$. ■

Recall that the models of T are related to the models of $\text{EXT}(T)$ (by Theorems 2.1.3 and 2.1.4). Moreover, since $T(\neg \bigvee \mathcal{S})$ is formed from T by a modification to the extension,

it is reasonable to compare the models of T with those of $T(\neg\bigvee\mathcal{S})$ by comparing the models of their extensions.

2.5.9 Theorem. Suppose that $\text{EXT}(T) \models \bigvee\mathcal{S}$, then M is a minimal model of $\text{EXT}(T(\neg\bigvee\mathcal{S}))$ iff either

- (a) M is a minimal model of $\text{EXT}(T)$ such that $M \not\supseteq \text{EXT}(\mathcal{L}) - \mathcal{S}$, or
- (b) $M = \text{EXT}(\mathcal{L}) - \mathcal{S}$.

2.5.10 Corollary. Suppose that $\text{EXT}(T) \models \bigvee\mathcal{S}$, then M is a model of $\text{EXT}(T(\neg\bigvee\mathcal{S}))$ iff M is a model of $\text{EXT}(T)$ or $M \supseteq \text{EXT}(\mathcal{L}) - \mathcal{S}$.

In view of this and Theorem 2.4.5, the following results are not surprising.

2.5.11 Theorem. Suppose that $T \not\models \bigvee\mathcal{P}$, \mathcal{T} is a maximal deduction tree for \mathcal{P} in $\text{INT}(T)$, and B is an unfactored branch through \mathcal{T} such that $\text{EXT}(T) \not\models \bigvee\text{EXT}(B)$. Then $T \subseteq T(\bigvee\mathcal{P})(\neg\bigvee\text{EXT}(B))$. In general it is not the case that $T \models T(\bigvee\mathcal{P})(\neg\bigvee\text{EXT}(B))$.

Proof. The first statement is trivial. For the second, consider for instance the database $T = \{P \leftarrow Q, R \vee S \vee T\}$, then $T(P)(\neg\{Q\}) = T \cup \{Q \vee R, Q \vee S, Q \vee T\}$. ■

2.5.12 Theorem. Suppose that $T \models \bigvee\mathcal{P}$ and that \mathcal{C} is a cover of \mathcal{P} in T . If $\mathcal{S} = \mathcal{C} \cap \text{EXT}(\mathcal{L})$ is not properly subsumed by any fact in $\text{EXT}(T)$ then T is equivalent to $T(\neg\bigvee\mathcal{S})(\bigvee\mathcal{P})$. In general, T and $T(\neg\bigvee\mathcal{S})(\bigvee\mathcal{P})$ are not equivalent.

Proof. The first statement is again trivial. For the second, consider the database $T = \{P \leftarrow Q, P \leftarrow R, Q, R \vee S\}$, then $T(\neg\bigvee\{Q, R\})(P) = (T - \{Q\}) \cup \{Q \vee S, Q \vee R\}$. ■

2.6 The Uniqueness of T_B .

The database(s) T_B (where B is an ext-minimal unfactored branch through \mathcal{T}) thus provide the means of deleting $\bigvee\mathcal{P}$ from the intensional database. We now examine the conditions under which there is a single such database (modulo equivalence).

By Lemma 2.5.2(d), if $\text{EXT}(B) \neq \text{EXT}(B')$, then T_B and $T_{B'}$ are not equivalent. From the results of the previous section we would therefore expect there to be a single such database if and only if $\text{EXT}(B) = \text{EXT}(B')$ whenever B and B' are ext-minimal unfactored branches through \mathcal{T} , which in turn is the case if and only if there is an unfactored branch

B such that

$$\forall B' (B' \text{ is an unfactored branch through } \mathcal{T} \implies \text{EXT}(B') \supseteq \text{EXT}(B)).$$

This fact is proven below in Corollary 2.6.3, and provides the basis for our view update algorithm which is given in Section 2.6.5. We first formulate the above idea in terms of covers.

2.6.1 Theorem. Suppose that $T \models \bigvee \mathcal{P}$, then the following are equivalent:

- (a) There is a database T^* such that
 - (i) $\text{INT}(T^*) = \text{INT}(T)$, $T \models T^*$, and $T^* \not\models \bigvee \mathcal{P}$, and
 - (ii) whenever $\text{INT}(T') = \text{INT}(T)$, $T \models T'$ with $T' \not\models \bigvee \mathcal{P}$, then $T^* \models T'$.
- (b) There is a cover \mathcal{C} of \mathcal{P} in T such that

$$\forall \mathcal{D} (\mathcal{D} \text{ is a cover of } \mathcal{P} \text{ in } T \implies \mathcal{D} \cap \text{EXT}(\mathcal{L}) \supseteq \mathcal{C} \cap \text{EXT}(\mathcal{L})).$$

- (c) There is an $\mathcal{S} \subseteq \text{EXT}(\mathcal{L})$ such that
 - (i) $\text{INT}(T) \models \bigvee \mathcal{S} \rightarrow \bigvee \mathcal{P}$, and
 - (ii) $\text{INT}(T) \not\models \bigwedge (\text{EXT}(\mathcal{L}) - \mathcal{S}) \rightarrow \bigvee \mathcal{P}$.

Proof (a) \rightarrow (b). By Theorem 2.5.8 there is a cover \mathcal{C} of \mathcal{P} in T such that T^* is equivalent $T(\neg \bigvee \mathcal{C} \cap \text{EXT}(\mathcal{L}))$.

Let \mathcal{D} be a cover of \mathcal{P} in T such that $\mathcal{D} \cap \text{EXT}(\mathcal{L}) \not\supseteq \mathcal{C} \cap \text{EXT}(\mathcal{L})$. By Corollary 2.3.9, $T \models \bigvee (\mathcal{C} \cap \text{EXT}(\mathcal{L}))$, and thus by Lemma 2.5.2(b), $T(\neg \bigvee \mathcal{D} \cap \text{EXT}(\mathcal{L})) \models \bigvee \mathcal{C} \cap \text{EXT}(\mathcal{L})$.

However, by condition (ii) we have that $T^* \models T(\neg \bigvee \mathcal{D} \cap \text{EXT}(\mathcal{L}))$, thus contradicting the fact that $T^* \not\models \bigvee \mathcal{C} \cap \text{EXT}(\mathcal{L})$.

(b) \rightarrow (c). Let \mathcal{C} be a cover of \mathcal{P} in T satisfying condition (b), and $\mathcal{S} = \mathcal{C} \cap \text{EXT}(\mathcal{L})$. Thus $\text{INT}(T) \models \bigvee \mathcal{S} \rightarrow \bigvee \mathcal{P}$ by Corollary 2.3.9.

Since \mathcal{C} is a cover of \mathcal{P} in $\text{INT}(T)$, it follows from Corollary 2.3.9 that $\text{INT}(T) \not\models \bigwedge (\text{EXT}(\mathcal{L}) - \mathcal{S}) \rightarrow \bigvee \mathcal{P}$.

(c) \rightarrow (a). We claim that $T(\neg \bigvee \mathcal{S})$ satisfies condition (a).

By condition (c)(ii), let M be a model of $\text{INT}(T) = \text{INT}(T(\neg \bigvee \mathcal{S}))$ such that $M \supseteq \text{EXT}(\mathcal{L}) - \mathcal{S}$ and $M \cap \mathcal{P} = \emptyset$. Clearly, $M \models T(\neg \bigvee \mathcal{S})$, and in particular, $T(\neg \bigvee \mathcal{S}) \not\models \bigvee \mathcal{P}$.

Suppose that $\text{INT}(T') = \text{INT}(T)$, $T \models T'$, $T' \not\models \bigvee \mathcal{P}$. Clearly $T' \not\models \bigvee \mathcal{S}$ (for otherwise $T' \models \bigvee \mathcal{P}$). The result then follows from Lemma 2.5.2(c). ■

2.6.2 Corollary. Suppose that $T \models \bigvee \mathcal{P}$.

- (a) If $\mathcal{S} \subseteq \text{EXT}(\mathcal{L})$ satisfies the conditions of Theorem 2.6.1 (c), then $\mathcal{S} = \{P \in \text{EXT}(\mathcal{L}) \mid \text{INT}(T) \models P \rightarrow \bigvee \mathcal{P}\}$.

- (b) Let $\mathcal{S} = \{P \in \text{EXT}(\mathcal{L}) \mid \text{INT}(T) \models P \rightarrow \bigvee \mathcal{P}\}$, then the following are equivalent:
- (i) There is a database T^* satisfying the conditions of Theorem 2.6.1(a).
 - (ii) $\text{INT}(T) \not\models \bigwedge(\text{EXT}(\mathcal{L}) - \mathcal{S}) \rightarrow \bigvee \mathcal{P}$.
 - (iii) $T(\neg \bigvee \mathcal{S})$ satisfies the conditions of Theorem 2.6.1(a).

Proof. (a) is trivial, and part (b) follows from Theorem 2.6.1. ■

2.6.3 Corollary. Suppose that $T \models \bigvee \mathcal{P}$, \mathcal{T} is a maximal deduction tree for \mathcal{P} in $\text{INT}(T)$ and $(B_i \mid 1 \leq i \leq j)$ enumerates the unfactored branches through \mathcal{T} . Then the following are equivalent:

- (a) There is a database T^* satisfying the conditions of Theorem 2.6.1(a).
- (b) There is an unfactored branch B through \mathcal{T} such that

$$\forall B' (B' \text{ is an unfactored branch through } \mathcal{T} \implies \text{EXT}(B') \supseteq \text{EXT}(B)).$$

- (c) If $\mathcal{S} = \bigcap_{i=1}^j \text{EXT}(B_i)$, then $\text{INT}(T) \not\models \bigwedge(\text{EXT}(\mathcal{L}) - \mathcal{S}) \rightarrow \bigvee \mathcal{P}$.

Proof (a) \rightarrow (b). follows from Theorem 2.3.6 and Theorem 2.6.1 (a) \rightarrow (b).

(b) \rightarrow (c). $\mathcal{S} = \text{EXT}(B)$ and $\text{ACT}(B)$ is a cover of \mathcal{P} . Thus by Corollary 2.3.9, $\text{INT}(T) \not\models \bigwedge(\text{EXT}(\mathcal{L}) - \mathcal{S}) \rightarrow \bigvee \mathcal{P}$.

(c) \rightarrow (a). Let $\mathcal{S} = \bigcap_{i=1}^j \text{EXT}(B_i)$ and suppose that $\text{INT}(T) \not\models \bigwedge(\text{EXT}(\mathcal{L}) - \mathcal{S}) \rightarrow \bigvee \mathcal{P}$. If $P \in \mathcal{S}$, then by Theorem 2.3.6, P belongs to every cover of \mathcal{P} in $\text{INT}(T)$, thus $\text{INT}(T) \models P \rightarrow \bigvee \mathcal{P}$ by Corollary 2.3.9. Thus \mathcal{S} satisfies the conditions of Theorem 2.6.1(c). ■

2.6.4 Example. Consider the following database:

- | | | |
|---------------------------------------|------------------------------|--------|
| 1. $Q_1 \vee Q \leftarrow S \wedge W$ | 2. $Q \vee S \leftarrow V$ | 3. U |
| 4. $W \leftarrow U$ | 5. $S \vee Q_1 \leftarrow U$ | |

where $\text{EXT}(\mathcal{L}) = \{U, V\}$, then a maximal deduction tree for $\{Q, Q_1\}$ in $\text{INT}(T)$ is depicted in Figure 2.6.4

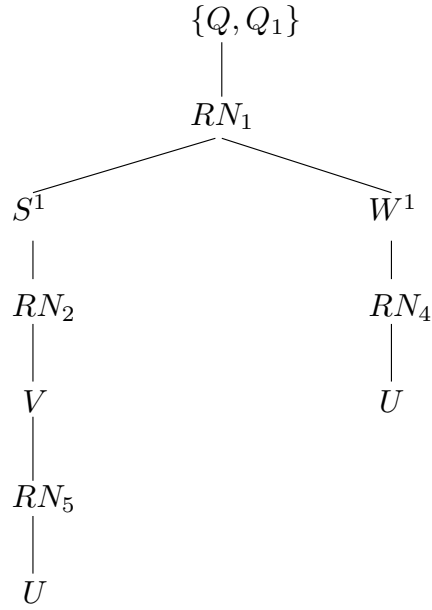


Figure 2.6.4.

Clearly $T(\neg U) = (T - \{U\}) \cup \{U \vee V\}$ is the unique database satisfying the conditions of Theorem 2.6.1(a), and $\{V, S\}$ is a model of $\text{INT}(T)$ which does not model $V \rightarrow Q \vee Q_1$ (cf. condition (c) of Corollary 2.6.3).

2.6.5. In Section 2.4.7 we presented an algorithm for generating (the unfactored branches of) a deduction tree. The following algorithm, based upon Corollary 2.6.3(b), generates a maximal deduction tree in $\text{INT}(T)$ and the \mathcal{S} satisfying Theorem 2.6.1(c) (if such exists). It also checks that $T \models \bigvee \mathcal{P}$, and if this is not the case, terminates with an appropriate message to this effect.

```

SOLVE(ACT, WAIT)
{ if WAIT factors ACT
  { REDUCE(ACT, WAIT);
    SOLVE(ACT, WAIT)
  }
else if ( $\exists C \in \text{INT}(T)$ )(conseq( $C$ )  $\subseteq$  ACT and antec( $C$ )  $\cap$  ACT =  $\emptyset$ )
  { EXTEND(ACT, C, WAIT);
    SOLVE(ACT, WAIT)
  }
else { PROC-BRANCH(EXT( $\mathcal{L}$ )  $\cap$  ACT); REDUCE(ACT, WAIT) }
}
  
```

```

PROC-BRANCH( $B$ )
{if ( $\nexists C \in \text{EXT}(T)$ )( $C \subseteq \text{EXT}(B)$ ) {print("∨  $\mathcal{P}$  is not provable"); STOP};
   $\mathcal{S} = \mathcal{S} \cap \text{EXT}(B)$ ;
  if  $\mathcal{S} == \text{EXT}(B)$  { $\mathcal{B} = B$ }
}
main ( )
{ push ({ $P \mid P \in \mathcal{P}$ }, ACT); WAIT =  $\emptyset$ ;  $\mathcal{S} = \text{EXT}(\mathcal{L})$ ; SOLVE(ACT, WAIT);
  if  $\mathcal{S} == \text{EXT}(\mathcal{B})$  { return  $\mathcal{S}$  }
  else { print("no such  $\mathcal{S}$ ") }
}

```

§3 THE FIRST ORDER LEVEL.

