

# Query compilation under the disjunctive well founded semantics

C. A. Johnson  
School of Computing  
University of Plymouth  
Plymouth, PL4 8AA  
England  
email : chrisj@soc.plymouth.ac.uk

**Abstract.** A top-down method is presented for compiling queries in propositional disjunctive deductive databases under the disjunctive well founded semantics. Compilation entails the construction of a set of goal trees, and the run-time processing of the compiled query then amounts to checking that some such tree can be extended within the extensional database to yield a tree that encapsulates a proof of the original query.

*Keywords.* Disjunctive deductive databases, disjunctive well-founded semantics, query compilation, top-down computation.

## Introduction.

Over the last few years there has been a great deal of interest in the study of unstratified deductive databases and logic programs, resulting in declarative semantics such as the disjunctive stable model semantics [20], the disjunctive well-founded semantics (DWFS) [4],  $WF^3$  [2], GDWFS [3] and others. Surveys of these semantics are to be found in [9, 19].

DWFS was introduced by Brass and Dix [1,4-8] as the weakest semantics which satisfies certain desirable properties, including the generalised principle of partial evaluation. DWFS itself has a complex definition making the development of associated methods somewhat more difficult than under other semantics. In [17,18] we presented a relatively simple characterisation of the DWFS in terms of the Gelfond-Lifschitz transformation, and using this presented a top-down procedure for query processing under the DWFS. In this paper we wish to attack a related problem - that of query compilation.

Query compilation is based upon two assumptions. Firstly that query processing itself can be computationally expensive - under DWFS it is  $\Pi_2^P$ -complete [10] for propositional programs. Secondly, that a deductive database is usually subdivided into an extensional database (containing facts and/or disjunctive facts) and an intensional database (the rule base) from which additional facts may be deduced. The intensional rule base is assumed to be much smaller and far more static than the extensional database, and this suggests that a commonly executed query might be pre-processed using

the intensional database to produce a transformed query that can then be answered more efficiently at run-time. The transformed query would remain valid unless the intensional database were itself modified, whence the pre-processing stage would need to be repeated. In the case when the transformed query can be processed against the extensional database only, it is known as a *compilation* of the original query. Notice that compilation is dependent upon a segregation into an extension and an intension, which is perhaps more suggestive of a deductive database than an arbitrary logic program, and as result we will refer henceforth to the former.

Compilation has been discussed under a number of other semantics, for example the GCWA [12], the perfect model semantics [15], and the disjunctive stable model semantics [16]. As suggested earlier, we will see that compilation under the DWFS is somewhat more difficult, and we will trace this difficulty through to the non-confluence of (our) top-down reasoning in DWFS.

In Section 2 we commence with a re-statement of our characterisation of the DWFS in terms of the Gelfond-Lifschitz transformation, and in Section 3 consider how we can adapt our minimal model reasoning techniques to the case when processing is restricted to the intensional database. Section 4 considers the analogous problem when handling positive disjunctive goals. In Section 5 we present the concept of a compilation tree which encapsulates our compilation process, and also defines the requirements for run-time processing of the compiled query.

A (first order) deductive database represents the set of its ground instances, and thus throughout this paper we work at the propositional level, thus enabling us to concentrate on the issues required for compilation without the distractions of unification. We comment further on the issue of the first order level in Section 6.

## §1. Preliminaries

Throughout  $\mathcal{L} = \{P_0, P_1, \dots\}$  denotes a finite propositional language, and we will assume that  $\mathcal{L}$  is a disjoint union  $\mathcal{L} = \text{EXT}(\mathcal{L}) \cup \text{INT}(\mathcal{L})$ .  $\text{EXT}(\mathcal{L})$  denotes the set of *extensional* predicates;  $\text{INT}(\mathcal{L})$  the *intensional* predicates.

$P, Q, \dots$  etc., will denote arbitrary predicates in  $\mathcal{L}$ , and  $\mathcal{P}, \mathcal{Q}, \dots$  will denote subsets of  $\mathcal{L}$ . A *literal* is a predicate or its negation. If  $\mathcal{I}$  is a set of literals, then  $\overline{\mathcal{I}} = \{\neg K : K \in \mathcal{I}\}$ . A rule is a formula  $C$  of the form  $A_1 \wedge A_2 \wedge \dots \wedge A_n \wedge \neg A_{n+1} \wedge \neg A_{n+2} \wedge \dots \wedge \neg A_{n+h} \rightarrow B_1 \vee B_2 \vee \dots \vee B_r$ , where each  $A_i, B_j \in \mathcal{L}$  and  $r > 0$ .  $\text{antec}(C) = \{A_1, A_2, \dots, A_n\}$ ,  $\mathcal{N}(C) = \{A_{n+1}, A_{n+2}, \dots, A_{n+h}\}$  and  $\text{conseq}(C) = \{B_1, B_2, \dots, B_r\}$ . As is usual in research into deductive databases, we assume that if  $\text{conseq}(C) \cap \text{INT}(\mathcal{L}) \neq \emptyset$ , then  $\text{conseq}(C) \subseteq \text{INT}(\mathcal{L})$  and  $\text{antec}(C) \neq \emptyset$ , and that if  $\text{conseq}(C) \cap \text{EXT}(\mathcal{L}) \neq \emptyset$ , then  $\text{conseq}(C) \subseteq \text{EXT}(\mathcal{L})$  and  $\mathcal{N}(C) = \text{antec}(C) = \emptyset$ .

A (deductive) database is a set of rules, which throughout will be denoted by  $T$ .  $T$  is said to be *positive* iff  $\mathcal{N}(C) = \emptyset$  for each  $C \in T$ . The *intensional database* is denoted by  $\text{INT}(T) = \{C \in T : \text{conseq}(C) \subseteq \text{INT}(\mathcal{L})\}$ , and the *extensional database* as  $\text{EXT}(T) = \{C \in T : \text{conseq}(C) \subseteq \text{EXT}(\mathcal{L})\}$ . Note that if  $C \in \text{EXT}(T)$ , then  $C = \bigvee \text{conseq}(C)$ , and in this case we will use  $C$  interchangeably with the set of

predicates  $\text{conseq}(C)$ .

**1.1 Definition.** Suppose that we have a set of literals  $\mathcal{A}$  such that  $\mathcal{E} = \{P \in \mathcal{L} : \neg P \in \mathcal{A}\} \subseteq \text{EXT}(\mathcal{L})$ . A *completion function* for  $\mathcal{A}$  (in  $\text{EXT}(T)$ ) is a function  $f : \mathcal{E} \rightarrow \text{EXT}(T)$  such that for each  $P \in \mathcal{E}$ ,  $f(P) \cap \mathcal{E} = \{P\}$ . We then define  $\mathcal{A}_f = (\mathcal{A} \cap \mathcal{L}) \cup \bigcup \{f(P) - \{P\} : P \in \mathcal{E}\}$ , and we refer to  $\mathcal{A}_f$  as a *completion* of  $\mathcal{A}$  (in  $\text{EXT}(T)$ ).

So for example if  $\mathcal{A} = \{\neg E_0, \neg E_1, E_2, P, Q\}$  and  $\text{EXT}(T) = \{E_0 \vee E_1, E_0 \vee E_2 \vee E_3, E_1 \vee E_4\}$ , then the only completion of  $\mathcal{A}$  in  $\text{EXT}(T)$  is  $\{E_2, E_3, E_4, P, Q\}$ .

Clearly if  $\mathcal{A} \subseteq \mathcal{L}$ , then  $\mathcal{A}$  is its own completion (and has no other completion). Note also that if  $R \in \mathcal{L}$ , then  $\mathcal{A}$  and  $\mathcal{A} \cup \{R\}$  have the same completion functions, and  $(\mathcal{A} \cup \{R\})_f = \mathcal{A}_f \cup \{R\}$ . Finally note that if  $\mathcal{A}$  is inconsistent (say  $\{P, \neg P\} \in \mathcal{A}$ ), then  $f(P) \subseteq \mathcal{A}_f$ .

## §2. DWFS

In [7], a definition of DWFS was given in terms of an increasing sequence  $\emptyset = D_0 \subseteq D_1 \subseteq D_2 \subseteq \dots$ , where  $\text{DWFS} = \bigcup_{\alpha} D_{\alpha}$ . Each  $D_{\alpha+1}$  is constructed as  $D_{\alpha+1} = D_{\alpha+1}^+ \cup D_{\alpha+1}^-$  and is defined in terms of  $T/D_{\alpha}$  where  $/$  is a reduction operator. In [18] we argued that to achieve a top-down approach to query processing under the DWFS (as is required for compilation) it is essential to express each of the iterative steps that constructs DWFS more directly in terms of  $T$  and  $D_{\alpha}$ . This also gives a simpler characterisation which is presented in Theorem 2.2 below.

