# Argumentation-theoretic proof procedures for default reasoning

7 May 2003

**P.M. Dung**
Division of Computer Science, Asian Institute of Technology
PO Box 2754, Bangkok 10501, Thailand
dung@cs.ait.ac.th

**R.A. Kowalski**, **F. Toni**
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
{rak,ft}@doc.ic.ac.uk

## Abstract

We present a succession of presentations of an argumentation-theoretic proof procedure that applies uniformly to a wide variety of logics for default reasoning, including Theorist, default logic, logic programming, autoepistemic logic, non-monotonic modal logic and certain instances of circumscription.

The proof procedure can be viewed as conducting a dispute, which is a sequence of alternating arguments:

$\mathcal{D}_0, \mathcal{A}_0, \mathcal{D}_1, \ldots, \mathcal{A}_i, \mathcal{D}_{i+1}, \ldots$

between a defendant and an opponent. The defendant goes first and wins the dispute if the defendant has a defence $\mathcal{D}_{i+1}$ against every attack $\mathcal{A}_i$ by the opponent.

The proof procedure generalises an abductive proof procedure for logic programming and implements the admissibility semantics. The admissibility semantics can be viewed as a variant of the stable semantics, which is the conventional semantics for many logics for default reasoning.

# 1 Introduction

In this paper, we present an argumentation-theoretic proof procedure for default reasoning. The proof procedure applies uniformly to any logic [31, 43, 32, 33] that can be defined by means of an abstract framework [4] for default reasoning, which includes default logic, logic programming, autoepistemic logic, non-monotonic modal logic and certain instances of circumscription.

The proof procedure can be viewed as conducting a dispute in which a defendant and an opponent exchange arguments. In its simplest form, the dispute can be viewed as a sequence of alternating arguments:

$\mathcal{D}_0, \mathcal{A}_0, \mathcal{D}_1, \ldots, \mathcal{A}_i, \mathcal{D}_{i+1}, \ldots$

The defendant puts forward some initial argument $\mathcal{D}_0$. For every argument $\mathcal{D}_i$ put forward by the defendant, the opponent attempts to respond with an attacking argument $\mathcal{A}_i$ against $\mathcal{D}_i$. For every attacking argument $\mathcal{A}_i$ put forward by the opponent, the defendant attempts to counter-attack with a defending argument $\mathcal{D}_{i+1}$ against $\mathcal{A}_i$. To win the dispute, the defendant needs to have a defending argument against every attacking argument of the opponent.

A winning dispute can be represented both in the form of a proof tree, and in the form of a derivation, which represents the top-down, step-by-step construction of the proof tree.

It can also be represented both abstractly, with arguments represented simply by sets of defeasible assumptions, and concretely, with arguments represented by complete proofs supported by defeasible assumptions.

These two options, proof tree versus derivation, and abstract versus concrete, give rise to four different but closely related presentations of our proof procedure. They allow us to develop our proof procedure in a succession of steps starting from a semantic specification and ending with the definition of concrete derivation, which is close to a Prolog implementation.

The specification of our proof procedure is the admissibility semantics, which generalises the admissibility semantics for logic programming [12, 13]. A set of defeasible assumptions is *admissible*, if it defends itself against all attacks (by counter-attacking all attacks against it) and does not attack itself. The proof procedure implements the semantics by constructing admissible sets incrementally.

The admissibility semantics is a credulous semantics for default reasoning. It contrasts with the stable semantics, which is the conventional semantics associated with most logics for default reasoning, including default logic, autoepistemic logic and nonmonotonic modal logic. Informally speaking, a set of defeasible assumptions is *stable*, if it attacks every assumption not in the set and does not attack itself.

The admissibility semantics is more general than the stable semantics, in the sense that every stable set of assumptions is a maximal (with respect to set inclusion) admisssible set of assumptions. However, not every maximal admissible set of assumptions is stable. Nonetheless, proof procedures for the admissibility semantics can also be used to compute the stable semantics in certain cases.

In the general case, the "global" character of the stable semantics (that a stable set must attack every assumption not in the set) presents a major obstacle to its efficient implementation. In contrast with the stable semantics, the admissibility semantics requires only the "local" construction of a set of assumptions that is relevant to establishing that a given query is a default consequence of a given theory. The local character of the admissibility semantics facilitates the development of an efficient proof procedure.

This paper is a sequel to the earlier paper [4], which described the abstract framework and its applications in detail. The paper [29] presents further examples, complementing those in [4]. Because these papers provide motivating examples, in this paper we focus mainly on the technical details of the proof procedure.

The rest of the paper has the following structure: Section 2 describes the abstract framework and its concrete instances. Section 3 describes the stable and admissibility semantics, while section 4 discusses and illustrates the computational problems associated with the stable semantics. Section 5 prepares the ground for the abstract proof trees for the admissibility semantics presented in section 6. The following three sections 7, 8, 9 present successive refinements of the proof procedure. Section 10 compares the proof procedure with other proof procedures for default reasoning, and is followed by the conclusion. Proofs of the theorems are in the appendix.

# 2 Assumption-based frameworks

In this section we briefly review the notion of assumption-based framework [5, 4, 29, 28], showing how it can be used to extend any deductive system for a monotonic logic to a non-monotonic logic for default reasoning.

A *deductive system* is a pair $(\mathcal{L}, \mathcal{R})$ where

- $\mathcal{L}$ is a formal language consisting of countably many sentences, and
- $\mathcal{R}$ is a set of inference rules of the form

$$\frac{\{\alpha_1, \ldots, \alpha_n\}}{\alpha}$$

where $\alpha, \alpha_1, \ldots, \alpha_n \in \mathcal{L}$ and $n \geq 0$. If $n = 0$, then the inference rule represents an axiom[1].

A set of sentences $T \subseteq \mathcal{L}$ is called a *theory*.

A *deduction* from a theory $T$ is a sequence $\beta_1, \ldots, \beta_m$, where $m > 0$, such that, for all $i = 1, \ldots, m$,

- $\beta_i \in T$, or
- there exists $\frac{\alpha_1, \ldots, \alpha_n}{\beta_i}$ in $\mathcal{R}$ such that $\alpha_1, \ldots, \alpha_n \in \{\beta_1, \ldots, \beta_{i-1}\}$.

$T \vdash \alpha$ means that there is a deduction (of $\alpha$) from $T$ whose last element is $\alpha$. $T \vdash Q$, for a set of sentences $Q$, means that $T \vdash \alpha$ for every $\alpha \in Q$. $Th(T)$ is the set $\{\alpha \in \mathcal{L} \,|\, T \vdash \alpha\}$.

To simplify the definition of the proof procedure later on in the paper, from section 8 we will assume that the theory $T$ is empty. This assumption can be made without loss of generality, because domain-specific theories can be simulated by domain-specific inference rules. More precisely, the consequence relation $T \vdash \sigma$ in a deductive system $(\mathcal{L}, \mathcal{R})$ is equivalent to the consequence relation $\{\} \vdash \sigma$ in a deductive system $(\mathcal{L}, \mathcal{R}')$ where $\mathcal{R}' = \mathcal{R} \cup \{\frac{}{\alpha} | \alpha \in T\}$.

[1] For simplicity, we simply write $\frac{\alpha_1, \ldots, \alpha_n}{\alpha}$ instead of $\frac{\{\alpha_1, \ldots, \alpha_n\}}{\alpha}$

Deductive systems are *monotonic*, in the sense that $T \subseteq T'$ implies $Th(T) \subseteq Th(T')$. They are also *compact*, in the sense that $T \vdash \alpha$ implies $T' \vdash \alpha$ for some finite subset $T'$ of $T$.

Given a deductive system $(\mathcal{L}, \mathcal{R})$, an *assumption-based framework* with respect to $(\mathcal{L}, \mathcal{R})$ is a tuple $\langle T, Ab, \overline{\ }\rangle$ where

- $T$, $Ab \subseteq \mathcal{L}$, $Ab \neq \{\}$
- $\overline{\ }$ is a mapping from $Ab$ into $\mathcal{L}$. $\overline{\alpha}$ is called the *contrary* of $\alpha$.

The theory $T$ can be viewed as a given set of beliefs, and $Ab$ as a set of candidate *assumptions* that can be used to extend $T$. An *extension* of a theory $T$ is a theory $Th(T \cup \Delta)$, for some $\Delta \subseteq Ab$. Sometimes, informally, we refer to the extension simply as $T \cup \Delta$ or $\Delta$.

We understand the problem of determining whether a given *sentence* $\sigma$ in $\mathcal{L}$ (also called a *query*) is a *default consequence* of $\langle T, Ab, \overline{\ }\rangle$ as the problem of determining whether there exists an "appropriate" $\Delta \subseteq Ab$ of $T$ such that $T \cup \Delta \vdash \sigma$. We also say that a *set $Q$ of sentences* is a *default consequence* of $\langle T, Ab, \overline{\ }\rangle$ if $\sigma$ is a *default consequence* of $\langle T, Ab, \overline{\ }\rangle$ for every $\sigma$ in $Q$. [2] This is a *credulous* semantics for default reasoning, in the sense that it requires only that $\sigma$ belong to *some* "appropriate" extension of $T$.

Many logics for default reasoning are credulous in this same sense, differing however in the way they understand what it means for an extension to be "appropriate". Some logics, in contrast, are *sceptical*, in the sense that they require that $\sigma$ belong to *all* "appropriate" extensions. However, the semantics of any of these logics can be made sceptical or credulous, simply by varying whether a sentence is deemed to be a consequence of a theory if it belongs to all "appropriate" extensions or if it belongs only to some "appropriate" extension.

Different logics for default reasoning differ, however, not only in whether they are credulous or sceptical and how they interpret the notion of what it means to be an appropriate extension, but also in their underlying frameworks.

*Theorist* [39] can be understood as a framework $\langle T, Ab, \overline{\ }\rangle$ where $T$ and $Ab$ are both arbitrary sets of sentences of first-order logic and the contrary $\overline{\alpha}$ of an assumption $\alpha$ is just its negation. The deductive system $(\mathcal{L}, \mathcal{R})$ is any deductive system for first-order logic.

*Circumscription* [31] can be understood similarly, except that the set of candidate assumptions is the set of all negations of (variable-free) atomic sentences $\neg p(t)$, for all predicates $p$ which are minimised, and of all (variable-free) atomic sentences $q(t)$ or their negations, for all predicates $q$ which are fixed. Both Theorist and Circumscription agree on their interpretation of what it means for an extension to be appropriate. However, the semantics of Theorist is credulous, whereas the semantics of Circumscription is sceptical.

[2] Whenever we assume $T$ to be empty, $T \cup \Delta \vdash \sigma$ is equivalent to $\Delta \vdash \sigma$.

*Logic programs* can be viewed as frameworks $\langle \{\}, Ab, \overline{\ } \rangle$ where the language $\mathcal{L}$ of the deductive system $(\mathcal{L}, \mathcal{R})$ consists of variable-free (also called "ground") atomic formulae and their negations and the inference rules $\mathcal{R}$ consist of "domain specific" rules of the form [3]

$$\frac{\alpha_1, \ldots, \alpha_m, not\beta_1, \ldots, not\beta_n}{\gamma}$$

For notational convenience, we also write such inference rules in the linear notation

$$\gamma \leftarrow \alpha_1, \ldots, \alpha_m, not\beta_1, \ldots, not\beta_n$$

We also write $\gamma$ instead of $\frac{-}{\gamma}$ or $\gamma \leftarrow$.

The intuitive meaning of $not\,\beta$ is that $\beta$ can not be shown. The set of candidate assumptions $Ab$ is the set of all negations $not\,p$ of ground atomic sentences $p$, and the contrary $\overline{not\,p}$ of an assumption is $p$. Most semantics for logic programs, including both credulous and sceptical semantics, can be understood in terms of this framework.

*Default logic* [43] is the case where the underlying monotonic logic is first-order logic augmented with domain-specific inference rules of the form

$$\frac{\alpha_1, \ldots, \alpha_m, M\beta_1, \ldots, M\beta_n}{\gamma}$$

where $\alpha_i$, $\beta_j$, $\gamma$ are first-order sentences. [4] $T$ is a first-order theory and $Ab$ consists of all expressions of the form $M\beta$ where $\beta$ is a sentence of first-order logic. The contrary $\overline{M\beta}$ of an assumption $M\beta$ is $\neg\beta$. The intuitive meaning of $M\beta$ is that $\neg\beta$ can not be shown. The semantics of default logic is normally understood in credulous terms.

In *autoepistemic logic* [33] the language $\mathcal{L}$ is the language of modal logic, with a modal operator $L$, but the inference rules in $\mathcal{R}$ are those of classical logic. The assumptions have the form $\neg L\alpha$ or $L\alpha$. The contrary of $\neg L\alpha$ is $\alpha$, and the contrary of $L\alpha$ is $\neg L\alpha$. The intuitive meaning of $L\alpha$ is that $\alpha$ can be shown. *Non-monotonic modal logics* [32] are similar, except the inference rules are those of some underlying modal logic and only assumptions of the form $\neg L\alpha$ are considered. The semantics of both *autoepistemic logic* and *non-monotonic modal logics* are normally understood in credulous terms.

