

GENERATING ALMOST MINIMAL ANSWERS IN INDEFINITE STRATIFIED DEDUCTIVE DATABASES

C A Johnson

Computer Science Department
University of Keele
Staffs, ST5 5BG
England
email: chrisj@cs.keele.ac.uk

Abstract. A method is presented for processing queries in first order (function free) indefinite deductive databases. Database rules may contain negative atoms within their bodies, and databases are assumed to be stratified and interpreted using the semantics of perfect models. Our method constructs query answers by repeatedly extending partial answers with new atoms. Careful validation of these new atoms helps to eliminate the generation of a large number of redundant (ie., non-minimal) answers. The method is sound and complete. It also admits powerful search space pruning methods, which, together with mild constraints on the database are sufficient to guarantee termination, and also ensure that some of the more complex aspects of the query answering process are grounded.

Keywords: Deductive databases, indefinite stratified databases, query processing, minimal answers, perfect models, cyclic trees, deduction trees.

INTRODUCTION.

Indefinite (or disjunctive) deductive databases [Gr86, He88, Jo98a, Lb92] allow the representation of data and data relationships using logical formulae of the form

$$A_1 \wedge A_2 \wedge \dots \wedge A_h \wedge \neg A_{h+1} \wedge \neg A_{h+2} \wedge \dots \wedge \neg A_{h+r} \rightarrow B_1 \vee B_2 \vee \dots \vee B_k$$

where each A_i, B_j is a first order (positive) atom. Given such a representation, we need to choose declarative semantics (which dictate what is “true” in the database), and develop procedures (procedural semantics) to enable us to decide if a given statement is true in the database, and in particular to answer queries.

In this paper, we will work throughout with first order stratified databases: rules may contain negative atoms in their body (as in the rule above), but such negative atoms are taken from a lower stratum than the atoms in the head of the rule. We also assume the commonly accepted natural semantics for such databases defined by perfect models [Pr88, Pr89]. Thus a statement is true in a database T iff it is true in every perfect model of T .

We present a query processing method for such databases. Our method constructs answers by repeatedly extending partial answers with new atoms. Potential new atoms are generated using cyclic trees, and their suitability is validated using extended deduction trees. The generation of answers can thus be seen as a bottom-up process, whilst the

construction of cyclic and extended deduction trees employs the database rules in a top-down fashion. The features of our method are as follows.

1. Answers to a query in an indefinite database may themselves be indefinite. However non-minimal (ie., subsumed) answers are completely redundant in giving new information about the query, hence an important consideration is the removal of such answers. A simple strategy is to generate all answers and then to eliminate non-minimal answers by subsumption. Unfortunately this method is highly inefficient as the number of non-minimal answers is likely to be extremely large. Our method therefore validates the suitability of new atoms (within a partial answer), and this in turn limits the number of non-minimal answers that are generated.
2. Our method admits powerful search space pruning methods which have the effect of reducing redundant computation, guaranteeing termination, and reducing much of the more complex processing to the ground level.
3. In a database we typically expect that the amount of indefinite information will be small in comparison to that that is definite. We thus make heavy use of definite and semi-definite predicates [Jo96, Jo97]. Both of our tree constructions employ a linear strategy with respect to definite and semi-definite subgoals, in the sense that such subgoals are always used immediately to generate further nodes within the tree. This linearity contributes significantly towards termination, and in addition simplifies the search space. It also allows us to take advantage of any natural partitioning of the database into definite and indefinite components, by employing a strategy akin to SLD - resolution [Lb92], if query processing passes into the definite part.
4. There is a trade-off between the constraints that we impose on our database, and the efficiency of the query processing methods that the database will admit. We thus adopt database constraints which represent what seems to be a reasonable compromise in relation to this trade-off. (In this respect we can view the assumption of stratification itself as such a compromise.) In particular, given our assumption in (3) above, we aim to impose weaker constraints on the definite part of the database.

In Section 2 we present an overview of our query processing method, and also contrast it with the purely top-down method presented in [Jo98a]. The method constructs answers by repeatedly extending partial answers with new atoms. Potential new atoms are generated by the construction of cyclic trees. Such trees were introduced in [Jo96] as a means of representing the complex relationships between predicates lying inside and outside of a perfect model. In Section 3 we first review the motivation and basic properties of such trees, and then detail their usage in our query answering procedure. Cyclic trees possess a certain amount of linearity within their definition. Two important consequences of this are that cyclic trees are always naturally ground, and moreover that a terminating top-down construction of such trees (Section 3.3) is feasible, again, as a result of the fact that large parts of the tree become ground during the construction.

The validation of a potential new atom requires us to show that a particular set of ground literals is not true in the database, and for this we will employ a limited form of the query processing method presented in [Jo98a] which is based upon the top-down construction of extended deduction trees. The motivation and features of such trees,

as they relate to the validation procedure, are reviewed in Section 4. In Section 5 we address the efficiency of the validation procedure: specifically we show that the procedure is ground unless the processing enters the definite/semi-definite part of the database, wherein the validation procedure becomes linear. It also follows from results of [Jo98a] that the validation procedure (and hence query processing as a whole) is terminating.

In Section 6 we consider some possible extensions to our method, and in Section 7 we discuss related research. Finally Section 8 presents our conclusions and suggestions for future research. The material of the current paper is adapted from the extended technical report [Jo98].

§1. TERMINOLOGY

1.1 Language and rules.

Throughout we assume that \mathcal{L} is a finite (function free) first order language

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

where $\{P_1, P_2, \dots, P_n\}$ is the set of predicate symbols and $\{c_1, c_2, \dots, c_m\}$ is the set of constant symbols, $n, m > 0$. We (implicitly) assume the existence of countably many variables x_i for the construction of formulae in \mathcal{L} . A *term* in \mathcal{L} is either a constant symbol or a variable. Sequences of terms will be denoted by $\mathbf{t}, \mathbf{x}, \mathbf{y}, \mathbf{z}$, etc, and sequences of constants by $\mathbf{a}, \mathbf{b}, \mathbf{c}$, etc. A *positive atom* is an expression of the form $P(\mathbf{t})$, where P is a predicate symbol, and \mathbf{t} has length $\text{arity}(P)$. A *negative atom* is the negation of a positive atom, and is denoted by $\neg P(\mathbf{t})$. An *atom* is either a positive or a negative atom.

An expression is *ground* if it contains no variables, and \mathcal{H} (the Herbrand base) denotes the set of ground positive atoms in \mathcal{L} . Given a first order expression $expr$, $gr(expr)$ denotes the set of ground instances of $expr$. $VAR(expr)$ denotes the set of variables that appear in $expr$.

1.1.1 Stratification.

We assume throughout the existence of a *level function*

$$\ell : \{P_1, P_2, \dots, P_n\} \rightarrow \{0, 1, 2, \dots, n\}.$$

A predicate P is said to be *extensional* iff $\ell(P) = 0$. If P is not extensional, it is *intensional*. $\text{EXT}(\mathcal{L})$ denotes the set of extensional predicates, and $\text{INT}(\mathcal{L})$, the set of intensional predicates. If $P(\mathbf{t})$ is an atom, then $\ell(P(\mathbf{t})) = \ell(\neg P(\mathbf{t})) = \ell(P)$. \mathcal{P} is a set of atoms/predicates, then define $\mathcal{P}_\alpha = \{K \in \mathcal{P} \mid \ell(K) = \alpha\}$, $\mathcal{P}_{<\alpha} = \{K \in \mathcal{P} \mid \ell(K) < \alpha\}$, etc.

1.1.2 Definition. A *rule* C is a formula of the form

$$A_1 \wedge A_2 \wedge \dots \wedge A_h \wedge \neg A_{h+1} \wedge \neg A_{h+2} \wedge \dots \wedge \neg A_{h+r} \rightarrow B_1 \vee B_2 \vee \dots \vee B_k$$

such that:

- (i) each A_i and each B_j is a positive atom in \mathcal{L} and $k > 0$,
- (ii) for each $j \leq k$, $\ell(B_j) = \ell(B_1)$,
- (iii) for each $i \leq h$, $\ell(A_i) \leq \ell(B_1)$,
- (iv) for $1 \leq i \leq r$, $\ell(A_{h+i}) < \ell(B_1)$,
- (v) if $h + r > 0$, then each B_j is intensional, and
- (vi) if $h = r = 0$, then each B_j is extensional.

For each such rule, $\text{antec}(C) = \{A_1, A_2, \dots, A_h\}$, $\mathcal{N}(C) = \{A_{h+1}, A_{h+2}, \dots, A_{h+r}\}$, $\overline{\mathcal{N}}(C) = \{\neg A_{h+1}, \neg A_{h+2}, \dots, \neg A_{h+r}\}$, and $\text{conseq}(C) = \{B_1, B_2, \dots, B_k\}$. Define $\ell(C) = \ell(B_1)$. We assume that rules are *range restricted* in the following sense.

- (vii) $\text{VAR}(\mathcal{N}(C)) \cup \text{VAR}(\text{conseq}(C)) \subseteq \text{VAR}(\text{antec}(C))$.

1.1.3 Definition. A *deductive database* is a set of rules, which throughout will be denoted by T . We also define $T_\alpha = \{C \in T \mid \ell(C) = \alpha\}$, etc. $\text{EXT}(T) = T_0$ is the *extension* of T , and $\text{INT}(T) = T_{>0}$ its *intension*. Note that the extension is ground by property (vii) of Definition 1.1.2.

We implicitly assume that T contains the tautology $K \vee \neg K$ for each $K \in \mathcal{H}$. Note that such tautologies are not of the form given in Definition 1.1.2.

1.1.4 Definition [Pr88]. Let C be a rule and $M \subseteq \mathcal{H}$, then M *models* C (written $M \models C$) iff M satisfies every ground instance of C . M *models* T (written $M \models T$) iff $M \models C$ for each $C \in T$.

A model M of T is *perfect* iff whenever $\alpha \leq n$ and $M' \subset M_\alpha$ (cf. Definition 1.1.1), then $M' \cup M_{<\alpha} \not\models T_\alpha$.

1.1.5 Definition (a) If Φ is a variable free formula in \mathcal{L} , then $T \models \Phi$ iff Φ is true in every perfect model of T .

(b) Given a query $?Q(\mathbf{x})$, an *answer* is a set $\mathcal{A} \subseteq \text{gr}(Q(\mathbf{x}))$ such that $T \models \bigvee \mathcal{A}$. The answer \mathcal{A} is *minimal* iff it is not properly subsumed by any other answer to $?Q(\mathbf{x})$.

1.2 Semi-definite predicates.

Recall that a rule C is definite iff $|\text{conseq}(C)| = 1$. We may thus (loosely) define a predicate P to be definite iff whenever $C \in T$ with $P(\mathbf{t}) \in \text{conseq}(C)$, then C is definite and each predicate appearing in $\text{antec}(C) \cup \mathcal{N}(C)$ is definite. Formally we assume the existence of a set $\text{Def}(\mathcal{L}) \subseteq \mathcal{L}$ satisfying the following condition.

1.2.1 Definiteness. If $P \in \text{Def}(\mathcal{L})$ and P appears in the consequent of some rule $C \in T$, then C is definite, and each predicate appearing in $\text{antec}(C) \cup \mathcal{N}(C)$ is in $\text{Def}(\mathcal{L})$.

A predicate appearing in $\text{Def}(\mathcal{L})$ is said to be *definite*.

When we address the issue of termination, the main problem ([Jo96, Jo97]) is intensional predicates. We thus assume the existence of sets $\text{SD}(\mathcal{L}) \subseteq \text{WSD}(\mathcal{L}) \subseteq \text{INT}(\mathcal{L})$ whose predicates satisfy the following variant of definiteness.

1.2.2 Semi-definiteness. If $P \in SD(\mathcal{L})$ and P appears in the consequent of some rule $C \in T$, then C is definite, and each intensional predicate appearing in $\text{antec}(C) \cup \mathcal{N}(C)$ is in $SD(\mathcal{L})$.

A predicate appearing in $SD(\mathcal{L})$ is said to be *semi-definite*. Clearly we may assume that $Def(\mathcal{L}) \cap \text{INT}(\mathcal{L}) \subseteq SD(\mathcal{L})$.

1.2.3 Weak semi-definiteness. If $P \in WSD(\mathcal{L})$ and P appears in the consequent of some rule $C \in T$, then C is definite, and if R appears in $\text{antec}(C)$ with $\ell(R) = \ell(C)$, then $R \in WSD(\mathcal{L})$.

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

By an abuse of the above notation, if $P(\mathbf{t})$ is a positive atom with $P \in Def(\mathcal{L})$ ($SD(\mathcal{L}), WSD(\mathcal{L})$), we may also write $P(\mathbf{t}) \in Def(\mathcal{L})$ ($SD(\mathcal{L}), WSD(\mathcal{L})$).

1.2.4 Definition. Let $Def(T) = \{C \in T \mid \text{conseq}(C) = \{P(\mathbf{t})\}, \text{ where } P \in Def(\mathcal{L})\}$.

Similar definitions may be made for $SD(T)$ and $WSD(T)$. It is clear that a definite atom in \mathcal{H} is either in all perfect models of T or none, thus we make the following definition.

1.2.5 Definition. Let $\mathcal{M}_{Def(T)} = \{P(\mathbf{a}) \in \mathcal{H} \cap Def(\mathcal{L}) \mid T \models P(\mathbf{a})\}$.

Clearly if $P(\mathbf{a}) \in \mathcal{H} \cap Def(\mathcal{L})$, then $T \models P(\mathbf{a})$ iff $Def(T) \models P(\mathbf{a})$. Also it is intuitively clear that if M is a perfect model of T , then $M_0 \cup (M \cap SD(\mathcal{L}))$ is formed by “closing” M_0 under $SD(T)$, and for each $0 < \alpha \leq n$, $M_{<\alpha} \cup (M_\alpha \cap WSD(\mathcal{L}))$ is formed by closing $M_{<\alpha}$ under $WSD(T)_\alpha$ (cf. Definitions 1.1.3/4).

1.3 Predicate-rule trees.

Both deduction and cyclic trees have a common structure, and it will thus simplify our later definitions if we present that common structure once. Motivation for the following definition appears in [Jo96, Jo97, Jo98, Jo98a], and will also appear in later sections.

1.3.1 Definition. A *predicate-rule tree* is a finite tree \mathcal{T} consisting of predicate nodes, set nodes and rule nodes satisfying the following conditions.