In this section we present some examples to illustrate the construction of deduction trees at the first order level. In particular, we show that the use of null values provides a simple way in which the tree construction can be phrased. As was seen at the propositional level, the branches through a suitably defined deduction tree provide the required information to perform view updates. The precise definition of such a deduction tree, and the proof of its required properties is to be found in Sections 4 and 6.

3.1 Terminology.

A first order function free language with nulls and equality has the form

$$\mathcal{L} = \{P_1, P_2, \dots, P_n, =, c_1, c_2, \dots, c_m, \omega_1, \omega_2, \dots\}$$

where $\{P_1, P_2, \dots, P_n, =\}$ is the set of predicate symbols, $\{c_1, c_2, \dots, c_m\}$ is the set of constant symbols ($m > 0$), and $\{\omega_i \mid i = 1, 2, 3, \dots\}$ is a (countably infinite) set of nulls. We implicitly assume the existence of countably many variables x_1, x_2, \dots for the construction of formulae in \mathcal{L} . Again, predicates in \mathcal{L} are either extensional or intensional. Since no defining formulae are needed for the equality predicate, we shall assume that $=$ is extensional.

A *term* in \mathcal{L} is a variable, a constant or a null, and an atomic formula is an expression of the form $K(\mathbf{t})$ where K is a predicate symbol P or its negation $\neg P$, and \mathbf{t} is a sequence of terms of length $\text{arity}(P)$. In the former case $K(\mathbf{t})$ is said to be a *positive atom*.

3.1.1 Definition. A *rule* in \mathcal{L} is a formula C of the form

$$B_1 \vee B_2 \vee \dots \vee B_m \leftarrow A_1 \wedge A_2 \wedge \dots \wedge A_n$$

where:

- (i) Each A_i and each B_j is a positive atom, $m > 0$. If $n > 0$, then each predicate in $\text{conseq}(C)$ is intensional, else if $n = 0$, then each predicate in $\text{conseq}(C)$ is extensional.
- (ii) C is range restricted, meaning that every variable that appears in $\text{conseq}(C)$ also appears in $\text{antec}(C)$.

We say that a rule is *tidy* iff $\text{conseq}(C)$ contains exactly the same variables as does $\text{antec}(C)$. We will denote the set of *untidy* variables in C by

$$\mathcal{U}(C) = \{x \mid x \text{ appears in } \text{antec}(C) \text{ but not in } \text{conseq}(C)\}.$$

$\text{INT}(T)$ again consists of a set of rules with non-empty antecedent. For simplicity we shall assume that $\text{INT}(T)$ does not contain nulls. $\text{EXT}(T)$ consists of (disjunctive) facts of the form

$$B_1 \vee B_2 \vee \dots \vee B_m$$

where each B_i is an extensional positive atom (including expressions of the form $\omega_i = \omega_j$, $\omega_i = c_j$, or $c_i = c_j$). By condition (ii), each such B_i must contain no variables.

As is usual in research into deductive databases, we implicitly assume that T satisfies the domain closure axiom $\forall x \bigvee_i x = c_i$ and the unique name axiom $\bigwedge_{i \neq j} c_i \neq c_j$. The latter will be achieved by the inclusion of rules of the form $\emptyset \leftarrow c_i = c_j$ (for $i \neq j$) in $\text{INT}(T)^*$ (see below). A result of this is that the theory of the first order level is slightly more complicated than a straightforward lifting of the propositional level.

It will simplify our discussion if we assume that T contains the formulae $c_i = c_i$ ($i \leq m$). Thus the fact $c_i = c_j \vee B_1 \vee B_2 \vee \dots \vee B_m$ is trivially true if $i = j$, and equivalent to $B_1 \vee B_2 \vee \dots \vee B_m$ otherwise.

3.1.2 Definition. Let $P(\mathbf{t})$ be a positive atom where \mathbf{t} is a sequence of variables and constants containing x . An *eq-rule* is a rule having the form:

- (i) $P(\mathbf{t}) \leftarrow P(\mathbf{t}(x/y)) \wedge x = y$, where y is a variable in \mathbf{t} , or
 - (ii) $P(\mathbf{t}) \leftarrow P(\mathbf{t}(x/c)) \wedge x = c$, where c is a constant,
- and $\mathbf{t}(x/y)$ ($\mathbf{t}(x/c)$) is obtained from \mathbf{t} by replacing each occurrence of x by y (c).

Note that (because of the existence of infinitely many variables) there are infinitely many eq-rules in \mathcal{L} . In order to guarantee finiteness, we can regard two eq-rules as identical if they are variants of each other. Note that such eq-rules do not satisfy the conditions given in Definition 3.1.1, since P may be extensional. Let

$$\text{INT}(T)^* = \text{INT}(T) \cup \{C \mid C \text{ is an eq-rule in } \mathcal{L}\} \cup \{\emptyset \leftarrow c_i = c_j \mid i \neq j\}.$$

3.1.3 Ground instances and nulls.

Given $C \in \text{INT}(T)^*$, a *null instance* of C is an instance of C of the form $C\theta$, where $\theta : \{x_i \mid 1 \leq i\} \rightarrow \{\omega_i \mid 1 \leq i\} \cup \{c_1, c_2, \dots, c_m\}$. (We shall, in the interests of convenience use both prefix and postfix notation to denote substitution application.) The instance is *ground* if $C\theta$ contains no nulls. Notice that any instance of an eq-rule is trivially valid by the unique name axiom. Obviously a set of rules is regarded as representing its ground instances, thus define

$$gr(\text{INT}(T)) = \{C\theta \mid C \in \text{INT}(T), \theta : \{x_i \mid 1 \leq i\} \rightarrow \{c_1, c_2, \dots, c_m\}\}$$

and

$$gr(\text{INT}(T)^*) = \{C\theta \mid C \in \text{INT}(T)^*, \theta : \{x_i \mid 1 \leq i\} \rightarrow \{c_1, c_2, \dots, c_m\}\}.$$

Null values (appearing in $\text{EXT}(T)$ only) of course represent unknown constants, and when inferring information from T , we require that such information be derivable under

any (consistent) assignment of constants to nulls. Thus let $\exists\omega \text{EXT}(T)$ be the formula

$$\exists u_1 \exists u_2 \dots \exists u_r \text{EXT}(T)(\omega_i/u_i)$$

where $\{\omega_i \mid i \leq r\}$ are those nulls occurring in $\text{EXT}(T)$, $\{u_i \mid i \leq r\}$ are variables not appearing in T , and $\text{EXT}(T)(\omega_i/u_i)$ is formed from $\text{EXT}(T)$ by replacing each occurrence of ω_i by u_i . $\text{EXT}(T)$ is then defined as representing $\exists\omega \text{EXT}(T)$, which in turn is logically equivalent to

$$\bigvee_{\theta} \text{EXT}(T)\theta = \bigvee \{\text{EXT}(T)\theta \mid \theta : \{\omega_i \mid i \leq r\} \rightarrow \{c_1, c_2, \dots, c_m\}\}.$$

Thus T is regarded as representing the ground database

$$\text{gr}(T) = \text{gr}(\text{INT}(T)^*) \cup \bigvee_{\theta} \text{EXT}(T)\theta.$$

3.1.4 Definition. The Herbrand base \mathcal{H} is the set of all ground positive atoms (without nulls). The extended Herbrand base \mathcal{H}_ω consists of the set of all null instances of positive atoms (i.e. allowing nulls but not variables). $\text{EXT}(\mathcal{H})$ denotes those atoms in \mathcal{H} whose predicate is extensional.

3.1.5 Definition. A *model* for \mathcal{L} is a pair (M, f) such that $M \subseteq \mathcal{H} - \{c_i = c_j \mid i \neq j\}$, and $f : \{\omega_i \mid i = 1, 2, 3, \dots\} \rightarrow \{c_1, c_2, \dots, c_m\}$.

Given a (variable free) formula C , and such a function f , $f(C)$ is obtained from C by replacing each null ω by $f(\omega)$. $(M, f) \models C$ iff $M \models f(C)$, in which case (M, f) is said to be a *model* of C .

(M, f) is a *model* of T iff $(M, f) \models C$ for each $C \in T$. If ϕ is a (variable free) formula, then we write $T \models \phi$ iff every model of T is a model of ϕ .

It is easy to show that if ϕ contains no nulls, then $T \models \phi$ iff $\text{gr}(T) \models \phi$. The problem of deciding whether one database is stronger than another is somewhat more complicated. For example, if $T_1 = \{P(\omega_1), Q(\omega_2)\}$, then $T_1 \not\models P(\omega_2) \wedge Q(\omega_1)$ in the sense defined above. However the database T_1 is clearly equivalent to $T_2 = \{P(\omega_2), Q(\omega_1)\}$ in the sense that they both represent the statement $\exists x P(x) \wedge \exists y Q(y)$.

We thus regard T_1 as being stronger than T_2 iff $\text{gr}(T_1) \models \text{gr}(T_2)$. T_1 and T_2 are equivalent iff $\text{gr}(T_1)$ and $\text{gr}(T_2)$ are equivalent.

3.1.6 View updates. Let T and $\mathcal{P} \subseteq \mathcal{H}$ be given.

(a) If $T \not\models \bigvee \mathcal{P}$, then the problem of inserting $\bigvee \mathcal{P}$ into T is to find a database $T(\bigvee \mathcal{P})$ such that

- (i) $\text{INT}(T(\bigvee \mathcal{P})) = \text{INT}(T)$, and $gr(T(\bigvee \mathcal{P})) \models gr(T)$,
 - (ii) $T(\bigvee \mathcal{P}) \models \bigvee \mathcal{P}$, and
 - (iii) if T' satisfies conditions (i) and (ii), then $gr(T') \models gr(T(\bigvee \mathcal{P}))$.
- (b) If $T \models \bigvee \mathcal{P}$, then the problem of deleting $\bigvee \mathcal{P}$ is to generate database(s) T^* such that
- (i) $\text{INT}(T^*) = \text{INT}(T)$, and $gr(T) \models gr(T^*)$,
 - (ii) $T^* \not\models \bigvee \mathcal{P}$, and
 - (iii) whenever T' satisfies (i) and (ii) and $gr(T') \models gr(T^*)$, then $gr(T^*) \models gr(T')$.

As a result of this relationship between the propositional and first order levels, we shall see that our solution to the view update problem at the propositional level forms the basis for our first order method.

The following rephrase for the first order level some of the results needed to build and reason about deduction trees (cf. Section 2.2).

3.1.7 Lemma. Suppose that $\mathcal{C} \subseteq \mathcal{H}$.

- (a) \mathcal{C} is a cover in $gr(\text{INT}(T^*))$ iff $\{c_i = c_j \mid i \neq j\} \subseteq \mathcal{C}$ and \mathcal{C} is a cover in $gr(\text{INT}(T))$.
- (b) If \mathcal{C} is a cover in $gr(\text{INT}(T))$ then $\mathcal{C} \cup \{c_i = c_j \mid i \neq j\}$ is a cover in $gr(\text{INT}(T)^*)$.

3.1.8 Theorem. Let $\mathcal{P} \subseteq \mathcal{H}$, then the following are equivalent:

- (a) $T \models \bigvee \mathcal{P}$.
- (b) Whenever \mathcal{C} is a cover of \mathcal{P} in $gr(\text{INT}(T))$, $\text{EXT}(T) \models \bigvee \mathcal{C}$.
- (c) Whenever \mathcal{C} is a cover of \mathcal{P} in $gr(\text{INT}(T)^*)$, $\text{EXT}(T) \models \bigvee \mathcal{C}$.

3.1.9 Theorem. Let $\mathcal{C} \subseteq \mathcal{H}$, then the following are equivalent:

- (a) $\text{EXT}(T) \models \bigvee \mathcal{C}$.
- (b) For every function $f : \{\omega_i \mid i = 1, 2, 3, \dots\} \rightarrow \{c_1, c_2, \dots, c_m\}$, there is a fact $C \in \text{EXT}(T)$ such that $f(C) \subseteq \mathcal{C} \cup \{c_i = c_j \mid i \neq j\}$.
- (c) There is a fact $E \in \bigvee_{\theta} \text{EXT}(T)\theta$ such that $E \subseteq \mathcal{C} \cup \{c_i = c_j \mid i \neq j\}$.

3.2 Deduction Trees at the First Order Level.

When we attempt to extend our results of Section 2 to the first order level, three main problems present themselves.

- (a) For certain types of database, construction of the appropriate deduction tree may require multiple applications of some rules from $\text{INT}(T)$. Under these circumstances however, it becomes very difficult to assess how many applications are needed in order to

construct a “maximal” tree. In Section 4, we restrict the form of our database in such a way as to prevent the necessity of multiple applications, and this then enables us to generate an appropriate maximal tree via a terminating construction.

(b) As mentioned in the remark following Definition 2.4.2, redundancy in the updated database may only be removed by a consideration of the existing extension. We examine a method employing eq-rules and the multiple applications of rules in $\text{INT}(T)$ whereby such redundancy may be removed. For the sake of simplicity, we do not attempt to incorporate the removal of redundancy into our formal discussion in Section 4.