All examples used in the paper are drawn from logic programming, because of its simplicity. However, the same examples can also be expressed in any of the other formalisms (see [29]).

With the possible exception of logic programming, where many different notions of appropriate extension have been investigated, almost all of these logics have focused on the notion of stable extension, presented in the next section 3. They also require that an appropriate extension $\Delta$ be closed.

[3] We assume, as is conventional, that a logic program containing variables stands for the set of all its ground instances over the Herbrand universe of the program.

[4] As in the case of logic programs, inference rules containing free variables are interpreted as standing for the set of all their ground instances.

**Definition 2.1** A set of assumptions $\Delta \subseteq Ab$ is *closed* if and only if $\Delta = \{\alpha \in Ab \,|\, T \cup \Delta \vdash \alpha\}$. An assumption-based framework is *flat* if and only if every set of assumptions $\Delta \subseteq Ab$ is closed.

Our proof procedure is defined for flat assumption-based frameworks. However, as we will see later, it can also be adapted to non-flat frameworks.

All frameworks for logic programming and default logic are flat. Informally speaking, flat frameworks are ones in which there are no "hidden" assumptions.

The framework with $T = \{\neg Lp\}$, in both auto-epistemic and non-monotonic modal logic, is an example of a non-flat framework. In this framework, the empty set of assumptions implies, together with $T$, the "hidden" assumption $\neg Lp$.

# 3 Argumentation semantics for assumption-based frameworks

In this section we review the argumentation semantics of stable and admissible extensions for assumption-based frameworks, as given in [12, 13, 5, 4, 29, 28]. In these semantics a deduction of a conclusion from a theory $T$ augmented by assumptions $A$ is regarded as an argument for the conclusion, supported by $A$. A set of assumptions $A$ is regarded as attacking another set of assumptions $\Delta$ if $A$ supports an argument for the contrary of some assumption in the $\Delta$. More formally:

**Definition 3.1** Given any assumption-based framework $\langle T, Ab, \overline{\ }\rangle$ and a set of assumptions $A \subseteq Ab$,

- *$A$ attacks an assumption $\alpha \in Ab$* if and only if $T \cup A \vdash \overline{\alpha}$. The corresponding deduction of $\overline{\alpha}$ is said to be an *argument* for $\overline{\alpha}$ and against $\alpha$, *supported* by $A$.
- *$A$ attacks a set of assumptions $\Delta \subseteq Ab$* if and only if $A$ attacks some assumption $\alpha \in \Delta$. The corresponding deduction of $\overline{\alpha}$ is said to be an *argument* against $\Delta$, *supported* by $A$. Any such assumption $\alpha \in \Delta$ is said to be a *culprit* in $\Delta$ attacked by $A$.

**Definition 3.2** A set of assumptions $\Delta$ is *stable* if and only if

1. $\Delta$ is closed,
2. $\Delta$ does not attack itself, and
3. $\Delta$ attacks every assumption $\alpha \notin \Delta$.

Equivalently, $\Delta$ is stable if and only if $\Delta$ is closed and $\Delta$ is the set of all assumptions it does not attack. Thus, the stable semantics forces $\Delta$ to take a stand for, or against, every assumption, no matter whether or not that assumption is relevant to a given

candidate query. The admissibility semantics, on the other hand, requires only that $\Delta$ takes a stand against those (relevant) assumptions that participate in closed attacks against $\Delta$.

**Definition 3.3** A set of assumptions $\Delta$ is an *admissible* extension of a given theory $T$ if and only if

1. $\Delta$ is closed,
2. $\Delta$ does not attack itself, and
3. for each closed set of assumptions $A \subseteq Ab$,
   if $A$ attacks $\Delta$ then $\Delta$ attacks $A$.

In other words, an admissible set of assumptions is a closed set of assumptions that does not attack itself and that counter-attacks every closed set of assumptions that attacks it.

For an important class of frameworks, namely stratified frameworks, there exist unique stable extensions. These extensions coincide with the unique maximal admissible extensions of these frameworks [4]. Because every admissible extension is contained in some maximal admissible extension, admissible extensions for stratified frameworks can always be extended to stable extensions, and proof procedures for the admissibility semantics can, therefore, be used to approximate the computation of the stable semantics.

**Definition 3.4** A flat framework is *stratified* if and only if there exists no infinite sequence of assumptions of the form $\alpha_1, \ldots, \alpha_n, \ldots$, where for every $n \geq 1$, $\alpha_{n+1}$ belongs to a minimal (with respect to set inclusion) attack against $\alpha_n$.

**Example 3.1** The logic program

$$p \leftarrow not\, q$$
$$q \leftarrow not\, p$$

is not stratified, because there exists an infinite sequence of the form:
$not\, p, not\, q \ldots, not\, p, not\, q, \ldots$.

In the next section we discuss some of the problems inherent in the computation of credulous default consequences under the stable semantics.

# 4 Computation of stable semantics

In the general case, default reasoning is non-computable no matter what semantics is used to determine the notion of default consequence. This is because, formulated in argumentation-theoretic terms, all such semantics require the construction of sets of assumptions $\Delta$ that do not attack themselves. In the case of Theorist and Circumscription, this is equivalent to showing that $\Delta$ is consistent; and consistency in general is not even semi-decidable. The situation is no better for the other logics for default reasoning.

Because of the non-computability of default reasoning in general, all formal complexity results for default reasoning apply only to the case of finite propositional theories (e.g. see [11]). Our comparison of the relative efficiency of determining stable versus admissible consequences, therefore, will be largely informal and will be illustrated by means of examples.

**Example 4.1** It is a principle of most systems of law that a person who is accused of committing a crime is deemed to be innocent unless the person is proven to be guilty. Suppose Mary's father is accused of committing a crime. These assumptions can be formalised by means of the theory $T = \{\}$ with the following domain specific inference rules, written in logic programming form.

$$innocent(X) \leftarrow accused(X),\ \ not\, guilty(X)$$
$$accused(father(Mary)).$$

Consider whether $innocent(father(Mary))$ is a default consequence of $T$ according to the stable model semantics. For this purpose, it is necessary to ensure the existence of a stable set of assumptions $\Delta$ such that $innocent(father(Mary)) \in Th(T \cup \Delta)$. However, only the assumption $not\, guilty(father(Mary))$ is "relevant", in the sense that it alone, together with the given theory, monotonically implies $innocent(father(Mary))$. Thus it is necessary to show that this assumption belongs to some stable set.

There is only one stable set of assumptions for the given theory, and the assumption does indeed belong to it. However, the stable set is the infinite set

$$\{not\, guilty(father^i(Mary)) | i \geq 0\}.$$

In contrast, the singleton set

$$\{not\, guilty(father(Mary))\}$$

consisting of only the relevant assumption, is admissible.

The "global" character of the stable semantics affects the efficiency of its proof procedures, which need to guarantee the existence of an entire stable extension, independently of the query to be answered. In contrast, our proof procedure for the

admissibilty semantics is query-oriented. It uses the monotonic inference rules backwards, logic programming style, not only to find assumptions $\Delta_0$ to support the given query, but also to find attacks and to generate defences against attacks, ending with an admissible set of assumptions $\Delta$ that includes these defences and extends $\Delta_0$.

Some proof procedures for the stable semantics do manage to restrict their attention to the "local" computation of relevant assumptions (e.g. [45]), while guaranteeing nonetheless that they belong to some stable set. However, the following example shows that this is not always possible.

**Example 4.2** Consider whether $innocent(father(Mary))$ is a default consequence according to the stable semantics of the theory $T = \{\}$ and the following domain specific rules of inference.

$$innocent(X) \leftarrow accused(X), not\ guilty(X)$$
$$accused(father(Mary))$$
$$accused(Mary) \leftarrow not\ accused(Mary).$$

The last rule can be seen as a "bug" in the program. As in example 4.1, only the set of assumptions $\Delta_0 = \{not\ guilty(father(Mary))\}$ is relevant to the conclusion.

However, according to the stable semantics it is necessary to ensure the existence of a stable set of assumptions $\Delta$ that contains $\Delta_0$. To be a stable set, $\Delta$ either must contain $not\ accused(Mary)$ or must imply $accused(Mary)$ (but not both). But the only way to derive $accused(Mary)$ is by means of the assumption $not\ accused(Mary)$. Consequently, there exists no such stable set $\Delta$.

Thus, the example shows that in order to determine whether or not the conclusion $innocent(father(Mary))$ holds according to the stable semantics, it is necessary to consider assumptions, such as $not\ accused(Mary)$, that are not relevant to the conclusion in any way. In contrast, the admissibility semantics completely avoids the need to consider such irrelevant assumptions, since $\Delta_0$ alone is admissible.

Because stable sets of assumptions are infinite in the general case, most proof procedures for the stable semantics of logic programs restrict attention to the finite case (e.g. see [7, 36, 35, 3, 41, 22, 2]). To achieve this restriction, they either prohibit the use of function symbols or they restrict the number of ground instances of the program in some other way.

Despite these restrictions, the stable semantics has been successfully applied to a large class of problem domains, including combinatorics and planning [10]. Typically in these applications, the input and output are represented by sets of ground relations, instead of by terms as in ordinary logic programming. It is not obvious, however, to what extent the success of this approach is due to the semantics and to what extent it is due to the associated programming style. To the extent that it is due to the programming style, it is possible that similar results can be achieved with proof procedures for

other semantics, such as the proof procedure for the admissibility semantics presented in this paper.

As we have already noted, our proof procedure for the admissibility semantics can be used for many of the applications for which the stable semantics has proved to be useful. However, it can also be used for more conventional logic programming applications, in which stable extensions are often infinitely large and can not be computed.

# 5 Towards a proof procedure for the admissibility semantics

Both the stable and the admissibility semantics are a semantics in the sense that they provide a non-constructive specification of default consequence. A proof procedure, on the other hand, needs not only to be constructive but also to be as efficient as possible.

A potential source of non-constructivity and inefficiency is the need, in the general case, to generate closed sets $\Delta$ and closed attacks $A$ against $\Delta$. This necessitates generating potentially all deductive consequences of a set of assumptions to determine what other hidden assumptions might be entailed.

This problem of generating hidden assumptions does not exist in flat assumption-based frameworks. However, even in non-flat frameworks, it is not always prohibitively expensive. For example in the non-flat auto-epistemic and non-monotonic modal logic theory $T = \{\neg Lp\}$, the "hidden" assumption $\neg Lp$ belongs to every closed set of assumptions. Therefore, in this example, to turn any set of assumptions $\Delta$ into a closed set, it suffices simply to add this hidden assumption to $\Delta$.

For the sake of simplicity, we shall focus on developing a proof procedure for flat assumption-based frameworks. The first two presentations of the proof procedure (in terms of sets of attacking and defending assumptions) can be adapted to non-flat frameworks, simply by replacing every reference to "attack" by "closed attack".

Another source of non-constructivity and inefficiency is the requirement that an admissible set $\Delta$ attacks *all* attacks $A$ against $\Delta$. This requirement is non-constructive because, given any attack $A_0$, any superset $A$ of $A_0$ is also an attack and there can be infinitely many such supersets.

Ideally, it would be sufficient to generate and counter-attack only the *minimal* attacks against $\Delta$. But in the general case it is not even semi-decidable to ensure that an attack is minimal. Therefore instead of minimal attacks we generate and counter-attack only covering sets of attacks.

**Definition 5.1** A set $\mathcal{C}$ of sets of assumptions is a *covering set of attacks* against a set of assumptions $\Delta$ if and only if each element of $\mathcal{C}$ is an attack against $\Delta$ and for every attack $A$ against $\Delta$, there exists $C \in \mathcal{C}$ such that $C \subseteq A$.

Requiring $\Delta$ to counter-attack all attacks in some covering set of attacks greatly reduces the number of attacks that $\Delta$ needs to counter-attack, and therefore it greatly improves the prospects of developing an efficient proof procedure.

**Example 5.1** Consider the logic program

$$\begin{array}{lcl} q & \leftarrow & not\ \ r, not\ \ p \\ q & \leftarrow & not\ \ r \\ r & \leftarrow & \end{array}$$

All attacks against the admissible set $\Delta = \{not\, q\}$ are supersets of the attack $A_0$=$\{not\, r\}$. Thus, in order to check that $\Delta$ is admissible, it is necessary only to check that $\Delta$ counter-attacks the covering set $\{\{not\, r\}\}$ of attacks against $\Delta$.

Note that if the empty set attacks $\Delta$, then the set consisting of only the empty set is a covering set of attacks, and consequently there is no way for $\Delta$ to defend itself against all attacks. On the other hand, if there are no attacks against $\Delta$, then the empty set is the only covering set of attacks, and therefore $\Delta$ trivially defends itself against all attacks.

The following theorem is simply a restatement of the definition of admissiblity in terms of counter-attacking a covering set of attacks.