- (i) The root node $root(\mathcal{T})$ (at the top of \mathcal{T}) is either a predicate or a set node.
- (ii) Each predicate node is labelled with an atom, and each set node is labelled with a set of atoms. If N is either a predicate or a set node, then the label of N is denoted by $lab(N)$.
- (iii) If N is a predicate node and $lab(N)$ is a positive atom, then we refer to N as *positive*, else it is *negative*.
- (iv) Let N be either a predicate node or a set node. N has at most one child node, which is a rule node if it exists. If N is not the root node, then the parent of N is a rule

node.

Predicate and set nodes will be denoted via N and M (possibly with sub and superscripts), or in diagrams possibly via their label alone.

- (v) If RN is a rule node, then the parent of RN is a predicate or set node. Each child node of RN (if they exist) is either a predicate or a set node. RN is again labelled, and we may write RN_S where S is the label of RN .

1.3.2 Example.

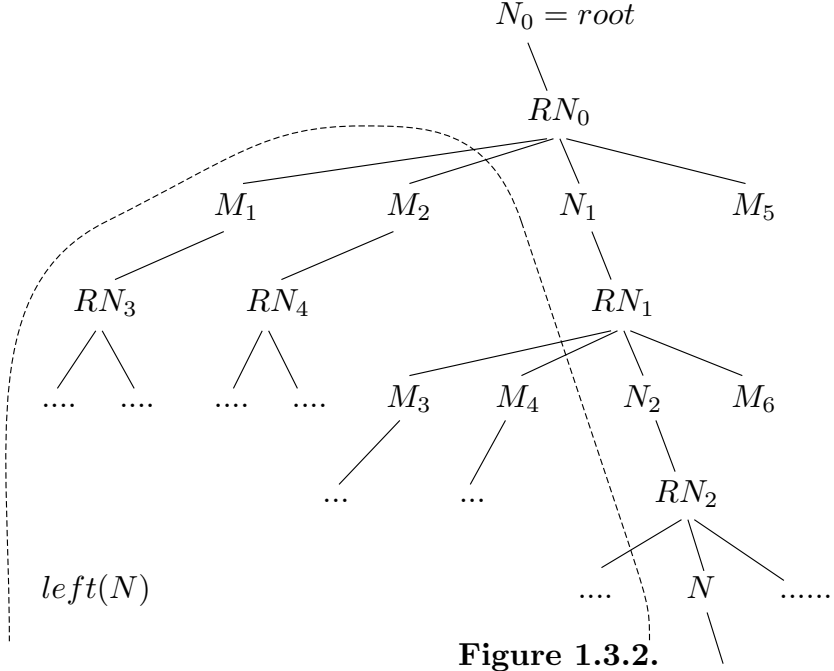


Figure 1.3.2.

Figure 1.3.2 depicts the general structure of a predicate-rule tree. N_i, M_j denote predicate or set nodes, and RN_i denote rule nodes. No labels are depicted.

1.3.3 Definition. Let \mathcal{T} be a predicate-rule tree and N a predicate or set node in \mathcal{T} .

- (a) We write $N > RN$ if RN is the child of N and $RN > M$ if M is a child of RN . The $>$ relation is assumed transitive.

- (b)

$$below(N) = \{N' \mid N' \text{ is a predicate or set node in } \mathcal{T}, N \geq N'\}, \text{ and}$$

$$left(N) = \bigcup \{below(N') \mid N' \text{ has a right sibling } N'' \geq N\}$$

(see Figure 1.3.2).

- (c) $ACT(N)$ denotes the set of atoms labelling nodes above N . ie.,

$$ACT(N) = \{lab(M) \mid M \text{ is a predicate node, } M \geq N\} \cup$$

$$\bigcup \{lab(M) \mid M \text{ is a set node, } M \geq N\}.$$

Alternatively we may think of $\text{ACT}(N)$ as the set or sequence of predicate and set nodes above N . Thus in Figure 1.3.2, $\text{ACT}(M_3) = (N_0, N_1, M_3)$.

§2. GENERATING ANSWERS: AN OVERVIEW

In [Jo98a] we presented a top-down query processing method applicable to first order stratified databases. The method had two main shortcomings. Firstly the size of the answer to be extracted had to be pre-specified, thus in order to generate all answers we have to generate answers of size 1, then answers of size 2, etc. Non-minimal answers can of course be eliminated via subsumption, although because of the potential number of non-minimal answers that might be generated, this would in general be very costly. Worse still, there is no apparent way to readily determine when all answers have been generated (and hence stop the iteration). The method for query processing to be presented in the current paper attempts to overcome these shortcomings.

The following proposition is well known.

2.1 Proposition (a) Let $P(\mathbf{a}) \in \mathcal{H}$, then $P(\mathbf{a})$ belongs to some perfect model of T iff there is a set $\mathcal{P} \subseteq \mathcal{H}$ such that $T \models \bigvee \mathcal{P}$, $P(\mathbf{a}) \in \mathcal{P}$ and there is no $\mathcal{P}' \subset \mathcal{P}$ such that $T \models \bigvee \mathcal{P}'$.

(b) If $\mathcal{A} \subseteq \text{gr}(Q(\mathbf{x}))$ is an answer to the query $?Q(\mathbf{x})$, then \mathcal{A} is minimal iff for each $Q(\mathbf{a}) \in \mathcal{A}$, we may find a perfect model M of T such that $M \cap \mathcal{A} = \{Q(\mathbf{a})\}$.

As indicated above, we will construct query answers by repeatedly extending partial answers with new atoms. In order to limit the number of non-minimal answers generated, we will ensure that each new atom $Q(\mathbf{a})$ that is used to extend a partial answer \mathcal{A} is contained in some perfect model that is disjoint from \mathcal{A} (cf. Definition 2.2 below). A by-product of this strategy is that the extension of \mathcal{A} stops as soon as \mathcal{A} itself becomes a full answer, whence guaranteeing termination.

Note of course that this procedure is not guaranteed to generate only minimal answers. (Indeed in some cases, perfect model membership is not necessarily sufficient to guarantee membership in any minimal answer of the query under consideration.) However, the above-mentioned strategy will eliminate many non-minimal answers, and as such represents a compromise between the desire to compute only minimal answers, and the desire for computational efficiency.

2.2 Definition. Let $\mathcal{A} \subseteq \text{gr}(Q(\mathbf{x}))$, then define $\text{PM}(T, Q(\mathbf{x}), \mathcal{A})$ iff there is an $\mathbf{a} \in \text{gr}(\mathbf{x})$ and a perfect model M of T such that $Q(\mathbf{a}) \in M$ and $M \cap \mathcal{A} = \emptyset$.

2.3 Proposition. Suppose that $T \models \bigvee \{Q(\mathbf{a}) \mid \mathbf{a} \in \text{gr}(\mathbf{x})\}$, ie., the query $?Q(\mathbf{x})$ has an answer. Let $\mathcal{A} \subseteq \text{gr}(Q(\mathbf{x}))$ be such that $\neg \text{PM}(T, Q(\mathbf{x}), \mathcal{A})$. Then $T \models \bigvee \mathcal{A}$.

Proof. Let M be a perfect model of T , then M contains some ground instance $Q(\mathbf{a})$ of $Q(\mathbf{x})$. If $M \cap \mathcal{A} = \emptyset$, then M witnesses that $\text{PM}(T, Q(\mathbf{x}), \mathcal{A})$. ■

2.4 Generating answers.

In Sections 3-6 we present a method which, given $Q(\mathbf{x})$ and \mathcal{A} , attempts to generate an instance $Q(\mathbf{a})$ of $Q(\mathbf{x})$ such that $Q(\mathbf{a})$ witnesses $\text{PM}(T, Q(\mathbf{x}), \mathcal{A})$.

If we refer to this procedure Proc-PM, and assume that the procedure generates “nil” if unsuccessful, then sets \mathcal{A} satisfying $\neg\text{PM}(T, Q(\mathbf{x}), \mathcal{A})$ can be computed via the following algorithm.

```

answer( $T, Q(\mathbf{x})$ )
{
   $\mathcal{A} = \emptyset$ ;
  while ( $Q(\mathbf{a}) \leftarrow \text{Proc-PM}(T, Q(\mathbf{x}), \mathcal{A}) \neq \text{nil}$ )
    { $\mathcal{A} \leftarrow \mathcal{A} \cup \{Q(\mathbf{a})\}$ };
  return( $\mathcal{A}$ );
}

```

In order to ensure that the sets so generated are indeed answers we need to be sure that $T \models \bigvee\{Q(\mathbf{a}) \mid \mathbf{a} \in gr(\mathbf{x})\}$. The following proposition shows that this may be verified by simply testing whether $T \models \bigvee \mathcal{A}$ for the first \mathcal{A} (satisfying $\neg\text{PM}(T, Q(\mathbf{x}), \mathcal{A})$) that is generated.

2.5 Proposition. Let $\mathcal{A} \subseteq gr(Q(\mathbf{x}))$ be such that $\neg\text{PM}(T, Q(\mathbf{x}), \mathcal{A})$. Then $T \models \bigvee\{Q(\mathbf{a}) \mid \mathbf{a} \in gr(\mathbf{x})\}$ iff $T \models \bigvee \mathcal{A}$.

Proof. The implication from right to left is trivial. The implication from left to right follows from Proposition 2.3. ■

2.6 Termination.

Clearly the method terminates (assuming that Proc-PM terminates) since T has only a finite number of perfect models. Indeed it is clear that as a partial answer grows, it becomes harder to extend (since it is harder to find a perfect model that is disjoint from it).

§3. CYCLIC TREES

Cyclic trees were introduced in [Jo96] as a means of testing minimal and perfect model membership. They can thus be used, together with the validation procedure described in Sections 4/5 to generate atoms satisfying the PM relationship (Definition 2.2).

Section 3.1 reviews the motivation for and definition of cyclic trees, and their relationship to PM is detailed in Section 3.2. Section 3.3 describes a method of constructing cyclic trees for first order databases.

3.1 Motivation.

We first present an example, modified from [Jo96], in order to motivate the definition of cyclic tree which follows below. Further motivation is to be found in [Jo96, Jo98, Jo98a, Jo98b].

3.1.1 Example. Suppose that T is propositional, and consists of the following rules:

- | | | |
|--|--|--|
| 1. $Q_2 \wedge Q_3 \wedge \neg R_1 \rightarrow Q_1 \vee Q_5$ | 2. $Q_1 \wedge \neg R_2 \rightarrow Q_2$ | 3. $S_2 \wedge \neg R_3 \rightarrow Q_3$ |
| 4. $S_3 \rightarrow Q_1 \vee Q_2 \vee Q_6$ | 5. $S_2 \vee R_5$ | 6. $S_1 \rightarrow Q_3 \vee Q_2$ |
| 7. $S_3 \vee R_7$ | 8. $R_1 \rightarrow Q_2$ | |

with $\ell(Q_i) = 1$, and $\ell(S_i) = \ell(R_j) = 0$. Suppose that Q_1 lies in the perfect model M , then $M_1 = \{K \in M \mid \ell(K) = 1\}$ is minimal and thus we may find a rule $C \in T_1$ such that $M_0 \cup (M_1 - \{Q_1\}) \not\models C$. There are two possibilities for C , namely rules 1 and 4. Suppose that C is rule 1, then $\{Q_1, Q_2, Q_3\} \subseteq M \subseteq \mathcal{H} - \{Q_5, R_1\}$. This “application” of rule 1 is represented in the top 3 levels of the predicate-rule tree \mathcal{T}_1 (Figure 3.1.1). (Only Q_2 and Q_3 (ie., the antecedents) are depicted as child nodes of RN_1 , since we wish to examine these atoms further.)

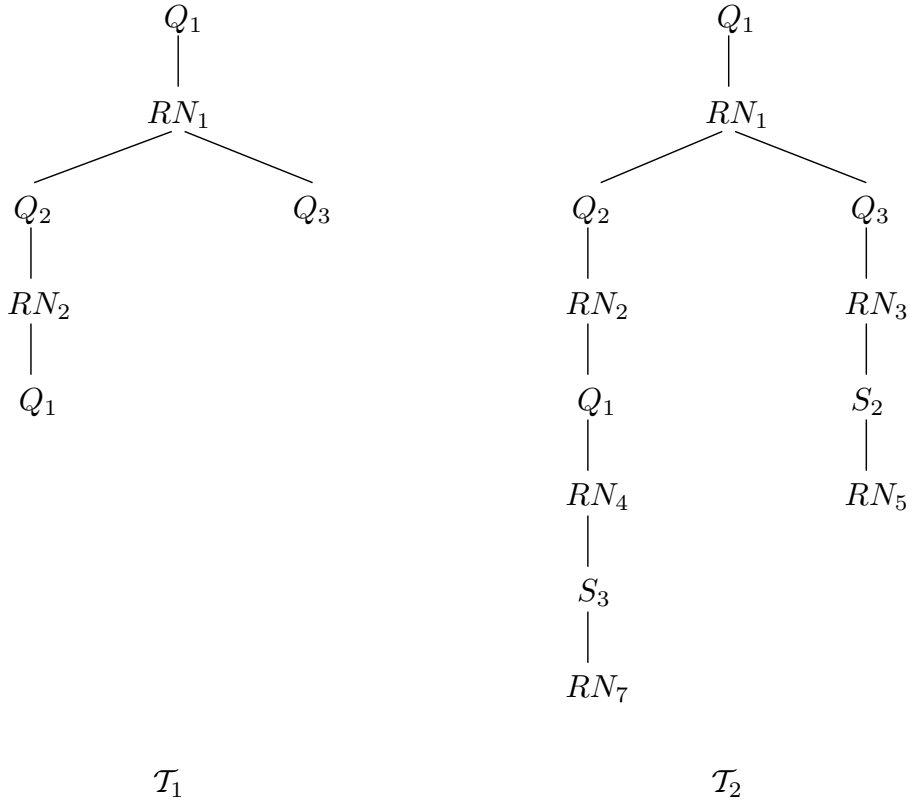


Figure 3.1.1.

Suppose now that we apply the same argument to Q_2 . If C is a rule in T_1 such that $M_0 \cup (M_1 - \{Q_2\}) \not\models C$, then (since $\{Q_1, Q_2, Q_3\} \subseteq M \subseteq \mathcal{H} - \{Q_5, R_1\}$) C must be rule 2, thus yielding the lower part of the left branch in \mathcal{T}_1 , and the new constraint $\{Q_1, Q_2, Q_3\} \subseteq M \subseteq \mathcal{H} - \{Q_5, R_1, R_2\}$.