**2.1 Definition** [11]. If  $C$  is a rule, then  $\text{pos}(C) = \bigwedge \text{antec}(C) \rightarrow \bigvee \text{conseq}(C)$ .

If  $\mathcal{K} \subseteq \mathcal{L}$ , then the *Gelfond-Lifschitz transformation* is given by

$$T|_g(\mathcal{L} - \mathcal{K}) = \{\text{pos}(C) : C \in T, \mathcal{N}(C) \subseteq \mathcal{K}\}.$$

Notice that  $T|_g(\mathcal{L} - \mathcal{K})$  is positive, and can thus be interpreted naturally using the minimal model semantics.

**2.2 Theorem** [18].  $D_{\alpha+1}^+ = \{\bigvee \mathcal{P} : \mathcal{P} \subseteq \mathcal{L}, T|_g(\mathcal{L} - \overline{D_{\alpha}^-}) \models \bigvee \mathcal{P}\}$ , and  $D_{\alpha+1}^- = \{\neg Q : Q \in \mathcal{L}, (\forall N \subseteq \mathcal{L} - \overline{D_{\alpha}^-})(N \models D_{\alpha+1}^+ \implies T|_g N \models_{\text{min}} \neg Q)\}$ .

Recall that  $\overline{D_{\alpha}^-} = \{Q : \neg Q \in D_{\alpha}^-\}$ .  $\models_{\text{min}}$  refers to minimal model reasoning, thus  $T|_g N \models_{\text{min}} \neg Q$  iff  $Q$  belongs to no minimal model of  $T|_g N$ .

### §3. Handling minimal model reasoning.

In [17,18] we presented a top-down query processing method under the DWFS, using (unfactored) quasi cyclic trees to handle minimal model reasoning. In the present context, we wish to pre-process a query using  $\text{INT}(T)$  only, whence we make the following definition which provides an appropriate variant of UQC trees. The motivation behind the details of this definition are presented in [13-15], and are also summarised in the comments and examples that follow.

**3.1 Definition** [17,18]. Let  $\mathcal{T}$  be a finite tree containing predicate nodes and rule nodes satisfying the following conditions.

- (i) The root node (at the top of  $\mathcal{T}$ ) is a predicate node.
- (ii) If  $n$  is a predicate node then  $n$  is labelled with a predicate, denoted by  $\text{lab}(n)$ . If  $n$  is not a leaf node, then  $n$  has a single child node which is a rule node. If  $n$  is a leaf node, then  $\text{lab}(n) \in \text{EXT}(\mathcal{L})$ . If  $n$  is not the root node, then its parent node is a rule node.
- (iii) If  $rn$  is a rule node, then  $rn$  is labelled with a rule  $C \in T$  (written  $rn_C$ ). For each  $K \in \text{antec}(C)$ ,  $rn_C$  has a (predicate) child node labelled with  $K$  (and these are its only child nodes).

Then we make the following definitions.

- (1) Suppose that  $n$  is a predicate node in  $\mathcal{T}$ . Let

$$\text{CYC}(n) = \{\text{lab}(n') \mid n' \text{ is a predicate node, } n' \geq n, \text{ and} \\ \exists n'' \geq n', n'' \text{ is a predicate node, } \text{lab}(n'') = \text{lab}(n)\}.$$

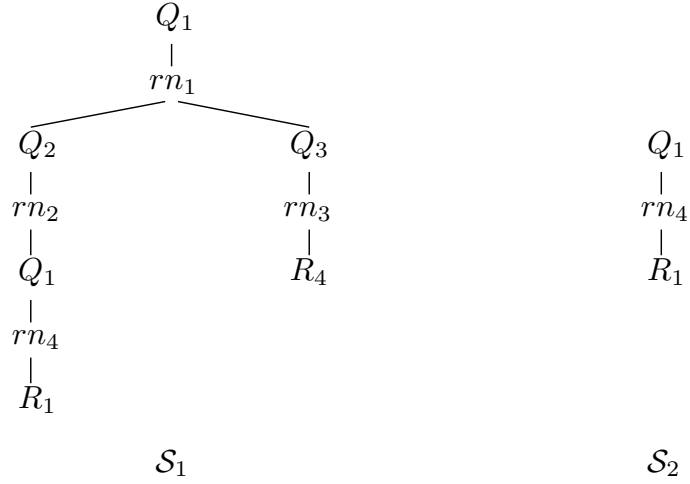
If the child node of  $n$  is  $rn_C$ , then define  $\mathcal{O}(rn_C) = \text{conseq}(C) - \text{CYC}(n)$ .

- (2) Let  $\mathcal{O}(\mathcal{T}) = \bigcup\{\mathcal{O}(rn_C) \mid rn_C \text{ is a rule node in } \mathcal{T}\}$ ,  
 $\mathcal{N}(\mathcal{T}) = \bigcup\{\mathcal{N}(C) \mid rn_C \text{ is a rule node in } \mathcal{T}\}$ , and  
 $\text{Pred}(\mathcal{T}) = \{\text{lab}(n) \mid n \text{ is a predicate node in } \mathcal{T}\}$ .
- (3) Let  $P \in \mathcal{L}$ , then  $\mathcal{T}$  is said to be a *weak quasi cyclic* (WQC) tree for  $P$  in  $T$  iff
  - (a) The root node is labelled with  $P$ ,
  - (b)  $\text{Pred}(\mathcal{T}) \cap \mathcal{O}(\mathcal{T}) = \emptyset$ ,
  - (c) whenever  $rn_C$  is a rule node in  $\mathcal{T}$  with parent  $n$ , then
    - (i)  $\text{conseq}(C) \cap \text{CYC}(n) \neq \emptyset$ , and
    - (ii)  $\text{antec}(C) \cap \text{CYC}(n) = \emptyset$ .

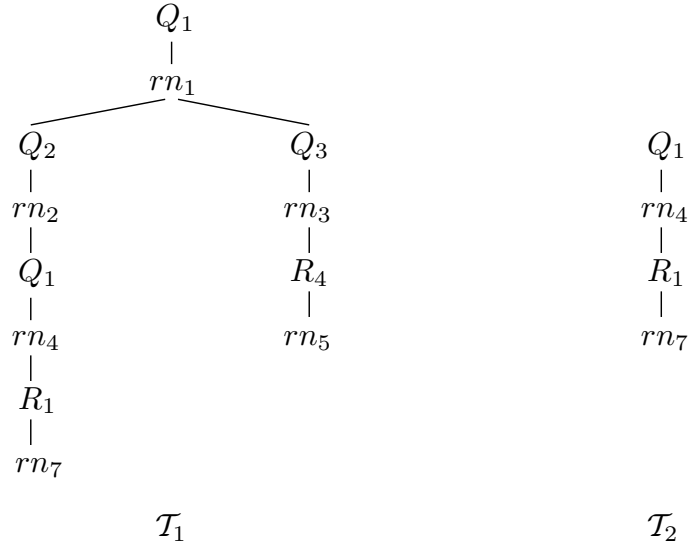
**3.2 Example.** Suppose that  $T$  consists of the following rules

- |  |  |  |
|--|--|--|
| 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. $R_4 \wedge \neg R_3 \rightarrow Q_3$ |
| 4. $R_1 \rightarrow Q_1 \vee Q_2 \vee Q_4$                   | 5. $R_4 \vee R_5$                        | 6. $R_6 \rightarrow Q_3 \vee Q_2$        |
| 7. $R_1 \vee R_7$  | 8. $Q_5 \rightarrow Q_2$                 |  |

where  $\text{EXT}(T) = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7\}$ . Then the only WQC trees for  $Q_1$  in  $T$  are depicted in Figure 3.2(i) and Figure 3.2(ii) below.



**Figure 3.2(i).**



**Figure 3.2(ii).**

Note that the only predicate node  $n$  for which  $\text{CYC}(n) \neq \{\text{lab}(n)\}$  is the lower instance of  $Q_1$  on the left hand branch of  $\mathcal{S}_1$  and  $\mathcal{T}_1$  (with  $\text{CYC}(n) = \{Q_1, Q_2\}$ ).