**Theorem 5.1** Given a (flat) assumption-based framework $\langle T, Ab, {}^{-}\rangle$, a set of assumptions $\Delta \subseteq Ab$ is admissible if and only if

1. $\Delta$ does not attack itself, and
2. there exists some covering set $\mathcal{C}$ of attacks against $\Delta$,
   such that for every $A \in \mathcal{C}$, $\Delta$ attacks $A$.

# 6 Proof trees for flat assumption-based frameworks

We decompose the task of establishing whether a given set of sentences $Q$ is a credulous default consequence of $\langle T, Ab, {}^{-}\rangle$ under the admissibility semantics into two subtasks:

1. find a set $\Delta_0 \subseteq Ab$ such that $T \cup \Delta_0 \vdash Q$;
2. find a set $\Delta \subseteq Ab$ such that $\Delta_0 \subseteq \Delta$ and $\Delta$ is admissible.

**Example 6.1** Consider the problem of showing that the query $not\, p$ is a default consequence of the program

$$\begin{array}{lcl} p & \leftarrow & not\ \ q \\ q & \leftarrow & not\ \ r \end{array}$$

We do this in two stages: First construct the set $\Delta_0 = \{not\, p\}$. Then extend $\Delta_0$ to the admissible set $\Delta = \{not\, p,\ not\, r\}$.

In this section we focus on the second task, of extending a given $\Delta_0$ to an admissible set $\Delta$. The computation of $\Delta_0$ will be incorporated into the proof procedure in section 8.

The notion of abstract proof tree in this section captures the essence of our proof procedure for the admissibility semantics. In later sections, we will show how to derive proof trees one step at a time, top-down, starting from the root node and how to incorporate monotonic inference into the proof tree.

Abstract proof trees demonstrate the incremental extension of an initial set of assumptions $\Delta_0$ to an admissible set $\Delta$. Each node in the proof tree is labelled [5] by a set of assumptions and has the status of either a *defence* node or an *attack* node, but not both. The set $\Delta$ is the set of all assumptions belonging to the defence nodes in the tree. For every defence node labelled by a set of assumptions $D$, the children of the node are a set of attack nodes labelled by a covering set of attacks against $D$. For every attack node labelled by a set of assumptions $A$, there is a single child node that is a defence node labelled by an attack against $A$.

A proof tree can be seen as representing a dispute between a defendant and an opponent who take turns in exchanging arguments, in the form of sets of assumptions. The defendant and opponent share the same beliefs, in the form of the assumption-based framework underlying the construction of the tree. The set of assumptions $\Delta_0$, put forward by the defendant, is the starting point of the dispute. To win the dispute, the defendant must find a defence against every attack by the opponent.

Thus, viewed top-down, breadth-first and level by level, the dispute is a sequence of alternating defences and attacks:

$\mathcal{D}_0 = \Delta_0, \mathcal{A}_0, \mathcal{D}_1, \ldots, \mathcal{A}_i, \mathcal{D}_{i+1}, \ldots$

The defendant puts forward some defence $\mathcal{D}_i$, and the opponent responds with a covering set of attacks $\mathcal{A}_i$ against $\mathcal{D}_i$. The defendant counter-attacks with some further defence $\mathcal{D}_{i+1}$ sufficient to counter-attack every attack in $\mathcal{A}_i$. The defender wins the dispute either if it never ends or if, for some $n$, there are no attacks against $\mathcal{D}_n$.

A proof tree is an abstraction of a dispute, in that it does not show how the proof tree is constructed, and it does not show the search for defences. It is also an abstraction, in that it does not specify complete arguments, but only the assumptions underlying those arguments.

It is worth noting, however, that proof trees can be constructed not only top-down, but also bottom-up, as well as mixed top-down and bottom-up. Thus proof trees can form the basis for the development of a wide variety of different types of proof procedures.

**Example 6.2** Consider the logic program

$$p \leftarrow not\ q$$
$$q \leftarrow not\ r, not\ p$$

[5] We distinguish between nodes and their labels, because the same set of assumptions can label different nodes. Alternatively and equivalently, we could treat proof trees as multi-sets of nodes, that is to say as nodes that are occurrences of sets of assumptions.

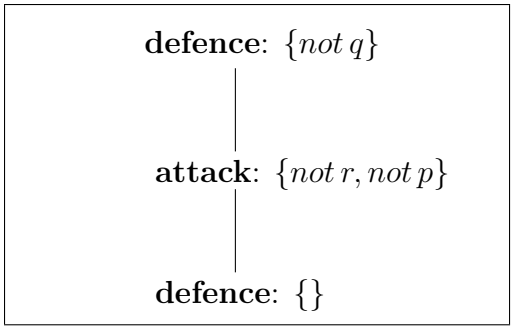


Figure 1: Proof tree for $\Delta_0 = \{not\, q\}$ in example 6.2

$$
\begin{array}{lll}
q & \leftarrow & not\ \ t, u \\
r & \leftarrow & \\
t & \leftarrow &
\end{array}
$$

The problem is to show that p is a default consequence of the program. To do so, we construct a proof tree whose root is a defence node labelled by $\{not\, q\}$. The root has a single child, which is an attack node, labelled by $\{not\, r, not\, p\}$. This attack node also has a single child, which is a defence node, labelled by the empty set, which attacks the culprit $not\, r$ in $\{not\, r, not\, p\}$. Therefore it is also a leaf node, because there is no attack against an empty set. The full tree is given in figure 1

Note that $not\, p$ is another possible choice of culprit in $\{not\, r, not\, p\}$, and this is attacked by $\{not\, q\}$. Therefore, there is an infinite proof tree in which the defence $\{not\, q\}$ and the attack $\{not\, r, not\, p\}$ alternate. This infinite proof tree is another successful proof for the defendant, because the defendant has a counter-attack for every attack put forward by the opponent.

A proof tree can be viewed as an and-tree ("and" because it includes *all* attacks in a covering set for all attacks against *all* defence nodes in the tree). However, the search space for a proof can be viewed as an and-or-tree ("or" because it includes *some* defence against every attack node in the tree). To obtain a proof procedure from the definition of proof tree, we need to specify a strategy for searching the and-or tree, to find a defence against every attack.

**Definition 6.1** A **proof tree** for a set of assumptions $\Delta_0$ is a (possibly infinite) tree $\mathcal{T}$ such that

1. Every node of $\mathcal{T}$ is labelled by a set of assumptions and is assigned the status of **defence** node or **attack** node, but not both.

2. The root is a defence node labelled by $\Delta_0$.

3. For every defence node $N$ labelled by a set of assumptions $D$, the children of $N$ are a set of attack nodes labelled by a covering set $\mathcal{C}$ of attacks against $D$. (One such child for every attack in $\mathcal{C}$.)

   Note that if there are no attacks against $D$, then $N$ is a terminal node. In particular, $N$ is a terminal node, if $D$ is an empty set of assumptions.

4. For every attack node $N$ labelled by a set assumptions $A$, there exists exactly one child of $N$ which is a defence node labelled by an attack against an assumption $\delta \in A$. $\delta$ is often referred to as the *culprit* in A.

The set of all assumptions belonging to the defence nodes in $\mathcal{T}$ is called the **defence set** of $\mathcal{T}$.

The definition of proof tree incorporates the requirement that an admissible set $\Delta$ attacks every attack against $\Delta$. Moreover, it shows how $\Delta$ can be constructed incrementally, starting from $\Delta_0$. However, it does not incorporate the further requirement that $\Delta$ does not attack itself. This further requirement is incorporated in the definition of *admissible* proof tree:

**Definition 6.2** A proof tree $\mathcal{T}$ for $\Delta_0$ is **admissible** if and only if no culprit in the label of an attack node belongs to the defence set of $\mathcal{T}$.

Notice that the definition of admissibility does not require that attack nodes and defence nodes have no assumptions in common. This is because the opponent can use the defendant's own assumptions against the defendant. However, if the opponent can attack the defendant using only the defendant's own assumptions, then the defendant looses the dispute. To win the dispute, the defendant needs to identify and counter-attack in every attack of the opponent some culprit that does not belong to the defendant's own defence.

The admissibility requirement does not necessarily imply that a proof procedure that searches for admissible proof trees needs to incorporate an explicit admissibility check. As we will see in theorem 6.2, finite proof trees are guaranteed to be admissible even without such a check.

The first part of the following theorem is a soundness result, stating that the defence set of an admissible proof tree is an admissible superset of $\Delta_0$. The second part is a completeness result, guaranteeing that, for any given admissible superset $\Delta$ of $\Delta_0$, there exists a proof tree for $\Delta_0$ whose defence set is $\Delta$.

**Theorem 6.1** Given a set of assumptions $\Delta_0 \subseteq Ab$:

i) If $\mathcal{T}$ is an admissible proof tree for $\Delta_0$ and $\Delta$ is the defence set of $\mathcal{T}$,
   then $\Delta_0 \subseteq \Delta$ and $\Delta$ is admissible.

ii) If $\Delta \subseteq Ab$ is an admissible set of assumptions such that $\Delta_0 \subseteq \Delta$,
   then there exists an admissible proof tree for $\Delta_0$ with defence set $\Delta$.

Admissible proof trees can be non-constructive, both because they can be infinite in breadth and because they can be infinite in depth: They can be infinite in breadth, because covering sets can be infinite. They can be infinite in depth, because there can be infinitely long branches of alternating attack and defence nodes. The following examples illustrate these two possibilities.

**Example 6.3** Consider the logic program

$$p \leftarrow q(X), \mathit{not}\, r(X)$$
$$q(\mathit{succ}(X)) \leftarrow q(X)$$
$$q(0)$$
$$r(X)$$

and the set of assumptions $\Delta_0 = \{\mathit{not}\, p\}$. The Herbrand universe is the infinite set $\{\mathit{succ}^i(0) | i \geq 0\}$. There are infinitely many (finite) attacks against $\Delta_0$, i.e.

$\{\mathit{not}\, r(0)\}$, $\{\mathit{not}\, r(\mathit{succ}(0))\}$, $\{\mathit{not}\, r(\mathit{succ}(\mathit{succ}(0)))\}$, etc.

Therefore, the smallest covering set $\mathcal{C}$ for all attacks against $\Delta_0$ is

$\{\{\mathit{not}\, r(\mathit{succ}^i(0))\} | i \geq 0\}$

and every proof tree for $\Delta_0$ is infinite in width. Note that every attack in $\mathcal{C}$ is counter-attacked by the empty set, and thus there exists an admissible proof tree for $\Delta_0$ with defence set $\Delta_0$; and therefore $\Delta_0$ is admissible.

**Example 6.4** Consider the logic program

$$p(X) \leftarrow \mathit{not}\, q(\mathit{succ}(X))$$
$$q(X) \leftarrow \mathit{not}\, p(\mathit{succ}(X))$$

and the set of assumptions $\Delta_0 = \{\mathit{not}\, p(0)\}$. There exists an admissible proof tree for $\Delta_0$, which consists of a single infinite branch of alternating defence and attack nodes. For every defence node labelled by a set of assumptions of the form $\{\mathit{not}\, p(\mathit{succ}^{2i}(0))\}$, there is a single child node, which is an attack node labelled by $\{\mathit{not}\, q(\mathit{succ}^{2i+1}(0))\}$. Similarly, for every attack node labelled by a set of assumptions of the form $\{\mathit{not}\, q(\mathit{succ}^{2i+1}(0))\}$ there is a single child node, which is a defence node labelled $\{\mathit{not}\, p(\mathit{succ}^{2i+2}(0))\}$. The set of assumptions

$\Delta = \{\mathit{not}\, p(\mathit{succ}^{2i}(0)) | i \geq 0\}$

belonging to the labels of defence nodes is an admissible superset of $\Delta_0$.

In the special case of proof trees that are finite in depth, the admissibility check is unnecessary [6]:

[6] However, as example example 7.2 later shows, although an explicit admissibility check is unnecessary for such proof trees, it can none-the-less increase the efficiency of recognizing that a partial proof tree can not be extended into a full proof tree, and therefore it can increase the efficiency of proof procedures that search for proof trees.

**Theorem 6.2** Any proof tree that has no infinite branches is an admissible proof tree.

All proof trees for stratified frameworks (see section 4) are finite in depth. This is because the well-ordering of assumptions guarantees the absence of infinite branches. Therefore, an explicit admissibilty check is unnecessary for such frameworks.

Admissible proof trees represent the idealised output of a proof procedure, which, to be complete, would have to be capable of constructing infinitely large proof trees. In practice, therefore, any feasible proof procedure, restricted to constructing finite proof trees, will necessarily be incomplete in the general case. The definition of derivation in the next section is restricted to finite derivations. Therefore, any proof procedure that generates such derivations is incomplete in the general case.

# 7 Derivations with Filtering by Defences

Informally, a derivation is a step by step, top-down construction of a proof tree, starting from the root node. Each step corresponds to selecting a node in the frontier of the proof tree and generating all of its children. Any node can be selected for this purpose. Different selections give rise to different derivations, but do not affect completeness, because they simply represent different ways of generating the same proof tree. This selection can be formalised by means of a *selection function*, analogous to the use of such functions in the formal definitions of SLD and SLDNF in logic programming.