The left-hand branch of \mathcal{T}_1 forms a “cycle”. There is no point working with Q_1 or Q_2 alone (since we have already done so), thus we look for a rule $C \in T_1$ such that $M_0 \cup (M_1 - \{Q_1, Q_2\}) \not\models C$, the only candidate being rule 4. Rule 7 then terminates the branch (since rule 7 has no antecedents), thus yielding the left branch in \mathcal{T}_2 , and the constraint $\{Q_1, Q_2, Q_3, S_3\} \subseteq M \subseteq \mathcal{H} - \{Q_5, R_1, R_2, Q_6, R_7\}$.

We thus move on to examine Q_3 : if $C \in T_1$ and $M_0 \cup (M_1 - \{Q_3\}) \not\models C$, then C must be rule 3. Rule 5 can then be used to handle S_2 , and again this terminates the branch, yielding the right-hand branch in \mathcal{T}_2 , and the constraint $\{Q_1, Q_2, Q_3, S_3, S_2\} \subseteq M \subseteq \mathcal{H} - \{Q_5, R_1, R_2, Q_6, R_7, R_3, R_5\}$.

Note also that if M is any model of T that is disjoint from $\{Q_5, R_1, R_2, Q_6, R_7, R_3, R_5\}$, then (using the rules in the tree) we may infer that $\{Q_1, Q_2, Q_3, S_3, S_2\} \subseteq M$.

During the entire process, the only point at which we had a choice of rule was during the first step, when C might have been rule 4 (instead of rule 1). If this had been the case, then we would have generated an alternative tree, and an alternative constraint on M , namely $\{Q_1, S_3\} \subseteq M \subseteq \mathcal{H} - \{Q_2, Q_6, R_7\}$. Again it is easy to show that if M is any model of T that is disjoint from $\{Q_2, Q_6, R_7\}$, then $\{Q_1, S_3\} \subseteq M$.

We are thus able to see, for example, that Q_1 lies in some perfect model of T iff $T \not\models \bigvee\{Q_5, R_1, R_2, Q_6, R_7, R_3, R_5\}$ or $T \not\models \bigvee\{Q_2, Q_6, R_7\}$.

Notice the main features of the above predicate-rule trees. First, there are no set nodes or negative predicate nodes. Secondly, it is the intention that $lab(N)$ lies in the intended perfect model M for each predicate node N . Each rule node is labelled with a rule $C \in T_\alpha$, and if RN_C has parent N , then there is a subset $\mathcal{S} \subseteq \mathcal{H}_\alpha \cap \text{ACT}(N)$ such that $M - \mathcal{S} \not\models C$. Thus, $\text{antec}(C) \subseteq M - \mathcal{S}$, $M \cap ((\text{conseq}(C) - \mathcal{S}) \cup \overline{\mathcal{N}}(C)) = \emptyset$, and since $M \models C$ we must have that $\text{conseq}(C) \cap \mathcal{S} \neq \emptyset$.

The following definitions capture these features, and also provides a precise definition of the subset $\mathcal{S} = \text{CYC}(N) \subseteq \text{ACT}(N)$ that we wish to work with.

3.1.2 Definition. Let \mathcal{T} be a predicate-rule tree satisfying the following conditions.

- (i) \mathcal{T} contains no set nodes, and each predicate node is positive.
- (ii) If RN is a rule node, then RN is labelled with an instance $C\theta$ of a rule $C \in T$ (denoting application of the rule). For each $K \in \text{antec}(C\theta)$, $RN_{C\theta}$ has a (predicate) child node labelled with K , and these are the only child nodes of $RN_{C\theta}$.

Then we make the following definitions.

- (a) Suppose that N is a predicate node in \mathcal{T} . Let $top(N)$ be the unique predicate node such that $top(N) \geq N$, $lab(top(N)) = lab(N)$ and for each predicate node $N' > top(N)$, $lab(N') \neq lab(N)$. Define

$$\text{CYC}(N) = \{lab(N') \mid N' \text{ is a predicate node, } top(N) \geq N' \geq N\}.$$

If N has child $RN_{C\theta}$, then define $\mathcal{O}(RN_{C\theta}) = \text{conseq}(C\theta) - \text{CYC}(N)$.

- (b) Let $\mathcal{O}(\mathcal{T}) = \bigcup\{\mathcal{O}(RN_{C\theta}) \mid RN_{C\theta} \text{ occurs in } \mathcal{T}\}$,
 $\mathcal{N}(\mathcal{T}) = \bigcup\{\mathcal{N}(C\theta) \mid RN_{C\theta} \text{ occurs in } \mathcal{T}\}$, and
 $Pred(\mathcal{T}) = \{lab(N) \mid N \text{ is a predicate node in } \mathcal{T}\}$.

3.1.3 Definition. Let \mathcal{T} be a predicate rule tree satisfying conditions (i) and (ii) of the above definition. Suppose that $lab(root(\mathcal{T})) = K$, then \mathcal{T} is said to be a *cyclic tree* for K in T iff

- (a) $Pred(\mathcal{T}) \cap (\mathcal{O}(\mathcal{T}) \cup \mathcal{N}(\mathcal{T})) = \emptyset$,
(b) whenever $RN_{C\theta}$ is a rule node in \mathcal{T} with parent N , then
 (i) $conseq(C\theta) \cap CYC(N) \neq \emptyset$,
 (ii) $antec(C\theta) \cap CYC(N) = \emptyset$,
 (iii) $antec(C\theta) \cap WSD(\mathcal{L}) \cap ACT(N) = \emptyset$, and
(c) \mathcal{T} has no predicate leaf node.

\mathcal{T} is said to be a *partial cyclic tree* iff it satisfies conditions (a) and (b).

The following proposition states some of the structural properties of cyclic trees.

3.1.4 Proposition [Jo98]. Suppose that \mathcal{T} is a partial cyclic tree in which $N > RN_{C\theta} > M$, where N is the parent of $RN_{C\theta}$ which in turn is the parent of M . Then

- (i) $\ell(lab(N)) = \ell(C) \geq \ell(lab(M))$,
(ii) $CYC(N) \subseteq \{lab(N') \mid N' \geq N, \ell(lab(N')) = \ell(lab(N))\}$,
(iii) if $\ell(lab(M)) < \ell(C)$, then $CYC(M) = \{lab(M)\}$,
(iv) if $lab(N) \in WSD(\mathcal{L})$, then $CYC(N) = \{lab(N)\} = conseq(C\theta)$ and $C \in WSD(T)$.

As indicated in the introduction, it is our aim to impose the weakest possible constraints on the database, and in particular on the definite part of the database. Since $CYC(N) \subseteq ACT(N)$, condition (b)(iii) in Definition 3.1.3 can be seen as a strengthening of (b)(ii) for the special case of weakly semi-definite predicates. Proposition 3.1.4(iv) follows as a consequence, and this partial linearity allows us (in Section 3.3 below) to guarantee the termination of cyclic tree constructions using significantly weaker constraints on $WSD(T)$ than are necessary for $T - WSD(T)$. (Condition (b)(iii) is not included in the definition of cyclic tree given in [Jo96].)

Also as a result of Proposition 3.1.4(i), the depth of the tree is dependent to a large extent on the relative volume of indefinite and recursive information in the database, and the number of levels in the database's stratification. In a database, we would not typically expect these parameters to be large, thus promoting the efficient construction of cyclic trees.

3.2 Perfect model membership.

In this section we present the basic relationships between cyclic trees and the property PM. Theorems 3.2.1/2 were illustrated in Example 3.1.1.

3.2.1 Theorem [Jo96]. If \mathcal{T} is a cyclic tree in $gr(T)$, and $M \models T$ with $M \cap (\mathcal{O}(T) \cup \mathcal{N}(T)) = \emptyset$, then $Pred(\mathcal{T}) \subseteq M$.

3.2.2 Theorem [Jo98]. If M is a perfect model of T , then for each $P(\mathbf{a}) \in M$ there is a cyclic tree \mathcal{T} for $P(\mathbf{a})$ in $gr(T)$ such that $Pred(\mathcal{T}) \subseteq M \subseteq \mathcal{H} - (\mathcal{O}(T) \cup \mathcal{N}(T))$.

3.2.3 Corollary [Jo98]. Let \mathcal{P} be a non-complementary set of ground atoms, and $K \in \mathcal{H}$. Then $T \not\models \neg K \vee \bigvee \mathcal{P}$ iff there is a cyclic tree \mathcal{T} for K in T such that

$$T \not\models \bigvee (\mathcal{P} \cup \{\neg L \mid L \in Pred(\mathcal{T})\} \cup \mathcal{O}(T) \cup \mathcal{N}(T)).$$

3.2.4 Corollary. $PM(T, Q(\mathbf{x}), \mathcal{A})$ iff there is a cyclic tree \mathcal{T} in $gr(T)$ for some $Q(\mathbf{a}) \in gr(Q(\mathbf{x}))$ such that $T \not\models \bigvee (\mathcal{A} \cup \{\neg L \mid L \in Pred(\mathcal{T})\} \cup \mathcal{O}(T) \cup \mathcal{N}(T))$.

Returning to our procedure Proc-PM, we see that it entails generating cyclic trees (in order to generate potential atoms $Q(\mathbf{a})$), and then validating that $T \not\models \bigvee (\mathcal{A} \cup \{\neg L \mid L \in Pred(\mathcal{T})\} \cup \mathcal{O}(T) \cup \mathcal{N}(T))$. In Section 3.3 we discuss the generation of cyclic trees, and Sections 4/5 are devoted to the validation procedure.

The validation procedure makes particular use of the fact that the set $\mathcal{A} \cup \{\neg L \mid L \in Pred(\mathcal{T})\} \cup \mathcal{O}(T) \cup \mathcal{N}(T)$ is ground, which in turn follows from the theorem below.

3.2.5 Theorem. If \mathcal{T} is a cyclic tree in T , then \mathcal{T} is ground.

Proof. Suppose not, then pick a predicate node N_0 such that $lab(N_0)$ is non-ground, and such that if M is a predicate node with $M > N_0$, then $lab(M)$ is ground. Also pick a predicate node N_1 such that $N_1 \leq N_0$, $lab(N_1)$ is non-ground, and such that if RN is the child of N_1 and M is a child of RN , then $lab(M)$ is ground. Notice that $CYC(N_1) \subseteq \{lab(M) \mid N_0 \geq M \geq N_1\}$, and that $CYC(N_1)$ consists solely of non-ground atoms.

Let $RN_{D\phi}$ be the child node of N_1 . If $R \in antec(D\phi)$, then R labels some child node of $RN_{D\phi}$, whence by assumption R is ground. But then since D is range restricted (Definition 1.1.2) $conseq(D\phi)$ is ground, and hence cannot intersect $CYC(N_1)$, thus contradicting condition (b)(i) of Definition 3.1.3. ■

3.3 Constructing cyclic trees.

In this section we present a method of generating cyclic trees for first order databases. Our results here extend those of [Jo96] in that significantly weaker constraints are imposed on $WSD(T)$, in line with our aims stated in the introduction.

Our construction will build up the cyclic tree from partial cyclic trees via a left-to-right series of extensions. We first illustrate some of the problems associated with such a construction.

3.3.1 Example. Suppose that T contains the following rules

$$C: S(x, y) \wedge P(x) \wedge Q(y, x) \rightarrow Q(x, y) \vee P(y)$$

$$D_1: P(x) \wedge R(x, y) \rightarrow S(x, y) \vee Q(x, y)$$

$$D_2: P(y) \wedge Q(y, x) \rightarrow S(x, y)$$

$$D_3: R(x) \rightarrow S(x, x)$$

and we wish to construct cyclic trees for instances of $Q(u, v)$. We start with an initial partial cyclic tree \mathcal{T}_0 consisting of just the root node N_0 which is labelled with $Q(u, v)$. Suppose that we then apply rule 1, using the substitution $\mu = \{u \rightarrow x, v \rightarrow y\}$, to yield \mathcal{T}_1 (Figure 3.3.1). Notice that \mathcal{T}_1 is formed by appending the rule node RN_C as a child of N_0 , appending the antecedent(s) of C as the child nodes of RN_C , and finally applying the substitution μ to the entire tree. \mathcal{T}_1 is clearly itself a partial cyclic tree.

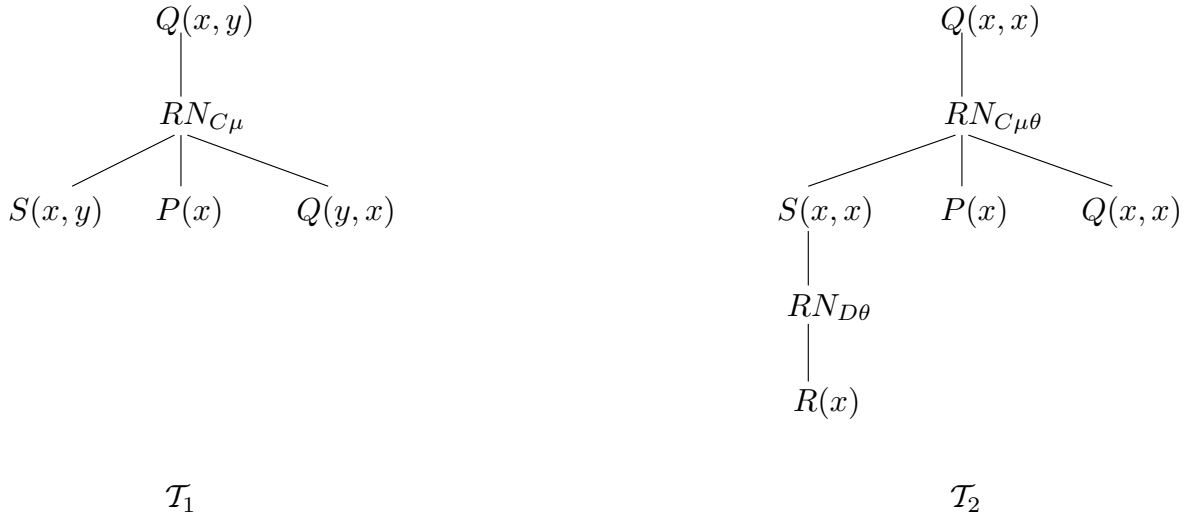


Figure 3.3.1.

Suppose now that we consider applying one of the rules D_1, D_2 or D_3 to $S(x, y)$. Firstly D_1 and D_2 are inapplicable, since $Q(x, y)/P(y)$ would become a member of both \mathcal{O} and $Pred$. D_3 is inapplicable for two reasons (cf., \mathcal{T}_2). Firstly, the unifying substitution $\theta = \{y \rightarrow x\}$ would result in $P(y\theta) = P(x) \in \mathcal{O}(\mathcal{T}_2)$ and $P(x\theta) = P(x) \in Pred(\mathcal{T}_2)$. Secondly $Q(y, x)$ would be unified with $Q(x, y)$, thus contravening condition (b)(ii) of Definition 3.1.3.