(c) How should existentially quantified variables appearing in the “output” of our tree traversal be handled?

For the remainder of this section we present several informal examples to illustrate the above-mentioned problems, and to provide motivation for the definitions which follow in Section 4. In the interest of simplicity, we shall usually omit to explicitly represent the rule nodes in deduction trees : in each case it should be obvious which rule is being applied, and the inclusion of the rule node only serves to clutter the diagram.

We first discuss the use of nulls as a method of handling untidy variables.

3.2.1 Example. Suppose that T contains the rules $Q \leftarrow P(x)$, and we wish to insert Q into T . Intuitively the correct formula to insert into T is $\exists x P(x)$. There are various ways in which the existential quantifier can be handled. The simplest approach is to add the fact $P(a_1) \vee P(a_2) \vee \dots \vee P(a_n)$ (where a_1, a_2, \dots, a_n are the constants in \mathcal{L}) to $\text{EXT}(T)$. In practice n will be large, and thus this option is likely to be expensive. Rossi and Naqvi [Ro89] employed null values (thus suggesting that we add the formula $P(\omega)$ to T , where ω is a new null value not already appearing in T). Alternatives to the use of nulls have also been considered [At91, De90, Ka90], each involving some mechanism of deciding upon a suitable value for the unknown variable x .

For the most part, we shall adopt the simplest approach, simply adding $P(\omega)$ to the extension, although our methods could be trivially modified to for example prompt the user for unknown values. In this context, the deduction tree construction enables the system to decide what question the user needs to be asked.

3.2.2 Example. Suppose that T contains the rule $Q \leftarrow P(x, y) \wedge S(y, x)$, and we wish to add Q to T . A deduction tree for Q in $\text{INT}(T)$ is depicted in Figure 3.2.2.

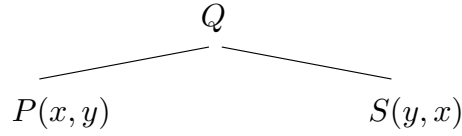


Figure 3.2.2.

Employing null values, we add to $\text{EXT}(T)$ the facts $P(\omega_1, \omega_2)$ and $S(\omega_2, \omega_1)$. As mentioned above, an alternative strategy might be to prompt the user to enter values for ω_1 and ω_2 .

The impracticability of using the constants directly is even more evident in this example. Such an approach would require us to add 2^{n^2} facts, each containing n^2 atoms (where n is the number of constants in the language)!

The following example illustrates the problems in eliminating redundancy from the updated database.

3.2.3 Example. Suppose that T consists of the following rules and facts:

1. $Q \leftarrow P(y)$
2. $Q \leftarrow R(x) \wedge S(x)$
3. $P(a) \vee R(b)$

where P , R and S are extensional, and that we wish to add Q to T . Applying the rules in $\text{INT}(T)$ yields the deduction tree depicted in Figure 3.2.3(i)

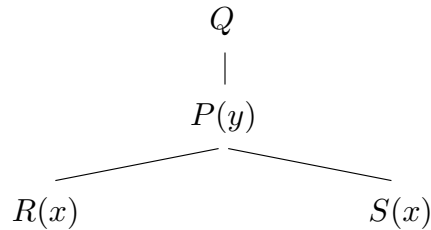


Figure 3.2.3(i).

thus indicating that Q may be inferred from $T_1 = T \cup \{P(\omega_0) \vee R(\omega_1), P(\omega_0) \vee S(\omega_1)\}$ (which we will see later to be equivalent to $T(Q)$).

The inclusion of the two new facts into T does however introduce redundancy. To see this, note that the facts $P(\omega_0) \vee R(\omega_1)$ and $P(\omega_0) \vee S(\omega_1)$ are equivalent to

$$\bigvee \{ [P(c_i) \vee R(c_j)] \wedge [P(c_i) \vee S(c_j)] \mid 1 \leq i, j \leq m \}$$

which in turn reduces to

$$\bigvee_i P(c_i) \vee \bigvee_j (R(c_j) \wedge S(c_j)) \quad \dots\dots(\heartsuit)$$

When converted to conjunctive normal form, (\heartsuit) generates many (disjunctive) facts that are subsumed by $P(a) \vee R(b)$, and hence redundant (since T already contains the fact $P(a) \vee R(b)$).

We may prevent the redundancy by instead inserting (the conjunctive normal form of)

$$S(b) \vee \bigvee_i P(c_i) \vee \bigvee \{R(c_j) \wedge S(c_j) \mid 1 \leq j \leq m, c_j \neq b\} \quad \dots\dots(*)$$

i.e. $P(a) \vee R(b) \models (*) \leftrightarrow (\heartsuit)$.

However, when converted to conjunctive normal form, it easy to see that the formula $(*)$ yields 2^{m-1} facts. Certainly the redundancy would be preferable to allowing T to grow in this way.

A more efficient means of preventing the redundancy would be to suitably modify the new facts (i.e. $P(\omega_0) \vee R(\omega_1)$ and $P(\omega_0) \vee S(\omega_1)$) before we add them to T . This can be achieved by employing eq-rules, cf. Section 3.1.2. (This idea will also prove useful in our construction of first order deduction trees.) Applying $P(y) \leftarrow P(a) \wedge y = a$ and $R(x) \leftarrow R(b) \wedge x = b$ yields the tree depicted in Figure 3.2.3(ii).

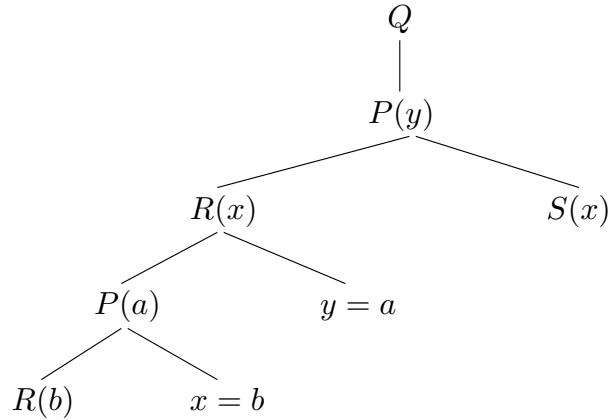


Figure 3.2.3(ii).

The left-most branch is now solved using fact 3, thus suggesting that $T(Q)$ if formed via $T_2 = T \cup \{P(\omega_0) \vee R(\omega_1) \vee P(a) \vee \omega_1 = b, P(\omega_0) \vee R(\omega_1) \vee \omega_0 = a, P(\omega_0) \vee S(\omega_1)\}$, the three new facts being equivalent to

$$\bigvee \{ [P(c_i) \vee R(c_j) \vee P(a) \vee c_j = b] \wedge [P(c_i) \vee R(c_j) \vee c_i = a] \wedge [P(c_i) \vee S(c_j)] \mid 1 \leq i, j \leq m \}$$

.....(†)

It is straightforward to check that the formulae (†) and (*) are equivalent. Hence T_1 and T_2 are equivalent, and the inclusion of the three facts in T_2 does not introduce redundancy in the following sense. Let T' be the set of facts obtained by converting (†) to conjunctive normal form. Then for each fact C in T' , if $\{P(a), R(b)\} \subseteq C$ then either C contains an atom of the form $c = c$, or is properly subsumed by some other fact in T' .

Again, we could consider simplifying the update to T still further by disambiguating the unknowns ω_0 and ω_1 . [Ka90] suggested employing a minimality criteria, which in this case would have the effect of instantiating ω_0 to a and ω_1 to b (since this minimises the amount of new information being inserted into the database).

Notice that when we extract the relevant formula from the deduction tree, the (untidy) variables are instantiated with nulls. It will simplify our presentation to perform this instantiation as soon as the rule is applied in the tree.

3.2.4 Example. Suppose that \mathcal{L} contains just the two constants a and b , and that T consists of the following rules and facts:

1. $Q(x) \leftarrow P(x) \wedge U(x)$
2. $P(x) \leftarrow H(x, y) \wedge R(y)$
3. $Q(x) \vee U(x) \leftarrow S(x, y)$
4. $H(a, a) \vee H(a, b)$

where H , S and R are extensional, and we wish to add $Q(a)$ to T . The tree produced by applying the rules in $\text{INT}(T)$ is depicted in Figure 3.2.4(i).

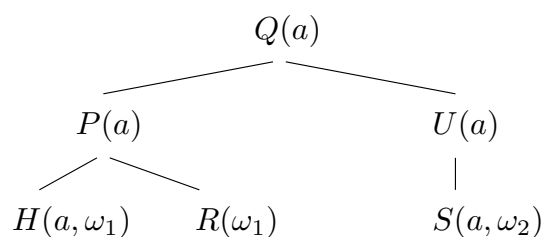


Figure 3.2.4(i).

thus $Q(a)$ may be inferred from $T \cup \{H(a, \omega_1), R(\omega_1), S(a, \omega_2)\}$ (and again we will see later that this is equivalent to $T(Q(a))$). The first two facts are equivalent to

$$(H(a, a) \wedge R(a)) \vee (H(a, b) \wedge R(b))$$

which in turn is equivalent to

$$(H(a, a) \vee H(a, b)) \wedge (H(a, a) \vee R(b)) \wedge (R(a) \vee H(a, b)) \wedge (R(a) \vee R(b))$$

where again the first of these facts is redundant due to the existence of $H(a, a) \vee H(a, b)$ in T . In order to remove the redundancy, we (apparently) need to re-apply rule 2, and then employ eq-rules of the form $H(a, x) \leftarrow H(a, b) \wedge x = b$. The tree produced is depicted in Figure 3.2.4(ii).

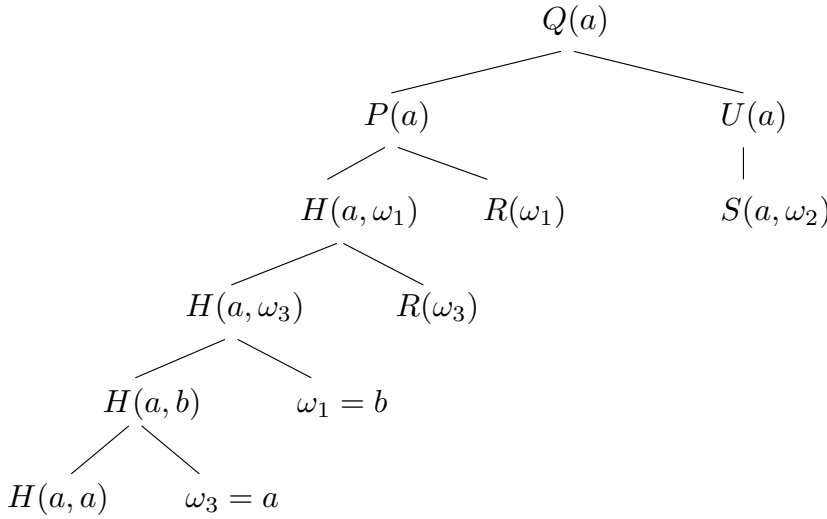


Figure 3.2.4(ii).

and once again we can check that the formula generated is equivalent to

$$(H(a, a) \vee R(b)) \wedge (R(a) \vee H(a, b)) \wedge (R(a) \vee R(b)) \wedge S(a, \omega_2)$$

and thus (as in the preceding example) does not introduce redundancy.

In the above examples we were able to generate $T(\bigvee \mathcal{P})$ via an appropriate deduction tree in $\text{INT}(T)$ without multiple applications of rules in $\text{INT}(T)$. (The multiple applications of rules from $\text{INT}(T)$ were only needed to remove redundancy.) We will see in Section 4 that this was in fact due to the simple structure of the databases considered. The following example illustrates that in general, multiple applications of rules in $\text{INT}(T)$ may be needed to construct the tree.

3.2.5 Example. Suppose that T contains the following:

1. $Q(z) \leftarrow P(z, x)$
2. $P(z, x) \vee P(z, y) \leftarrow R(z, w_0, w_1) \wedge S(x, y)$
3. $R(a, b, c)$

where R and S are extensional, and we wish to add $Q(a)$ to T , then the generation of the deduction tree (depicted in Figure 3.2.5) requires the use of rule 1 twice.

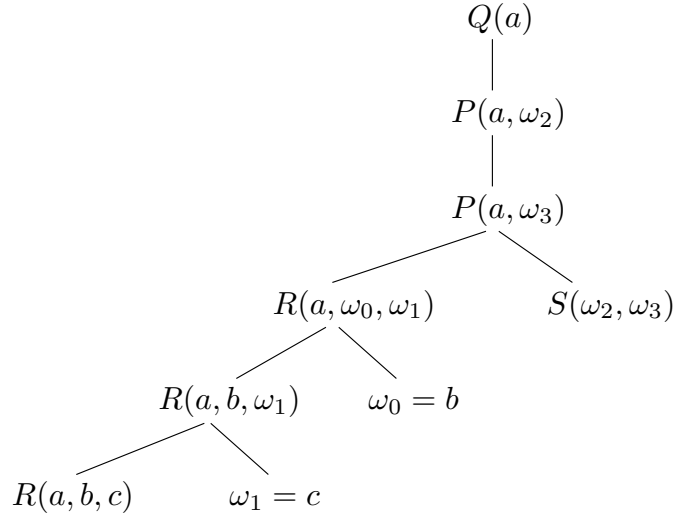


Figure 3.2.5.

thus we can see that to add $Q(a)$ we need to add the formulae

$$(R(a, \omega_0, \omega_1) \vee R(a, b, \omega_1) \vee \omega_1 = c) \wedge (R(a, \omega_0, \omega_1) \vee \omega_0 = b) \wedge S(\omega_2, \omega_3)$$

Moreover, since

$$\exists x_0 \exists x_1 \exists x_2 \exists x_3 [(R(a, x_0, x_1) \vee R(a, b, x_1) \vee x_1 = c) \wedge (R(a, x_0, x_1) \vee x_0 = b) \wedge S(x_2, x_3)]$$

is equivalent to $\exists x_2 \exists x_3 S(x_2, x_3)$, we may disregard the first two disjunctive facts.