The selection of nodes in the proof tree is different from the search for defences against attacks. The search for defences requires a search strategy, which turns the definition of derivation into the specification of a proof procedure for finding derivations.

Because our ultimate goal is to develop effective proof procedures, we restrict the definition of derivation to derivations of finite length. However, as we will see later, the corresponding proof trees can sometimes be infinite.

Each step in a derivation corresponds to the selection of a node in the frontier of a proof tree and generating a new frontier. We represent this frontier by a data structure of the form $\langle \Delta_i, \mathcal{D}_i, \mathcal{A}_i \rangle$, where $\Delta_i$ is the set of defence assumptions generated so far, $\mathcal{D}_i$ is the set of defence assumptions that have not yet been selected, and $\mathcal{A}_i$ is the set of attacks that have not yet been selected. [7]

The set $\Delta_i$ is used to filter defence assumptions in $\mathcal{D}_i$, so they are not considered redundantly, more than once, and to filter culprits $\sigma \in A$ in attacks $A \in \mathcal{A}_i$, so that the defence set $\Delta$ constructed by the derivation does not attack itself.

As we will see later, because of filtering of defence assumptions, finite derivations can sometimes correspond to infinite proof trees. Therefore, because finite derivations can correspond to infinite proof trees, filtering of attack assumptions is necessary to guarantee admissibility.

[7] The "sets" $\Delta_i, \mathcal{D}_i, \mathcal{A}_i$ can be true sets or multi-sets (i.e. occurrences of sets), depending upon whether or not new assumptions and new attacks are filtered to determine whether or not they already belong to the set or multi-set to which they are added.

**Definition 7.1** A **filtered derivation of a defence set** $\Delta$ for a set of assumptions $\Delta_0$ is a finite sequence of triples
$\langle \Delta_0, \mathcal{D}_0, \mathcal{A}_0 \rangle, \ldots, \langle \Delta_i, \mathcal{D}_i, \mathcal{A}_i \rangle, \ldots, \langle \Delta_n, \mathcal{D}_n, \mathcal{A}_n \rangle$
where
$\mathcal{D}_0 = \Delta_0$
$\mathcal{A}_0 = \mathcal{D}_n = \mathcal{A}_n = \{\}$,
$\Delta = \Delta_n$,
and for every $0 \leq i < n$, only one $\sigma$ in $\mathcal{D}_i$ or one $A$ in $\mathcal{A}_i$ is selected, and:

1. If $\sigma \in D_i$ is selected then
   $\mathcal{D}_{i+1} = \mathcal{D}_i - \{\sigma\}$,
   $\Delta_{i+1} = \Delta_i$,
   $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \mathcal{C}$
   where $\mathcal{C}$ is a covering set of attacks against $\sigma$.

2. If $A \in \mathcal{A}_i$ is selected then there is an attack $D$ against some culprit $\sigma \in A$ such that
   $\sigma \notin \Delta_i$
   $\mathcal{A}_{i+1} = \mathcal{A}_i - \{A\}$
   $\mathcal{D}_{i+1} = \mathcal{D}_i \cup D - \Delta_i$,
   $\Delta_{i+1} = \Delta_i \cup D$.

The use of defence assumptions $\Delta_i$ to filter defence assumptions $\mathcal{D}_i$ can turn an infinite proof tree into a finite derivation. This is illustrated by the following example:

**Example 7.1** Consider the logic program

$$p \leftarrow not\, q$$
$$q \leftarrow not\, p$$

and the query $\{not\, p\}$. There exists a two-step derivation of the defence set $\{not\, p\}$:
$\langle \{not\, p\}, \{not\, p\}, \{\} \rangle$,
$\langle \{not\, p\}, \{\}, \{\{not\, q\}\} \rangle$,
$\langle \{not\, p\}, \{\}, \{\} \rangle$.
Without filtering defence assumptions, it would be necessary to generate an infinite sequence of alternating defences $\{not\, p\}$ and covering sets of attacks $\{\{not\, q\}\}$, and the derivation would never terminate. Thus, filtering defence assumptions allows finite derivations to be constructed in some cases where the corresponding proof tree is infinite.

Because filtering of defence assumptions can turn an infinite proof tree into a finite derivation, filtering of culprits is necessary for admissibility :

**Example 7.2** Consider the same logic program

$$p \leftarrow not\, q$$
$$q \leftarrow not\, p$$

but this time with the query $\{not\, p, not\, q\}$. With filtering of defence assumptions, but without filtering of culprits, there would exist a derivation:
$\langle\{not\, p, not\, q\}, \{not\, p, not\, q\}, \{\}\rangle$,
$\langle\{not\, p, not\, q\}, \{not\, q\}, \{\{not\, q\}\}\rangle$,
$\langle\{not\, p, not\, q\}, \{not\, q\}, \{\}\rangle$,
$\langle\{not\, p, not\, q\}, \{\}, \{\{not\, p\}\}\rangle$,
$\langle\{not\, p, not\, q\}, \{\}, \{\}\rangle$.
This derivation corresponds to an infinite proof tree that is not admissible.

Filtering of culprits can also improve efficiency by turning an infinitely failed attempt to generate a derivation into a finite failure.

**Example 7.3** Consider the logic program $p \leftarrow not\ p$ and the query $\{not\, p\}$.
The only partial derivation that can be generated by any proof procedure terminates in failure after only one step:

$\langle\{not\, p\}, \{not\, p\}, \{\}\rangle$,
$\langle\{not\, p\}, \{\}, \{\{not\, p\}\}\rangle$.

This partial derivation can not be extended, because the only candidate culprit in the only attack in $\mathcal{A}_1$ belongs to $\Delta_1$.
Without filtering culprits, a proof procedure would attempt to generate an infinite sequence of alternating defences $\{not\, p\}$ and covering sets of attacks $\{\{not\, p\}\}$, and would never terminate.

Note that the definition of derivation could be modified to incorporate also filtering by means of culprits, by adding an extra component $\mathcal{C}_i$ into the data structure $\langle\Delta_i, \mathcal{D}_i, \mathcal{C}_i, \mathcal{A}_i\rangle$, where $\mathcal{C}_i$ is the set of potential culprits already selected for counter-attack earlier in the derivation. The definition of derivation would then need to be modified in step 2, where $\sigma \in A$ is selected as potential culprit and $D$ is chosen to counter-attack $\sigma$, by adding the extra conditions $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{\sigma\}$ and $\mathcal{C}_{i+1} \cap D = \{\}$.
This modification might seem to be necessary to guarantee admissibility, but it is not. The situation is analogous to the situation with finite proof trees: Incorporating such an additional admissibility check can shorten the search for derivations, but is not necessary for admissibility.
The correspondence between derivations and proof trees is obvious from the similarity between their definitions: the step from $\langle\Delta_i, \mathcal{D}_i, \mathcal{A}_i\rangle$ to $\langle\Delta_{i+1}, \mathcal{D}_{i+1}, \mathcal{A}_{i+1}\rangle$ in a derivation corresponds to the relationship between a node and its children in a proof tree. Consequently, derivations have the same defence sets as their corresponding proof

trees. However, many different derivations with different selection functions can correspond to the same proof tree. If there is a derivation with one selection function, then there is a derivation with any other selection function. These observations justify the following theorems.

**Theorem 7.1** For every filtered derivation of a defence set $\Delta$ for a set of assumptions $\Delta_0$, there exists a (possibly infinite) admissible proof tree for $\Delta_0$ with defence set $\Delta$.

The following corollary is a soundness result for derivations, which follows directly from theorem 6.1.

**Corollary 7.1** For every filtered derivation of a defence set $\Delta$ for a set of assumptions $\Delta_0$, the defence set $\Delta$ is an admissible set such that $\Delta_0 \subseteq \Delta$.

Completeness holds for finite derivations, in the sense that for every finite proof tree there exists a "corresponding" finite derivation. However, because of filtering, the correspondence is not exact.

**Theorem 7.2** For every finite proof tree for a set of assumptions $\Delta_0$ with defence set $\Delta$, and for every selection function, there exists a derivation for $\Delta_0$ of a defence set $\Delta' \subseteq \Delta$.

The notions of proof tree and derivation presented in this and the previous section are defined for flat assumption-based frameworks. As mentioned earlier, the definitions can be adapted to non-flat frameworks, simply by replacing all references to "attack" by references to "closed attack".

However, the definitions of concrete proof tree and concrete derivation in the next two sections can not be so readily adapted to non-flat frameworks. This is because they incorporate the use of monotonic inference to generate sets of assumptions. To adapt the definitions to non-flat frameworks, it would be necessary to show how the monotonic inference rules can be used to generate hidden assumptions, and this is not straight-forward in the general case.

# 8 Proof trees incorporating monotonic inference

In this section, we use the monotonic inference rules backwards both to generate attacks and defences and to generate the supporting set of assumptions $\Delta_0$ for an initial query $Q$. For this purpose, we generalise the labels of nodes from sets of assumptions to sets of sentences.

To apply the inference rules backwards, we select some non-assumption sentence $\sigma$ in the set of sentences $R$ labelling a node $N$. We then create a child of $N$ labelled by the set of sentences $R - \{\sigma\} \cup S$ for some inference rule $\frac{S}{\sigma} \in \mathcal{R}$. Depending on

whether $N$ is a defence node or an attack node, $N$ has either one such child (when $N$ is a defence node) or it has all such children (when $N$ is an attack node).

In the logic programming case, these labels of the children of $N$ are just ground resolvents of the label at $N$. In the more general case, they generalise the notion of resolvent. Just as in the case of resolution, their definition can be further generalised to incorporate unification.

This generalisation to incorporate unification is especially important in the cases of logic programming and default logic. Without unification, proof procedures for generating derivations need to consider all ground instances of the inference rules. With unification, they can employ the inference rules, containing variables, directly. We will return to this issue in the next section.

To minimise the use of new terminology, when there is no possibility of confusion, we will use the same terminology, *proof tree* and *derivation*, as we have before. However, where confusion might arise, we use the terms *abstract proof tree* and *abstract derivation* for the trees and derivations of the earlier sections and *concrete proof tree* and *concrete derivation* for those defined in this section and the next section.

A concrete proof tree can be seen as a concrete representation of a dispute, which displays, not only the assumptions made by the two adversaries, but also the monotonic inference steps that the adversaries use to construct their arguments.

**Definition 8.1** A **proof tree** for a set of sentences $Q$ is a (possibly infinite) tree $\mathcal{T}$ such that

1. Every node of $\mathcal{T}$ is labelled by a set of sentences and is assigned the status of **defence** node or **attack** node, but not both.

2. The root of $\mathcal{T}$ is a defence node labelled by Q.

3. Let $N$ be a defence node labelled by $D$. If $D$ is empty, then $N$ is a terminal node. If $D$ is not empty and the selected sentence in $D$ is an assumption $\delta$, then there exists one child of $N$, which is an attack node labelled by $\{\overline{\delta}\}$ and a second child of $N$ that is a defence node labelled by $D$ - $\{\delta\}$. If $\delta$ is not an assumption, then there exists some inference rule $\frac{S}{\delta} \in \mathcal{R}$ and there exists exactly one child of $N$, which is a defence node labelled by $D - \{\delta\} \cup S$.

4. Let $N$ be an attack node labelled by $A$. There exists some selected sentence $\alpha$ in $A$. If $\alpha$ is an assumption, then $\alpha$ is the culprit at $N$, and there exists exactly one child of $N$, which is a defence node labelled by $\{\overline{\alpha}\}$. If $\alpha$ is not an assumption and there exists no inference rule $\frac{S}{\alpha} \in \mathcal{R}$, then $N$ is a terminal node. Otherwise, the children of $N$ are a set of attack nodes labelled by the sets of sentences $A - \{\alpha\} \cup S$, where $\frac{S}{\alpha} \in \mathcal{R}$. (There being one such child for each such inference rule.)

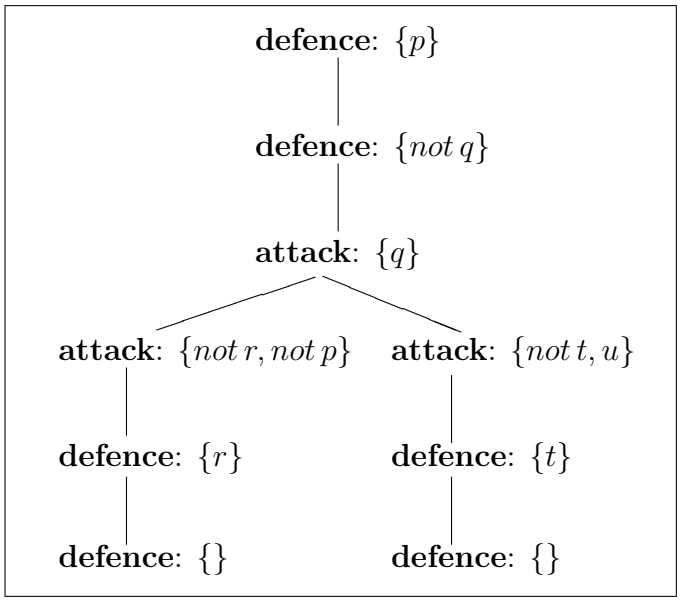


Figure 2: Proof tree for $Q = \{p\}$ in example 8.1

5. There is no infinite branch consisting only of defence nodes.

The set $\Delta$ of all assumptions belonging to defence nodes in $\mathcal{T}$ is called the **defence set of** $\mathcal{T}$.