We thus see that the construction of cyclic trees at the first order level shares similarities with the propositional level, although we have to be careful that the applied substitution does not violate the properties of the existing tree, and that new nodes that are added to the tree likewise satisfy the required properties. The following definition is aimed at ensuring that existing nodes do not violate the required properties.

If N is a predicate node, then we may write $\text{CYC}(\mathcal{T}, N)$ to denote $\text{CYC}(N)$ computed in \mathcal{T} , etc. If μ is a substitution, then $\mathcal{T}\mu$ is the tree formed from \mathcal{T} by applying μ to all labels.

3.3.2 Definition. Let \mathcal{T} be a partial cyclic tree and μ a substitution. Then μ is *consistent* with \mathcal{T} iff

- (a) $\text{Pred}(\mathcal{T})\mu \cap (\mathcal{O}(\mathcal{T}) \cup \mathcal{N}(\mathcal{T}))\mu = \emptyset$,
- (b) for all predicate nodes M, N in \mathcal{T} , if $M > RN_{C\theta} > N$ (where M is the parent of $RN_{C\theta}$ which in turn is the parent of N), then
 - $\text{lab}(N) \notin \text{WSD}(\mathcal{L}) \implies N \notin \text{CYC}(\mathcal{T}\mu, M)$, and
 - $\text{lab}(N) \in \text{WSD}(\mathcal{L}) \implies N \notin \text{ACT}(\mathcal{T}\mu, M)$.

Conditions (a) and (b) are clearly related to conditions (a) and (b) (respectively) of Definition 3.1.3(b).

3.3.3 Lemma [Jo96]. If \mathcal{T} is a partial cyclic tree and μ is a substitution that is consistent with \mathcal{T} , then $\mathcal{T}\mu$ is a partial cyclic tree with $\text{Pred}(\mathcal{T})\mu = \text{Pred}(\mathcal{T}\mu)$, $\mathcal{O}(\mathcal{T})\mu = \mathcal{O}(\mathcal{T}\mu)$, and $\mathcal{N}(\mathcal{T})\mu = \mathcal{N}(\mathcal{T}\mu)$. If N is a predicate node in \mathcal{T} , then $\text{ACT}(\mathcal{T}, N)\mu = \text{ACT}(\mathcal{T}\mu, N)$.

In order to guarantee the termination of our tree construction we will, for the remainder of this section, assume that T satisfies the following constraints.

- (a) If $C \in \text{WSD}(T)$ then

$$\text{VAR}(\text{antec}(C)) - \text{VAR}(\text{conseq}(C)) \subseteq \text{VAR}(\{R(\mathbf{t}) \in \text{antec}(C) \mid \ell(R) < \ell(C)\}).$$

- (b) If $C \in \text{INT}(T) - \text{WSD}(T)$, then

$$\text{VAR}(\text{antec}(C)) \subseteq \text{VAR}(\{R(\mathbf{t}) \in \text{antec}(C) \mid \ell(R) < \ell(C)\}).$$

Our motivation in formulating these constraints is that they should be as weak as is possible. In particular we make note of our earlier assumption that $\text{INT}(T) - \text{WSD}(T)$ will typically be relatively small, and hence that less stringent constraints on $\text{WSD}(T)$ are desirable.

During our construction we will also assume that the child nodes N_1, N_2, \dots, N_r of a rule node $RN_{C\theta}$ are ordered so that if $\ell(\text{lab}(N_i)) < \ell(\text{lab}(N_j)) = \ell(C)$, then N_i is a left sibling of N_j .

The effect of these constraints will be to force relevant parts of the tree to become ground during the construction (Theorem 3.3.4), which in turn will ensure that the CYC sets do not change once used (Lemma 3.3.5) and also allow us to guarantee the termination of the construction (Theorem 3.3.10).

The following theorem is similar in spirit to [Jo96, Theorem 6.4.7].

3.3.4 Theorem [Jo98]. Suppose that \mathcal{T} is a partial cyclic tree in T in which $M > RN_{C\theta} > N$, where M is the parent of $RN_{C\theta}$ which in turn is the parent of N . Suppose also that $\ell(\text{lab}(N)) = \ell(C)$ and $\text{left}(N)$ contains no predicate leaf node. Then:

- (a) $\{lab(N') \mid N' \in left(N)\}$ is ground,
- (b) if $C \in WSD(T)$, then $VAR(lab(N)) \subseteq VAR(ACT(M))$, and
- (c) if $C \notin WSD(T)$, then each of the following sets is ground:
 - (i) $\{lab(N') \mid N' \text{ is a child node of } RN_{C\theta}\} = \text{antec}(C\theta)$,
 - (ii) $\text{conseq}(C\theta) \cup \mathcal{N}(C\theta)$, and
 - (iii) $\{lab(N') \mid N' \geq N, \ell(lab(N')) = \ell(lab(N))\}$.

3.3.5 Lemma. Let \mathcal{T} be a partial cyclic tree and μ a substitution that is consistent with \mathcal{T} . If N is a predicate node in \mathcal{T} such that $left(N)$ contains no predicate leaf node, then $CYC(\mathcal{T}\mu, N) = CYC(\mathcal{T}, N)\mu$.

Proof. Suppose that N has parent RN_C with $\ell(lab(N)) = \ell(C)$. By Proposition 3.1.4(ii), $CYC(N) \subseteq \{lab(N') \mid N' \geq N, \ell(lab(N')) = \ell(lab(N))\}$, and if $lab(N) \notin WSD(\mathcal{L})$, then this set is ground in \mathcal{T} (by Theorem 3.3.4(c)), whence the result is trivial.

In all other cases, it follows from Proposition 3.1.4 that $CYC(N) = \{lab(N)\}$, whence $CYC(\mathcal{T}\mu, N) = \{lab(\mathcal{T}\mu, N)\} = \{lab(\mathcal{T}, N)\}\mu = CYC(\mathcal{T}, N)\mu$. ■

The following provides our basic mechanism for extending (and thus constructing) partial cyclic trees.

3.3.6 Definition. Let \mathcal{T} be a partial cyclic tree in T and μ a substitution that is consistent with \mathcal{T} . Suppose that N is the predicate leaf node in \mathcal{T} such that $left(N)$ contains no predicate leaf node.

Let \mathcal{T}' be formed from \mathcal{T} by appending RN_C as a child of N (and appending the appropriate child nodes of RN_C from $\text{antec}(C)$), and finally applying the substitution μ to all nodes in the resultant tree.

\mathcal{T}' is said to be an *extension* of \mathcal{T} , written $\mathcal{T}' = \text{EXTEND}(\mathcal{T}, N, C, \mu)$. Notice that $Pred(\mathcal{T}') = Pred(\mathcal{T})\mu \cup \text{antec}(C)\mu$, $\mathcal{O}(\mathcal{T}') = \mathcal{O}(\mathcal{T})\mu \cup \mathcal{O}(\mathcal{T}', RN_{C\mu}) = \mathcal{O}(\mathcal{T})\mu \cup (\text{conseq}(C)\mu - CYC(\mathcal{T}, N)\mu)$, and $\mathcal{N}(\mathcal{T}') = \mathcal{N}(\mathcal{T})\mu \cup \mathcal{N}(C)\mu$.

We thus say that \mathcal{T}' is a *valid extension* of \mathcal{T} iff the following conditions hold.

- (a) $Pred(\mathcal{T})\mu \cup \text{antec}(C)\mu$ and $\mathcal{O}(\mathcal{T})\mu \cup (\text{conseq}(C)\mu - CYC(\mathcal{T}, N)\mu) \cup \mathcal{N}(\mathcal{T})\mu \cup \mathcal{N}(C)\mu$ are disjoint,
- (b) (i) $\text{conseq}(C)\mu \cap CYC(\mathcal{T}, N)\mu \neq \emptyset$,
- (ii) $\text{antec}(C)\mu \cap CYC(\mathcal{T}, N)\mu = \emptyset$, and
- (iii) $\text{antec}(C)\mu \cap WSD(\mathcal{L}) \cap ACT(\mathcal{T}, N)\mu = \emptyset$.

As one would expect, we can also insist that μ is a unifier of some subset of $\text{conseq}(C)$ with some subset of $CYC(N)$, which we note from Theorem 3.3.4/5 is either ground, or a singleton set. This in turn reduces the unification process and choices.

Condition (a) guarantees that $Pred(\mathcal{T}') \cap (\mathcal{O}(\mathcal{T}') \cup \mathcal{N}(\mathcal{T}')) = \emptyset$, and condition (b) ensures that the new nodes meet condition (b) of Definition 3.1.3. The following results show that valid extensions provide a correct, complete and terminating method of constructing cyclic trees for the first order level. Proposition 3.3.7 follows trivially from the conditions placed upon a valid extension.

3.3.7 Proposition (Correctness). If \mathcal{T} is a partial cyclic tree and \mathcal{T}' is a valid extension of \mathcal{T} , then \mathcal{T}' is a partial cyclic tree.

3.3.8 Theorem (Completeness). Let $P(\mathbf{t}\theta) \in \mathcal{H}$ and \mathcal{T} be a cyclic tree for $P(\mathbf{t}\theta)$ in $gr(\mathcal{T})$. Then there is a sequence $(\mathcal{T}_i \mid 0 \leq i \leq n)$ such that

- (i) $\mathcal{T}_0 = \{N_{P(\mathbf{t})}\}$,
- (ii) for each $i < n$, \mathcal{T}_{i+1} is a valid extension of \mathcal{T}_i , and
- (iii) $\mathcal{T} = \mathcal{T}_n$.

The proof is similar to that of [Jo96, Theorem 6.5.5], hence we omit the details.

3.3.9 Lemma. Suppose that $(\mathcal{T}_i \mid i = 0, 1, 2, \dots)$ is a sequence such that each \mathcal{T}_{i+1} is a valid extension of \mathcal{T}_i , say $\mathcal{T}_{i+1} = \text{EXTEND}(\mathcal{T}_i, N_i, C_i, \mu_i)$.

Suppose also that $N_0 > \dots > N_{r_0} > RN_{C_{r_0}} > N_{r_1} > RN_{C_{r_1}} > N_{r_2} > \dots > N_{r_j} > RN_{C_{r_j}} > N_{r_{j+1}} > \dots$ with $lab(N_{r_0}) \in WSD(\mathcal{L})$. Then there is a $j > 0$ such that $\ell(lab(N_{r_j})) < \ell(lab(N_{r_0}))$.

Proof. We present an outline sketch. Suppose that for each j , $\ell(lab(N_{r_j})) = \ell(lab(N_{r_0}))$. By Proposition 3.1.4(iv), each $C_{r_j} \in WSD(\mathcal{T})$ with $\text{conseq}(C_{r_j}) = \{lab(N_{r_j})\}$, thus $\ell(lab(N_{r_j})) = \ell(C_{r_j}) = \ell(lab(N_{r_{j+1}}))$. In particular, $lab(N_{r_{j+1}}) \in WSD(\mathcal{L})$. But then for each $j \geq 0$, we have that

- (i) $VAR(lab(\mathcal{T}_{r_{j+1}}, N_{r_{j+1}})) \subseteq VAR(lab(\mathcal{T}_{r_j}, N_{r_j}))$ (by Theorem 3.3.4(b)), and
- (ii) $lab(\mathcal{T}_{r_{j+1}}, N_{r_{j+1}}) \notin \{lab(\mathcal{T}_{r_i}, N_{r_i}) \mid 0 \leq i \leq j\}$ (by Definition 3.3.2).

However, clearly an infinite number of distinct atoms cannot be created from a finite number of predicates and variables. ■

3.3.10 Theorem (Termination). Let $\mathcal{T}_0 = \{N_{P(\mathbf{t})}\}$. Then there is no infinite sequence $(\mathcal{T}_i \mid i = 0, 1, 2, \dots)$ such that each \mathcal{T}_{i+1} is a valid extension of \mathcal{T}_i .

Proof. Suppose that $(\mathcal{T}_i \mid i = 0, 1, 2, \dots)$ is such that each \mathcal{T}_{i+1} is a valid extension of \mathcal{T}_i , say $\mathcal{T}_{i+1} = \text{EXTEND}(\mathcal{T}_i, N_i, C_i, \mu_i)$.

Clearly an infinite branch $N_0 > \dots > N_{r_0} > RN_{C_{r_0}} > N_{r_1} > RN_{C_{r_1}} > N_{r_2} > \dots > N_{r_j} > RN_{C_{r_j}} > N_{r_{j+1}} > \dots$, must be generated, and thus we may find a $k > 0$ such that $\ell(lab(N_{r_j})) = \ell(lab(N_{r_k}))$ for each $j > k$. By Lemma 3.3.9, if $j \geq k$ then $lab(N_{r_j}) \notin WSD(\mathcal{L})$. Note that

- (i) for $j \geq k$, $\{lab(\mathcal{T}_{r_{j+1}}, N') \mid N' \geq N_{r_{j+1}}, \ell(lab(N')) = \ell(lab(N_{r_{j+1}}))\}$ is ground (by Theorem 3.3.4(c)), and
- (ii) for $j \geq k$, $lab(\mathcal{T}_{r_{j+1}}, N_{r_{j+1}}) \notin \text{CYC}(\mathcal{T}_{r_{j+1}}, N_{r_j})$, by the consistency of the substitutions.

But then we may find $k < j < h$ such that $lab(\mathcal{T}_{r_h}, N_{r_h}) = lab(\mathcal{T}_{r_j}, N_{r_j})$ and $lab(\mathcal{T}_{r_{h+1}}, N_{r_{h+1}}) = lab(\mathcal{T}_{r_{j+1}}, N_{r_{j+1}})$. Hence $lab(\mathcal{T}_{r_{h+1}}, N_{r_{h+1}}) \in \{lab(\mathcal{T}_{r_{h+1}}, N_i) \mid j \leq i \leq h\} \subseteq \text{CYC}(\mathcal{T}_{r_{h+1}}, N_{r_h})$, thus contradicting condition (ii). ■

§4. VALIDATION

We now turn our attention to the process of validation, ie., showing that $T \models \bigvee (\mathcal{A} \cup \{\neg L \mid L \in \text{Pred}(\mathcal{T})\} \cup \mathcal{O}(\mathcal{T}) \cup \mathcal{N}(\mathcal{T}))$, where \mathcal{T} is a cyclic tree, and \mathcal{A} is the partial answer generated to-date.