We will see in Section 4, that rule 1 needed to be applied more than once precisely because P is not semi-definite. Note also that the eq-rules are being employed so that the fact $R(a, b, c)$ can be applied (in this case to remove redundancy). In fact eq-rules may also be needed in order to construct the tree, as is illustrated in the following example.

3.2.6 Example. Suppose that T contains the following:

1. $Q(z) \leftarrow P(z, x)$

2. $P(z, b) \leftarrow R(z, w) \wedge S(w)$

where R and S are extensional, and we wish to add $Q(a)$ to T . Applying rule 1 yields the tree depicted in Figure 3.2.6(i).

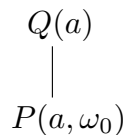


Figure 3.2.6(i).

In order to apply rule 2, we now need to identify ω_0 with the constant b , and hence we use the eq-rule $P(a, x) \leftarrow P(a, b) \wedge x = b$. Applying rule 2 then yields the tree depicted in Figure 3.2.6(ii).

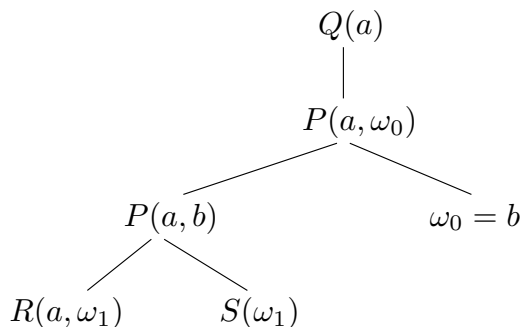


Figure 3.2.6(ii).

Thus we need to add the formula $R(a, \omega_1) \wedge S(\omega_1) \wedge \omega_0 = b$. Clearly the last atom is redundant, since $\exists u_0 \exists u_1 (R(a, u_1) \wedge S(u_1) \wedge u_0 = b)$ is equivalent to $\exists u_1 (R(a, u_1) \wedge S(u_1))$.

Note that in other cases (eg., Example 3.2.3) atoms of the form $\omega = c$ are not redundant in the formula generated.

§4 SEMI-DEFINITE PREDICATES.

As pointed out in the previous section, it may be very difficult to see how many times a rule should be applied in order to generate a “maximal” tree. Note however that if all rules are tidy, then deduction trees are ground (since the root node is ground). An important consequence of this is that (by condition (4) of Definition 2.2.4) each rule needs to be applied at most once along a given branch. The need for multiple applications is thus due to the existence of untidy variables. If these untidy variables appear in semi-definite predicates however, then it can be shown that multiple applications are not needed. This gives us a means of terminating branches through the deduction tree, for which we also need some weak form of stratification.

4.1 Stratification.

Recall (Definition 2.1.5) that $SD(\mathcal{L}) \subseteq \text{INT}(\mathcal{L})$ denotes the set of semi-definite predicates, and $SD(T) = \{C \in \text{INT}(T) \mid \text{conseq}(C) = \{P(\mathbf{t})\}, P \in SD(\mathcal{L})\}$.

Throughout Section 4, we assume that T satisfies the following conditions.

- (a) Suppose that $C \in \text{INT}(T) - SD(T)$. If $R(\mathbf{t}) \in \text{antec}(C)$, R is intensional, and \mathbf{t} contains a variable in $\mathcal{U}(C)$, then $R \in SD(\mathcal{L})$.
- (b) There is a function

$$\ell : SD(\mathcal{L}) \rightarrow \{1, 2, \dots, n\}$$

(where n equals the number of predicates in \mathcal{L}) such that whenever $C \in SD(T)$, $R(\mathbf{t}) \in \text{antec}(C)$ and R is intensional then:

- (i) $\ell(R) \leq \ell(P)$, and
- (ii) if \mathbf{t} contains a variable in $\mathcal{U}(C)$, then $\ell(R) < \ell(P)$.

Notice that in (b), R must be semi-definite by Definition 2.1.5. It will prove useful to extend ℓ to all predicates in \mathcal{L} , thus we define $\ell(P) = 0$ for each $P \in \text{EXT}(\mathcal{L})$, and $\ell(P) = n + 1$ for each intensional predicate that is not semi-definite.

We may also extend ℓ to the rules in $\text{INT}(T)^*$ in the obvious way. Given $C \in \text{INT}(T)^*$, define

$$\ell(C) = \begin{cases} n + 1, & \text{if } C \text{ is indefinite;} \\ \ell(P), & \text{if } C \text{ is definite with } \text{conseq}(C) = \{P(\mathbf{u})\}; \\ 0, & \text{if } C \text{ is of the form } \emptyset \leftarrow c_i = c_j. \end{cases}$$

The reader may also note that eq-rules satisfy similar conditions to those given in (a) and (b), since such rules are tidy, and the ℓ value of each predicate in the antecedent is \leq the ℓ value of the predicate in the consequent. We thus have the following theorem, whose proof is trivial.

4.1.1 Theorem. Let C be a rule in $\text{INT}(T)^*$. If $R(\mathbf{t}) \in \text{antec}(C)$, then

- (a) $\ell(R) \leq \ell(C)$, and
- (b) if \mathbf{t} contains a variable in $\mathcal{U}(C)$, then $\ell(R) < \ell(C)$ and R is semi-definite or extensional.

4.2 The Deduction Tree.

For the sake of simplicity, we shall throughout Section 4 ignore the problems of redundancy, thus concentrating on defining an appropriate deduction tree in $\text{INT}(T)^*$ (cf. the remark following Definition 2.4.2).

In the previous section we saw the need to identify nulls with constants in order to allow some database rule to be applied. For instance in Example 3.2.6 we applied the eq-rule $P(a, x) \leftarrow P(a, b) \wedge x = b$ to the atom $P(a, \omega_0)$ specifically so that the rule $P(z, b) \leftarrow R(z, w) \wedge S(w)$ could be applied. The following definition formalises the idea of “instantiating” nulls for this purpose.

4.2.1 Definition. Suppose that \mathbf{s} is a sequence of variables and constants, \mathbf{t} is a sequence of nulls and constants (of the same length as \mathbf{s}) and that there is a mapping $\eta : \{\omega_i \mid i \geq 1\} \rightarrow \{c_1, c_2, \dots, c_m\}$ such that $\mathbf{t}\eta$ is an instance of \mathbf{s} .

Let the nulls which appear in \mathbf{t} be $\omega_1, \omega_2, \dots, \omega_r$, and let \mathbf{t}' be formed from \mathbf{t} by replacing each ω_i by u_i where u_i is a variable not appearing in \mathbf{s} . Note that \mathbf{s} and \mathbf{t}' are unifiable, thus we may find a most general unifier ρ of \mathbf{s} and \mathbf{t}' . Define

$$\mu_{\mathbf{s}, \mathbf{t}} = \{(\omega_i, c) \mid i \leq r, u_i \rho = c\} \cup \{(\omega_i, \omega_j) \mid 1 \leq i, j \leq r, i \neq j, u_i \rho = u_j \rho \notin \{c_1, c_2, \dots, c_m\}\}.$$

Thus intuitively $\mu_{\mathbf{s}, \mathbf{t}}$ defines the set of minimum instantiations of \mathbf{t} in order to transform \mathbf{t} into a null instance of \mathbf{s} . It is easy to check that $\mu_{\mathbf{s}, \mathbf{t}}$ is well defined (i.e. is independent of the choice of ρ) and moreover that \mathbf{t} is a null instance of \mathbf{s} iff $\mu_{\mathbf{s}, \mathbf{t}} = \emptyset$.

For instance in Example 3.2.6, we had $\mathbf{s} = (z, b)$ and $\mathbf{t} = (a, \omega_0)$. In order to transform \mathbf{t} into a (null) instance of \mathbf{s} we need to identify ω_0 with b , thus $\mu_{\mathbf{s}, \mathbf{t}} = \{(\omega_0, b)\}$.

Most of the features of a first order deduction tree should now be clear from our discussions in Sections 2 and 3. The following definition is followed by two further illustrative examples.

4.2.2 Definition. Given $\mathcal{P} \subseteq \mathcal{H}$, a deduction tree for \mathcal{P} in $\text{INT}(T)$ satisfies the conditions of a deduction tree given in Definition 2.2.4 with the following exceptions.

- (a) Predicate nodes are labelled with an element of \mathcal{H}_ω . No null that appears in \mathcal{T} appears in T .
- (b) If RN is a rule node in \mathcal{T} , then RN has a label of the form $lab(RN) = (C, \chi)$, where $C \in \text{INT}(T)$ or C is an eq-rule, and
- (i) $\chi : \{x \mid x \text{ appears in } C\} \rightarrow \{\omega_i \mid 1 \leq i\} \cup \{c_1, c_2, \dots, c_m\}$.
 - (ii) $(\forall z \in \mathcal{U}(C))(\chi(z) \text{ is a null})$.
 - (iii) $(\forall z \in \mathcal{U}(C))(\forall y \in \{x \mid x \text{ appears in } C\})(y \neq z \implies \chi(y) \neq \chi(z))$.
 - (iv) If $(C, \chi) \neq (D, \psi)$ then $\mathcal{U}(C)\chi \cap \mathcal{U}(D)\psi = \emptyset$.
 - (v) If RN has parent node N , then $\text{conseq}(C\chi) \subseteq \text{ACT}(N)$.
- (c) If RN is a rule node with $lab(RN) = (C, \chi)$, where C is an eq-rule, then $C\chi$ is of the form $P(\mathbf{t}) \leftarrow P(\mathbf{t}(\omega/\alpha)) \wedge \omega = \alpha$ (where ω is a null appearing in \mathbf{t} and α is either a null (appearing in \mathbf{t}) or a constant), and there is some rule $C \in \text{SD}(T)$ with $\text{conseq}(C) = \{P(\mathbf{s})\}$ such that
- (i) $(\exists \eta : \{\omega_i \mid 1 \leq i\} \rightarrow \{c_1, c_2, \dots, c_m\})(\mathbf{t}\eta \text{ is an instance of } \mathbf{s})$, and
 - (ii) $(\omega, \alpha) \in \mu_{\mathbf{s}, \mathbf{t}}$.

The reader should note that some of the features of the above definition have been chosen specifically to meet the needs of databases satisfying the conditions of Section 4.1. We leave it to the reader to define a deduction tree that would be applicable to more general forms of database.

The following example and notes illustrate some of the above points.

4.2.3 Example. Let T consist of the following rules:

- | | |
|---|---------------------------------|
| 1. $Q(x) \leftarrow P(x) \wedge U(x)$ | 4. $H(a, a, x) \leftarrow T(x)$ |
| 2. $P(x) \leftarrow H(x, y, z) \wedge R(y)$ | 5. $P(x) \leftarrow V(z, y, x)$ |
| 3. $Q(x) \vee U(x) \leftarrow S(x, y)$ | 6. $S(a, a)$ |

where R, S, T and V are extensional, then a deduction tree for $Q(a)$ in $\text{INT}(T)$ is depicted in Figure 4.2.3. (The rule node $RN_{\omega_0=a}$ denotes application of the obvious eq-rule).

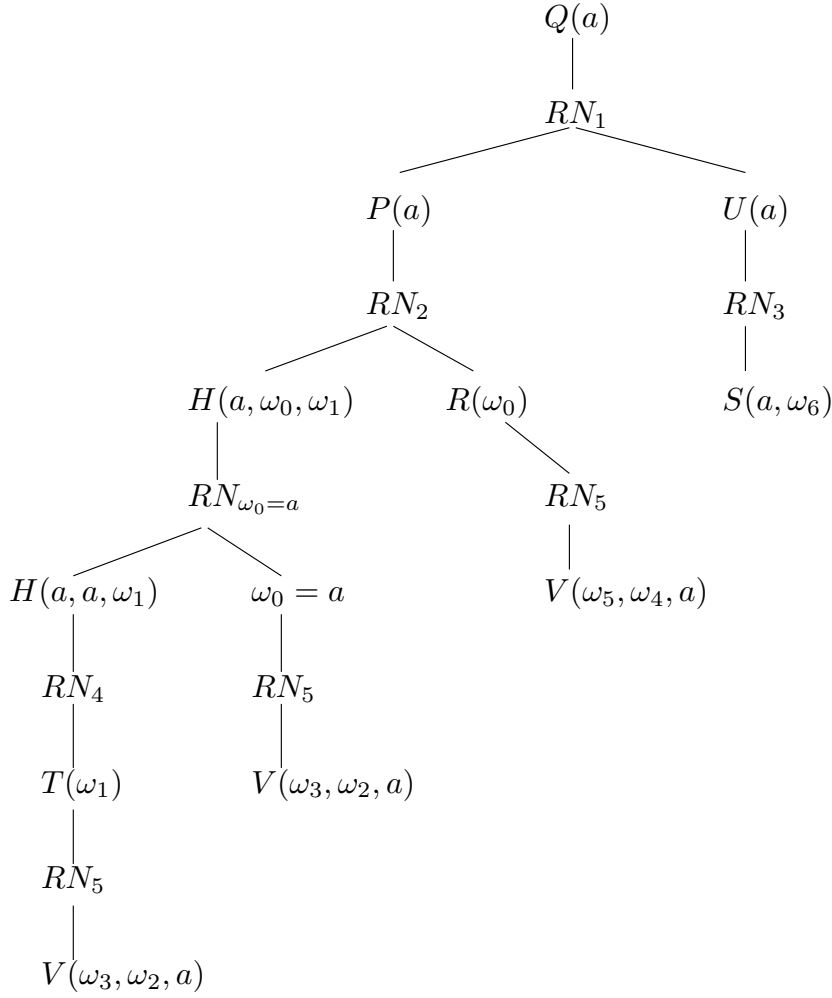


Figure 4.2.3.