**Definition 8.2** A proof tree $\mathcal{T}$ for a set of sentences $Q$ is **admissible** if and only if no culprit at an attack node belongs to the defence set of $\mathcal{T}$.

**Example 8.1** Consider the logic program

$$
\begin{array}{lll}
p & \leftarrow & not\ q \\
q & \leftarrow & not\ r, not\ p \\
q & \leftarrow & not\ t, u \\
r & \leftarrow & \\
t & \leftarrow &
\end{array}
$$

The program is the same as that of example 6.2, and the problem is the same, namely to show that $p$ is a default consequence of the program. The concrete proof tree is given in figure 2.

Note that the proof tree contains a branch that does not correspond to any branch in the abstract proof tree of example 6.2. This is because the branch unnecessarily

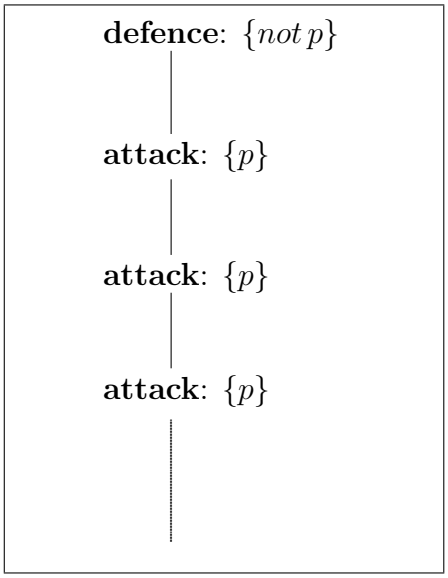


Figure 3: Concrete proof tree for $Q = \{not\, p\}$ for the program $p \leftarrow p$

defends the potential attack $\{not\, t, u\}$, which cannot be developed into a complete attack.

In concrete proof trees, any node $N$ whose selected sentence is an assumption $\sigma$, whether a defence node or an attack node, always has a single child $N'$, which is a node of the opposite status, labelled by $\{\overline{\sigma}\}$. This node represents all potential attacks against $\sigma$. Monotonic inference rules are applied backwards to such nodes, to generate descendant nodes of the same status as $N'$.

In the case where these descendant nodes are defence nodes, they represent a choice of one way to counter-attack $\sigma$. For such a counter-attack to succeed, it must generate a finite argument for $\overline{\sigma}$. For this reason, a proof tree may not contain any infinite branches consisting of defence nodes only. For example, given the logic program $p \leftarrow p$, there exists no proof tree for the query $\{p\}$.

In the case where these descendant nodes are attack nodes, they represent a potential covering set of attacks against $\sigma$. An infinite branch consisting of a succession of such attack nodes indicates a failure to generate an attack. A proof tree may, therefore, contain such infinite branches consisting of attack nodes only. For example, given the logic program $p \leftarrow p$, the query $\{not\, p\}$ has an infinite proof tree, the initial part of which is shown in figure 3.

In addition to finitely generated attacks that are defeated and to infinitely failed attempts to generate an attack, a proof tree may also contain finitely failed attempts to generate an attack. This happens when the selected sentence in the label of an attack node is a non-assumption $\sigma$ that is the conclusion of no inference rule $\dfrac{S}{\sigma} \in \mathcal{R}$.

To obtain a proof procedure from the definition of proof tree, we construct the proof tree top-down and specify the search strategy for finding defences. The simplest alternatives are to employ a depth-first or a breadth-first search. Other more sophisticated strategies are also possible. It is also possible to regard such choices as non-deterministic, in which case they can be implemented in a non-deterministic programming language, such as Prolog. In fact, our implementation of the proof procedure in Prolog specifically makes use of this possibility. The use of Prolog, of course, means that the search is depth-first, but controlled by the Prolog compiler or interpreter, rather than by the Prolog program.

Although it may be more obvious to construct proof trees top-down, as in the case of abstract proof trees, concrete proof trees can also be constructed bottom-up, as well as mixed top-down and bottom-up. Thus concrete proof trees, like abstract proof trees, can form the basis for the development of a great variety of types of proof procedure.

The following theorem states the relationship between abstract and concrete proof trees: For any concrete proof tree there corresponds an abstract proof tree having the same or smaller defence set. Similarly, for any abstract proof tree there is a corresponding concrete proof tree having the same or smaller defence set.

**Theorem 8.1** Given a set of sentences $Q$

i) For every concrete proof tree $\mathcal{T}$ for $Q$ there exists a set of assumptions $\Delta_0 \subseteq Ab$ and an abstract proof tree $\mathcal{T}'$ for $\Delta_0$ such that

   - $T \cup \Delta_0 \vdash Q$.
   - The defence set of $\mathcal{T}'$ is a subset of teh defence set of $\mathcal{T}$.
   - $\mathcal{T}$ is an admissible proof tree if and only if $\mathcal{T}'$ is an admissible proof tree.

ii) For every $\Delta_0 \subseteq Ab$ such that $T \cup \Delta_0 \vdash Q$, and for every abstract proof tree $\mathcal{T}'$ for $\Delta_0$, there exists a concrete proof tree $\mathcal{T}$ for $Q$ such that

   - The defence set of $\mathcal{T}$ is a subset of the defence set of $\mathcal{T}'$.
   - $\mathcal{T}$ is an admissible proof tree if and only if $\mathcal{T}'$ is an admissible proof tree.

The following corollary follows directly from the theorem and the soundness and completeness result for abstract proof trees. It states the soundness and a strong form of completeness for concrete proof trees. The completeness result is actually an improvement over the simple completeness we have in the abstract case. Indeed, the fact that not every possible admissible extension $\Delta$ of $\Delta_0$ can be generated as a defence set of an admissible concrete proof tree is actually an advantage, because arbitrary extensions $\Delta$ can contain assumptions that are not relevant to the defence of $\Delta_0$.

**Corollary 8.1** Given a set of sentences $Q$:

i) If $\mathcal{T}$ is an admissible concrete proof tree for $Q$ and $\Delta$ is the defence set of $\mathcal{T}$, then, $T \cup \Delta \vdash Q$ and $\Delta$ is admissible.

ii) If $\Delta \subseteq Ab$ is an admissible set of assumptions such that $T \cup \Delta \vdash Q$, then, there exists an admissible concrete proof tree for $Q$ with defence set $\Delta' \subseteq \Delta$ such that $\Delta'$ is admissible and $T \cup \Delta' \vdash Q$.

As is the case with abstract proof trees, the admissibility check is unnecessary for concrete proof trees that are finite in depth:

**Theorem 8.2** Any concrete proof tree that has no infinite branches is an admissible concrete proof tree.

For a large class of frameworks, generalising the class of acyclic logic programs [1], all branches of all concrete proof trees are finite. A framework is acyclic if there is a well-ordering of all sentences in the language of the framework such that, whenever a sentence belongs to the premise of an inference rule, then the sentence is lower in the ordering than the consequent of the inference rule.

Although proof trees can be constructed top-down, bottom-up or mixed top-down-bottom-up, derivations restrict that choice to the top-down direction.

# 9 Concrete Derivations with Filtering by Defences

Concrete derivations bear to concrete proof trees the same relationship that abstract derivations bear to abstract proof trees. As in the case of abstract derivations, we restrict concrete derivations to derivations of finite length. However, because of filtering by defences, concrete derivations sometimes correspond to infinite concrete proof trees. Concrete derivations generalise the derivations for logic programming of [19].

**Definition 9.1** A **filtered derivation of a defence set** $\Delta$ for a non-empty set of sentences $Q$ is a finite sequence of triples
$\langle \Delta_0, \mathcal{D}_0, \mathcal{A}_0 \rangle, \ldots, \langle \Delta_i, \mathcal{D}_i, \mathcal{A}_i \rangle, \ldots, \langle \Delta_n, \mathcal{D}_n, \mathcal{A}_n \rangle$
where
$\mathcal{D}_0 = Q$
$\Delta_0 = Ab \cap Q$
$\mathcal{A}_0 = \mathcal{D}_n = \mathcal{A}_n = \{\}$,
$\Delta = \Delta_n$, and
for every $0 \leq i < n$, either $\sigma$ in $\mathcal{D}_i$ or $A$ in $\mathcal{A}_i$ is selected but not both, and:

1. If $\sigma \in D_i$ is selected and $\sigma$ is an assumption, then
$\mathcal{D}_{i+1} = \mathcal{D}_i - \{\sigma\}$,
$\Delta_{i+1} = \Delta_i$,
$\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{\{\overline{\sigma}\}\}$.

2. If $\sigma \in D_i$ is selected and $\sigma$ is not an assumption, then there exists some inference rule $\frac{S}{\sigma} \in \mathcal{R}$ and
   $\mathcal{D}_{i+1} = \mathcal{D}_i - \{\sigma\} \cup S - \Delta_i$,
   $\Delta_{i+1} = \Delta_i \cup (Ab \cap S)$,
   $\mathcal{A}_{i+1} = \mathcal{A}_i$.

3. If $A \in \mathcal{A}_i$ and $\alpha \in A$ are selected and $\alpha$ is an assumption, then
   $\alpha \notin \Delta_i$
   $\mathcal{A}_{i+1} = \mathcal{A}_i - \{A\}$
   $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{\overline{\alpha}\}$,
   $\Delta_{i+1} = \Delta_i$.

4. If $A \in \mathcal{A}_i$ and $\alpha \in A$ are selected and $\alpha$ is not an assumption, then
   $\mathcal{A}_{i+1} = \mathcal{A}_i - \{A\} \cup \{A - \{\alpha\} \cup S \mid \frac{S}{\alpha} \in \mathcal{R}\}$
   $\mathcal{D}_{i+1} = \mathcal{D}_i$,
   $\Delta_{i+1} = \Delta_i$.

Note that the definition of concrete derivation could be modified, like the definition of abstract derivation, to incorporate also filtering by means of culprits. As in the case of abstract derivations, this modification has the advantage that it can result in earlier termination of unsuccessful searches.

The definition can also be modified in steps 2 and 4 above, to incorporate unification of the selected non-assumption with the conclusion of the inference rules. This modification is necessary in cases like logic programming and default logic, where infinitely many inference rules are represented by finitely many inference rule schemata.

This modification is illustrated in the following two examples:

**Example 9.1** Consider the query $\{even(s(s(0)))\}$ to the program:

$$even(0)$$
$$even(succ(X)) \leftarrow not\, even(X)$$

The second rule is actually an inference rule schema, standing for the infinitely many ground instances: $even(succ^{i+1}(0)) \leftarrow not\, even(succ^i(0))$, over the Herbrand universe $\{succ^i(0) | i \geq 0\}$.
Using unification to match selected non-assumptions with the conclusion of the inference rule schema, we obtain the following derivation:

$\langle\{\}, \{even(succ(succ(0)))\}, \{\}\rangle$,
$\langle\{\}, \{not\, even(succ(0))\}, \{\}\rangle$,

$\langle\{not\, even(succ(0))\}, \{\}, \{\{even(succ(0))\}\}\rangle,$
$\langle\{not\, even(succ(0))\}, \{\}, \{\{not\, even(0)\}\}\rangle,$
$\langle\{not\, even(succ(0))\}, \{even(0)\}, \{\}\rangle,$
$\langle\{not\, even(succ(0))\}, \{\}, \{\}\rangle.$

**Example 9.2** Consider the query $\{not\, p\}$ to the logic program:

$$p \leftarrow q(X), not\ \ r(succ(X))$$
$$q(0)$$
$$r(succ(0))$$
$$r(succ(X)) \leftarrow r(X)$$

The only way to attack the query is by using the inference rule schema $p \leftarrow q(X), not\ r(succ(X))$. Without unification, a proof procedure would have to consider separately the infinitely many potential attacks $\{q(succ^i(0)), not\ r(succ^{i+1}(0))\}$, where $i \geq 0$, associated with the ground instances of the inference rule schema.

However, by using unification instead of ground instantiation, it is possible to consider instead only the single schematic potential attack $\{q(X), not\ \ r(succ(X))\}$.

Suppose the selection rule chooses $q(X)$ in this schematic potential attack. With unification, this selection rule gives rise to the following finite derivation:

$\langle\{not\, p\}, \{not\, p\}, \{\}\rangle,$
$\langle\{not\, p\}, \{\}, \{\{q(X), not\ r(succ(X))\}\}\rangle,$
$\langle\{not\, p\}, \{\}, \{\{not\ r(succ(0))\}\}\rangle,$
$\langle\{not\, p\}, \{r(succ(0))\}, \{\}\rangle,$
$\langle\{not\, p\}, \{\}, \{\}\rangle.$

Our Prolog implementation is a meta-logic program that directly implements the unmodified definition above. It represents the potentially infinitely many ground instances of an inference rule schema by a single meta-level term, directly representing the schema itself. Thus, it is the Prolog compler/interpreter, rather than the Prolog implementation of the proof procedure, that unifies selected non-assumptions with the conclusions of the inference rules.