In order to achieve this we will employ extended deduction trees, which were introduced in [Jo98, Jo98a] as a means of performing top-down processing of queries of the form $?Q(\mathbf{x})$. We wish to show in particular that we can take advantage of two properties of the set $\mathcal{A} \cup \{\neg L \mid L \in \text{Pred}(\mathcal{T})\} \cup \mathcal{O}(\mathcal{T}) \cup \mathcal{N}(\mathcal{T})$ in order to make such testing efficient in that:

1. The testing process is entirely ground unless one enters the semi-definite part of the database, and
2. A linear procedure can be employed if the processing enters the definite/semi-definite part of the database.

In order to achieve these goals we will, in Section 5, place a further (mild) constraint on the database (akin to the constraints already introduced in Section 3.3). This constraint also has the effect of guaranteeing termination.

In this section we will first review the basic properties [Jo98, Jo98a] of extended deduction trees; Section 5 and 6 will discuss their applicability at the first order level.

4.1 Motivation.

We first present an example to motivate the definition (4.1.3) of extended deduction trees. The following theorem is trivial.

4.1.1 Theorem [Jo98a]. Suppose that \mathcal{P} is a set of ground atoms and $C \in \text{gr}(T)$ with $\text{conseq}(C) \subseteq \mathcal{P}$. Then $T \models \bigvee \mathcal{P}$ iff $T \models A \vee \bigvee \mathcal{P}$ for each $A \in \text{antec}(C) \cup \overline{\mathcal{N}}(C)$.

4.1.2 Example. Suppose that T is propositional and contains the following rules

- | | | |
|--|---|--|
| 1. $A_2 \wedge E_0 \wedge \neg B_0 \rightarrow A_0 \vee A_1$ | 2. $E_1 \wedge \neg E_3 \rightarrow B_0 \vee B_1$ | 3. $E_1 \wedge \neg E_5 \rightarrow B_0$ |
| 4. $E_4 \wedge E_1 \rightarrow B_4 \vee B_1$ | 5. $E_4 \vee E_3$ | 6. $E_0 \vee E_2$ |
| 7. $E_1 \vee E_2$ | 8. $E_5 \vee E_2$ | 9. $E_4 \rightarrow A_2$. |

where $\ell(A_i) = 2$, $\ell(B_i) = 1$ and $\ell(E_i) = 0$. Suppose that we wish to decide if $T \models \bigvee \mathcal{P}$, where $\mathcal{P} = \{\neg A_2, A_0, A_1, B_4, E_2\}$. The consequent of rule 1 is contained in \mathcal{P} , thus by Theorem 4.1.1, $T \models \bigvee \mathcal{P}$ iff $T \models A_2 \vee \bigvee \mathcal{P}$, $T \models E_0 \vee \bigvee \mathcal{P}$, and $T \models \neg B_0 \vee \bigvee \mathcal{P}$. This is represented in the top 3 levels in the predicate rule tree depicted in Figure 4.1.2(i).

applied to the left-hand child, and the two child nodes are solved by rule 5 and by the rule $E_1 \vee \neg E_1$. This then shows that $T \models \bigvee \mathcal{P}$. The tree illustrated in Figure 4.1.2(i) is an extended deduction tree.

4.1.3 Definition. Let \mathcal{P} be a set of atoms. An *extended deduction tree* \mathcal{T} for \mathcal{P} is a predicate-rule tree satisfying the following conditions.

- (a) $root(\mathcal{T})$ is a set node with $lab(root(\mathcal{T})) = \mathcal{P}$,
- (b) If N is a set node or a positive predicate node with child RN , then RN is labelled with an instance $C\theta$ of a rule $C \in T$ (and we write $RN_{C\theta}$) where $conseq(C\theta) \subseteq ACT(N)$. For each $K \in antec(C\theta) \cup \overline{\mathcal{N}}(C\theta)$, $RN_{C\theta}$ has a (predicate) child node labelled with K , and these are the only child nodes of $RN_{C\theta}$.
- (c) Suppose that N is a negative predicate node with $lab(N) = \neg K$. If N has child node RN , then K is ground, RN is labelled with $\rightarrow \neg K$ (written $RN_{\rightarrow \neg K}$), and for each cyclic tree \mathcal{T} for K in T , $RN_{\rightarrow \neg K}$ has a (set) child node with label $\{\neg L \mid L \in Pred(\mathcal{T})\} \cup \mathcal{O}(\mathcal{T}) \cup \mathcal{N}(\mathcal{T})$. $RN_{\rightarrow \neg K}$ has no other child nodes.

As indicated above, a rule node of the form $RN_{\rightarrow \neg K}$ denotes the generation of cyclic trees in order to process the negative subgoal $\neg K$. By Theorem 3.2.5, cyclic trees are ground, hence the insistence that K is ground. Theorems 4.1.4 and 4.2.2 below show that extended deduction trees can be used to characterise the property $T \models \bigvee \mathcal{P}$.

4.1.4 Theorem [Jo98]. Suppose that \mathcal{P} is a set of ground atoms. If \mathcal{P} has an extended deduction tree \mathcal{T} in $gr(T)$ containing no predicate leaf node, then $T \models \bigvee \mathcal{P}$.

4.2 Cyclic sets and extended deduction trees.

The converse of Theorem 4.1.4 does not hold in general, since negative atoms need to be “expanded” using cyclic trees. We thus make the following definition.

4.2.1 Definition [Jo98, Jo98b]. Let \mathcal{P} be a non-complementary set of ground atoms (ie., \mathcal{P} contains no complementary pair). Then \mathcal{P} is said to be *cyclic* iff there is a set of cyclic trees $\{\mathcal{T}_i \mid 0 \leq i \leq m\}$ in $gr(T)$ such that

- (i) $\{K \in \mathcal{H} \mid \neg K \in \mathcal{P}\} = \bigcup_{i=0}^m Pred(\mathcal{T}_i)$, and
- (ii) $\{K \in \mathcal{H} \mid K \in \mathcal{P}\} \supseteq \bigcup_{i=0}^m (\mathcal{O}(\mathcal{T}_i) \cup \mathcal{N}(\mathcal{T}_i))$.

Notice that sets of the form $\mathcal{A} \cup \{\neg L \mid L \in Pred(\mathcal{T})\} \cup \mathcal{O}(\mathcal{T}) \cup \mathcal{N}(\mathcal{T})$ (ie., sets to be validated) are indeed cyclic. Notice also that a set of positive ground atoms is always cyclic (whence the query processing required by the remarks prior to Proposition 2.5 can also be handled by extended deduction trees).

When we move to the first order level we will wish to impose various constraints on the construction of extended deduction trees in order to guarantee termination and promote efficiency. Of course these constraints should not compromise completeness in that if \mathcal{P} is cyclic with $T \models \bigvee \mathcal{P}$, then an extended deduction tree for \mathcal{P} should be constructable

under the given constraints. The additional conditions (b)-(g) in Theorem 4.2.2 below are included specifically for this purpose. In particular, conditions (b)-(e) indicate that we may adopt a linear construction should the processing pass into the definite/semi-definite part of the database. Conditions (f) and (g) guarantee termination by limiting the length of branches.

4.2.2 Theorem [Jo98]. Suppose that \mathcal{P} is cyclic and $T \models \bigvee \mathcal{P}$. Then there is an extended deduction tree \mathcal{T} for \mathcal{P} in $gr(T)$ such that

- (a) each leaf node in \mathcal{T} is a rule node,
- (b) whenever N is a positive predicate node with $lab(N) \in Def(\mathcal{L})$, then $lab(N) \in \mathcal{M}_{Def(T)}$,
- (c) whenever N is a negative predicate node with $\neg lab(N) \in Def(\mathcal{L})$, then $\neg lab(N) \notin \mathcal{M}_{Def(T)}$,
- (d) whenever N is a positive predicate node with $lab(N) \in SD(\mathcal{L}) \cup Def(\mathcal{L})$ and RN_C is the child node of N , then $conseq(C) = \{lab(N)\}$,
- (e) whenever N is a set node with parent $RN_{\rightarrow \neg P(a)}$ where $P \in Def(\mathcal{L})$ and RN_C is the child node of N , then $C \in Def(T)$ and $conseq(C) \subseteq lab(N)$.
- (f) if N is a predicate node in \mathcal{T} , then $lab(N)$ does not appear in the label of any predicate or set node M such that $M > N$, and
- (g) if N is a positive predicate node in \mathcal{T} with $\ell(lab(N)) = 0$, then $lab(N)$ occurs in some rule in $EXT(T)$.

4.3 Search space pruning.

Search space pruning of extended deduction trees is vital, if we are to ensure that the validation process is grounded outside of the semi-definite part of the database. The following example illustrates the form of pruning to be employed.

4.3.1 Example. Suppose that our database contains the following rules.

- | | | | |
|-------------------------------|---------------|-------------------------------|--------|
| 1. $S \wedge R \rightarrow Q$ | 3. $V \vee S$ | 5. $P \wedge V \rightarrow Q$ | 7. ... |
| 2. $W \wedge U \rightarrow Q$ | 4. R | 6. $P \vee S$ | 8. ... |

with query $?Q$. An extended deduction tree for $\{Q\}$ is shown in Figure 4.3.1.

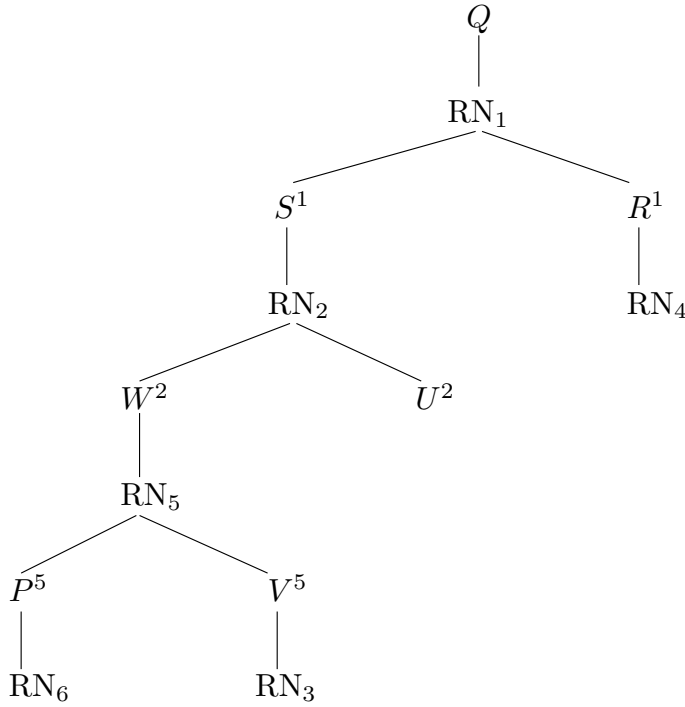


Figure 4.3.1.

Atoms are superscripted simply to indicate which rule they come from, and to aid the description below. Notice that $\text{ACT}(W^2)$ is solved using rule 5 and then rules 6 and 3. The atom W is not needed (for the applications of rules 5,6 and 3) and thus the use of rule 2 is actually redundant. There is thus no need to examine those branches emanating from the node U^2 : the subtree below W^2 solves the branch to S^1 irrespective of which antecedent from rule 2 is used. We will say that U^2 has been *pruned*.

The form of redundancy discussed above is most easily analysed by examining, for a given predicate (or set) node N , which atoms *must* lie on $\text{ACT}(N)$ in order to satisfy condition (b) of Definition 4.1.3 with respect to the rule nodes below N . For example in the tree depicted in Figure 4.3.1, the branch above RN_6 must contain P and S , whereas the branch above RN_5 must contain S . For a given predicate/set node N , the essential predicates on $\text{ACT}(N)$ will be denoted by $\text{ESS}(N)$. It is precisely because $W \notin \text{ESS}(W^2)$, that we regard W^2 as *redundant* (in enabling the tree to satisfy condition (b) of Definition 4.1.3).

A precise definition of these concepts is provided in [Jo98a]; for the purposes of the present paper, the somewhat technical details are not needed.

4.3.2 Definition [Jo98a]. An extended deduction tree \mathcal{T} for \mathcal{P} in $gr(T)$ is said to be *pruned* iff it satisfies the following conditions.

- (a) If N is redundant, then N is not a leaf node, and for each $K \in \text{lab}(N)$ either $K \notin \text{ESS}(N)$ or $(\exists M > N)(K \in \text{lab}(M))$.

- (b) If M is a predicate or set leaf node, then M is a leaf iff M is pruned. If M is pruned then M has a redundant left sibling.

Clearly if $K \in \text{lab}(N)$ also occurs in the label of some node M above N , then both occurrences of K are not necessary in order for the tree to satisfy condition (b) of Definition 4.1.3 with respect to the rule nodes below N . Condition (a) of the above definition dictates that under such circumstances, we regard the lower occurrence of K as redundant.

Thus as illustrated in the previous example, the right siblings of a redundant predicate node are pruned. Notice that the status of a node N (as redundant/non-redundant) cannot be determined until the construction of the tree below N is complete (since only then do we know which rules were used in that part of the tree).

The following theorem shows that pruning does not compromise correctness.

4.3.3 Theorem [Jo98]. Let \mathcal{P} be a set of ground atoms. If \mathcal{P} has a pruned deduction tree in $gr(T)$, then $T \models \bigvee \mathcal{P}$.

§5. FIRST ORDER DATABASES

5.1 Deduction sequences.

In this section we illustrate the construction of extended deduction trees at the first order level. The construction builds up the tree, left-to-right, as a sequence of partial trees. As indicated in Section 4.2 we constrain the construction to build pruned trees satisfying the conditions (b)-(g) of Theorem 4.2.2. Our aim is to illustrate the properties of this construction that result from the groundedness of the initial goal and from the pruning described in the previous section.

5.1.1 Example. Let T contain the following rules

1. $S(y, x) \wedge Q(y, x) \wedge \neg R(y, x) \rightarrow Q(x, b)$
2. $D(y, x, z) \wedge S(x, y) \rightarrow S(y, x)$
3. $D(b, a, c)$
4. $E(x, y) \rightarrow S(x, b)$
5. $E(a, b) \vee E(a, c)$
6. $E(a, b) \vee E(b, a)$
7. $E(x, y) \rightarrow Q(x, y) \vee Q(y, x)$
8.

and suppose that we wish to determine whether $T \models Q(a, b)$. x, y, z are variables; a, b, c are constants. D and E are extensional, and S is semi-definite.

Applying rule 1 to $Q(a, b)$ yields the predicate-rule tree \mathcal{T}_1 (Figure 5.1.1(i)). Suppose that we now attack the left-most child of RN_1 using rule 2 thus resulting in the predicate-rule tree \mathcal{T}_2 .