$\text{Pred}(\mathcal{T}_1) = \text{Pred}(\mathcal{S}_1) = \{Q_1, Q_2, Q_3, R_1, R_4\}$ ,  $\mathcal{N}(\mathcal{T}_1) = \mathcal{N}(\mathcal{S}_1) = \{R_1, R_2, R_3\}$ ,  $\mathcal{O}(\mathcal{T}_1) = \{Q_5, Q_4, R_7, R_5\}$ , and  $\mathcal{O}(\mathcal{S}_1) = \{Q_5, Q_4\}$ .

Similarly,  $\text{Pred}(\mathcal{T}_2) = \text{Pred}(\mathcal{S}_2) = \{Q_1, R_1\}$ ,  $\mathcal{N}(\mathcal{T}_2) = \mathcal{N}(\mathcal{S}_2) = \emptyset$ ,  $\mathcal{O}(\mathcal{T}_2) = \{Q_2, Q_4, R_7\}$ , and  $\mathcal{O}(\mathcal{S}_2) = \{Q_2, Q_4\}$ .

Notice that in each tree, if  $n$  is a predicate node with  $\text{lab}(n) \in \text{EXT}(\mathcal{L})$ , then either  $n$  is a leaf, or  $n$  has a child of the form  $rn_C$ , where  $C \in \text{EXT}(T)$ ,  $\text{lab}(n) \in C$ , and

$rn_C$  is a leaf (since  $\text{antec}(C) = \emptyset$ ) with  $\mathcal{O}(rn_C) = C - \{lab(n)\}$ . Note also that if  $lab(n) \in \text{INT}(\mathcal{L})$ , then  $n$  has a child node of the form  $rn_C$ , where  $C \in \text{INT}(T)$  (whence  $rn_C$  is not a leaf since  $\text{antec}(C) \neq \emptyset$ , cf., Section 1).

### 3.3 Definition.

- (a) A *semi-factored quasi cyclic* (SQC) tree in  $T$  is a WQC tree in  $\text{INT}(T)$ .
- (b) A WQC tree  $\mathcal{T}$  is said to be an *unfactored quasi cyclic* (UQC) tree iff  $\mathcal{T}$  has no predicate leaf node.

If  $\mathcal{S}$  is an SQC tree, then clearly  $\mathcal{O}(\mathcal{T}) \subseteq \text{INT}(\mathcal{L})$ , and for each node  $n$ ,  $n$  is a leaf iff  $n$  is a predicate node and  $lab(n) \in \text{EXT}(\mathcal{L})$ . We define  $\mathcal{R}(\mathcal{S}) = \mathcal{O}(\mathcal{S}) \cup \{\neg lab(n) : n \text{ is a predicate leaf node in } \mathcal{S}\}$ . Note that  $\mathcal{R}(\mathcal{S})$  is of the form given in Definition 1.1. If  $f$  is a completion function for  $\mathcal{R}(\mathcal{S})$ , then we define  $\mathcal{S}_f$  to be the tree obtained by extending each leaf node  $n$  with a rule node labelled with  $f(lab(n))$ . We refer to  $\mathcal{S}_f$  as a *completion* of  $\mathcal{S}$ .

So in Example 3.2,  $\mathcal{S}_1$  and  $\mathcal{S}_2$  are the only SQC trees for  $Q_1$  in  $T$ , and  $\mathcal{T}_1$  and  $\mathcal{T}_2$  are the only UQC trees for  $Q_1$  in  $T$ .  $\mathcal{R}(\mathcal{S}_1) = \{\neg R_1, \neg R_4, Q_5, Q_4\}$ , and  $\mathcal{R}(\mathcal{S}_2) = \{\neg R_1, Q_2, Q_4\}$ .

The example above illustrates that UQC trees can only be formed by extending SQC trees. This then gives us a two-phase means of constructing UQC trees: firstly by constructing SQC trees (using  $\text{INT}(T)$  only) and then extending these to UQC trees (using  $\text{EXT}(T)$  only). This two-phase approach is exactly what is required for compilation.

The motivation for building WQC trees for  $P$  in  $T$  is to establish whether  $P$  belongs to some minimal model of  $T|_g(\mathcal{L} - \mathcal{K})$  (for some  $\mathcal{K}$ ). If  $M$  is such a model, then we can iteratively build a WQC tree for  $P$  in  $T$  such that each predicate (node) in the tree belongs to  $M$  (i.e.  $\text{Pred}(\mathcal{T}) \subseteq M$ ), and (using the minimality of  $M$ ) in which each predicate node  $n$  is extended via a rule  $C$  which would witness that  $M - \text{CYC}(n) \not\models T|_g(\mathcal{L} - \mathcal{K})$ . This requires that  $\mathcal{N}(C) \subseteq \mathcal{K}$ ,  $\text{antec}(C) \subseteq M - \text{CYC}(n)$  (thus yielding condition (c)(ii) from Definition 3.1, and again requiring that all predicate nodes in the tree belong to  $M$ ), and  $(M - \text{CYC}(n)) \cap \text{conseq}(C) = \emptyset$  (whence  $M \cap \mathcal{O}(rn_C) = M \cap (\text{conseq}(C) - \text{CYC}(n)) = \emptyset$ , thus yielding condition (b) from Definition 3.1). Since  $M \models T|_g(\mathcal{L} - \mathcal{K})$ , we would then have that  $M \cap \text{conseq}(C) \neq \emptyset$ , whence  $\text{conseq}(C) \cap \text{CYC}(n) \neq \emptyset$  (thus yielding condition (c)(i) from Definition 3.1).

The reason for choosing to witness  $\text{CYC}(n)$  (rather than some other subset of the branch above  $n$ ) is that doing so then allows to us to prove the following properties [13,18].

- (i) If  $M$  is a minimal model of  $T|_g(\mathcal{L} - \mathcal{K})$  and  $P \in M$ , then we may find a UQC tree  $\mathcal{T}$  for  $P$  in  $T$  such that  $\text{Pred}(\mathcal{T}) \subseteq M \subseteq \mathcal{L} - \mathcal{O}(\mathcal{T})$  and  $\mathcal{N}(\mathcal{T}) \subseteq \mathcal{K}$ , and
- (ii) if  $\mathcal{T}$  is a UQC tree in  $T$ ,  $\mathcal{N}(\mathcal{T}) \subseteq \mathcal{K}$ , and  $M \models T|_g(\mathcal{L} - \mathcal{K})$  with  $M \cap \mathcal{O}(\mathcal{T}) = \emptyset$ , then  $\text{Pred}(\mathcal{T}) \subseteq M$ .

**3.4 Proposition.** If  $\mathcal{S}$  is an SQC tree, then any completion  $\mathcal{S}_f$  of  $\mathcal{S}$  is a UQC tree, with

$Pred(\mathcal{S}_f) = Pred(\mathcal{S})$ ,  $\mathcal{N}(\mathcal{S}_f) = \mathcal{N}(\mathcal{S})$ ,  $\mathcal{O}(\mathcal{S}_f) = \mathcal{R}(\mathcal{S})_f$ , and  $\mathcal{O}(\mathcal{S}_f) \cap \text{INT}(\mathcal{L}) = \mathcal{O}(\mathcal{S})$ .

Conversely any UQC tree is “subsumed” by the completion of some SQC tree.