The correspondence between concrete derivations and concrete proof trees is similar to that between abstract derivations and abstract proof trees: One step in a derivation corresponds to the relationship between a node and its children in a proof tree. Consequently, concrete derivations have the same defence sets as their corresponding concrete proof trees:

**Theorem 9.1** For every concrete filtered derivation of a defence set $\Delta$ for a set of sentences $Q$, there exists a (possibly infinite) admissible concrete proof tree for $Q$ with defence set $\Delta$.

The following corollary is a soundness result for derivations, which follows directly from theorem 9.1 above and corollary 8.1.

**Corollary 9.1** For every filtered derivation of a defence set $\Delta$ for a set of sentences $Q$, the defence set $\Delta$ is an admissible set such that $\Delta_0 \subseteq \Delta$.

As in the case of abstract derivations, completeness holds for finite derivations:

**Theorem 9.2** For every finite concrete proof tree for a set of sentences $Q$ with defence set $\Delta$, and for every selection function, there exists a concrete filtered derivation for $Q$ of defence set $\Delta' \subseteq \Delta$.

The proofs of theorems theorem 9.1 and theorem 9.2 for the concrete case are similar to those for theorem 7.1 and theorem 7.2 for the abstract case. Because of this similarity they are not included in the appendix.

# 10 Comparisons

The proof procedure presented in this paper is a further development of the proof procedure of Dung, Kowalski and Toni [17] for computing the admissibility semantics for abstract assumption-based frameworks. The proof procedure of [17] is developed in the form of a metalogic program that is derived systematically from specifications formalised in logic, by means of logic program transformation techniques [37].

In developing the proof procedures in [17], we discovered that the notion of proof tree is the basic notion upon which all the other notions depend. For this reason, we have temporarily set aside the transformation techniques of [17], to develop the proof tree approach presented in this paper.

Kakas and Toni [48, 28] also develop an argumentation-theoretic proof procedure for the admissibility semantics as well as for the well-founded semantics [51], the weak stability semantics [26] and the acceptability semantics [27]. Their proof procedure is explicitly defined only for logic programs, but, as the authors remark and as this paper shows, they can be generalised to any flat assumption-based framework.

Kakas and Toni [48, 28] present a number of presentations of their proof procedure, parameterised for different semantics. Although they use the terminology of proof trees, all of their presentations of their proof procedure are, in fact, based on derivations, rather than on proof trees in the sense that we have defined them. Therefore, although their proof procedure is similar to our derivations, they do not have a presentation in terms of proof trees in our sense. Moreover, their use of proof tree terminology arguably obscures the nature of their proof procedure.

The proof procedures of [17] and [48, 28], like that of this paper all generalise the proof procedure for logic programming of [19]. Although the proof procedures of [48, 28] incorporate both filtering by defences and filtering by culprits, in this paper we analyse the extent to which such filtering is needed both to turn some infinite proof trees into finite derivations and to guarantee admissibility.

Although our proof procedure computes credulous default consequences relative to the admissibility semantics, it can also be used as a basis for computing (credulous

and/or sceptical) default consequences relative to other semantics. In particular, since every admissible extension is contained in a maximal admissible extension, our proof procedure is sound with respect to (credulous) maximal admissible extension semantics, i.e. partial stable model [44]/preferred extension semantics [12] in the logic programming case. Dung, Mancarella and Toni [16] present a number of proof procedures for computing sceptical consequences under the partial stable model/preferred extension semantics and credulous and sceptical consequences under the stable semantics that are parametric with respect to any procedure for computing credulous consequences under admissibility.

Various proof procedures have been defined for the stable semantics for a number of concrete logics for default reasoning. However, to the best of our knowledge, there currently exists no proof procedure for any logic for default reasoning that computes default consequences under the (credulous or sceptical) stable semantics consequences that does not have significant restrictions.

Satoh and Iwayama [45] define a proof procedure for computing the stable semantics for range-restricted logic programs that admit at least one stable model. Therefore, the proof procedure cannot be applied to the program in example 4.2. However, for example 4.1, it generates only the one relevant assumption needed to establish the desired conclusion and, therefore, it avoids in this case the need to explicitly consider the infinitely many irrelevant assumptions.

Chen and Warren [7] define a proof procedure for computing the stable semantics for finite propositional logic programs. However, they show that the proof procedure can also be used to compute credulous default consequences of non-propositional logic programs under the partial stable model semantics [44]. In example 4.2, the sentence $innocent(father(Mary))$ is a default consequence under the partial stable model semantics, and the proof procedure sucessfully demonstrates the conclusion, generating only the one relevant assumption that is needed for the conclusion.

Niemelä and Simmons [36] define proof procedures for computing default consequences under both the credulous and sceptical stable semantics for range-restricted, function-free logic programs. Because of these restrictions, the proof procedures cannot be applied to examples 4.1 and 4.2.

Satoh [46] adapts the (credulous) proof procedure in [45] to determine default consequences with respect to the semantics of default logic. The proof procedure applies to consistent default theories whose default rules are expressed in clausal form. The procedure is defined only for the propositional case, but can be applied to the non-propositional case, by appropriately instantiating variables.

Niemelä [35] also defines proof procedures for default logic. These procedures are defined for the propositional case and for a decidable fragment of first-order logic.

Barback and Lobo [3] similarly define a proof procedure for default logic. The proof procedure is sound and complete for finite semi-normal default theories, whose every subtheory is guaranteed to admit a (stable) extension.

Dung [14] introduced the idea of viewing proof procedures in logic programming as dialogue games in which an opponent and proponent attack and defend their argu-

ments. This view of proof procedures is similar to the view presented in this paper.

# 11 Conclusion

In this paper, we have presented a succession of presentations of an argumentation-theoretic proof procedure for credulous default reasoning. The most abstract of these is based upon the notion of abstract proof tree, which displays the tree structure of attacks and defences, put forward by a defendant and opponent in the course of a dispute ending as a win for the defendant.

Abstract proof trees are neutral with respect to whether they are constructed top-down, bottom-up or mixed top-down-bottom-up, and also with respect to the search strategy for searching for successful proof trees.

Derivations are obtained from proof trees, by restricting the direction for constructing proof trees to the top-down direction. Each step in a derivation corresponds to the generation of the children of a selected child at the frontier of a proof tree and the construction of a new frontier. New defence assumptions and culprits are filtered, using the accumulated set of defence assumptions already generated in the derivation. Filtering new defence assumptions sometimes results in finite derivations whose corresponding proof trees are infinite. Filtering of culprits is necessary in such cases to ensure that the final defence set generated by the derivation does not attack itself.

Like abstract proof trees, abstract derivations are also neutral with respect to the search strategy for searching for successful derivations.

Although we define abstract proof trees and abstract derivations explicitly only for flat frameworks, they can be adapted for non-flat frameworks by replacing all references to "attacks" by references to "closed attacks".

We define concrete proof trees and concrete derivations by specifying how to incorporate monotonic inference into abstract proof trees and abstract derivations, respectively. This restricts the resulting proof procedures to flat assumption-based frameworks. In other respects, however, concrete proof trees and concrete derivations bear a similar relationship to that between abstract proof trees and abstract derivations.

We have implemented in Prolog a proof procedure that searches for concrete derivations. The implementation is a non-deterministic meta-logic program. The proof procedure inherits its depth-first search strategy and selection function from the Prolog interpreter/compiler; and it similarly inherits its use of unification for applying inference rule schemata from the Prolog interpreter/compiler.

Our argumentation-theoretic proof procedure is defined for the admissibility semantics. However, as Kakas and Toni have shown [48, 28], it is possible to define proof procedures, for a number of different semantics, including the admissibility semantics, in a uniform, parametric manner. It would be interesting, therefore, to adapt the argumentation-theoretic proof procedure of this paper to the range of semantics investigated in [48, 28]. It would also be interesting to explore the applicability of these proof procedures to the legal reasoning domain and to study their relationships

to the argumentation-theoretic proof procedures that have been developed specifically for legal reasoning [40].

## Acknowledgements

This research was supported by the Fujitsu Research Laboratories and by the EEC activity KIT011-LPKRR.

# References

[1] K.R. Apt and R. Bol, Logic programming and negation: a survey, *Journal of Logic Programming* 19-20, pp. 9-71, (1994).

[2] A.B. Baker, M.L. Ginsberg, A theorem prover for prioritised circumscription. *Proc. IJCAI'89*, Morgan Kaufmann, pages 463–467

[3] M.D. Barback, J. Lobo, A resolution-based procedure for default theories with extensions. *Proc. Non-monotonic Extensions of Logic Programs: Theory and Applications* (J. Dix, L.M. Pereira and T. Przymusinski eds.) Springer-Verlag, LNAI 927 (1995) pages 101–126

[4] A. Bondarenko, P.M. Dung, R.A. Kowalski, F. Toni, An abstract, argumentation-theoretic framework for default reasoning. *Artificial Intelligence* 93 (1,2) (1997) pages 63–101

[5] A. Bondarenko, F. Toni, R.A. Kowalski, An assumption-based framework for non-monotonic reasoning. *Proc. 2nd International Workshop on Logic Programming and Non-monotonic Reasoning* (A. Nerode and L. Pereira eds.) MIT Press (1993) pages 171–189

[6] G. Brewka, J. Dix. K. Konolige, *Nonmonotonic reasoning: an overview*. CLSI Lecture Notes 73 (Standford, California) (1997)

[7] W. Chen, D.S. Warren, Computation of stable models and its integration with logical query processing.
Available by ftp `ftp.ms.uky.edu:pub/lpnmr/chen2.ps` (1994)

[8] J.C. Dernbach, R.V. Singleton II, *A practical guide to legal writing and legal method*. Rothman & Co. (1981)

[9] Y. Dimopoulos, A.C. Kakas, Logic programming without negation as failure. *Proc. ILPS'95*, MIT Press

[10] Y. Dimopoulos, B. Nebel, and J. Koehler, Encoding planning problems in non-monotonic logic programs, Proc. European Conference on Planning 1997 (ECP-97), Springer Verlag, 1997, 169-181

[11] Y. Dimopoulos, B. Nebel, F. Toni, On the Computational Complexity of Assumption-based Argumentation for Default Reasoning, Artificial Intelligence 141, pp 57-78, October 2002

[12] P.M. Dung, An argumentation theoretic foundation of logic programming *J. Logic Programming* 22 (1995), 151–177

[13] P.M. Dung, The acceptability of arguments and its fundamental role in non-monotonic reasoning and logic programming and n-person game *Artificial Intelligence* 77 (1995), 321-357

[14] P.M. Dung, Logic Programming as Dialog-Game *Technical Report* (1993)

[15] P.M. Dung, P. Mancarella, F. Toni, Argumentation-based proof procedures for credulous and sceptical non-monotonic reasoning, *Computational Logic: Logic Programming and Beyond – Essays in Honour of Robert A. Kowalski*, Springer Verlag LNAI 2408, 289–310, 2002

[16] P.M. Dung, R.A. Kowalski, F. Toni, Synthesis of proof procedures for default reasoning. *Proc. LOPSTR'96*, Springer Verlag LNCS 1207

[17] K. Eshghi, R.A. Kowalski, Abduction through deduction. Imperial College Technical Report (1988)

[18] K. Eshghi, R.A. Kowalski, Abduction compared with negation as failure. *Proc. ICLP'89*, MIT Press

[19] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming. *Proc. ICSLP'88*, MIT Press

[20] M.A. Gilbert, *How to win an argument*. J. Wiley & Sons, Inc. (1995)

[21] M.L. Ginsberg, A circumscriptive theorem prover. *Artificial Intelligence* 39 (1989) pages 209–230

[22] V.A. Howard, J.H. Barton, *thinking on paper*. Quill (1986)

[23] A.C. Kakas, R.A. Kowalski, F. Toni, Abductive logic programming. *Journal of Logic and Computation* 2(6) (1993) pages 719-770

[24] A.C. Kakas, R.A. Kowalski, F. Toni, The role of abduction in logic programming. *Handbook of Logic in Artificial Intelligence and Logic Programming* 5 (1998) pages 235-324, Oxford University Press

[25] A.C. Kakas, P. Mancarella, Stable theories for logic programs. *Proc. ISLP'91*, MIT Press

[26] A.C. Kakas, P. Mancarella, P.M. Dung, The acceptability semantics for logic programs. *Proc. ICLP'94*, MIT Press, pages 504–519

[27] A.C. Kakas, F. Toni, Computing argumentation in logic programming. *Journal of Logic and Computation* 9 pages 515-562 (1999)

[28] R. A. Kowalski, F. Toni, Abstract argumentation. *Journal of Artificial Intelligence and Law* 4(3–4) pages 275–296, 1996, Special Issue on Logical Models of Argumentation, Kluwer Academic Publishers

[29] V.W. Marek, M. Truszczynski, *Nonmonotonic logic: context-dependent reasoning.* Springer Verlag (1993)

[30] J. McCarthy, Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence* (13) (1980) pages 27–39

[31] D. McDermott, Nonmonotonic logic II: non-monotonic modal theories. *JACM* 29(1) (1982)

[32] R. Moore, Semantical considerations on non-monotonic logic. *Artificial Intelligence* 25 (1985)

[33] L. Morgenstern, Inheritance comes of age: applying nonmonotonic techniques to problems in industry. *Artificial Intelligence* (103)1–2 (1998) pages 237–271

[34] I. Niemelä, Towards efficient default reasoning. *Proc. IJCAI'95*, Morgan Kaufman, pages 312–318

[35] I. Niemelä, P. Simons, Efficient implementation of the well-founded and stable model semantics. *Proc. JICSLP'96*, MIT Press, pages 289–303.