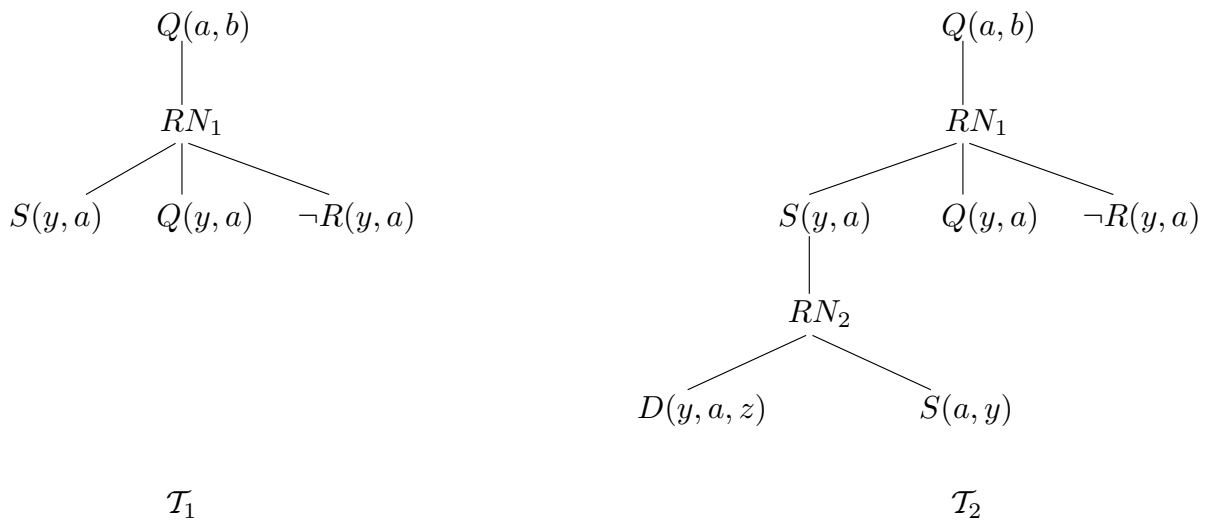


Figure 5.1.1(i).

The left-most child of RN_2 is solved by applying the fact $D(b, a, c)$, thus resulting in \mathcal{T}_3 (Figure 5.1.1(ii)). We then attack the atom $S(a, b)$ for which the only applicable rule is rule 4. (Notice that rule 2 is *not* applicable, since it would introduce a duplicate of the node $S(b, a)$, thus contravening condition (f) of Theorem 4.2.2.)

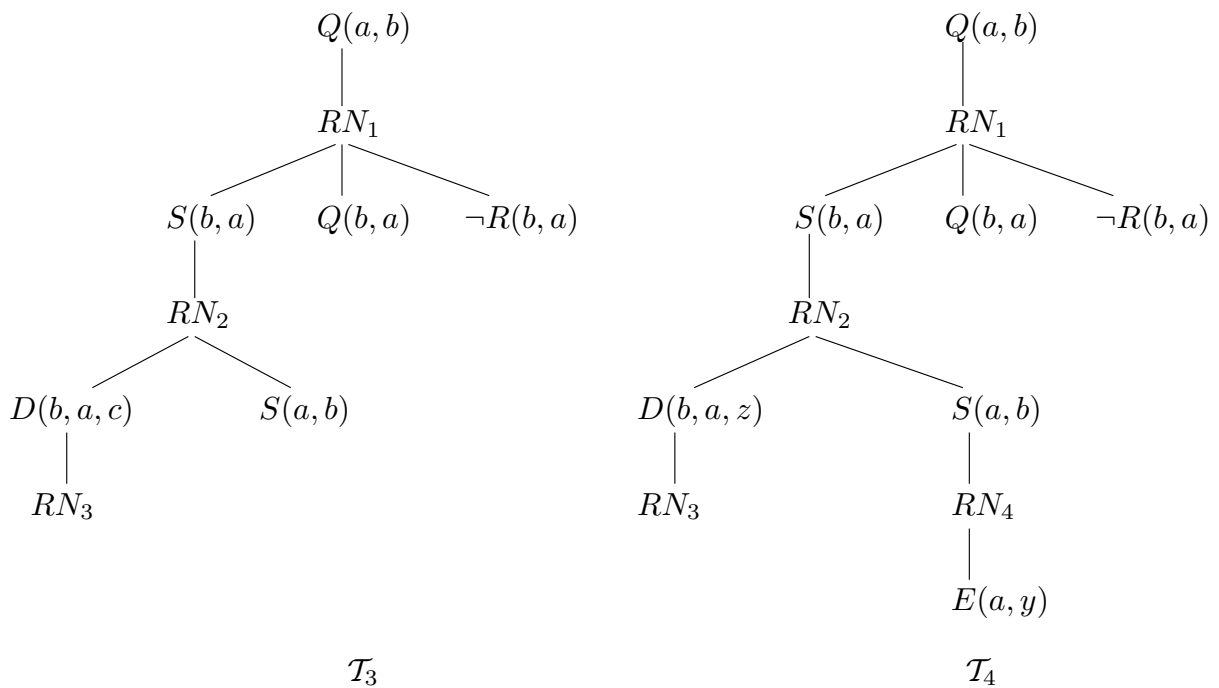


Figure 5.1.1(ii).

Applying rule 4 again yields the tree \mathcal{T}_5 (Figure 5.1.1(iii)). (Note again that at this stage, rule 2 is not applicable by condition (f) of Theorem 4.2.2, and that rules 5-7 are not applicable as there is no suitable unifier.)

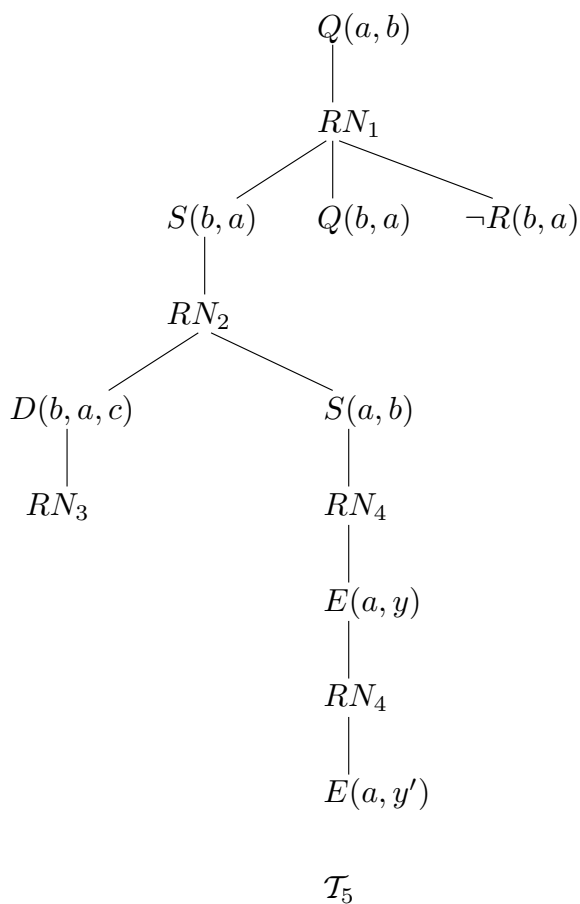


Figure 5.1.1(iii).

The branch to $E(a, y')$ is then solved using rule 5 (and the substitution $\{y \rightarrow b, y' \rightarrow c\}$). We would then continue to attack the goal $Q(a, b) \vee Q(b, a)$, etc.

We thus note the following features of the construction to-date. Firstly the definite predicate $D(y, a, z)$ is solved directly using $D(b, a, c)$, and any partial branch terminating in S is extended in a linear fashion in line with condition (d) of Theorem 4.2.2. (Note that because S is *semi*-definite, S cannot be *solved* in a purely linear fashion, in that we have to apply rule 4 a second time, the second application being non-linear.)

The unification process is potentially more complex during the application of a disjunctive rule. However, after the processing of the sub-tree below $S(y, a)$ we see that $S(y, a)$ and the subgoals $Q(a, b) \vee Q(y, a)$ have become ground whence the unification choices involved in the application of an indefinite rule (eg., rule 7) are limited and simplified, thus reducing the search space, and the complexity of the unification process.

Also note that the subgoal $\neg R(y, a)$ has become ground. This is particularly important: if the subgoal had not become ground we would have had to arbitrarily choose a ground instance of $R(y, a)$ before generating cyclic trees.

5.1.2 Example. Let T contain the following rules

1. $V(y, x) \wedge Q(y, x) \wedge \neg R(y, x) \rightarrow Q(x, b)$
2. $D(x, y, z) \wedge Q(x, y) \rightarrow Q(y, x)$
3. $D(b, a, c)$
4. $E(a, b)$
5. $E(x, y) \rightarrow Q(x, y) \vee Q(y, x)$
6.

and suppose again that we wish to determine whether $T \models Q(a, b)$. Applying rule 1 to $Q(a, b)$ yields \mathcal{T}_1 (Figure 5.1.1(i)). Suppose that we now attack the left-most child of RN_1 using rule 2, thus resulting in the tree \mathcal{T}_2 .

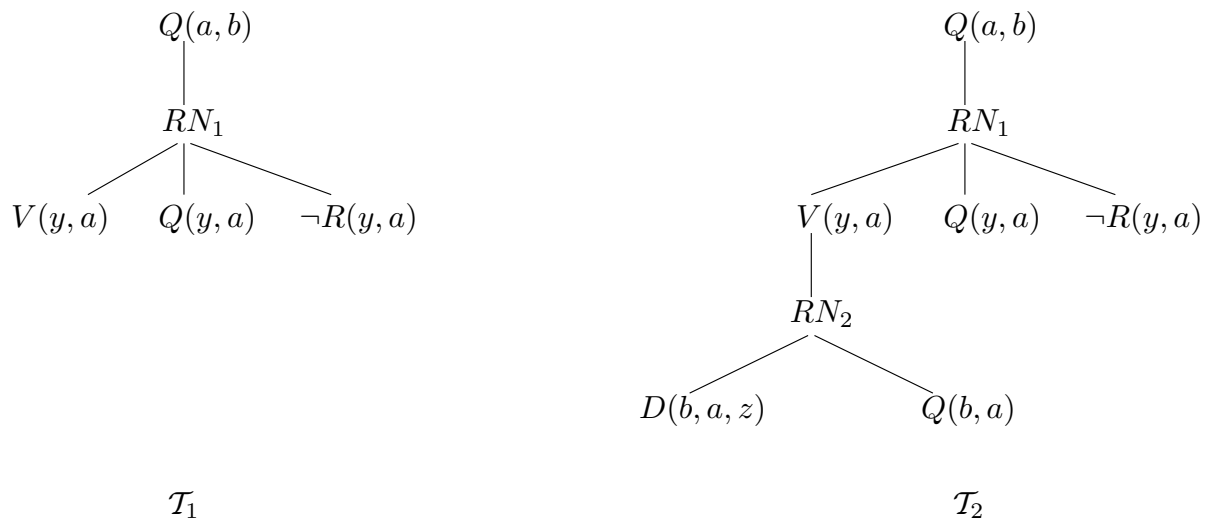


Figure 5.1.2(i).

Applying rules 3-5 yields the tree \mathcal{T}_3 depicted in Figure 5.1.2(ii).

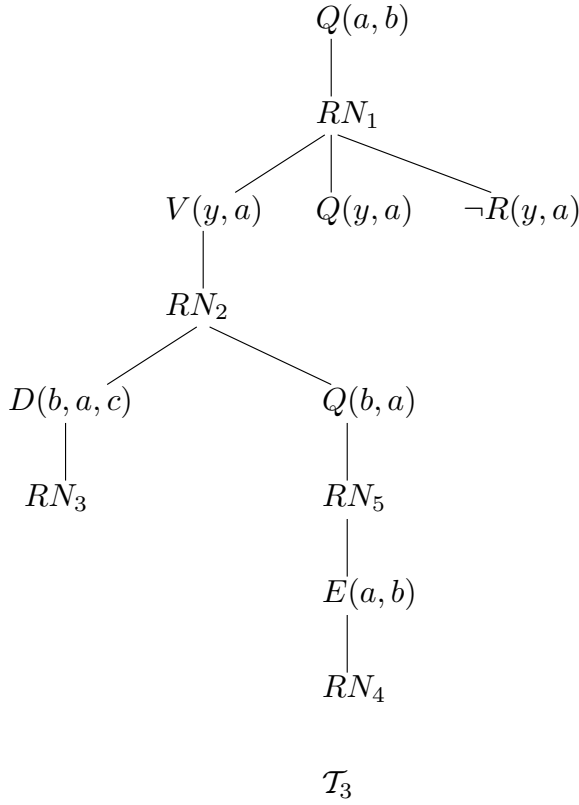


Figure 5.1.2(ii)

Notice now that $V(y, a)$ has *not* become ground. However it is redundant, hence the non-groundedness of the remaining two subgoals is not problematic, since they can be pruned.

We thus see that extended deduction trees are constructed in a left-to-right fashion via a series of partial trees \mathcal{T}_i . At each stage we extend the left-most predicate/set leaf node N_i in \mathcal{T}_i (assuming that N_i has not been pruned). We refer to the sequence $(\mathcal{T}_i, N_i \mid i = 0, 1, 2, \dots)$ as a *deduction sequence*. The precise technical details of such sequences ([Jo98, Jo98a]) are not needed for the current discussion.

As indicated earlier, given N_i we can only decide whether N_i is redundant once extensions of $\text{ACT}(N_i)$ have been examined. We denote by \mathcal{T}_{i^*} the tree which results from the examination of such extensions. The following theorem, illustrated in the previous two examples, is a special case of [Jo98a, Theorem 5.2.1].

5.1.3 Theorem. If N_i is not redundant, then in \mathcal{T}_{i^*} the label of N_i is ground.

5.2 Groundedness

At the end of Example 5.1.1, we indicated the importance of negative subgoals becoming ground. The following theorem shows that pruning has this effect.

5.2.1 Theorem. Let $(\mathcal{T}_i, N_i \mid 0 \leq i \leq n)$ be a deduction sequence, and N_i a negative predicate node. If N_i is unpruned in \mathcal{T}_i , then $lab(\mathcal{T}_i, N_i)$ is ground.

Proof. Suppose that $lab(\mathcal{T}_i, N_i)$ is not ground. Let $RN_{C\theta}$ be the parent node of N_i in \mathcal{T}_i , then since $VAR(\mathcal{N}(C\theta)) \subseteq VAR(\text{antec}(C\theta))$ (cf. Section 1.1.2 (vii)), we may find a left sibling N_k of N_i such that $lab(\mathcal{T}_i, N_k)$ is a non-ground positive atom.