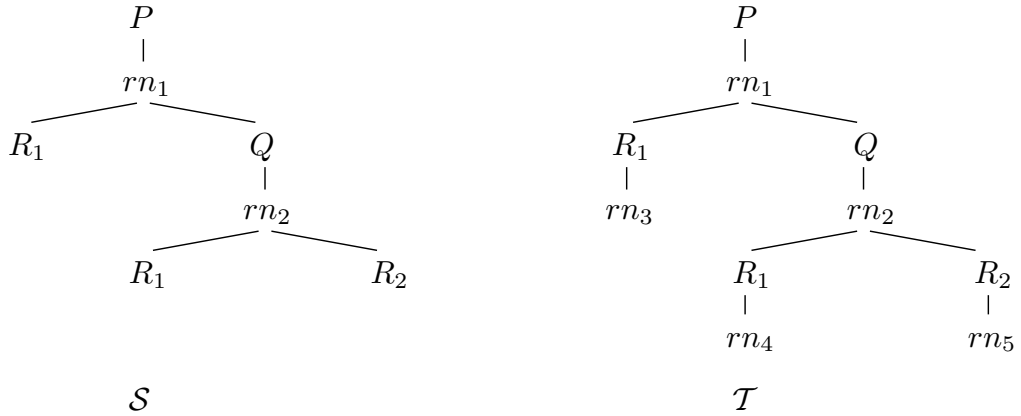
**3.5 Proposition.** Let  $\mathcal{T}$  be a UQC tree, and  $\mathcal{S}$  be the tree produced by removing the leaf nodes. Then  $\mathcal{S}$  is an SQC tree, and there is a completion  $\mathcal{S}_f$  of  $\mathcal{S}$  such that  $Pred(\mathcal{S}) = Pred(\mathcal{S}_f) = Pred(\mathcal{T})$ ,  $\mathcal{N}(\mathcal{S}) = \mathcal{N}(\mathcal{S}_f) = \mathcal{N}(\mathcal{T})$ , and  $\mathcal{O}(\mathcal{S}) = \mathcal{O}(\mathcal{T}) \cap \text{INT}(\mathcal{L}) \subseteq \mathcal{O}(\mathcal{S}_f) = \mathcal{R}(\mathcal{S})_f \subseteq \mathcal{O}(\mathcal{T})$ .

Note that we cannot guarantee that  $\mathcal{R}(\mathcal{S})_f = \mathcal{O}(\mathcal{T})$ , as is illustrated by the following example.

**3.6 Example.** Suppose that  $\mathcal{T}$  contains the following rules.

1.  $R_1 \wedge Q \rightarrow P$
2.  $R_1 \wedge R_2 \rightarrow Q$
3.  $R_1 \vee R_3$
4.  $R_1 \vee R_4$
5.  $R_2$

with  $\text{EXT}(\mathcal{L}) = \{R_1, R_2, R_3, R_4\}$ , then there is precisely one SQC tree for  $P$  in  $\mathcal{T}$  depicted as  $\mathcal{S}$  in Figure 3.6. The completions of  $\mathcal{R}(\mathcal{S})$  are  $\{R_3\}$  and  $\{R_4\}$ , none of which match  $\mathcal{O}(\mathcal{T}) = \{R_3, R_4\}$ .



**Figure 3.6.**

Our compilation process will require us to express DWFS membership in terms of SQC trees. Firstly we recall the following theorem detailing the relationship to UQC trees.

**3.7 Theorem** [18].  $\neg Q \in D_{\alpha+1}^-$  iff for each UQC tree  $\mathcal{T}$  for  $Q$  in  $\mathcal{T}$ , either

- (i)  $\bigvee \mathcal{N}(\mathcal{T}) \in D_{\alpha+1}^+$ , or
- (ii)  $T|_g(\mathcal{L} - \mathcal{N}(\mathcal{T}) \cup \overline{D_\alpha^-}) \models \bigvee \mathcal{O}(\mathcal{T})$ .

Propositions 3.4 and 3.5 then yield the following result.

**3.8 Corollary.**  $\neg Q \in D_{\alpha+1}^-$  iff for each SQC tree  $\mathcal{S}$  for  $Q$  in  $T$ , either

- (i)  $\bigvee \mathcal{N}(\mathcal{S}) \in D_{\alpha+1}^+$ , or
- (ii) for each completion  $\mathcal{R}(\mathcal{S})_f$  of  $\mathcal{R}(\mathcal{S})$ ,  $T|_g(\mathcal{L} - \mathcal{N}(\mathcal{S}) \cup \overline{D_\alpha^-}) \models \bigvee \mathcal{R}(\mathcal{S})_f$ .

**Proof.** Suppose that  $\neg Q \in D_{\alpha+1}^-$ , and that  $\mathcal{S}$  is an SQC tree for  $Q$  in  $T$ . Let  $f$  be a completion function for  $\mathcal{R}(\mathcal{S})$ , then by Proposition 3.4 and Theorem 3.7,  $\bigvee \mathcal{N}(\mathcal{S}_f) = \bigvee \mathcal{N}(\mathcal{S}) \in D_{\alpha+1}^+$  or  $T|_g(\mathcal{L} - \mathcal{N}(\mathcal{S}_f) \cup \overline{D_\alpha^-}) = T|_g(\mathcal{L} - \mathcal{N}(\mathcal{S}) \cup \overline{D_\alpha^-}) \models \bigvee \mathcal{O}(\mathcal{S}_f)$ , where  $\mathcal{O}(\mathcal{S}_f) = \mathcal{R}(\mathcal{S})_f$ .

Suppose that the right hand side is true, and that  $\mathcal{T}$  is a UQC tree for  $Q$  in  $T$ . Using Proposition 3.5, pick an SQC tree  $\mathcal{S}$  and a completion  $\mathcal{S}_f$  of  $\mathcal{S}$  which subsumes  $\mathcal{T}$  in the sense given in Proposition 3.5. If  $\bigvee \mathcal{N}(\mathcal{T}) = \bigvee \mathcal{N}(\mathcal{S}) \notin D_{\alpha+1}^+$ , then by hypothesis we have that  $T|_g(\mathcal{L} - \mathcal{N}(\mathcal{T}) \cup \overline{D_\alpha^-}) = T|_g(\mathcal{L} - \mathcal{N}(\mathcal{S}) \cup \overline{D_\alpha^-}) \models \bigvee \mathcal{R}(\mathcal{S})_f$ , where  $\mathcal{R}(\mathcal{S})_f \subseteq \mathcal{O}(\mathcal{T})$ . The result then follows from Theorem 3.7. QED.

The following definition encapsulates the property identified in condition (ii) above.

**3.9 Definition.** Suppose that  $\mathcal{A}$  is a set of literals of the form given in Definition 1.1, then we write  $T|_g(\mathcal{L} - \mathcal{K}) \models^* \bigvee \mathcal{A}$  iff for each completion  $\mathcal{A}_f$  of  $\mathcal{A}$  in  $\text{EXT}(T)$ ,  $T|_g(\mathcal{L} - \mathcal{K}) \models \bigvee \mathcal{A}_f$ .

Since a subset of  $\mathcal{L}$  has only itself as a completion, the following proposition follows trivially from Theorem 2.2.

**3.10 Proposition.**  $\bigvee \mathcal{P} \in D_{\alpha+1}^+$  iff  $T|_g(\mathcal{L} - \overline{D_\alpha^-}) \models^* \bigvee \mathcal{P}$ .

**3.11 Corollary.**  $\neg Q \in D_{\alpha+1}^-$  iff for each SQC tree  $\mathcal{S}$  for  $Q$  in  $T$ , either

- (i)  $T|_g(\mathcal{L} - \overline{D_\alpha^-}) \models^* \bigvee \mathcal{N}(\mathcal{S})$ , or
- (ii)  $T|_g(\mathcal{L} - \mathcal{N}(\mathcal{S}) \cup \overline{D_\alpha^-}) \models^* \bigvee \mathcal{R}(\mathcal{S})$ .

#### §4. Handling positive subgoals.

The following trivial theorem, amended from [13,14], presents our basic mechanism for handling positive disjunctive goals in positive databases.

**4.1 Theorem.** Let  $\mathcal{P}, \mathcal{K} \subseteq \mathcal{L}$ .

- (a) If  $T|_g(\mathcal{L} - \mathcal{K}) \models \bigvee \mathcal{P}$ , then we may find a  $C \in T$  such that  $\text{conseq}(C) \subseteq \mathcal{P}$ ,

$\text{antec}(C) \cap \mathcal{P} = \emptyset$  and  $\mathcal{N}(C) \subseteq \mathcal{K}$ .

(b) If  $C \in T$  with  $\text{conseq}(C) \subseteq \mathcal{P}$  and  $\mathcal{N}(C) \subseteq \mathcal{K}$ , then  $T|_g(\mathcal{L} - \mathcal{K}) \models \bigvee \mathcal{P}$  iff for each  $Q \in \text{antec}(C)$ ,  $T|_g(\mathcal{L} - \mathcal{K}) \models Q \vee \bigvee \mathcal{P}$ .

Of course property (b) is trivial if  $\text{antec}(C) \cap \mathcal{P} \neq \emptyset$ . The following theorem extends Theorem 4.1 to cover  $\models^*$ .

**4.2 Theorem.** Suppose that  $\mathcal{K} \subseteq \mathcal{L}$  and  $\mathcal{A}$  is a set of literals of the form given in Definition 1.1,  $C \in \text{INT}(T)$ ,  $\text{conseq}(C) \subseteq \mathcal{A}$ , and  $\mathcal{N}(C) \subseteq \mathcal{K}$ . Then the following are equivalent.