This tree in turn generates the formula

$$(T(\omega_1) \vee V(\omega_3, \omega_2, a)) \wedge (\omega_0 = a \vee V(\omega_3, \omega_2, a)) \wedge (R(\omega_0) \vee V(\omega_5, \omega_4, a)) \wedge S(a, \omega_6).$$

Note (1). The same instance $P(a) \leftarrow V(\omega_3, \omega_2, a)$ of rule 5 was applied along the two left-most branches. We could alternatively have used different instances since the formulae

$$\exists(x_0, x_1, x_2, x_3, x_4, x_5) [(T(x_1) \vee V(x_3, x_2, a)) \wedge (x_0 = a \vee V(x_3, x_2, a)) \wedge (R(x_0) \vee V(x_5, x_4, a))]$$

and

$$\exists(x, y, x_0, x_1, x_2, x_3, x_4, x_5)$$

$$[(T(x_1) \vee V(x_3, x_2, a)) \wedge (x_0 = a \vee V(y, x, a)) \wedge (R(x_0) \vee V(x_5, x_4, a))]$$

are equivalent. The usage of different instances does have the disadvantage of introducing more nulls. An advantage however is seen in note (5) below.

Notice that (by condition (b)(iv) of Definition 4.2.2) we could not for instance have used say the instance $P(a) \leftarrow V(\omega_2, \omega_4, a)$ along the third left-most branch.

(2). Rule 4 cannot be applied against $H(a, \omega_0, \omega_1)$ since it has ω_0 as its second argument, thus necessitating the application of the eq-rule $H(a, x, y) \leftarrow H(a, a, y) \wedge x = a$. In this case only one application of an eq-rule is necessary to enable rule 4 to be used, but of course in general, several eq-rule applications may be needed (if there are several nulls that require instantiating, cf., Example 3.2.5). Condition (c) in Definition 4.2.2 effectively says that an eq-rule is applicable if there is some sequence of eq-rule applications (including the one under consideration) that are necessary in order to allow the application of some definite rule in $\text{INT}(T)$.

Notice also that rule 4 is not applicable against the second left-most branch, since $RN_{\omega_0=a}$ has already been applied (and thus may not (by condition (4) of Definition 2.2.4) be re-applied).

(3). Clearly there is no reason to assume that the value (ω_0) that might satisfy R is the same as the value (ω_6) that might satisfy S , hence the use of different nulls (cf. conditions (b) (ii) - (iv) of Definition 4.2.2).

(4). In this section we are only considering trees in $\text{INT}(T)$, and as mentioned earlier, this allows potential redundancy. If we wished, redundancy could be eliminated (cf. Example 3.2.5) by the application of the eq-rule $S(a, x) \leftarrow S(a, a) \wedge x = a$ to the right-most branch, thus resulting in the formula

$$(T(\omega_1) \vee V(\omega_3, \omega_2, a)) \wedge (\omega_0 = a \vee V(\omega_3, \omega_2, a)) \wedge (R(\omega_0) \vee V(\omega_5, \omega_4, a)).$$

(5). Similarly, if T had contained the fact $V(c, b, a)$, then by applications of eq-rules equating ω_3/ω_5 and ω_2/ω_4 with c and b respectively, we could remove the three left-most branches from the formula generated.

If on the other hand, $\text{EXT}(T)$ had contained (say) the fact $T(c) \vee V(c, c, a)$, then eq-rules equating ω_1, ω_2 and ω_3 with c could have been applied to the left-most branch, but this would not have enabled us to eliminate this branch from the formula generated (since the second left-most branch also contains ω_3 and ω_2).

This points to the advantage of using a different instance of rule 5 along the second left-most branch.

4.2.4 Example. Let T contain the following rules and facts.

1. $Q(x, y) \leftarrow P(x, y) \wedge P(y, x)$
2. $P(z, w) \leftarrow S(z, w, u)$
3. $Q(x, y) \leftarrow H(y, x, v)$

and suppose that we wish to insert the tuple $Q(a, b)$. Applying the rules in T gives the tree depicted in Figure 4.2.4.

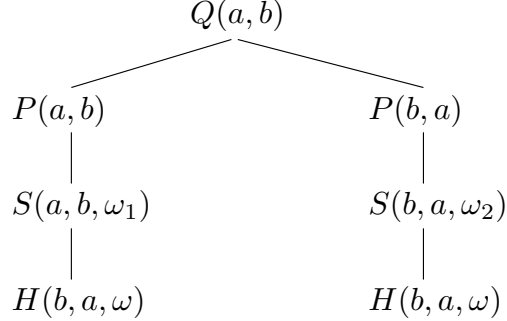


Figure 4.2.4.

Notice that the same instance of rule 3 may be applied to both branches. The same cannot be said for rule 2, since the consequents required are different. In particular, there is no reason why the value (ω_1) which satisfies $S(a, b, u)$ should be same as that which satisfies $S(b, a, u)$ (i.e. ω_2), thus different nulls must be employed (cf. condition (b)(iv) of Definition 4.2.2).

4.2.5 The maximal tree.

Throughout the remainder of Section 4, \mathcal{T} will denote a deduction tree for \mathcal{P} in $\text{INT}(T)$ such that

- (a) For each branch B through \mathcal{T} , if B contains two rule nodes of the form $RN_{(C,\phi)}$ and $RN_{(C,\theta)}$, then $\text{conseq}(C)\theta \neq \text{conseq}(C)\phi$.
- (b) No unfactored branch through \mathcal{T} can be extended by a rule in $\text{INT}(T)^*$ without contravening either condition (a) or the conditions of Definition 4.2.2.

Condition (a) is the precise formalism of our requirement that multiple applications should not be employed, and condition (b) of course expresses maximality. Since eq-rules are tidy the following property is a consequence of (a) and condition (4) of Definition 2.2.4.

- (c) For each branch B through \mathcal{T} , if B contains two rule nodes of the form $RN_{(C,\phi)}$ and $RN_{(D,\theta)}$, where C and D are variants of the same eq-rule, then $\text{conseq}(C)\phi \neq \text{conseq}(D)\theta$.

It is relatively easy to show (see Section 6) that \mathcal{T} is finite. We aim to show (in Sections 4.3 and 4.4 respectively) that

- (i) if $T \not\models \bigvee \mathcal{P}$, then a traversal of \mathcal{T} can be used to construct $T(\bigvee \mathcal{P})$, and

- (ii) if $T \models \bigvee \mathcal{P}$, then a traversal of \mathcal{T} can be used to decide if there is a database satisfying the conditions of Theorem 2.6.1(a), and to construct this database (if it exists).

4.2.6 Traversing \mathcal{T} .

Notice that \mathcal{T} is actually variable free, thus \mathcal{T} may be constructed (traversed) in a similar fashion to that seen at the propositional level (Sections 2.4.7 and 2.6.5). The *only* difference at the first order level lies in the conditions imposed on a rule used in an extension step. At the propositional level the conditions on an intensional rule were that $\text{conseq}(C) \subseteq \text{ACT}$ and $\text{antec}(C) \cap \text{ACT} = \emptyset$ (cf. conditions (4) and (5) of Definition 2.2.4). At the first order level, we have to pick a rule $C \in \text{INT}(T)$ and a substitution θ where that $\text{conseq}(C\theta) \subseteq \text{ACT}$, $\text{antec}(C\theta) \cap \text{ACT} = \emptyset$, and such that the conditions given in Definitions 4.2.2 and Section 4.2.5 are satisfied.

It is worth noting certain features of this construction which promote and suggest efficiency.

- (i) Firstly, if $P(\mathbf{t})$ is an atom labelling a predicate node in the tree with $P \in \text{INT}(\mathcal{L}) - \text{SD}(\mathcal{L})$, then \mathbf{t} contains no nulls, and hence $P(\mathbf{t})$ is ground. Thus if C is indefinite (then $C \notin \text{SD}(T)$) and (C, θ) is used to extend ACT, then $C\theta$ is obtained from C by mapping each variable in $\text{conseq}(C)$ to a constant, and each variable in $\mathcal{U}(C)$ to a new null. The application of indefinite rules is thus somewhat simplified.
- (ii) When applying a rule $C \in \text{SD}(T)$ to an existing atom $P(\mathbf{t})$ in the tree, the behaviour of the nulls in \mathbf{t} is similar to that of variables in a Horn clause resolution step, with the exception that such nulls are “instantiated” via the eq-rules. Such usage of the eq-rules makes the presentation somewhat clearer (since existing atoms within the tree do not change as a result of subsequent substitutions), and also permits a single structure to handle the application of other rules (involving other instantiations) to $P(\mathbf{t})$.
- (iii) The general structure of a deductive database will normally be a large extension, and a relatively smaller intension. It is worth noting that the tree construction only requires access to the intension.
- (iv) It is not necessarily the case that any of the branches through \mathcal{T} will be redundant (and we hope that factorisation will catch some of those that are), nor does any branch contain duplicate atoms, thus (following Theorem 4.2.8 below) we might argue that the tree is as complex as it needs to be in order to compute the covers.

The following two theorems prove the vital relationship between branches through \mathcal{T} and covers of \mathcal{P} . Let $(B_i \mid 1 \leq i \leq j)$ enumerates the unfactored branches through \mathcal{T} , and let $(\psi_k \mid 1 \leq k \leq k')$ list those mappings of nulls (that appear in \mathcal{T}) to constants. Unfortunately, $\mathcal{T}\psi_k$ might fail to be a deduction tree for \mathcal{P} in $\text{gr}(\text{INT}(T)^*)$, since the

instantiation ψ_k might unify the labels of two different predicate nodes along the same branch. This however is a minor point, and we can still prove the following theorems.

4.2.7 Theorem. Let $\mathcal{C} \subseteq \mathcal{H}$ be a cover of \mathcal{P} in $gr(INT(T)^*)$, then for each $k \leq k'$, there is an unfactored branch B through \mathcal{T} such that $ACT(B)\psi_k \subseteq \mathcal{C}$.

Proof. Let $RN_{(C,\theta)}$ be a rule node in \mathcal{T} , and N be the parent node of $RN_{(C,\theta)}$, then by condition (b)(v) of Definition 4.2.2, $conseq(C\theta\psi_k) \subseteq ACT(N)\psi_k$. Also, if $\{N_1, N_2, \dots, N_e\}$ are the child nodes of $RN_{(C,\theta)}$, then $\{lab(N_d)\psi_k \mid d \leq e\} = antec(C\theta\psi_k)$.

Thus we may construct a branch B through \mathcal{T} such that for each predicate node N on B , $lab(N)\psi_k \in \mathcal{C}$, and for each right sibling M of N , $lab(M)\psi_k \notin \mathcal{C}$. Since $B\psi_k$ is not factored in $\mathcal{T}\psi_k$, B cannot be factored in \mathcal{T} . ■

4.2.8 Theorem. For each function $f : \{1, 2, \dots, k'\} \rightarrow \{1, 2, \dots, j\}$, either

- (a) $\bigcup_{k=1}^{k'} ACT(B_{f(k)}\psi_k)$ contains an atom of the form $c_i = c_i$, or
- (b) $\bigcup_{k=1}^{k'} ACT(B_{f(k)}\psi_k)$ contains a cover of \mathcal{P} in $gr(INT(T))$.

Because of its complexity, the proof of the above theorem is presented in the Appendix (Section 6).

4.3 View Updates : Insertions.

In this section we consider the case of insertions, thus we show that if $T \not\models \bigvee \mathcal{P}$, then (as in Section 2.4.2) $T(\bigvee \mathcal{P}) = T \cup \{\bigvee EXT(B_i) \mid i \leq j\}$. Deletions are considered in Section 4.4.

4.3.1 Theorem.

- (a) for each $k \leq k'$, $INT(T) \cup \{\bigvee EXT(B_i)\psi_k \mid i \leq j\} \models \bigvee \mathcal{P}$.
- (b) $INT(T) \cup \{\bigvee EXT(B_i) \mid i \leq j\} \models \bigvee \mathcal{P}$.

Proof (a). Suppose that $INT(T) \cup \{\bigvee EXT(B_i)\psi_k \mid i \leq j\} \not\models \bigvee \mathcal{P}$, then by Theorem 3.1.8, we may find a cover \mathcal{C} of \mathcal{P} in $gr(INT(T)^*)$ such that $INT(T) \cup \{\bigvee EXT(B_i)\psi_k \mid i \leq j\} \not\models \bigvee \mathcal{C}$. By Theorem 4.2.7, we may find an unfactored branch B such that $ACT(B)\psi_k \subseteq \mathcal{C}$, thus $INT(T) \cup \{\bigvee EXT(B_i)\psi_k \mid i \leq j\} \not\models \bigvee ACT(B)\psi_k$, an obvious contradiction since $B = B_i$ for some $i \leq j$.

(b). This follows trivially from part (a) since $\{\bigvee EXT(B_i) \mid i \leq j\}$ is equivalent to

$\bigvee_{k=1}^{k'} \bigwedge_{i=1}^j \bigvee \text{EXT}(B_i)\psi_k$. ■

Hence $T \cup \{\bigvee \text{EXT}(B_i) \mid i \leq j\} \models \bigvee \mathcal{P}$. Theorem 4.2.8 allows us to deduce that this database is no stronger than is necessary.