[36] A. Pettorossi, M. Proietti, Transformation of logic programs: foundations and techniques. *Journal of Logic Programming* 19/20, 1994, pages 261–320

[37] A. Peczenik, Jumps and logic in law. *Journal of Artificial Intelligence and Law* 4(3–4) pages 141–174, 1996, Special Issue on Logical Models of Argumentation, Kluwer Academic Publishers

[38] D. Poole, A logical framework for default reasoning. *Artificial Intelligence* 36 (1988)

[39] H. Prakken and G. Sartor, The role of logic in computational models of legal argument: a critical survey, *Computational Logic: Logic Programming and Beyond – Essays in Honour of Robert A. Kowalski*, Springer Verlag LNAI 2408, 289–310, 2002

[40] T. Przymusinski, An algorithm to compute circumscription. *Artificial Intelligence* 38 (1989) pages 49–73

[41] T. Przymusinski, Semantics of disjunctive logic programs and deductive databases. *Proc. DOOD '91*

[42] R. Reiter, A logic for default reasoning. *Artificial Intelligence* 13 (1980)

[43] D. Saccà, C. Zaniolo, Stable models and non-determinism for logic programs with negation. *Proc. ACM SIGMOD-SIGACT Symposium on Principles of Database Systems* (1990)

[44] K. Satoh and N. Iwayama. A correct top-down proof procedure for general logic programs with integrity constraints. *Proc. 3rd International Workshop on Extensions of Logic Programming*, (E. Lamma, P. Mello, eds) Springer Verlag LNAI 660 (1992) pages 19–34

[45] K. Satoh, A top-down proof procedure for default logic by using abduction. *Proc. ECAI'94*

[46] M. Thielscher, A nonmonotonic disputation-based semantics and proof procedure for logic programs. *JICSLP'96*, MIT Press, pages 483–497

[47] F. Toni, A.C. Kakas, Computing the acceptability semantics. *Proc. 3rd International Workshop on Logic Programming and Non-monotonic Reasoning,* (V. W. Marek, A. Nerode and M. Truszczynski eds.) Springer Verlag LNAI 928 pages 401–415 (1995)

[48] A. Torres. Negation as failure to support. *Proc. 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning* MIT press (Pereira and Nerode eds.) 223–243 (1993)

[49] S.E. Toulmin, *The uses of arguments*. Cambridge University Press (1958)

[50] A. Van Gelder, K.A. Ross, J.S. Schlipf, Unfounded sets and the well-founded semantics for general logic programs. *Proc. ACM SIGMOD-SIGACT, Symposium on Principles of Database Systems* (1988)

[51] A. Weston, *A rulebook for arguments*. Hackett Publishing Company, Inc. (1992)

# A Abstract Proof Trees

Given a proof tree, an attack node $N$ in the tree, labelled by a set of assumptions $A$, and a child of $N$ labelled by a set of assumptions $D$, let $culprit(A)$ be an arbitrary assumption in $A$ attacked by $D$.

**Theorem** 6.1
Given a set of assumptions $\Delta_0 \subseteq Ab$:

i) If $\mathcal{T}$ is an admissible proof tree for $\Delta_0$ and $\Delta$ is the defence set of $\mathcal{T}$,
then $\Delta_0 \subseteq \Delta$ and $\Delta$ is admissible.

ii) If $\Delta \subseteq Ab$ is an admissible set of assumptions such that $\Delta_0 \subseteq \Delta$,
then there exists an admissible proof tree for $\Delta_0$ with defence set $\Delta$.

**PROOF**

1. It is obvious that $\Delta_0 \subseteq \Delta$. It remains to prove that $\Delta$ does not attack itself and attacks every attack against it.

   From the definition of admissible proof tree, it follows that for each label $A$ of an attack node in $\mathcal{T}$, $culprit(A) \notin \Delta$

   Assume that $\Delta$ attacks itself. Let $\delta \in \Delta$ such that $\Delta$ attacks $\delta$. From the definition of $\Delta$, there is a defence node $N$ labelled by $D$ such that $\delta \in D$. Since $\Delta$ attacks $\delta$, $\Delta$ attacks $D$. Because the labels of the children of $N$ form a covering set of attacks against $D$, there is a child of $N$ that is labelled with $A$ such that $A \subseteq \Delta$. Hence $culprit(A) \in \Delta$. Contradiction.

   Assume that $S$ is an attack against some assumption $\delta \in \Delta$. From the definition of $\Delta$, there is a defence node $N$ labelled with $D$ such that $\delta \in D$. Since $S$ attacks $\delta$, $S$ attacks $D$. Therefore there is a child $M$ of $N$ which is labelled by some $A$ such that $A \subseteq S$. Therefore $A$ is attacked by the label $E$ of its only child. Since $E \subseteq \Delta$, $\Delta$ attacks $A$ and therefore $\Delta$ attacks $S$.

2. Let $\mathcal{C}$ be a covering set of attacks against $\Delta$. Note that $\mathcal{C}$ is also a covering set of attacks against $\Delta_0$. Construct a proof tree as follows:

   (a) The root is labelled by $\Delta_0$. This node is of rank 0.

   (b) The children of the root are labelled by the attacks in $\mathcal{C}$ - there being one such node for every such attack. These children are all attack nodes and have rank 1.

   (c) For each node $N$ of rank $k \geq 1$:

   - If $N$ is a defence node, then $N$ is labelled by $\Delta$ and the children of $N$ are labelled by the attacks in $\mathcal{C}$ - there being one such node for every such attack.

- If $N$ is an attack node, then $N$ has a single child, which is a defence node of rank k+1, labelled with $\Delta$.

It is obvious that this tree is an admissible proof tree whose set of defence nodes is $\Delta$.

**Theorem** 6.2
Any proof tree that has no infinite branches is an admissible proof tree.

**PROOF:** Two nodes $(N, M)$ are said to be conflicting if $N$ is a defence node and $M$ is an attack node such that the culprit of $M$ belongs to the label of $N$. Let $(N, M) \sqsubset (N', M')$ if $(N, M) and (N', M')$ are conflicting pairs of nodes, $N'$ is a child of $N$, and $M'$ is a child of $M$.

It is clear that the existence of an infinite sequence $(N_0, M_0) \sqsubset (N_1, M_1) \sqsubset ....$ implies that the proof tree is not finite.

To prove the theorem, we prove that any non-admissible proof tree is infinite. This follows directly from the following claim:

**Claim** Let $(N, M)$ be a conflicting pair of nodes. There is another conflicting pair $(M', N')$ such that $(N, M) \sqsubset (M', N')$.

**Proof**: Let $\alpha$ be the culprit at $M$. Let $D$ be the label of the only child $M$' of $M$. Then $D$ attacks $\alpha$.

Because $\alpha$ belongs to the label of $N$, and because the labels of the children of $N$ are a covering set of attacks against the label of $N$, there is a child $N'$ of $N$ that is labelled by some $D'$ such that $D' \subseteq D$. Because $N'$ is an attack node, $D' \neq \{\}$. Hence it is clear that the culprit at $N'$ belongs to $D$' and therefore also belongs to $D$. This implies that $(M', N')$ is a conflicting pair of nodes and $(N, M) \sqsubset (M', N')$.

# B Transformation of Filtered Abstract Derivation Into Abstract Proof Trees

The transformation consists of two steps: First a filtered abstract derivation is transformed into a filtered abstract proof tree, then a filtered abstract proof tree is transformed into a abstract proof tree.

## B.1 Filtered Abstract Proof Tree

**Definition B.1** A **filtered abstract proof tree** for a set of assumptions $\Delta_0$ is a finite tree $\mathcal{T}$ such that:

1. Every node of $\mathcal{T}$ is labelled by a set of assumptions and is assigned the status of **defence** node or **attack** node, but not both.

2. The root is an defence node labelled by $\Delta_0$

3. At every defence node $N$, the assumptions belonging to its label are classified as one of type *"filtered"* or *"defended"*. For each assumption belonging to the label of a defence node, there is exactly one defence node at which it has the status "defended".

4. For every defence node $N$ labelled by a set of assumptions $D$, the children of $N$ are a set of attack nodes, labelled by a covering set of attacks against the set consisting of the defended assumptions in the label of $N$.

5. For every attack node $N$ labelled by a set of assumptions $A$, there exists exactly one child of $N$, which is a defence node labelled by an attack against $A$.

The set of all assumptions belonging to defence nodes in $\mathcal{T}$ is called the **defence set** of $\mathcal{T}$.

The definition of admissible filtered proof tree is similar to the definition of admissible proof trees.

## B.2 Transformation of Filtered Abstract Derivations Into Filtered Abstract Proof Trees

Let

$\langle \Delta_0, \mathcal{D}_0, \mathcal{A}_0 \rangle, \ldots, \langle \Delta_i, \mathcal{D}_i, \mathcal{A}_i \rangle, \ldots, \langle \Delta_n, \mathcal{D}_n, \mathcal{A}_n \rangle$

be a filtered derivation.

We will construct a sequence of partial filtered proof trees:
$\mathcal{T}_0, \ldots, \mathcal{T}_i, \ldots, \mathcal{T}_n$,
such that $\mathcal{T}_i$ corresponds to $\langle \Delta_i, \mathcal{D}_i, \mathcal{A}_i \rangle$ in the sense that:

**Lemma B.1**

- The set of assumptions belonging to the labels of defence nodes of $\mathcal{T}_i$ coincides with $\Delta_i$.
- The set of assumptions that belong to the labels of defence nodes of $\mathcal{T}_i$ and are neither of type *filtered* nor of type *defended* coincides with $\mathcal{D}_i$.
- The set of labels of attack nodes of $\mathcal{T}_i$ that are leaves in $\mathcal{T}_i$ coincides with $\mathcal{A}_i$.

**Proof**: The trees $\mathcal{T}_i$ are constructed inductively as follows:

- $\mathcal{T}_0$ consists of exactly one node that is also its root and is a defence node. The label of the root is $\Delta_0$.

  Assume that $\mathcal{T}_i$ has been constructed and corresponds to $\langle \Delta_i, \mathcal{D}_i, \mathcal{A}_i \rangle$. We now construct $\mathcal{T}_{i+1}$. There are two cases:

- $\sigma \in D_i$ is selected and

  $\mathcal{D}_{i+1} = \mathcal{D}_i - \{\sigma\}$,

  $\Delta_{i+1} = \Delta_i$,

  $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \mathcal{C}_\sigma$

  where $\mathcal{C}_\sigma$ is a covering set of attacks against $\sigma$.

  Let $N$ be a defence node such that $\sigma$ belongs to the label of $N$. Let $|\mathcal{C}_\sigma| = k$. Add k children to $N$ and label them with the attacks in $\mathcal{C}_\sigma$. Mark the type of $\sigma$ as "defended". The new nodes are attack nodes.

  The resulting tree $\mathcal{T}_{i+1}$ corresponds to $\langle \Delta_{i+1}, \mathcal{D}_{i+1}, \mathcal{A}_{i+1} \rangle$.

- $A \in \mathcal{A}_i$ is selected and there is an attack $D$ against some culprit $\sigma \in A$ such that

  $\sigma \notin \Delta_i$

  $\mathcal{A}_{i+1} = \mathcal{A}_i - \{A\}$

  $\mathcal{D}_{i+1} = \mathcal{D}_i \cup D - \Delta_i$,

  $\Delta_{i+1} = \Delta_i \cup D$.

  Let $N$ be a leaf in $\mathcal{T}_i$ such that $N$ is an attack node and $A$ is the label of $N$. Add a child of $N$ and label it with $D$. Mark the assumptions belonging to $D \cap \Delta_i$ as "filtered".

  The resulting tree $\mathcal{T}_{i+1}$ corresponds to $\langle \Delta_{i+1}, \mathcal{D}_{i+1}, \mathcal{A}_{i+1} \rangle$.

**Corollary**

$\mathcal{T}_n$ is a admissible filtered proof tree whose defence set is $\Delta_n$.