- (i)  $T|_g(\mathcal{L} - \mathcal{K}) \models^* \bigvee \mathcal{A}$ ,
- (ii) for each  $Q \in \text{antec}(C)$ ,  $T|_g(\mathcal{L} - \mathcal{K}) \models^* Q \vee \bigvee \mathcal{A}$ .

**Proof.** Since every completion of  $\{Q\} \cup \mathcal{A}$  contains a completion of  $\mathcal{A}$ , (i)  $\rightarrow$  (ii) is trivial.

For the converse, suppose that  $\mathcal{A}_f$  is a completion of  $\mathcal{A}$ , then for each  $Q \in \text{antec}(C)$ ,  $\{Q\} \cup \mathcal{A}_f$  is a completion of  $\{Q\} \cup \mathcal{A}$ , whence  $T|_g(\mathcal{L} - \mathcal{K}) \models Q \vee \bigvee \mathcal{A}_f$ . Since  $\text{conseq}(C) \subseteq \mathcal{A}_f$ , the result then follows from Theorem 4.1(b). QED.

Again it is clear that there is no point attacking  $\mathcal{A}$  using a rule  $C$  for which  $\text{antec}(C) \cap \mathcal{A} \neq \emptyset$ .

Theorem 4.2 indicates that we can attack each of the completions of  $\mathcal{A}$  using  $C$  simultaneously. Moreover it is easy to see that there is nothing to be gained from using rules in  $\text{INT}(T)$  to attack completions individually, since if  $\mathcal{A}_f$  is a completion of  $\mathcal{A}$ , then  $\mathcal{A}_f - \mathcal{A} \subseteq \text{EXT}(\mathcal{L})$ , whence if  $C \in \text{INT}(T)$  is such that  $\text{conseq}(C) \subseteq \mathcal{A}_f$  and  $\text{antec}(C) \cap \mathcal{A}_f = \emptyset$ , then  $\text{conseq}(C) \subseteq \mathcal{A}$  and  $\text{antec}(C) \cap \mathcal{A} = \emptyset$ . i.e.,  $C$  can be used to attack all completions simultaneously.

When no rule in  $\text{INT}(T)$  can be employed to attack  $\mathcal{A}$  we must of course look to  $\text{EXT}(T)$  as dictated by the following two theorems.

**4.3 Theorem.** Suppose that  $\mathcal{A}$  is a set of literals of the form given in Definition 1.1, and  $\mathcal{E} = \{K \in \mathcal{A} : K \in \text{EXT}(\mathcal{L}) \cup \overline{\text{EXT}(\mathcal{L})}\}$ . Then  $\text{EXT}(T) \models^* \bigvee \mathcal{E}$  iff  $\text{EXT}(T) \models_{\text{min}} \bigvee \mathcal{E}$ .

**Proof** ( $\rightarrow$ ). Let  $M \subseteq \text{EXT}(\mathcal{L})$  be a minimal model of  $\text{EXT}(T)$  such that  $M \not\models \bigvee \mathcal{E}$ , then for each negative atom  $\neg P \in \mathcal{E}$ ,  $P \in M$  whence we may find a rule  $f(P) \in \text{EXT}(T)$  such that  $M \cap f(P) = \{P\}$ . But then  $P \in \{Q \in f(P) : \neg Q \in \mathcal{E}\} \subseteq M \cap f(P) = \{P\}$ , and  $f$  therefore defines a completion function for  $\mathcal{E}$  with  $(\mathcal{E}_f - \mathcal{E}) \cap M = \emptyset$ . Thus  $M \not\models \bigvee \mathcal{E}_f$  contradicting property (i).

( $\leftarrow$ ). Suppose that  $\mathcal{E}_f$  is a completion of  $\mathcal{E}$  and  $M$  is a model of  $\text{EXT}(T)$  such that  $M \not\models \bigvee \mathcal{E}_f$ . Let  $M' \subseteq M$  be a minimal model of  $\text{EXT}(T)$ . For each negative atom  $\neg P \in \mathcal{E}$ ,  $(f(P) - \{P\}) \cap M = \emptyset$ , whence (since  $M' \models f(P)$ ) we must have that  $P \in M'$ . Thus  $M' \models \bigvee \mathcal{E}$ . QED.

**4.4 Theorem.** Suppose that  $\mathcal{K} \subseteq \mathcal{L}$  and  $\mathcal{A}$  is a set of literals of the form given in Definition 1.1 with  $\mathcal{E} = \{K \in \mathcal{A} : K \in \text{EXT}(\mathcal{L}) \cup \overline{\text{EXT}(\mathcal{L})}\}$ .

(a) If  $\text{EXT}(T) \models^* \bigvee \mathcal{E}$ , then  $T|_g(\mathcal{L} - \mathcal{K}) \models^* \bigvee \mathcal{A}$ ,

(b) Suppose that  $T|_g(\mathcal{L} - \mathcal{K}) \models^* \bigvee \mathcal{A}$  and there is no rule  $C \in \text{INT}(T)$  such that  $\text{conseq}(C) \subseteq \mathcal{A}$ ,  $\text{antec}(C) \cap \mathcal{A} = \emptyset$ , and  $\mathcal{N}(C) \subseteq \mathcal{K}$ . Then  $\text{EXT}(T) \models^* \bigvee \mathcal{E}$ .

**Proof (a).** Suppose that  $\mathcal{A}_f$  is a completion of  $\mathcal{A}$  and  $M$  is a model of  $T|_g(\mathcal{L} - \mathcal{K})$  such that  $M \not\models \bigvee \mathcal{A}_f$ . Clearly  $M \cap \text{EXT}(\mathcal{L}) \models \text{EXT}(T)$ , and  $\mathcal{A}_f \cap \text{EXT}(\mathcal{L})$  is a completion of  $\mathcal{E}$ , whence  $M \models \bigvee \mathcal{A}_f$ .

(b). Suppose that  $\mathcal{E}_f$  is a completion of  $\mathcal{E}$ , then it is clear that  $\mathcal{C} = (\mathcal{A} \cap \mathcal{L}) \cup \mathcal{E}_f$  is a completion of  $\mathcal{A}$  with  $\mathcal{C} \cap \text{EXT}(\mathcal{L}) = \mathcal{E}_f \supseteq \mathcal{A} \cap \text{EXT}(\mathcal{L})$  and  $\mathcal{C} \cap \text{INT}(\mathcal{L}) = \mathcal{A} \cap \text{INT}(\mathcal{L})$ . But then clearly there is no rule  $C \in \text{INT}(\mathcal{L})$  such that  $\text{conseq}(C) \subseteq \mathcal{C}$ ,  $\text{antec}(C) \cap \mathcal{C} = \emptyset$ , and  $\mathcal{N}(C) \subseteq \mathcal{K}$ . By Theorem 4.1(a) we may find a rule  $C_0 \in \text{EXT}(T)$  such that  $C_0 \subseteq \mathcal{C}$ , whence  $C_0 \subseteq \mathcal{C} \cap \text{EXT}(\mathcal{L}) = \mathcal{E}_f$ . QED.

## §5. The compilation tree

In this section we combine the results of the previous two sections to develop a top-down method of query compilation. As indicated in Proposition 3.10 and Corollary 3.11, there are two types of goals that we need to consider. The first has the form  $T|_g(\mathcal{L} - \mathcal{N} \cup \overline{D_\alpha^-}) \models^* \bigvee \mathcal{P}$  (where  $\mathcal{N} = \emptyset$  in the condition (i) of Corollary 3.11), and we will denote this by a goal node of the form  $(? \bigvee \mathcal{P}, \mathcal{N})$ . By Theorem 4.2 (and the remark following), these goals are attacked by a rule  $C \in \text{INT}(T)$  such that  $\text{conseq}(C) \subseteq \mathcal{P}$ ,  $\text{antec}(C) \cap \mathcal{P} = \emptyset$  and  $\mathcal{N}(C) \subseteq \mathcal{N} \cup \overline{D_\alpha^-}$  (whence for each  $Q \in \mathcal{N}(C) - \mathcal{N}$  we obtain a negative child goal  $\neg Q$  (to test whether  $Q \in \overline{D_\alpha^-}$ ). This is then the second type of goal, and by Corollary 3.11, these are attacked by computing SQC trees for  $Q$ .