N_k cannot be redundant (since N_i is unpruned), whence $lab(\mathcal{T}_{k^*}, N_k)$, and hence $lab(\mathcal{T}_i, N_k)$, are ground by Theorem 5.1.3. ■

We can also show that when extending a node N_i with a rule $C_i \notin SD(T)$, the unification process is considerably simplified since the relevant part of the branch $ACT(\mathcal{T}_i, N_i)$ is already ground. In order to achieve this we shall assume that for each $C \in INT(T)$

$$VAR(\text{antec}(C)) - VAR(\text{conseq}(C)) \subseteq VAR(\{R(\mathbf{t}) \in \text{antec}(C) \mid R \in SD(\mathcal{L}) \cup \text{EXT}(\mathcal{L}), \ell(R) < \ell(C)\})$$

A similar constraint is seen in the construction of cyclic trees (Section 3.3).

In [Jo98a] we showed that the construction of extended deduction trees can be guaranteed to terminate provided that the database satisfies the above condition, and provided that certain conditions are imposed in the construction (such as a linear treatment of semi-definite predicates). Only one of these constraints is of direct relevance to the following discussion, namely that the child nodes of a rule node of the form $RN_{C\theta}$ are ordered so that the semi-definite and extensional child nodes whose ℓ -value is strictly less than $\ell(C)$ appear to the left of all others.

5.2.2 Theorem. Let $(\mathcal{T}_i, N_i \mid 0 \leq i \leq n)$ be a deduction sequence such that $lab(\mathcal{T}_0, N_0)$ is ground. For each i , if N_i is unpruned in \mathcal{T}_i , then each positive atom in $ACT(\mathcal{T}_i, N_i) \cap (INT(\mathcal{L}) - SD(\mathcal{L}))$ is ground.

Proof. Suppose that each positive atom in $ACT(\mathcal{T}_i, N_i) \cap (INT(\mathcal{L}) - SD(\mathcal{L}))$ is ground. Let RN be the child node of N_i (in \mathcal{T}_{i+1}), then we show that the property holds for all child nodes of RN .

If N_i is a negative predicate node, then by Theorem 3.2.5 the label of each child of RN is ground, and hence the theorem holds for each child of RN .

Suppose that N_i is a positive predicate node or a set node, and that N_k is a positive child of RN , where $lab(N_k) \in INT(\mathcal{L}) - SD(\mathcal{L})$ and N_k is unpruned in \mathcal{T}_k . In \mathcal{T}_k , the label of RN is of the form $C\beta$, where $C \in T$ and β is some substitution. Note that $lab(\mathcal{T}_k, N_k) \in \text{antec}(C\beta)$, whence $C \in INT(T) - SD(T)$, and in particular

$$\text{conseq}(C\beta) \subseteq ACT(\mathcal{T}_k, N_k) \cap (INT(\mathcal{L}) - SD(\mathcal{L})) = ACT(\mathcal{T}_i, N_i)\beta \cap (INT(\mathcal{L}) - SD(\mathcal{L})).$$

Thus by our hypothesis, $\text{conseq}(C\beta)$ is ground.

Suppose that the variable x occurs in $\text{lab}(\mathcal{T}_k, N_k)$, then $x \in \text{VAR}(\text{antec}(C\beta)) - \text{VAR}(\text{conseq}(C\beta))$, whence by the conditions on T and the construction given at the beginning of this section, we may find a left sibling N_j of N_k such that x occurs in $\text{lab}(\mathcal{T}_k, N_j)$. However, since N_k is unpruned, N_j cannot be redundant, whence $\text{lab}(\mathcal{T}_{j^*}, N_j)$ and hence $\text{lab}(\mathcal{T}_k, N_j)$ are ground. ■

5.2.3 Unification.

We have seen that the number of alternative ways in which a branch may be extended (whilst constructing an extended deduction tree) can be limited by the conditions of Theorem 4.2.2. These alternatives are limited still further, and in particular the unification process itself is simplified, by the effect of pruning.

1. No unification process is involved in extending a negative predicate node, since the node is already ground.
2. When extending a branch using a rule in $\text{INT}(T) - \text{SD}(T)$, the relevant part of the branch is already ground.
3. When extending a branch using a rule in $\text{SD}(T)$, the unification process unifies the head of the rule with a single atom on the branch (by condition (f) of Theorem 4.2.2), and
4. When terminating a branch using a rule in $\text{EXT}(T)$, the rule itself is grounded (cf., Definition 1.1.3).

§6. EXTENSIONS

6.1 Factorisation

In [Jo98, Jo98a] we also studied *factorisation* in relation to extended deduction trees and the associated query processing. The following example illustrates what is meant by factorisation, and also the complications involved in employing factorisation together with pruning.

6.1.1 Example. Suppose that our database contains the following rules.

- | | | |
|-------------------------------|--------|-------------------------------|
| 1. $S \wedge R \rightarrow Q$ | 3. V | 5. $U \wedge V \rightarrow S$ |
| 2. $W \wedge U \rightarrow Q$ | 4. R | 6. |

with query $?Q$. Consider the extended deduction tree for $\{Q\}$ shown in Figure 6.1.1.

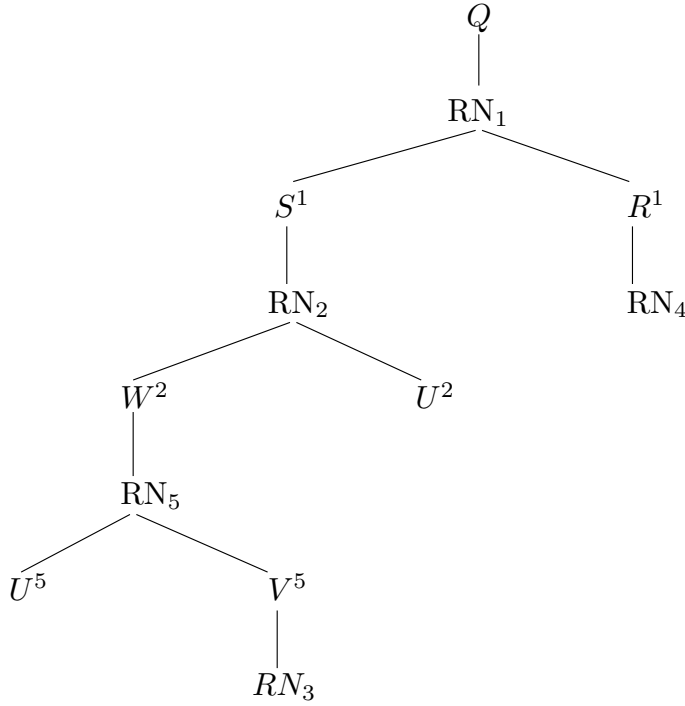


Figure 6.1.1.

Notice that $\text{ACT}(W^2)$ is extended by rule 5. The left child node, U^5 of RN_5 indicates that we need to show that $T \models Q \vee S \vee W \vee U$. However this subgoal is subsumed by $\text{ACT}(U^2)$ (ie., by the goal $Q \vee S \vee U$), and hence there is no need to consider extensions of $\text{ACT}(U^5)$, since extensions of $\text{ACT}(U^2)$ are scheduled to be considered later. We say that U^2 *factors* U^5 .

Following the decision to factor U^5 we would then continue to solve $\text{ACT}(V^5)$ using rule 3. Notice that W^2 is redundant, and hence U^2 can be pruned. However, extensions of U^5 were previously discarded on the assumption that extensions of U^2 would be considered later, which is not the case if U^2 is pruned.

A simple solution to this problem is to amend the pruning of the tree: the descendants of U^2 cannot be ignored (ie., U^2 cannot be pruned) if U^2 is used to factor some branch through W^2 .

The solution suggested above is shown in [Jo98,Jo98a] to provide a correct pruning rule that does not compromise termination. In addition, whilst factorisation does compromise Theorem 5.1.3 (and hence our *proof* of Theorem 5.2.2 given earlier), it can be shown [Jo98] that factorisation does not compromise Theorem 5.2.2 itself.

The preservation of the property given in Theorem 5.2.1 is far more problematic, as is illustrated by the following two examples.

6.1.2 Example. Let T contain the following rules.

1. $S(y, x) \wedge Q(y, x) \wedge \neg R(y, x) \rightarrow Q(x, b)$
2. $\neg R(y, x) \rightarrow S(y, x)$
3.

and suppose that we wish to determine whether $T \models Q(a, b)$. Applying rule 1 to $Q(a, b)$ yields the predicate-rule tree \mathcal{T}_1 (Figure 6.1.2). If we now attack the left-most child of RN_1 using rule 2, we generate the predicate-rule tree \mathcal{T}_2 .

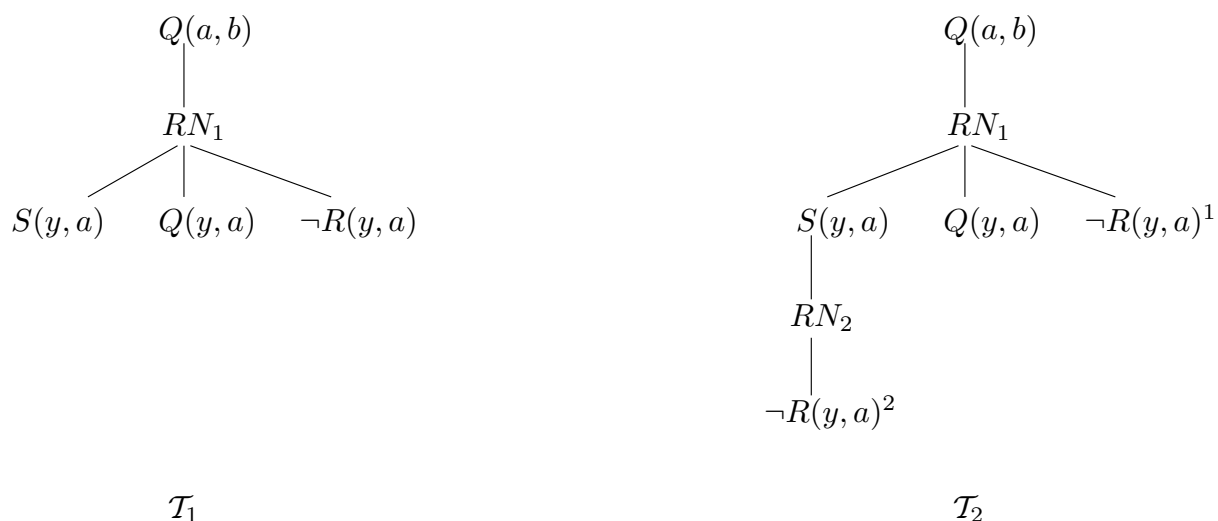


Figure 6.1.2.

Notice now that the subgoal $\neg R(y, a)^2$ can be factored (by $\neg R(y, a)^1$), in which case $S(y, a)$ becomes redundant and $Q(y, a)$ becomes pruned (whence, as in Example 5.1.2 the non-groundedness of $Q(y, a)$ causes no problem). However, because $\neg R(y, a)^1$ was used to factor a branch through the redundant node $S(y, a)$, $\neg R(y, a)^1$ cannot be pruned, and its extensions must be considered.

6.1.3 Example. Suppose that T contains the following rules.

1. $S(x, y) \wedge P(x, y) \rightarrow Q(y)$
2. $P(x, y) \wedge \neg R(x, y) \rightarrow S(x, y)$
3.

Consider the predicate-rule tree for $Q(a)$ illustrated in Figure 6.1.3.

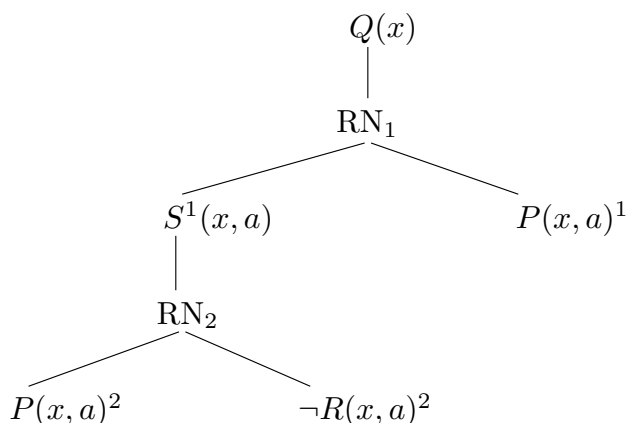


Figure 6.1.3.

We can now factor $P(x, a)^2$, whence $P(x, a)^2$ becomes a leaf (and in particular it cannot be regarded as redundant by Definition 4.3.2). Thus $\neg R(x, a)$ is not pruned, and hence again its extensions must be considered.

These examples raise the question as to how we can limit factorisation in such a way as to preserve Theorem 5.2.1. In fact the problem in the above examples is that factorisation is applied using a *non-ground* atom (ie., $\neg R(y, a)$ and $P(x, a)$). If we restrict factorisation so that it is only applied using ground atoms, then it can be shown ([Jo98, Section 5]) that Theorem 5.2.1 is preserved.

For atoms in $\text{INT}(\mathcal{L}) - \text{SD}(\mathcal{L})$ this restriction is of no consequence: we have already seen in Theorem 5.2.2 that such atoms become ground anyway. For atoms in $\text{Def}(\mathcal{L})$ we can also show (Section 6.2 below) that this restriction is irrelevant. However for other atoms the restriction is a real limitation, but one that we believe is worth paying in order to guarantee the groundedness of negative subgoals, and hence alleviate the potential problem described at the end of Example 5.1.1.

6.2 Bottom-up treatment of definite predicates

Conditions (b)-(e) of Theorem 4.2.2 indicate that when constructing an extended deduction tree, we can adopt an entirely linear strategy if we enter the definite part of the database. Suppose that we disallow the usage of definite predicates (positive or negative) in factorisation. If $P(\mathbf{x})$ is a predicate node with $P \in \text{Def}(\mathcal{L})$, then the tree constructed below $P(\mathbf{x})$ is a pruned tree for some instance $P(\mathbf{x}\theta)$ of $P(\mathbf{x})$, whence the tree witnesses that $T \models P(\mathbf{x}\theta)$ (or equivalently that $P(\mathbf{x}\theta) \in \mathcal{M}_{\text{Def}(T)}$). (Similarly for negative predicate nodes $\neg P(\mathbf{a})$, the tree below the node simply witnesses that $P(\mathbf{a}) \notin \mathcal{M}_{\text{Def}(T)}$.)