4.3.2 Corollary. Suppose that $\text{INT}(T') = \text{INT}(T)$ with $T' \models \bigvee \mathcal{P}$. Then

$$T' \models \exists \omega \bigwedge_{i=1}^j \bigvee \text{EXT}(B_i).$$

Proof. Again, $\bigvee_{k=1}^{k'} \bigwedge_{i=1}^j \bigvee \text{ACT}(B_i\psi_k)$ is logically equivalent to

$$\bigwedge_f \left\{ \bigvee_{k=1}^{k'} \bigvee \text{ACT}(B_{f(k)}\psi_k) \mid f : \{1, 2, \dots, k'\} \rightarrow \{1, 2, \dots, j\} \right\}.$$

The result then follows from Theorems 4.2.8 and 3.1.8. ■

4.3.3 Corollary. Suppose that $T \not\models \bigvee \mathcal{P}$, then $T \cup \{\bigvee \text{EXT}(B_i) \mid i \leq j\}$ is logically equivalent to $T(\bigvee \mathcal{P})$.

4.4 View Updates : Deletions.

4.4.1. Assume that $T \models \bigvee \mathcal{P}$, and we wish to delete the inference of $\bigvee \mathcal{P}$. For the sake of brevity, we shall only concern ourselves with the problem of establishing whether there is a unique modification T^* against the extension (cf. Section 2.6). Specifically T^* should satisfy the following conditions.

- (a) $\text{INT}(T^*) = \text{INT}(T)$, $gr(T) \models gr(T^*)$ and $T^* \not\models \bigvee \mathcal{P}$, and
- (b) whenever $\text{INT}(T') = \text{INT}(T)$, $gr(T) \models gr(T')$ with $T' \not\models \bigvee \mathcal{P}$, then $gr(T^*) \models gr(T')$.

For the remainder of Section 4.4, let $\mathcal{S} = \{K \in \text{EXT}(\mathcal{H}) \mid \text{INT}(T) \models K \rightarrow \bigvee \mathcal{P}\}$. As in Section 2.6 (see Corollary 2.6.2) this set plays a crucial role in that if such a T^* exists, then it is equivalent to a database of the form $T(\neg \bigvee \mathcal{S})$. Our aim is to show that \mathcal{T} may be used to construct \mathcal{S} , and also to test whether $T(\neg \bigvee \mathcal{S})$ satisfies conditions (a) and (b) above.

Notice that $\{c_r = c_s \mid r \neq s\} \subseteq \mathcal{S}$. If $\{c_r = c_r \mid r \leq m\} \cap \mathcal{S} \neq \emptyset$, then there is no database T' such that $\text{INT}(T') = \text{INT}(T)$ and $T' \not\models \bigvee \mathcal{P}$. Thus we assume that $\{c_r = c_r \mid r \leq m\} \cap \mathcal{S} = \emptyset$.

Lemma 4.4.2 below is an immediate consequence of Theorem 3.1.8.

4.4.2 Lemma. If $K \in \text{EXT}(\mathcal{H}) - \{c_r = c_s \mid r \neq s\}$, then $K \in \mathcal{S}$ iff whenever \mathcal{C} is a cover of \mathcal{P} in $gr(\text{INT}(T)^*)$, then $(\{K\} \cup \{c_r = c_r \mid r \leq m\}) \cap \mathcal{C} \neq \emptyset$.

Note in particular, that if \mathcal{C} is a cover of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $\mathcal{C} \cap \{c_r = c_r \mid r \leq m\} = \emptyset$, then $\mathcal{S} \subseteq \mathcal{C}$.

In view of the close relationship between covers of \mathcal{P} and branches through \mathcal{T} , it is no surprise that \mathcal{T} provides a membership test for \mathcal{S} .

4.4.3 Theorem. Suppose that $K \in \text{EXT}(\mathcal{H}) - \{c_r = c_s \mid r \neq s\}$, then $K \in \mathcal{S}$ iff

$$(\exists k \leq k')(\forall i \leq j) [(\{K\} \cup \{c_r = c_r \mid r \leq m\}) \cap \text{EXT}(B_i \psi_k) \neq \emptyset].$$

Proof (\rightarrow). Suppose that $K \in \mathcal{S}$ and for each $k \leq k'$ there is an $f(k) \leq j$ such that $(\{K\} \cup \{c_r = c_r \mid r \leq m\}) \cap \text{EXT}(B_{f(k)} \psi_k) = \emptyset$. Pick a cover \mathcal{D} of \mathcal{P} in $gr(\text{INT}(T))$ such that $\mathcal{D} \subseteq \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)} \psi_k)$, then $\mathcal{D} \cup \{c_r = c_s \mid r \neq s\}$ is a cover in $gr(\text{INT}(T)^*)$, and since $\mathcal{D} \cap \{c_r = c_r \mid r \leq m\} = \emptyset$ we must have that $K \in \mathcal{D}$ (by Lemma 4.4.2), a contradiction.

(\leftarrow). For the converse, let \mathcal{C} be a cover of \mathcal{P} in $gr(\text{INT}(T)^*)$, and suppose that $(\{K\} \cup \{c_r = c_r \mid r \leq m\}) \cap \mathcal{C} = \emptyset$. By Theorem 4.2.7 we may find a function $f : \{1, 2, \dots, k'\} \rightarrow \{1, 2, \dots, j\}$ such that $\mathcal{C} \supseteq \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)} \psi_k)$. Hence $(\{K\} \cup \{c_r = c_r \mid r \leq m\}) \cap \text{EXT}(B_{f(k)} \psi_k) = \emptyset$ for each $k \leq k'$, thus contradicting our hypothesis. ■

The following generalises Definition 2.5.1 to the first order level.

4.4.4 Definition. If $\mathcal{A} \subseteq \text{EXT}(\mathcal{H})$, then let

$$T(\neg \bigvee \mathcal{A}) = \text{INT}(T) \cup \{C \vee P \mid C \in \text{EXT}(T), P \in \text{EXT}(\mathcal{H}) - \mathcal{A}\} \cup \{c_r = c_r \mid r \leq m\}.$$

4.4.5 Lemma. If $\mathcal{A} \subseteq \text{EXT}(\mathcal{H})$, then

- (a) $gr(T(\neg \bigvee \mathcal{A}))$ and $(gr(T))(\neg \bigvee \mathcal{A})$ are equivalent, and
- (b) if $\{c_r = c_r \mid r \leq m\} \cap \mathcal{A} = \emptyset$ and $\{c_r = c_s \mid r \neq s\} \subseteq \mathcal{A}$, then $T(\neg \bigvee \mathcal{A}) \not\equiv \bigvee \mathcal{A}$.

The following is the first order analogue of Theorem 2.6.1. As mentioned earlier, the details of the first order level are slightly different to those of the propositional level, thus we sketch the details of the proof.

4.4.6 Theorem. The following are equivalent.

- (a) There is a database T^* satisfying conditions (a) and (b) of Section 4.4.1.
- (b) There is a cover \mathcal{C} of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $\mathcal{S} = \mathcal{C} \cap \text{EXT}(\mathcal{H})$.
- (c) $T(\neg \bigvee \mathcal{S})$ satisfies conditions (a) and (b) of Section 4.4.1.

Proof (a) \rightarrow (b). Since $T^* \not\models \bigvee \mathcal{P}$ we may (by Theorem 3.1.8) find an ext-minimal cover \mathcal{C} of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $gr(T^*) \not\models \bigvee(\mathcal{C} \cap \text{EXT}(\mathcal{H}))$.

By assumption, T^* contains the atoms $c_r = c_r$ ($r \leq m$), and thus each such atom cannot belong to \mathcal{C} . By Lemma 4.4.2, we must have that $\mathcal{S} \subseteq \mathcal{C}$.

Suppose that $\mathcal{S} \subset \mathcal{C} \cap \text{EXT}(\mathcal{H})$, say $K \in \mathcal{C} \cap \text{EXT}(\mathcal{H}) - \mathcal{S}$, then $K \notin \{c_r = c_s \mid 1 \leq r, s \leq m\}$. Since $K \notin \mathcal{S}$ we may (by Lemma 4.4.2) find a cover \mathcal{D} of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $(\{K\} \cup \{c_r = c_r \mid r \leq m\}) \cap \mathcal{D} = \emptyset$. Hence $\mathcal{C} \cap \text{EXT}(\mathcal{H}) \not\subseteq \mathcal{D} \cap \text{EXT}(\mathcal{H})$, and thus as in Lemma 2.5.2(b), $gr(T)(\neg \bigvee(\mathcal{D} \cap \text{EXT}(\mathcal{H}))) \models \bigvee(\mathcal{C} \cap \text{EXT}(\mathcal{H}))$.

By Lemma 4.4.5(b), $gr(T)(\neg \bigvee(\mathcal{D} \cap \text{EXT}(\mathcal{H}))) \not\models \bigvee(\mathcal{D} \cap \text{EXT}(\mathcal{H}))$, and thus by Theorem 3.1.8, $gr(T)(\neg \bigvee(\mathcal{D} \cap \text{EXT}(\mathcal{H}))) \not\models \bigvee \mathcal{P}$. Hence, by condition (b) of Section 4.4.1, $gr(T^*) \models gr(T)(\neg \bigvee(\mathcal{D} \cap \text{EXT}(\mathcal{H})))$, which in turn contradicts the fact that $gr(T^*) \not\models \bigvee(\mathcal{C} \cap \text{EXT}(\mathcal{H}))$.

(b) \rightarrow (c). Let \mathcal{C} be a cover of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $\mathcal{S} = \mathcal{C} \cap \text{EXT}(\mathcal{H})$. By Lemma 4.4.5(b), $T(\neg \bigvee \mathcal{S}) \not\models \bigvee \mathcal{S}$, and thus by Theorem 3.1.8, $T(\neg \bigvee \mathcal{S}) \not\models \bigvee \mathcal{P}$.

Let T' be a database such that $\text{INT}(T') = \text{INT}(T)$, $gr(T) \models gr(T')$ and $T' \not\models \bigvee \mathcal{P}$. By Theorem 3.1.8 we may find a cover \mathcal{D} of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $\text{EXT}(T') \not\models \bigvee \mathcal{D}$, and by Lemma 4.4.2, $\mathcal{S} \subseteq \mathcal{D}$.

Let $E \in \bigvee_{\theta} \text{EXT}(T')\theta$, then $E \not\subseteq \mathcal{D}$, and hence $E \not\subseteq \mathcal{S}$. But then as in Lemma 2.5.2(b), $T(\neg \bigvee \mathcal{S}) \models E$. ■

Again, using the relationship between covers of \mathcal{P} and branches through \mathcal{T} , we can easily show that \mathcal{T} may be used to check whether condition (b) of Theorem 4.4.6 holds.

4.4.7 Theorem. There is a cover \mathcal{C} of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $\mathcal{S} = \mathcal{C} \cap \text{EXT}(\mathcal{H})$ iff $(\forall k \leq k')(\exists i \leq j)(\text{EXT}(B_i\psi_k) \subseteq \mathcal{S})$.

Proof (\rightarrow). Suppose there is a cover \mathcal{C} of \mathcal{P} in $gr(\text{INT}(T)^*)$ such that $\mathcal{S} = \mathcal{C} \cap \text{EXT}(\mathcal{H})$, then by Theorem 4.2.7, we may find a function f such that $\mathcal{C} \supseteq \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k)$. But then $\mathcal{S} \supseteq \bigcup_{k=1}^{k'} \text{EXT}(B_{f(k)}\psi_k)$, whence the result follows.

(\leftarrow). Let f be such that $\mathcal{S} \supseteq \bigcup_{k=1}^{k'} \text{EXT}(B_{f(k)}\psi_k)$. Let \mathcal{D} be a cover of \mathcal{P} in $gr(\text{INT}(T))$ such that $\mathcal{D} \subseteq \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k)$. Let $\mathcal{C} = \mathcal{D} \cup \{c_r = c_s \mid r \neq s\}$, then \mathcal{C} is a cover in $gr(\text{INT}(T)^*)$ with $\mathcal{C} \cap \text{EXT}(\mathcal{H}) \subseteq \mathcal{S}$. By the hypothesis given at the beginning of this section, $\{c_r = c_r \mid r \leq m\} \cap \mathcal{S} = \emptyset$, thus $\{c_r = c_r \mid r \leq m\} \cap \mathcal{C} = \emptyset$ and hence by Lemma

4.4.2, $\mathcal{S} \subseteq \mathcal{C}$. This proves that $\mathcal{S} = \mathcal{C} \cap \text{EXT}(\mathcal{H})$. ■

To conclude, we present two examples illustrating the usage of the above theorems to perform view deletions.

4.4.8 Example. Suppose that T contains the following rules

1. $Q(x) \leftarrow R(y, x) \wedge S(x)$
2. $S(x) \leftarrow P(x, y)$
3. $R(u, x) \leftarrow P(x, u)$
4. $P(a, b) \vee P(a, c)$

and we wish to delete $Q(a)$. The maximal deduction tree is illustrated in Figure 4.4.8.

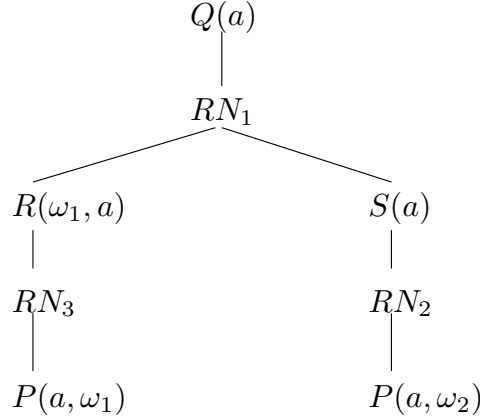


Figure 4.4.8

By Theorem 4.4.3, if $K \in \text{EXT}(\mathcal{H}) - \{c_r = c_s \mid r \neq s\}$, then $K \in \mathcal{S}$ iff $(\exists k \leq k')(\forall i \leq j) [K \in \text{EXT}(B_i\psi_k)]$, and thus $\mathcal{S} = \{c_r = c_s \mid r \neq s\} \cup \{P(a, a), P(a, b), P(a, c)\}$. Moreover, by Theorems 4.4.6 and 4.4.7, $T(\neg \bigvee \mathcal{S})$ satisfies the conditions of Section 4.4.1 iff for each mapping $\psi : \{\omega_1, \omega_2\} \rightarrow \{a, b, c\}$, either $P(a, \omega_1\psi) \in \mathcal{S}$ or $P(a, \omega_2\psi) \in \mathcal{S}$ (which is trivially the case).