**5.1 Definition.** A *goal node* can either have the form (i)  $(? \bigvee \mathcal{P}, \mathcal{N})$ , where  $\mathcal{P}$  is a set of literals of the form given in Definition 1.1, and  $\mathcal{N} \subseteq \mathcal{L}$ , or (ii)  $\neg Q$ , where  $Q \in \mathcal{L}$ .

A *rule node* can either have the form (i)  $rn_C$ , where  $C \in T$ , or (ii)  $rn_{\rightarrow \neg Q}$ , where  $Q \in \mathcal{L}$ .

A *goal tree* for the goal  $g^*$  is a finite tree of goal nodes and rule nodes satisfying the following conditions.

(a) The root node is  $g^*$ .

(b) Each goal node of the form  $\neg Q$  appears at most once along any branch.

(c) If  $g = (? \bigvee \mathcal{P}, \mathcal{N})$  then  $g$  has at most a single child node, which (if it exists) is of the form  $rn_C$ , where

- $\text{conseq}(C) \subseteq \mathcal{P}$ ,
- $\text{antec}(C) \cap \mathcal{P} = \emptyset$ ,
- $rn_C$  has child (goal) nodes of the form

- $(? \bigvee \mathcal{P} \cup \{A\}, \mathcal{N})$  for each  $A \in \text{antec}(C)$ , and
- $\neg Q$  for each  $Q \in \mathcal{N}(C) - \mathcal{N}$

and these are the only child nodes of  $rn_C$ .

- (d) If  $g = \neg Q$ , then  $g$  has a single child node of the form  $rn_{\rightarrow \neg Q}$ .  $rn_{\rightarrow \neg Q}$  has, for each SQC tree  $\mathcal{S}$  for  $Q$  in  $T$ , a single child node of the form  $(? \bigvee \mathcal{N}(\mathcal{S}), \emptyset)$  or  $(? \bigvee \mathcal{R}(\mathcal{S}), \mathcal{N}(\mathcal{S}))$ .

A goal tree is said to be *satisfied* in  $\text{EXT}(T)$  iff for each leaf node of the form  $(? \bigvee \mathcal{P}, \mathcal{N})$ , we have that  $\text{EXT}(T) \models_{\min} \bigvee \{K \in \mathcal{P} : K \in \text{EXT}(\mathcal{L}) \cup \overline{\text{EXT}(\mathcal{L})}\}$  (cf., Theorems 4.3/4).

Note that the definition of a goal tree does not allow duplicate negative goals to be duplicated as we pass down a branch: this overcomes circular arguments of the form  $\neg Q \in D_\alpha^-$  iff  $\neg Q \in D_{\alpha-1}^-$ , and also guarantees termination of goal tree constructions.

Which goal trees should we construct as part of our compilation effort? If  $\mathcal{T}$  is a goal tree,  $(? \bigvee \mathcal{P}, \mathcal{N})$  is a leaf node in  $\mathcal{T}$  and  $C \in \text{INT}(T)$  with  $\text{conseq}(C) \subseteq \mathcal{P}$ ,  $\text{antec}(C) \cap \mathcal{P} = \emptyset$ , and  $\mathcal{N}(C) \subseteq \mathcal{N}$ , then let  $\mathcal{T}'$  be the tree formed by extending  $(? \bigvee \mathcal{P}, \mathcal{N})$  with  $rn_C$  (and the corresponding child nodes). Then it is clear that for any  $\text{EXT}(T)$ , if  $\mathcal{T}$  is satisfied in  $\text{EXT}(T)$ , then so is  $\mathcal{T}'$ , i.e.,  $\mathcal{T}$  is redundant as far as the compilation effort is concerned. (As an aside we note that this is not the only redundancy when generating goal trees, although we will not pursue this issue further in the current paper. Further discussions on redundancy in these types of tree constructions are to be found in [14, 16]). The same feature does *not* occur when we extend via a rule  $C$  for which  $\mathcal{N}(C) \not\subseteq \mathcal{N}$ , since the resulting negative subgoals  $\neg Q$  generate branches that might not be satisfied in  $\text{EXT}(T)$  (if  $\neg Q \notin \text{DWFS}$ ).

Note also that if  $\mathcal{P}$  is inconsistent, then  $(? \bigvee \mathcal{P}, \mathcal{N})$  is satisfied in any  $\text{EXT}(T)$ , and if  $\mathcal{P} \subseteq \text{INT}(\mathcal{L})$ , then  $(? \bigvee \mathcal{P}, \mathcal{N})$  is satisfied in no  $\text{EXT}(T)$ . We therefore make the following definition.

**5.2 Definition.** A goal tree is a *compilation tree* iff for each node of the form  $(? \bigvee \mathcal{P}, \mathcal{N})$  in  $\mathcal{T}$

- (a) if  $\mathcal{P}$  is inconsistent, then  $(? \bigvee \mathcal{P}, \mathcal{N})$  is a leaf node,
- (b) if  $\mathcal{P}$  is consistent and there is a rule  $C \in \text{INT}(T)$  such that  $\text{conseq}(C) \subseteq \mathcal{P}$ ,  $\text{antec}(C) \cap \mathcal{P} = \emptyset$ , and  $\mathcal{N}(C) \subseteq \mathcal{N}$ , then  $(? \bigvee \mathcal{P}, \mathcal{N})$  is extended with a rule node labelled with some rule having these properties, and
- (c) if  $\mathcal{P} \subseteq \text{INT}(\mathcal{L})$ , then  $(? \bigvee \mathcal{P}, \mathcal{N})$  is not a leaf node in  $\mathcal{T}$ .

**5.3 Theorem.**

- (a)  $\bigvee \mathcal{P} \in \text{DWFS}$  iff  $(? \bigvee \mathcal{P}, \emptyset)$  has a compilation tree that is satisfied in  $\text{EXT}(T)$ .
- (b)  $\neg Q \in \text{DWFS}$  iff  $\neg Q$  has a compilation tree that is satisfied in  $\text{EXT}(T)$ .

**Proof.** The implication from right to left follows trivially from Proposition 3.10, Corol-

lary 3.11 and Theorems 4.2-4.4.

For the converse, for each goal  $g$  define  $\mu(g)$  as follows. If  $g = (? \bigvee \mathcal{P}', \mathcal{N})$ , and  $T|_g(\mathcal{L} - \mathcal{N} \cup \overline{D_\alpha^-}) \models^* \bigvee \mathcal{P}'$  for some  $\alpha$ , then let  $\mu(g) = (2\alpha_0, |\mathcal{L} - \mathcal{P}'|)$  where  $\alpha_0$  is the smallest such  $\alpha$ . (Note that  $D_\alpha^- \subseteq D_{\alpha+1}^-$  whence if  $T|_g(\mathcal{L} - \mathcal{N} \cup \overline{D_\alpha^-}) \models^* \bigvee \mathcal{P}'$ , then  $T|_g(\mathcal{L} - \mathcal{N} \cup \overline{D_{\alpha+1}^-}) \models^* \bigvee \mathcal{P}'$ .) If no such  $\alpha$  exists, then  $\mu(g)$  is undefined.

If  $g = \neg Q$  and  $\neg Q \in D_{\alpha+1}^- - D_\alpha^-$ , then  $\mu(g) = (2\alpha + 1, 0)$ . Again if  $\neg Q \notin \text{DWFS}$ , then  $\mu(g)$  is undefined.

Let  $<_{lex}$  denote the (well-founded) lexicographic ordering of  $\mu$  values. If  $\mu(g)$  is well defined, we show by induction on  $<_{lex}$  that we can develop a compilation tree for  $g$  that is satisfied in  $\text{EXT}(T)$  in which (i) for each goal node  $g'$ ,  $\mu(g')$  is defined, and (ii) the  $\mu$  values are  $<_{lex}$ -monotonic decreasing along any branch (which also guarantees condition (b) from Definition 5.1).