But then we can see that disallowing the use of definite predicates in factorisation is of no consequence. To see this, consider the tree illustrated in Figure 6.2.

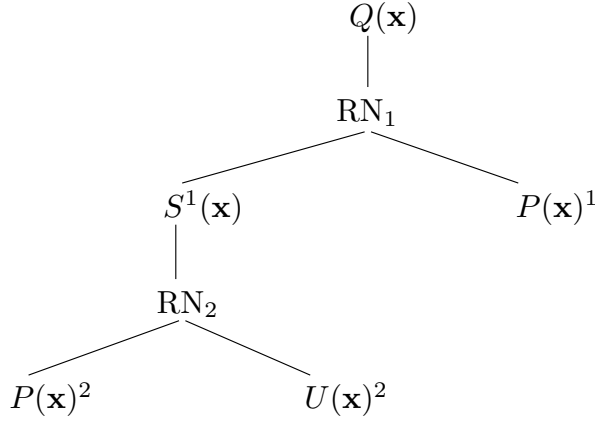


Figure 6.2.

We see that $P(\mathbf{x})^1$ can factor $P(\mathbf{x})^2$. However, if $P \in Def(\mathcal{L})$, we do not factor but press on and study the extensions of $P(\mathbf{x})^2$. As indicated above this results in \mathbf{x} being instantiated to some $\mathbf{x}\theta$ such that $P(\mathbf{x}\theta) \in \mathcal{M}_{Def(T)}$. But then the subtree generated below $P(\mathbf{x}\theta)^2$ can be used to solve the branch to $P(\mathbf{x}\theta)^1$, whence upon solving $P(\mathbf{x})^2$, we can in effect discard the branch to $P(\mathbf{x})^1$. We refer to this as *reverse factorisation* [Jo98].

6.2.1 Theorem. Let $(\mathcal{T}_i, N_i \mid 0 \leq i \leq n)$ be a deduction sequence in which we limit factorisation to be grounded (for non-definite atoms) or reverse (for definite atoms). If N_i a negative predicate node which is unpruned in \mathcal{T}_i , then $lab(\mathcal{T}_i, N_i)$ is ground.

Proof. Since factorisation is only applied with ground atoms, it follows from [Jo98a, Theorem 5.2.1] that if N_j is not redundant (in \mathcal{T}_{j^*}), then $lab(\mathcal{T}_{j^*}, N_j)$ is ground.

As in the proof of Theorem 5.2.1, if $lab(\mathcal{T}_i, N_i)$ is not ground, then we may find a left sibling N_k of N_i such that $lab(\mathcal{T}_i, N_k)$ is a non-ground positive atom. Thus N_k must be redundant, and since N_i is unpruned in \mathcal{T}_i , N_i must have been used to factor some branch through N_k . But then N_i must be ground by the given restriction on factorisation. ■

This also raises a further issue. If the subtree generated by extending a (say) positive definite goal $P(\mathbf{x})$ simply finds a ground instance of $P(\mathbf{x})$ in $\mathcal{M}_{Def(T)}$, then we could in fact adopt *any* method that achieves this end. In particular, the use of a bottom-up method would allow us to relinquish our constraints on $Def(T)$, in line with our aim stated in the introduction.

Similarly if $\neg P(\mathbf{a})$ is a negative definite goal, then we could adopt any method to show that $P(\mathbf{a}) \notin \mathcal{M}_{Def(T)}$ (ie., $P(\mathbf{a})$ is not true in $T/Def(T)$). Notice again here how important it is to have the negative subgoal grounded. If we were faced with an ungrounded subgoal of the form $\neg P(\mathbf{x})$, then there would typically be a huge number of instances $\mathbf{x}\theta$ for which $P(\mathbf{x}\theta) \notin \mathcal{M}_{Def(T)}$.

6.3 Pruning of cyclic trees

The computation of cyclic trees (below a negative predicate node) is one of the more costly aspects of our procedure. As indicated in Section 3, in a deductive database we would typically expect that the relative volume of indefinite and recursive information, and the number of levels in the database's stratification would not be large, which in turn would tend to limit the number (and size) of cyclic trees.

In addition, we can employ pruning as a means of limiting the number of such trees. Suppose that $N_{\neg B}$ is a negative predicate node with child $RN_{\rightarrow \neg B}$, and we are constructing a cyclic tree \mathcal{T} for B (in order to generate a child node of $RN_{\rightarrow \neg B}$). If P is a predicate node in \mathcal{T} such that $\neg P$ appears on $\text{ACT}(N_{\neg B})$, then \mathcal{T} can be pruned by allowing P to be a leaf in \mathcal{T} (cf., [Jo96, Theorem 7.4.2]).

Similarly, if P is a definite predicate node in \mathcal{T} , then we can insist that $P \in \mathcal{M}_{Def(\mathcal{T})}$, and again we can prune the tree at P .

§7. RELATED WORK

Semantics

Perfect models obviously extend minimal models [Gr86, He88] to the stratified case, where they are generally regarded as providing the most natural semantics. Whilst we believe that the vast majority of naturally occurring deductive databases are indeed stratified, there is currently much research interest into unstratified databases (and logic programs) to which the semantics defined by weakly perfect models [Pr90b], stable models [Ge88, Pr90, Pr94], stationary (otherwise known as partial or 3-valued stable) models [Pr94], and well-founded models [Ch93, Ch95, Pr89a, Pr90, Pr94, VanG91] apply. Each of these coincides with the perfect model semantics in the stratified case [Pr90b, Pr94, VanG91]. These concepts were extended to the disjunctive setting via the disjunctive stable semantics [Pr91], stationary semantics [Pr90a, Pr91a], and the static semantics [Pr95].

Query processing

For non-disjunctive programs, Przymusiński introduced SLS-resolution for the perfect model semantics [Pr89]. In [Ch93] Chen and Warren introduced XOLDTNF, a variant of SLS-resolution which employs loop checking to guarantee termination in the function free case, and these ideas were extended still further in SLG-resolution [Ch95] using subgoal dependencies to eliminate redundant computation.

For positive disjunctive databases both the deduction tree method and SLO-resolution [Lb92, Ra89, Ra89a, Ra92] are based upon hyperresolution. A comparison of these two methods is given in [Jo98, Section 10]. Solutions to the view update problem for deductive databases can be provided using deduction trees [Jo97] and cyclic covers [Jo98b].

In [Ya94, Fe95, Fe95a] the notion of a model-tree provides a bottom-up method of computing perfect models of ground databases. Top-down methods for testing perfect model membership are presented for first order databases in [Jo96] using cyclic trees and the relationship between cyclic sets and perfect models is explored further in [Jo98b]. To the best

of our knowledge, the methods of [Jo96, Jo97, Jo98, Jo98a, Jo98b] and those of the current paper represent the only top-down methods applicable to disjunctive databases under the perfect model semantics. The material in the current paper is adapted from the extended technical report [Jo98].

We know of no top-down query processing methods for disjunctive unstratified databases under either the stable, stationary or well-founded semantics. Of course, if such methods were to exist, then they would also be applicable to the perfect model semantics in the stratified case. However such a method would presumably be less efficient than a method specifically designed for the stratified case. As mentioned in the introduction, we see stratification as a compromise between representational flexibility and computational efficiency.

Pruning

Our pruning rule (introduced in Section 3.3) is an extension of a rule introduced in [Ho82] for Bibel's connection method [Bi87] and studied further in [Jo93]. The methods of [Ho82, Jo93] employ stack pointers rather than the ESS sets. Stack pointers have the advantage of simplicity, although they do not carry as much information as the ESS sets, and hence do not eliminate redundancy to the same effect, nor do they (as far as we know) yield termination.

§8. CONCLUSIONS

We have presented a complete and correct top down query answering method which is applicable to first order indefinite deductive databases under the semantics defined by perfect models. Our method is based upon a process of extending partial answers with potential new atoms, and then validating such atoms. The validation process is intending to limit the number of non-minimal answers generated by our procedure.

The generation of potential new atoms is based upon the top-down construction of cyclic trees. The validation process is based upon the top-down construction of extended deduction trees, and is a grounded process whilst processing the indefinite part of the intensional database, and linear otherwise. The entire querying process is finite, thus overcoming the main drawback of the query processing method presented in [Jo98a].

Problems which deserve further attention are the following.

1. How do the top-down methods presented in this paper compare with bottom-up methods for query answering? For databases more general than those considered in say Sections 3.3 and 5.2, can we adopt a combination of top-down and bottom-up processing of database rules? As mentioned above, this is possible when the processing enters the definite part of the database.
2. In order to generate all answers to a query $?Q(\mathbf{x})$ we need to employ some form of backtracking. This issue is not discussed in this paper.
3. Can the methods of this paper be extended to the unstratified case, using for instance the disjunctive stable semantics?
4. Let $?Q(\mathbf{x})$ be a query and $Q(\mathbf{a})$ be an instance of $Q(\mathbf{x})$. Under what conditions is

perfect model membership of $Q(\mathbf{a})$ sufficient to guarantee that $Q(\mathbf{a})$ belongs to some minimal answer of $?Q(\mathbf{x})$? [One obvious answer to this question is if each perfect model contains exactly one instance of $Q(\mathbf{x})$, this feature being commonly seen when functionally dependent attributes are involved in a query. Under these conditions our method would generate solely minimal answers.]

5. Is there a (reasonably) efficient method which computes minimal answers to queries without resorting to the use of subsumption? In this respect it should be noted that the problem of extracting answering (and in particular minimal answers) is computationally hard ([Jo96, Section 7]).

References

- [Bi87] W. Bibel, Automated Theorem Proving (Vieweg, Wiesbaden, 1987).
- [Ch93] W. Chen and D. S. Warren, A goal-oriented approach to computing the well-founded semantics, *Journal of Logic Programming*, vol. 17 (1993), 279-300.
- [Ch95] W. Chen, T. Swift and D. S. Warren, Efficient top-down computation of queries under the well-founded semantics, *Journal of Logic Programming*, vol. 24 (1995), 161-199.
- [Fe95a] J. Fernández and J. Minker, Computing perfect models of disjunctive stratified databases, *J. Logic Programming*, vol. 25 (1995), 33-51.
- [Fe95b] J. Fernández and J. Minker, Computing perfect and stable models using ordered model trees, *Computational Intelligence*, vol. 11 (1995), 89-112.
- [Ge88] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in : R. Kowalski and K. Bowen (eds.), *Proc. 5th Int'l Conference on Logic Programming*, Seattle (1988), 1070-1080.
- [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.
- [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).
- [Ho82] K. Hórning and W. Bibel, Improvements of a tautology testing algorithm, in: D. Loveland (Ed.), *Proceedings of the 6th Conference on Automated Deduction*, *Lecture Notes in Computer Science*, vol. 138 (Springer, Berlin, 1982), 326-341.
- [Jo93] C. A. Johnson, Factorisation and circuit in the connection method, *J. ACM*, vol. 40 (1993), 536-557.
- [Jo96] C. A. Johnson, On computing minimal and perfect model membership, *Data and Knowledge Engineering*, vol. 18 (1996), 225-276.
- [Jo97] C. A. Johnson, Deduction trees and the view update problem in indefinite deductive databases, *J. Automated Reasoning*, vol. 19 (1997), 31-85.
- [Jo98] C. A. Johnson, Extended deduction trees and query processing, Keele University Computer Science technical report TR98-07, available from <http://www.keele.ac.uk/depts/cs/cshome.html>
- [Jo98a] C. A. Johnson, Top down query processing in indefinite deductive databases, *Data and Knowledge Engineering*, vol. 26 (1998), 1-36.

- [Jo98b] C. A. Johnson, On cyclic covers and perfect models, submitted, Data and Knowledge Engineering. An extended technical report is available from <http://www.keele.ac.uk/depts/cs/cshome.html>
- [Lb92] J. Lobo, J. Minker, and A. Rajasekar, Foundations of Disjunctive Logic Programming, (MIT Press, Cambridge, Massachusetts, 1992).
- [Pr88] T. Przymusinski, On the declarative semantics of deductive databases and logic programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, (Morgan Kaufman, Washington, 1988).
- [Pr89] T. C. Przymusinski, On the declarative and procedural semantics of logic programs, J. Automated Reasoning, vol. 5 (1989), 167-205.
- [Pr89a] T. C. Przymusinski, An algorithm to compute circumscription, Artificial intelligence, vol. 38 (1989), 49-73.
- [Pr90] T. C. Przymusinski, The well-founded semantics coincides with the three-valued stable semantics, Fundamenta Informaticae, vol. 13 (1990), 445-464.
- [Pr90a] T. C. Przymusinski, Stationary semantics for disjunctive logic programs and deductive databases, in : S Debray and M Hermenegildo (eds), Proc. North American Logic Programming Conference, Austin, Texas (MIT Press, 1990), 40-59.
- [Pr90b] T. C. Przymusinski and H. Przymusinska, Weakly stratified logic programs, Fundamenta Informaticae, vol. 13 (1990), 51-65.
- [Pr91] T. C. Przymusinski, Stable semantics for disjunctive programs, New Generation Computing, vol. 9 (1991), 401-424.
- [Pr91a] T. C. Przymusinski, Semantics of disjunctive logic programs and deductive databases, in Proc. 2nd Int'l Conf. on Deductive and Object-Oriented Databases, DOOD'91, Munich, (Springer, 1991).
- [Pr94] T. C. Przymusinski, Well-founded and stationary models of logic programs, Annals of Mathematics and Artificial Intelligence, vol. 12 (1994), 141-187.
- [Pr95] T. C. Przymusinski, Static semantics for normal and disjunctive logic programs, Annals of Mathematics and Artificial Intelligence, vol. 14 (1995), 323-357.
- [Ra89] A. Rajasekar, Semantics for Disjunctive Logic Programs, PhD thesis, University of Maryland (1989).
- [Ra89a] A. Rajasekar, J. Lobo and J. Minker, in: Skeptical reasoning and disjunctive programs, in: Proceedings of the First International Conference on Knowledge Representation and Reasoning (1989), 349-357.
- [Ra92] A. Rajasekar and H. Yusuf, DWAM - A WAM model extension of disjunctive logic programs, in: Symposium on Logic in Databases, Knowledge Representation and Reasoning, University of Maryland.
- [VanG91] A. van Gelder, K. Ross and J. Schlipf, The well-founded semantics for general logic programs, J. Assoc. for Computing Machinery, vol. 38 (1991), 620-650.
- [Ya94] A. Yahya, J Fernández and J. Minker, Ordered model trees : A normal form for disjunctive deductive databases, J. Automated Reasoning, vol. 13 (1994), 117-143.