To perform the view deletion, we can either compute

$$T(\neg \bigvee \mathcal{S}) = \text{INT}(T) \cup \{a = a, b = b, c = c\} \cup \{P(a, b) \vee P(a, c) \vee P(a, a),$$

$$P(a, b) \vee P(a, c) \vee P(b, a), P(a, b) \vee P(a, c) \vee P(b, b), P(a, b) \vee P(a, c) \vee P(b, c),$$

$$P(a, b) \vee P(a, c) \vee P(c, a), P(a, b) \vee P(a, c) \vee P(c, b), P(a, b) \vee P(a, c) \vee P(c, c)\}$$

or we could (say) prompt the user to resolve the inconsistency between the existing fact $P(a, b) \vee P(a, c)$, and the requirement that $P(a, a) \vee P(a, b) \vee P(a, c)$ should not be inferred.

4.4.9 Example. Suppose that T contains the following rules

- | | | |
|--|------------------------------|---------------------------------|
| 1. $Q(x) \leftarrow R(y, x) \wedge S(x)$ | 2. $S(x) \leftarrow P(x, y)$ | 3. $R(u, x) \leftarrow P(x, u)$ |
| 4. $R(u, x) \leftarrow T(u, x)$ | 5. $S(x) \leftarrow V(x)$ | 6. $P(a, a) \vee P(a, b)$ |
| 7. $T(a, a)$ | 8. $V(a) \vee V(b)$ | |

and we wish to delete $Q(a)$. The maximal deduction tree is illustrated in Figure 4.4.9.

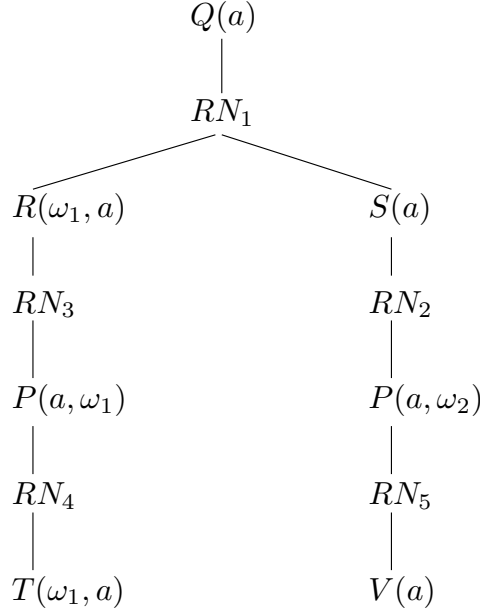


Figure 4.4.9

As in the previous example, $\mathcal{S} = \{c_r = c_s \mid r \neq s\} \cup \{P(a, a), P(a, b)\}$ and $T(\neg \bigvee \mathcal{S})$ satisfies the conditions of Section 4.4.1 iff for each mapping $\psi : \{\omega_1, \omega_2\} \rightarrow \{a, b\}$, either $\{T(\omega_1\psi, a), P(a, \omega_1\psi)\} \subseteq \mathcal{S}$ or $\{V(a), P(a, \omega_2\psi)\} \subseteq \mathcal{S}$ (which is trivially *not* the case).

Thus there is no single update against the extension. However, if \mathcal{C} is a cover of $Q(a)$ in $\text{INT}(T)$, then $\mathcal{C} \cap \text{EXT}(\mathcal{L})$ is of the form $\bigcup_{k=1}^{k'} \text{EXT}(B_{f(k)}\psi_k)$, and thus we might prompt the user to modify the existing facts $\{P(a, a) \vee P(a, b), T(a, a)\}$ in order to prevent the inference of either

- (i) $P(a, a) \vee P(a, b) \vee T(a, a) \vee T(b, a)$, or
- (ii) $P(a, a) \vee P(a, b) \vee V(a)$.

The above examples suggest that the complexities of performing the deletion are overwhelming (with or without the help of the user) in cases where the conditions of Theorem 4.4.7 are not met.

§5 CONCLUSIONS AND OPEN QUESTIONS.

We have seen that the concept of a deduction tree provides a powerful tool with which to attack the view update problem. At the propositional level, traversal of the appropriate tree yields sufficient information to perform view updates.

At the first order level three problems present themselves : constructing a “maximal” tree, existential quantifiers (and the associated problem of disambiguating the update against the extension) and the removal of redundancy. The first of these has lead us to make assumptions (concerning the structure of our database, cf., Section 4.1) under which a maximal deduction tree can be straightforwardly generated via a terminating construction. We have not directly addressed the second issue, although we have indicated that a disambiguation policies could be employed alongside the methods of the current paper.

We have also indicated that the problem of redundancy can only be addressed by an examination of the extension, possibly using eq-rules.

In [Jo96] we consider extensions of the results presented in this paper to the context of stratified disjunctive databases (in which rules bodies may contain negative atoms) under the semantics defined by perfect models.

We suggest the following problems as being worthy of further consideration.

(1). We have indicated that the equality predicate is extensional, but this then disallows T from containing integrity constraints of the form $y_0 = y_1 \leftarrow \phi(\mathbf{x}, y_0) \wedge \phi(\mathbf{x}, y_1)$. Moreover, if T is allowed to contain such rules, then should these be used actively in the generation of deduction trees, or should they be used to constrain the construction?

(2). Suppose that a rule C is of the form $\theta(\mathbf{x}) \leftarrow \phi(\mathbf{x}, \mathbf{y})$, i.e., \mathbf{x} denotes those variables that appear in $\text{conseq}(C)$, and $\mathbf{y} = y_1, y_2, \dots, y_k$ denotes the untidy variables in C . Suppose also that

$$T \models \mathbf{y} = \mathbf{y}_1 \leftarrow \phi(\mathbf{x}, \mathbf{y}) \wedge \phi(\mathbf{x}, \mathbf{y}_1).$$

It seems plausible that \mathbf{y} need not satisfy the requirement given in Section 4.1, since C can be converted into a tidy rule by the introduction of Skolem functions $\theta(\mathbf{x}) \leftarrow \phi(\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_k(\mathbf{x}))$.

(3). If T does not satisfy the conditions presented in Section 4.1, then how can we generate “maximal” deduction trees?

(4). Can the use of eq-rules and multiple applications of intensional rules be formalised into a method for eliminating redundancy from the updated extension?

(5). At the propositional level, a maximal tree in $\text{INT}(T)$ can easily be used to check whether $T \models \bigvee \mathcal{P}$. Can the same be said of the maximal tree introduced in Section 4.2.5?

§6 APPENDIX : PROOFS.

For the remainder of this section, \mathcal{T} will denote the maximal tree in $\text{INT}(T)$ introduced in Section 4.2. Recall that $(B_i \mid 1 \leq i \leq j)$ enumerates the unfactored branches through \mathcal{T} , and $(\psi_k \mid 1 \leq k \leq k')$ lists those mappings of nulls (that appear in \mathcal{T}) to constants.

Lemma. If $P(\mathbf{t})$ labels some predicate node in \mathcal{T} , and P is intensional but not semi-definite, then \mathbf{t} is ground.

The proof is trivial from the conditions of Section 4.1, since the root node \mathcal{P} is ground.

Theorem. \mathcal{T} is finite.

Proof. Suppose not, then there must be some infinite branch B . For each predicate $P \in \mathcal{L}$, let $Occ(P) = \{P(\mathbf{t}) : P(\mathbf{t}) \text{ labels some predicate node along } B\}$. By condition (4) of Definition 2.2.4, each atom $P(\mathbf{t}) \in Occ(P)$ labels exactly one predicate node along B , thus it suffices to show that each such $Occ(P)$ is finite. We show that for each $m \leq n + 1$:

$$\bigcup \{Occ(R) \mid \ell(R) = m\} \text{ is finite} \quad \dots\dots(*)$$

Suppose that $\ell(P) = n + 1$ (i.e. P is intensional but not semi-definite), then each instance of $P(\mathbf{t})$ on $\text{ACT}(B)$ is ground, and hence $Occ(P)$ is finite. This proves that $(*)$ holds when $m = n + 1$.

Suppose that $m < n + 1$, and $Occ(R)$ is finite whenever $\ell(R) > m$. $\dots\dots(\dagger)$.

Let ω be a null appearing in $\bigcup \{Occ(P) \mid \ell(P) = m\}$ but not in $\bigcup \{Occ(R) \mid \ell(R) > m\}$. Let $N_{P(\mathbf{t})}$ be the predicate node on B such that (i) $\omega \in \mathbf{t}$, (ii) $\ell(P) = m$ and (iii) no predicate node $M > N_{P(\mathbf{t})}$ satisfies conditions (i) and (ii).

Let $RN_{(C,\theta)}$ be the parent node of $N_{P(\mathbf{t})}$, then $\ell(C) \geq m$ by Theorem 4.1.1. If $\omega \notin \mathcal{U}(C)\theta$, then ω would appear in $\text{conseq}(C\theta)$, and thus by condition (b)(v) of Definition 4.2.2, ω appears in the label of some predicate node $N'_{Q(\mathbf{u})}$ with $\ell(Q) \geq m$ and $N'_{Q(\mathbf{u})} > N_{P(\mathbf{t})}$. However, this then contradicts condition (iii) above and that fact that ω does not appear in $\bigcup \{Occ(R) \mid \ell(R) > m\}$.

Thus $\omega \in \mathcal{U}(C)\theta$, and hence by Theorem 4.1.1(b) we must have that $\ell(C) > m$. However, by (\dagger) above and condition (a) of Section 4.2.5, $\bigcup \{\mathcal{U}(D)\psi : \ell(D) > m, (D, \psi) \text{ labels some rule node along } B\}$ is finite.

This proves that $(*)$ holds for m , and hence proves the theorem. ■

Definition. Suppose that $f : \{1, 2, \dots, k'\} \rightarrow \{1, 2, \dots, j\}$, $C \in \text{INT}(T)$ and (C, χ) labels some rule node on some branch B in \mathcal{T} . Suppose that N is the child node of $RN_{(C,\chi)}$ such that N lies on B , and let $ch(C, \chi, B) = \text{lab}(N)$. Define

$$\Gamma_f(C, \chi) = \{(\text{conseq}(C\chi\psi_k), \text{ch}(C, \chi, B_{f(k)})) : 1 \leq k \leq k' \text{ and } (C, \chi) \text{ labels some rule node on } B_{f(k)}\}$$

and

$$\Omega_f = \sum \{|\Gamma_f(C, \chi)| : C \in \text{INT}(T), (C, \chi) \text{ labels some rule node in } \mathcal{T}\}$$

4.2.8 Theorem. For each function $f : \{1, 2, \dots, k'\} \rightarrow \{1, 2, \dots, j\}$, either

- (a) $\bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k)$ contains an atom of the form $c_i = c_i$, or
- (b) $\bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k)$ contains a cover of \mathcal{P} in $gr(\text{INT}(T))$.

Proof. We proceed by induction on Ω_f . Suppose that conditions (a) or (b) are satisfied by every function h such that $\Omega_h < \Omega_f$, whilst neither (a) nor (b) is satisfied by f .

Claim 1: Suppose that $C \in \text{INT}(T)$, $C\theta$ is a ground instance of C , $k_0 \leq k'$, $\text{conseq}(C\theta) \subseteq \text{ACT}(B_{f(k_0)}\psi_{k_0})$ and $\text{antec}(C\theta) \cap \text{ACT}(B_{f(k_0)}\psi_{k_0}) = \emptyset$. Then there is a pair (C, χ) labelling some rule node along $B_{f(k_0)}$ such that $\text{conseq}(C\chi\psi_{k_0}) = \text{conseq}(C\theta)$.

Proof of Claim 1. Suppose that C is definite with $\text{conseq}(C) = \{K(\mathbf{s})\}$. Pick $K(\mathbf{t})$ on $\text{ACT}(B_{f(k_0)})$ such that:

- (i) $K(\mathbf{s})\theta = K(\mathbf{t})\psi_{k_0}$, and
- (ii) There is no atom $K(\mathbf{u})$ on $\text{ACT}(B_{f(k_0)})$ such that $K(\mathbf{s})\theta = K(\mathbf{u})\psi_{k_0}$, where \mathbf{u} contains (strictly) fewer nulls than \mathbf{t} .

Let $\mu_{\mathbf{s}, \mathbf{t}}$ be as defined in Section 4.2.1. Note that if $(\omega, c) \in \mu_{\mathbf{s}, \mathbf{t}}$, then $\omega\psi_{k_0} = c$, and if $(\omega_1, \omega_2) \in \mu_{\mathbf{s}, \mathbf{t}}$, then $\omega_1\psi_{k_0} = \omega_2\psi_{k_0}$.

Suppose that $\mu_{\mathbf{s}, \mathbf{t}}$ is non-empty, say containing (ω, c) . But then the eq-rule $K(\mathbf{t}) \leftarrow K(\mathbf{t}(\omega/c)) \wedge \omega = c$ must have been applied along $B_{f(k_0)}$ (unless $\text{ACT}(B_{f(k_0)}) \cap \{K(\mathbf{t}(\omega/c)), \omega = c\} \neq \emptyset$). In either case we have that $\text{ACT}(B_{f(k_0)}) \cap \{K(\mathbf{t}(\omega/c)), \omega = c\} \neq \emptyset$. If $\text{ACT}(B_{f(k_0)})$ contains the fact $\omega = c$, then $\text{ACT}(B_{f(k_0)}\psi_{k_0})$ contains $\omega\psi_{k_0} = c$, i.e., $c = c$, a contradiction. But then $K(\mathbf{t}(\omega/c))$ occurs on $\text{ACT}(B_{f(k_0)})$, contradicting (ii) above. Similarly, $\mu_{\mathbf{s}, \mathbf{t}}$ can contain no atom of the form (ω_1, ω_2) .