**Case 1.**  $g = \neg Q$  with  $\mu(g) = (2\alpha + 1, 0)$ , then we append  $rn_{\rightarrow, \neg Q}$  as a child of  $g$ . For each SQC tree  $\mathcal{S}$  for  $Q$  in  $T$ , we can (by Corollary 3.11) append either  $(? \bigvee \mathcal{N}(\mathcal{S}), \emptyset)$  (if  $T|_g(\mathcal{L} - \overline{D_\alpha^-}) \models^* \bigvee \mathcal{N}(\mathcal{S})$ ) or  $(? \bigvee \mathcal{R}(\mathcal{S}), \mathcal{N}(\mathcal{S}))$  (if  $T|_g(\mathcal{L} - \mathcal{N}(\mathcal{S}) \cup \overline{D_\alpha^-}) \models^* \bigvee \mathcal{R}(\mathcal{S})$ ). If  $g'$  is such a child, then  $\mu(g') = (2\delta, \sigma)$  where  $\delta \leq \alpha$ , whence  $\mu(g') <_{lex} \mu(g)$ , and the result follows by inductive hypothesis.

**Case 2.** Suppose that  $g = (? \bigvee \mathcal{P}', \mathcal{N})$  with  $\mu(g) = (2\alpha_0, |\mathcal{L} - \mathcal{P}'|)$ . If  $\mathcal{P}'$  is inconsistent, then we can leave  $g$  as a leaf.

If  $\text{EXT}(T) \not\models_{min} \bigvee \{K \in \mathcal{P}' : K \in \text{EXT}(\mathcal{L}) \cup \overline{\text{EXT}(\mathcal{L})}\}$ , then by Theorem 4.4(b) we can find a rule  $C \in \text{INT}(T)$  such that  $\text{conseq}(C) \subseteq \mathcal{P}'$ ,  $\text{antec}(C) \cap \mathcal{P}' = \emptyset$ , and  $\mathcal{N}(C) - \mathcal{N} \subseteq \overline{D_{\alpha_0}^-}$ . We then extend  $g$  with such a rule, giving preference to a rule for which  $\mathcal{N}(C) \subseteq \mathcal{N}$  if such exists (in order to satisfy condition (b) of Definition 5.2). Note that this also covers case (c) of Definition 5.2.

For each negative child node  $\neg Q'$  of  $g$ , we have that  $\neg Q' \in D_{\alpha_0}^-$ . Thus  $\mu(\neg Q') = (\beta, 0)$ , where  $\beta \leq 2(\alpha_0 - 1) + 1 < 2\alpha_0$ , whence  $\mu(\neg Q') <_{lex} \mu(g)$ , and the result then follows by inductive hypothesis.

In the case of child goals of the form  $g' = (? \bigvee (\mathcal{P}' \cup \{A\}), \mathcal{N})$  (where  $A \in \text{antec}(C)$ ), since  $T|_g(\mathcal{L} - \mathcal{N} \cup \overline{D_{\alpha_0}^-}) \models^* \bigvee \mathcal{P}'$ , it is trivially the case that  $T|_g(\mathcal{L} - \mathcal{N} \cup \overline{D_{\alpha_0}^-}) \models^* \bigvee (\mathcal{P}' \cup \{A\})$ . Thus  $\mu(g')$  is of the form  $(\delta, |\mathcal{L} - (\mathcal{P}' \cup \{A\})|)$ , where  $\delta \leq 2\alpha_0$  and  $|\mathcal{L} - (\mathcal{P}' \cup \{A\})| < |\mathcal{L} - \mathcal{P}'|$  (since  $\text{antec}(C) \cap \mathcal{P}' = \emptyset$ ). Thus  $\mu(g') <_{lex} \mu(g)$ , whence the result follows by inductive hypothesis. QED.

## 5.4 Compilation and run-time query processing.

Query compilation thus consists of computing the compilation trees for the given query. Only the leaf nodes of these trees need to be stored following compilation.

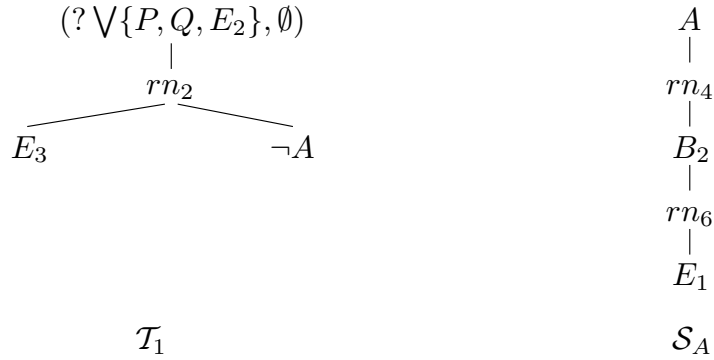
The run-time processing of the query then of course consists of checking whether any such tree is satisfied in  $\text{EXT}(T)$ .

**5.5 Example.** Let  $\text{INT}(T)$  consist of the following rules.

1.  $E_0 \wedge E_1 \wedge \neg E_3 \wedge \neg B \rightarrow B_3$
2.  $E_3 \wedge \neg A \rightarrow P \vee Q$
3.  $E_4 \wedge \neg E_3 \rightarrow B$
4.  $\neg B_1 \wedge B_2 \rightarrow A$
5.  $E_0 \wedge \neg A \rightarrow B_3$
6.  $E_1 \wedge \neg E_3 \rightarrow B_2 \vee B_3$
7.  $E_0 \wedge \neg B_2 \rightarrow B_1 \vee Q$

where  $\text{EXT}(\mathcal{L}) = \{E_0, E_1, E_2, E_3, E_4\}$ . Suppose that we wish to compile the query  $? \vee \{P, Q, E_2\}$ . Firstly note that rule 2 is the only rule whose consequent is a subset of  $\{P, Q, E_2\}$ , whence (since  $E_2 \in \text{EXT}(\mathcal{L})$ ) the goal tree consisting of the single root node  $(? \vee \{P, Q, E_2\}, \emptyset)$  is a compilation tree.

We now consider extending this tree via rule 2, yielding the tree  $\mathcal{T}_1$  in Figure 5.5(i). Note that for brevity, the goal  $(? \vee \{P, Q, E_2, E_3\}, \mathcal{N})$  is represented by the branch as a whole.  $rn_2$  denotes the application of rule 2, etc.

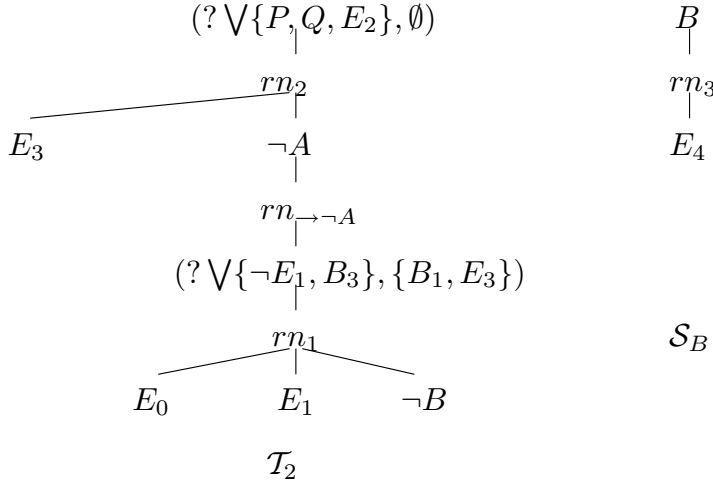


**Figure 5.5(i).**

The left hand branch cannot be extended further. The goal  $\neg A$  requires the construction of SQC trees for  $A$  in  $T$ , the only such tree being depicted in Figure 5.5(i) as  $\mathcal{S}_A$ , with  $\mathcal{R}(\mathcal{S}_A) = \{\neg E_1, B_3\}$  and  $\mathcal{N}(\mathcal{S}_A) = \{B_1, E_3\}$ .

The node  $rn_{\rightarrow \neg A}$  then has a child node of the form  $(? \vee \{B_1, E_3\}, \emptyset)$  or  $(? \vee \{\neg E_1, B_3\}, \{B_1, E_3\})$ . Notice that the former child would be a leaf (since no consequent is contained within  $\{B_1, E_3\}$  and  $E_3 \in \text{EXT}(\mathcal{L})$ ), thus yielding our second compilation tree.

The node  $(? \vee \{\neg E_1, B_3\}, \{B_1, E_3\})$  can only be extended with rule 1 and therefore it can be treated as leaf, and this yields our third compilation tree. Extending the node with rule 1 yields the tree depicted as  $\mathcal{T}_2$  in Figure 5.5(ii).



**Figure 5.5(ii).**

The left hand child of  $rn_1$  again forms a leaf, and the centre child can be ignored as a result of its inconsistency. The goal  $\neg B$  calls for the generation of SQC trees for  $B$ , the only such tree being depicted as  $\mathcal{S}_B$  in Figure 5.5(ii), with  $\mathcal{N}(\mathcal{S}_B) = \{E_3\}$  and  $\mathcal{R}(\mathcal{S}_B) = \{\neg E_4\}$ . In both cases, the resulting child nodes  $(?E_3, \emptyset)$  and  $(?\neg E_4, \{E_3\})$  cannot be extended, thus yielding 2 final compilation trees.

In total we have identified 5 such trees, whose leaf nodes are listed below.

1.  $(? \vee \{P, Q, E_2\}, \emptyset)$ ,
2.  $(? \vee \{P, Q, E_2, E_3\}, \emptyset)$ ,  $(? \vee \{B_1, E_3\}, \emptyset)$ ,
3.  $(? \vee \{P, Q, E_2, E_3\}, \emptyset)$ ,  $(? \vee \{\neg E_1, B_3\}, \{B_1, E_3\})$ ,
4.  $(? \vee \{P, Q, E_2, E_3\}, \emptyset)$ ,  $(? \vee \{E_0, \neg E_1, B_3\}, \{B_1, E_3\})$ ,  $(?E_3, \emptyset)$ ,
5.  $(? \vee \{P, Q, E_2, E_3\}, \emptyset)$ ,  $(? \vee \{E_0, \neg E_1, B_3\}, \{B_1, E_3\})$ ,  $(?\neg E_4, \{E_3\})$ .

Thus  $\vee\{P, Q, E_2\}$  is in DWFS iff either

1.  $\text{EXT}(T) \models_{\min} E_2$ , or
2.  $\text{EXT}(T) \models_{\min} (E_2 \vee E_3) \wedge E_3$ , or
3.  $\text{EXT}(T) \models_{\min} (E_2 \vee E_3) \wedge \neg E_1$ , or
4.  $\text{EXT}(T) \models_{\min} (E_2 \vee E_3) \wedge (E_0 \vee \neg E_1) \wedge E_3$ , or
5.  $\text{EXT}(T) \models_{\min} (E_2 \vee E_3) \wedge (E_0 \vee \neg E_1) \wedge \neg E_4$ .

## §6. Conclusions.

We have presented a top-down method for compiling queries in propositional deductive databases under the DWFS using the concept of a compilation tree. Our method is somewhat more complex than query compilation under (say) the disjunctive stable model semantics since we cannot compile a query by constructing a single tree. This

is a result of the non-confluence of the tree constructions in the following sense. In the case of disjunctive stable models, query processing can be achieved via a tree construction in which there are no dead-ends: if the query is indeed true, then any partial tree construction can always be extended to a full tree. In DWFS, the introduction of a negative node ( $\neg Q$ ) into our tree construction can have the effect of rendering subsequent extensions unsatisfiable in  $\text{EXT}(T)$  (if  $\neg Q \notin \text{DWFS}$ ).

Of course real databases are first order, and this then raises the question as to how we lift our methods to cover this case. The first point to note is that at the first order level, UQC trees are always ground [13, Theorem 6.4.7]. The construction of SQC trees at the first order level is detailed in [17]. Specifically the definition and construction of *complete pairs* is introduced, where a complete pair consists of a first order SQC tree  $\mathcal{T}$  and a set of constraints  $\mathcal{I}$  (satisfying certain conditions).  $\mathcal{S}$  is an SQC tree in  $\text{gr}(T)$  (the set of ground instances of  $T$ ) iff there is a complete pair  $(\mathcal{T}, \mathcal{I})$  and a ground instantiation  $\theta$  of  $\mathcal{T}$  such that  $\mathcal{T}\theta = \mathcal{S}$  and  $\theta$  does not violate  $\mathcal{I}$ .

In expanding a negative atom in a goal tree, we thus need to construct complete pairs. Each such complete pair  $(\mathcal{T}, \mathcal{I})$  then represents the set of completions  $\{(\mathcal{T}\theta)_f : \theta \text{ is a ground instantiation of } \mathcal{T} \text{ and does not violate } \mathcal{I}\}$ . The problem is that we cannot attack all of these completions simultaneously using rules in  $\text{INT}(T)$ . At the ground level, the reason why we can do this is precisely because for all completions  $\mathcal{S}_f, \mathcal{S}_h$  of an SQC tree  $\mathcal{S}$ ,  $\mathcal{O}(\mathcal{S}_f) \cap \text{INT}(\mathcal{L}) = \mathcal{O}(\mathcal{S}_h) \cap \text{INT}(\mathcal{L}) = \mathcal{O}(\mathcal{S})$ . Clearly this property fails when employing complete pairs as above, in that differing instantiations  $\theta$  can produce differing  $\mathcal{O}$  sets. Research on this issue is ongoing.

## References.

- [1] C. Aravindan, J. Dix and I. Niemelä, DISLOP: A research project on disjunctive logic programming, *AI communications*, vol. 10 (1997), 151-165.
- [2] C. Baral, J. Lobo and J. Minker, WF<sup>3</sup>: A semantics for negation in normal disjunctive logic programs, in Z. Ras and M. Zemankova (eds.), *Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems*, (Springer, 1991), 459-468.
- [3] C. Baral, J. Lobo and J. Minker, Generalised disjunctive well-founded semantics for logic programs, *Annals of Mathematics and Artificial Intelligence*, vol. 5 (1992), 89-132.
- [4] S Brass and J. Dix, A disjunctive semantics based upon unfolding and bottom-up evaluation, in : B Wolfinger, (ed.), *Innovationen bei Rechen- und Kommunikationssystemen (IFIP- Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases)*, (Springer, 1994), 83-91.
- [5] S. Brass and J. Dix, Disjunctive semantics based upon partial and bottom-up evaluation, in : L. Sterling (ed.), *Proc. 12th Int'l Conf. on Logic Programming*, Tokyo (MIT Press, 1995), 199-213.
- [6] S. Brass, J. Dix, I. Niemelä and T. C. Przymusiński, A comparison of the static and disjunctive well-founded semantics, in: A.G. Cohn, L.K. Schubert and S.C.

- Shapiro (eds.), Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (Morgan Kaufmann, 1998), 74-85.
- [7] S. Brass and J. Dix, Characterisations of the disjunctive well-founded semantics: confluent calculi and iterated GCWA, *J. Automated Reasoning*, vol. 20 (1998), 143-165.
  - [8] S. Brass and J. Dix, Semantics of disjunctive logic programs based upon partial evaluation, *J. Logic Programming*, vol. 40 (1999), 1-46.
  - [9] J. Dix, Semantics of logic programs: Their intuitions and formal properties, in A. Fuhrman and H. Rott (eds.), *Logic, Action and Information, Essays on Logic in Philosophy and Artificial Intelligence* (DeGruyter, 1995), 241-327.
  - [10] J. Dix, U. Furbach and I. Niemelä, Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations, in A. Robinson and A. Voronkov (eds.), *Handbook of Automated Reasoning*, vol. 2, (Elsevier Science, 2001), 1121-1234.
  - [11] 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.
  - [12] L. J. Henschen and H. Park, Compiling the GCWA in indefinite databases, in J. Minker, ed., *Foundations of Deductive Databases*, (Morgan Kaufmann, 1988), 395-438.
  - [13] C. A. Johnson, On computing minimal and perfect model membership, *Data and Knowledge Engineering*, vol. 18 (1996), 225-276.
  - [14] C. A. Johnson, Top-down query processing in indefinite stratified databases, *Data and Knowledge Engineering*, vol. 26 (1998), 1-36.
  - [15] C. A. Johnson, On cyclic covers and perfect models, *Data and Knowledge Engineering*, vol. 31 (1999), 25-65.
  - [16] C. A. Johnson, Processing deductive databases under the disjunctive stable model semantics, *Fundamenta Informaticae*, vol. 40 (1999), 31-51.
  - [17] C. A. Johnson, Top-down query processing in first order deductive databases under the DWFS, in Raś and Ohsuga (eds.), *Foundations of Intelligent Systems*, (Springer, 2000), 377-388.
  - [18] C. A. Johnson, On the computation of the disjunctive well-founded semantics, *Journal of Automated Reasoning*, vol. 26 (2001), 333-356.
  - [19] J. Lobo, J. Minker, and A. Rajasekar, *Foundations of Disjunctive Logic Programming*, (MIT Press, 1992).
  - [20] T. Przymusiński, Stable semantics for disjunctive programs, *New Generation Computing*, vol. 9 (1991), 401-424.