Thus $\mu_{\mathbf{s}, \mathbf{t}} = \emptyset$ and hence $K(\mathbf{t})$ is a null instance of $K(\mathbf{s})$. Let χ be a substitution defined on the variables in C such that $\mathbf{s}\chi = \mathbf{t}$ and if $x \in \mathcal{U}(C)$, then $x\chi$ is a null not appearing in T or \mathcal{T} .

Notice that $\mathbf{s}\chi\psi_{k_0} = \mathbf{t}\psi_{k_0} = \mathbf{s}\theta$. If $B_{f(k_0)}$ contains a rule node of the form $RN_{(C, \phi)}$ where $\text{conseq}(C\phi) = \text{conseq}(C\chi)$, then $\text{conseq}(C\phi\psi_{k_0}) = \text{conseq}(C\chi\psi_{k_0}) = \text{conseq}(C\theta)$, and we are finished. Suppose that no such rule node exists, then the reason for the inapplicability of the pair (C, χ) can only be because $\text{antec}(C\chi) \cap \text{ACT}(B_{f(k_0)}) \neq \emptyset$. Pick $K \in \text{antec}(C)$ such that $K\chi \in \text{ACT}(B_{f(k_0)})$. Since the nulls in $\mathcal{U}(C)\chi$ do not appear in \mathcal{T} ,

K cannot contain any untidy variables, and hence $K\chi\psi_{k_0} = K\theta$, thus contradicting the fact that $\text{antec}(C\theta) \cap \text{ACT}(B_{f(k_0)}\psi_{k_0}) = \emptyset$.

If C is indefinite, then by the above lemma, $\text{conseq}(C\theta) \subseteq \text{ACT}(B_{f(k_0)})$. Again define χ such that if x appears in $\text{conseq}(C)$, then $x\chi = x\theta$, and if $x \in \mathcal{U}(C)$, then $x\chi$ is a null not appearing in T or \mathcal{T} . The proof then proceeds as above.

Claim 2: Suppose that $C \in \text{INT}(T)$, $C\theta$ is a ground instance of C , $k_0 \leq k'$, $\text{conseq}(C\theta) \subseteq \text{ACT}(B_{f(k_0)}\psi_{k_0})$ and $\text{antec}(C\theta) \cap \text{ACT}(B_{f(k_0)}\psi_{k_0}) = \emptyset$.

Then $\text{antec}(C\theta) \cap \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k)$ contains an atom which is semi-definite or extensional.

Proof of Claim 2. By Claim 1, we may find an instance of C , $C\chi$ such that the rule node $RN_{(C,\chi)}$ appears on $B_{f(k_0)}$ with $\text{conseq}(C\chi\psi_{k_0}) = \text{conseq}(C\theta)$.

Clearly $\text{antec}(C\chi) \cap \text{ACT}(B_{f(k_0)}) \neq \emptyset$. Thus if $K \in \text{antec}(C)$, $K\chi = ch(C, \chi, B_{f(k_0)})$ and $K\chi$ contains no null from $\mathcal{U}(C)\chi$, then $K\chi\psi_{k_0} = K\theta$, contradicting the fact that $\text{antec}(C\theta) \cap \text{ACT}(B_{f(k_0)}\psi_{k_0}) = \emptyset$. Thus in particular, by the conditions of Section 4.1, K is semi-definite or extensional.

Suppose that $\text{antec}(C\theta) \cap \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k)$ contains no atom which is semi-definite or extensional. By conditions (b)(iii) and (b)(iv) of Definition 4.2.2 and the conditions of Section 4.1, if B is a branch through \mathcal{T} such that some null from $\mathcal{U}(C)\chi$ appears on $\text{ACT}(B)$, then B passes through a rule node labelled with (C, χ) where $ch(C, \chi, B)$ is semi-definite or extensional.

Let $l \leq k'$ be such that:

- (i) $B_{f(l)}$ contains a rule node labelled (C, χ) such that $\text{conseq}(C\chi\psi_l) = \text{conseq}(C\theta)$, and
- (ii) $ch(C, \chi, B_{f(l)})$ is semi-definite or extensional.

Let $\eta : \mathcal{U}(C)\chi \rightarrow \{c_1, c_2, \dots, c_m\}$ be such that for each $x \in \mathcal{U}(C)$, $x\chi\eta = x\theta$. Pick $g(l)$ such that for each $\omega \in \mathcal{U}(C)\chi$, $\psi_{g(l)}(\omega) = \eta(\omega)$, and for each null $\omega \notin \mathcal{U}(C)\chi$, $\psi_{g(l)}(\omega) = \psi_l(\omega)$. Notice in particular that $C\chi\psi_{g(l)} = C\theta$.

But then if $B_{f(g(l))}$ passes through a rule node labelled (C, χ) , and $ch(C, \chi, B_{f(g(l))})$ is semi-definite or extensional, then $\text{antec}(C\theta) \cap \text{ACT}(B_{f(g(l))}\psi_{g(l)})$ contains an atom which is semi-definite or extensional, contradicting our assumption.

In particular, no null in $\mathcal{U}(C)\chi$ appears on $\text{ACT}(B_{f(g(l))})$, thus by the definition of $g(l)$ we have that $\text{ACT}(B_{f(g(l))}\psi_{g(l)}) = \text{ACT}(B_{f(g(l))}\psi_l) \dots\dots\dots(\dagger)$.

Define $f^* : \{1, 2, \dots, k'\} \rightarrow \{1, 2, \dots, j\}$ via

$$f^*(l) = \begin{cases} f(g(l)), & \text{if } l \text{ satisfies (i) and (ii) above;} \\ f(l), & \text{otherwise.} \end{cases}$$

Suppose that (D, μ) labels some rule node on $B_{f^*(l)}$, whence $(\text{conseq}(D\mu\psi_l), ch(D, \mu, B_{f^*(l)})) \in \Gamma_{f^*}(D, \mu)$. By condition (b)(v) of Definition 4.2.2, $\text{conseq}(D\mu) \subseteq \text{ACT}(B_{f^*(l)})$.

If $f^*(l) = f(g(l))$, then by (†) above, $(\text{conseq}(D\mu\psi_l), \text{ch}(D, \mu, B_{f^*(l)})) = (\text{conseq}(D\mu\psi_{g(l)}), \text{ch}(D, \mu, B_{f(g(l))})) \in \Gamma_f(D, \mu)$. If $f^*(l) = f(l)$, then $(\text{conseq}(D\mu\psi_l), \text{ch}(D, \mu, B_{f^*(l)})) = (\text{conseq}(D\mu\psi_l), \text{ch}(D, \mu, B_{f(l)})) \in \Gamma_f(D, \mu)$. Thus $\Gamma_{f^*}(D, \mu) \subseteq \Gamma_f(D, \mu)$.

Moreover $\Gamma_{f^*}(C, \chi) \subset \Gamma_f(C, \chi)$ (since $(\text{conseq}(C\chi\psi_{k_0}), \text{ch}(C, \chi, B_{f(k_0)}))$ is in the latter but not the former). Thus $\Omega_{f^*} < \Omega_f$, and by inductive hypothesis, f^* must satisfy either condition (a) or (b). Clearly $\bigcup_{k=1}^{k'} \text{ACT}(B_{f^*(k)}\psi_k) \subseteq \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k)$, thus contradicting the fact that f satisfies neither (a) nor (b). This then proves Claim 2.

Claim 3: For each $l \leq k'$, $\mathcal{C} = \text{ACT}(B_{f(l)}\psi_l) \cup \{P(\mathbf{t}) \in \bigcup_{k=1}^{k'} \text{ACT}(B_{f(k)}\psi_k \mid \ell(P) \leq n\}$ (cf. Section 4.1) is a cover of \mathcal{P} in $\text{gr}(\text{INT}(T))$.

Proof of Claim 3. Clearly $\mathcal{P} \subseteq \text{ACT}(B_{f(l)}\psi_l) \subseteq \mathcal{C}$. Suppose that $C\theta$ is a ground instance of a rule in $C \in \text{INT}(T)$ with $\text{conseq}(C\theta) \subseteq \mathcal{C}$.

If $\ell(C) = n + 1$, then $\text{conseq}(C) \subseteq \text{ACT}(B_{f(l)}\psi_l$, and the result follows from Claim 2.

If $\ell(C) \leq n$, then C is definite, and thus there is an $l_0 \leq k'$ such that $\text{conseq}(C) \subseteq \text{ACT}(B_{f(l_0)}\psi_{l_0}$. The result again follows from Claim 2.

This completes the proof of Claim 3 and hence of the theorem. ■

References

- [Ab85] S. Abiteboul and G. Grahne, Update semantics for incomplete databases, in : Proc 11th VLDB, (1985), 1-12.
- [Ab88] S. Abiteboul, Updates, a new frontier, Proc. ICDT '88, Bruges, Springer LNCS vol. 326, (Springer, Berlin, 1988), 1-18.
- [Ab93] S. Abiteboul, S. Cluet and T. Milo, Querying and updating the file, in : Proc. 19th VLDB, (1993), 73-84.
- [At91] P. Atzeni and R. Torlone, Solving ambiguities in updating deductive databases, in : Mathematical Foundations of Database Systems, Lecture Notes in Computer Science, vol. 495 (Springer, Berlin, 1991), 104-118.
- [At92] P. Atzeni and R. Torlone, Updating intensional predicates in datalog, Data and Knowledge Engineering, vol. 8 (1992), 1-17.
- [Ba81] F. Bancilhon and N. Spyrtatos, Update semantics of relational views, ACM Transactions on Database Systems, vol. 6 (1981), 557 - 575.
- [Br90] F. Bry, Intensional updates : abduction via deduction, in : Proc. 7th Int'l Conference on Logic Programming (1990), 561-578
- [Da82] U. Dayal and P. Bernstein, On the correct translation of update operations on relational view, ACM Transactions on Database Systems, vol. 8 (1982), 382 - 416.
- [De90] H. Decker, Drawing updates from derivations, in : S. Abiteboul and P.C. Kanellakis (Eds.), Proceedings of the 3rd Int'l Conference on Database Theory, Paris, Lecture Notes in Computer Science, vol. 470 (Springer, Berlin, 1990), 437-451.
- [Fa83] R. Fagin, J. Ullman and M. Vardi, On the semantics of updates in databases, in Proceedings of the 2nd ACM Symposium on the Principles of Database Systems, (1983), 352-365.
- [Ga84] H. Gallaire, J. Minker and J.M. Nicolas, Logic and databases: A deductive approach, ACM Computing Surveys, vol. 16 (1984), 153-185.
- [Go88] G. Gottlob, P. Paolini and R. Zicari, Properties and update semantics of consistent views, ACM Transactions of Database Systems, vol. 13 (1988), 486 - 524.
- [Gr86] J. Grant and J. Minker, Answering queries in indefinite databases and the null value problem, Advances in Computing Research, vol. 3 (1986), 247-267.
- [Gr93] J. Grant, J. Horty, J. Lobo and J. Minker, View updates in stratified disjunctive databases, J. Automated Reasoning, vol. 11 (1993), 249-267.
- [He88] L. J. Henschen and H. Park, Compiling the GCWA in indefinite databases, in : J. Minker, ed., Foundations of Deductive Databases, pp 395-438, (Morgan Kaufmann, Washington, 1988)
- [Jo93] C. A. Johnson, Top down deduction in indefinite deductive databases, in : F. Bry (Ed.) Proceedings of the 1993 Journées Bases de Données Avancées, Toulouse, (INRIA,

- France), 119-138.
- [Jo96] C. A. Johnson, Conjunctive answers, view updates and perfect models, in preparation.
- [Ka90] A. Kakas and P. Mancarella, Database updates through abduction, in : Proc. 16th VLDB, (1990), 650 - 661.
- [Ke86] A. Keller, Choosing a view update translator by dialog at view definition time, in : Proc. 12th VLDB, (1986), 467 - 474.
- [Ke91] A. Keller, T. Barsalou, N. Siambela and G. Wiederhold, Updating relational databases through object-based views, in : Proc. ACM SIGMOD '91 (1991), 248 - 257.
- [Kr92] M. Kramer, G. Lausen and G. Saake, Updates in a rule-based language for objects, in : Proc. 18th VLDB, Vancouver (1992), 251-262.
- [Lo92] J. Lobo, J. Minker, and A. Rajasekar, Foundations of Disjunctive Logic Programming, (MIT Press, Cambridge, Massachusetts, 1992).
- [Ma88] S. Manchanda and D. Warren, A logic based language for database updates, in : J. Minker, ed., Foundations of Deductive Databases, 363-394, (Morgan Kaufmann, Washington, 1988).
- [Ra89] A. Rajasekar, Semantics for Disjunctive Logic Programs, PhD thesis, University of Maryland (1989).
- [Ro89] F. Rossi and S. Naqvi, Contributions to the view update problem, in : Proceedings of the 6th Int'l Conference on Logic Programming, (1989), 398-415.
- [Sc91] M. Scholl, C. Laasch and M. Tresch, updatable views in object oriented database systems, in Proc. 2nd Int'l Conference on Deductive and Object Oriented Databases, Munich, Lecture Notes in Computer Science, vol. 566 (Springer, Berlin, 1991), 189-207.
- [To88] A. Tomasic, View update annotation in definite deductive databases, Proc ICDT '88 (1988), 338 - 352.
- [Wi90] M. Winslett, Updating Logical Databases, Cambridge Tracts in Theoretical Computer Science, vol. 9 (Cambridge University Press, 1990).
- [Ya85] A. Yahya and L.J. Henschen, Deduction in non-Horn databases, J. Automated Reasoning, vol. 1 (1985), 141-160.