## B.3 Expanding Abstract Filtered Proof Trees into Abstract Proof Trees

Informally speaking, we expand a filtered proof tree $\mathcal{T}$ into a proof tree, by repeatedly adding below each defence node whose label contains one or more filtered assumptions $\delta$ additional subtrees rooted in attack nodes against $\delta$, extracted from the part of $\mathcal{T}$ where $\delta$ is marked as defended.

More precisely: For each assumption $\delta$ belonging to the defence set of $\mathcal{T}$, let $N_\delta$ be the unique defence node of $\mathcal{T}$ at which $\delta$ is marked as defended.

Let $\mathcal{C}$ be the set of all labels of the children of $N_\delta$.

It is clear that the set $\mathcal{C}_\delta = \{ A \in \mathcal{C} \mid$ A is an attack against $\delta \}$ is a covering set of attacks against $\delta$. For each of $A \in \mathcal{C}_\delta$, define $T_{\delta,A}$ to be the subtree of $\mathcal{T}$ whose root is the child of $N_\delta$ labelled by $A$.

The expansion of $\mathcal{T}$ is obtained as the limit of the sequence
$\mathcal{T}_0, \ldots, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots,$.

defined as follows:

1. $\mathcal{T}_0 = \mathcal{T}$.

2. Assume that we have already constructed $\mathcal{T}_i$, where $i \geq 0$. $\mathcal{T}_{i+1}$ is obtained by expanding $\mathcal{T}_i$ simultaneously at each defence node $N$ whose label contains one or more filtered assumptions as follows:

   Let $D$ be the set of all filtered assumptions belonging to the label of $N$. Let $\mathcal{C}_N = (\bigcup\{C_\delta \mid \delta \in D\}) - \mathcal{C}$ where $\mathcal{C}$ is the set of all labels of the children of $N$. Let $\mathcal{T}_{i+1}$ be $\mathcal{T}_i$ with new children added to $N$ that are attack nodes labelled by the elements of $\mathcal{C}_N$. Expand each of these children by the tree $T_{\delta,A}$.

3. The limit of $(\mathcal{T}_i)_{i=1,\ldots,\infty}$ is denoted by $expand(\mathcal{T})$.

It is obvious from the construction of $expand(\mathcal{T})$ that:

**Lemma B.2** 1. $expand(\mathcal{T})$ is a proof tree.

2. $expand(\mathcal{T})$ is admissible iff $\mathcal{T}$ is admissible.

3. The defence sets of $\mathcal{T}$ and $expand(\mathcal{T})$ coincide.

It follows immediately that:

**Lemma B.3** Given a set of assumptions $\Delta_0$, for every admissible filtered proof tree $\mathcal{T}$ for $\Delta_0$, there exists a admissible proof tree $\mathcal{T}'$ for $\Delta_0$ such that the defence sets of $\mathcal{T}$ and $\mathcal{T}'$ coincide.

It follows that:

**Theorem** 7.1
For every filtered derivation of a defence set $\Delta$ for a set of assumptions $\Delta_0$, there exists a (possibly infinite) admissible proof tree for $\Delta_0$ with defence set $\Delta$.

## B.4 Transformation of Finite Abstract Proof Trees into Filtered Abstract Proof Trees

To prove theorem 7.2, we give a method to translate every finite proof tree with a defence set $\Delta$ into a filtered proof tree $\mathcal{T}'$ with a defence set $\Delta' \subseteq \Delta$.

Let $\mathcal{T}$ be a finite proof tree with a defence set $\Delta$. For each assumption $\delta \in \Delta$, let $dis(\delta) = \min\{$ k $\mid$ there is a node N such that $\delta$ belongs to the label of N and k is the length of the path from the root to N $\}$.

We construct $\mathcal{T}'$ by starting from $\mathcal{T}$ and repeatedly cutting of nodes from $\mathcal{T}$ and labelling the assumptions accordingly:

- For each defence node N in $\mathcal{T}$, mark each assumption $\delta$ belonging to the label of N, as filtered if the length of the path from the root to N is greater than $dis(\delta)$. Mark all the other assumptions as defended.

  For each defence node N, it is obvious that the set of all labels of the children of N is covering set of attacks against the label of N. Let $\mathcal{C}_N$ be a subset of this set that constitutes a covering set of attacks against the assumptions marked as defended.

- Repeatedly for each defence node N, delete all the children of N whose labels do not belong to $\mathcal{C}_N$.

It is not difficult to see that $\mathcal{T}'$ is a filtered abstract proof tree.

# C Transformation from Concrete Proof Trees into Abstract Proof Trees and Vice Versa

## C.1 Preliminaries

A set of assumptions $A$ is said to *support* $\alpha$ if $A \vdash \alpha$. $A$ supports a set of sentences $S$ if $A$ supports each sentence in $S$. $A$ is also often called a support of $\alpha$ or $S$.

A *representative set of supports* for $\alpha$ is defined as a set of assumptions $\mathcal{B}$ such that $\mathcal{B} \cap A \neq \{\}$ for any support $A$ of $\alpha$.

It should be clear that R is a representative set of supports for an assumption $\alpha$ iff $\alpha \in R$. Further if $\alpha$ has no support then every set of assumptions is a representative set of supports of $\alpha$. If $\alpha$ has an empty support then there exist no representative set of support of $\alpha$.

It is easy to see that

**Lemma C.1** Let $\mathcal{C}$ be a covering set of attacks against an assumption $\alpha$. A set of assumptions $B$ is a representative set of supports for $\overline{\alpha}$ iff for each $A \in \mathcal{C}$, $A \cap B \neq \{\}$

**Lemma C.2**

1. Let $\alpha$ be a non-assumption sentence and $\mathcal{B}$ be a representative set of support of $\alpha$. Then for each rule of the form $\frac{S}{\alpha}$, there is at least a $\beta \in S$ such that $\mathcal{B}$ is also a representative set of supports of $\beta$

2. Let $\alpha$ be a non-assumption sentence. For each rule r of the form $\frac{S}{\alpha}$, let $B_r$ be a representative set of supports of at least one $\beta \in S$. Then

   $\mathcal{B} = \bigcup\{ B_r \mid \text{r is a rule of the form } \frac{S}{\alpha}\}$ is a representative set of supports of $\alpha$

3. Let $A$ be a set of sentences. $\mathcal{B}$ is a representative set of supports of $A$ iff $\mathcal{B}$ is a representative set of supports for some $\alpha \in A$

**PROOF**

1. Let $\mathcal{B}$ be a representative set of supports of $\alpha$. Let r be a rule of the form $\frac{S}{\alpha}$. We want to show now that $\mathcal{B}$ is also a representative set of support for some $\beta \in S$. Assume the contrary. Therefore for each $\beta \in S$ there is a set of assumptions $R_\beta$ such that $R_\beta$ supports $\beta$ and $\mathcal{B} \cap R_\beta = \{\}$. Therefore $R = \bigcup\{ R_\beta \,|\, \beta \in S \}$ is a set of assumptions that supports $\alpha$ and $R \cap \mathcal{B} = \{\}$. This is contrary to the fact that $\mathcal{B}$ is a representative set of supports of $\alpha$. Therefore, there is $\beta \in S$ s.t. $\mathcal{B}$ is a representative set of its supports.

2. Let $R$ be a set of assumption that supports $\alpha$. Therefore there is a rule r of the form $\frac{S}{\alpha}$ such that $R$ supports $S$. Therefore R also supports every element of $S$. Therefore, from the definition of $B_r$, it follows immediately $R \cap B_r \neq \{\}$. Hence $\mathcal{B} \cap R \neq \{\}$. We have proved that $\mathcal{B}$ is a representative set of supports of $\alpha$.

3. We only need to prove the only-if direction. Let $\mathcal{B}$ be a representative set of supports of G. Assume $\mathcal{B}$ is not a representative set of supports of any of the subgoals of G. Hence for any subgoal $\alpha$ of G there is a support $S_\alpha$ that is disjoint to $\mathcal{B}$. Therefore the union of all $S_\alpha$ is a support of G that is disjoint to $\mathcal{B}$. Contradiction

We introduce two kinds of partial proof trees which together constitute the concrete proof tree.

## C.2 Support Tree

A support tree of a set of sentences Q is a tree defined as follows:

1. Every node is labelled by a set of sentences and is assigned the status of a **defence** node.

2. The root is labelled by Q.

3. Let $N$ be a node labelled $D$. If $D$ is empty, then $N$ is a terminal node. If $D$ is not empty and the selected sentence in $D$ is an assumption $\delta$, then there exists exactly a child of $N$, which is labelled by $D$ - $\{\delta\}$. If $\delta$ is not an assumption, then there exists some inference rule $\dfrac{S}{\delta} \in \mathcal{R}$ and there exists exactly one child of $N$ labelled by $D - \{\delta\} \cup S$.

It is easy to see that

**Lemma C.3** 1. Let $\mathcal{T}$ be a finite support tree of $Q$. Then the set of assumptions in $\mathcal{T}$ supports $Q$

2. Let $S$ be a set of assumptions supporting $Q$. Then there exists a finite support tree of $Q$ whose set of assumptions is a subset of $S$.

## C.3 Representative Tree

A representative tree of a set of sentences $A$ is a finite tree defined as follows:

1. Every node is labelled by a set of sentences and is assigned the status of **attack** node.
2. The root is labelled by $A$
3. Let $N$ be an node labelled by $B$. Then $B \neq \{\}$. If the sentence selected in $B$ is an assumption $\alpha$, then$N$is a leaf. If the selected sentence $\alpha$ is not an assumption and there exists no inference rule $\frac{S}{\alpha} \in \mathcal{R}$, then $N$ is a terminal node. Otherwise, the children of $N$ are a set of attack nodes labelled by the sets of sentences $A - \{\alpha\} \cup S$, where $\frac{S}{\alpha} \in \mathcal{R}$. (There being one such child for each such inference rule.)

**Lemma C.4** Let $A$ be a set of sentences.

1. Let $\mathcal{B}$ be the set of assumptions selected in a representative proof tree$\mathcal{T}$ of $A$. Then $\mathcal{B}$ is a representative set of supports of $A$
2. Let $\mathcal{B}$ be a representative set of supports of $G$. Then there is a representative proof tree $\mathcal{T}$ of $A$ for some $\alpha \in A$ such that the set of assumptions selected in $\mathcal{T}$ is a subset of $\mathcal{B}$.

**Proof**

1. Let $R$ be a set of assumptions supporting $A$. We have to show that there exists a leaf of $\mathcal{T}$ whose selected sentence is an assumption belonging to R. Let $\alpha$ be the selected sentence at the root. If *alpha* is an assumption, we are done. Assume now that $\alpha$ is not an assumption.

   It is clear that there is a deduction of $A$ from $T \cup R$. For the purpose of the simplicity, we introduce the following notion of a simplified deduction.

   A simplified deduction of $A$ from R is a sequence $\beta_1, \ldots, \beta_n$ such that

   - $A \subseteq \{\beta_1, \ldots, \beta_n\} \cup R$, and
   - For every $i \leq k$ there is a rule of the form $\frac{S}{\beta_i}$ such that $S \subseteq R \cup \{\beta_1, \ldots, \beta_{i-1}\}$.

   It is clear that there is a simplified deduction $\beta_1, \ldots, \beta_n$ of $A$ from $R$. Because $\alpha$ is not an assumption, it is clear that $n \geq 1$.

   We prove the lemma by induction on n.

   Let$n = 1$. Therefore there is a rule of the form $\frac{S}{\alpha}$ such that $S \subseteq R$. Because $A \subseteq \{\beta_1, \ldots, \beta_n\} \cup R$, it follows $A \subseteq \{\alpha\} \cup R$. Since $\alpha \in A$, it follows immediately

$(A - \{\alpha\}) \cup S \subseteq R$. As $(A - \{\alpha\}) \cup S$ labels a child of the root, it is not empty. We are done.

Induction hypothesis. Let $n \geq 2$. Let $\frac{S}{\alpha}$ be the rule used in the first step of the simplified deduction. Let$N$be the node corresponding to this rule in the set of children of the root of $\mathcal{T}$. Let $\delta$ be the selected sentence at this node. If $\delta$ is an assumption then obviously $\delta \in R$ and $N$is a leaf of $\mathcal{T}$. Part 1 of the above lemma is therefore proved.

Let $\delta$ be a non-assumption sentence. It is clear that there is a simplified deduction of set $(A - \{\alpha\}) \cup S$ labelling$N$from R that is a subsequence of $\beta_1, \ldots, \beta_n$. Obviously this deduction has a length less than n. Therefore, according to the induction hypothesis, the intersection of R with the set of assumptions selected in the subtree of $\mathcal{T}$ which has$N$as its root is not empty. Hence the intersection of the set of assumptions labelling the leaves of $\mathcal{T}$ with R is not empty.

2. $\mathcal{T}$ is defined as follows:

   (a) Every node is labelled by a set of sentences and is assigned the status of **attack** node.

   (b) The root is labelled by $A$ such that $\mathcal{B}$ is a representative set of support of the selected sentence.

   (c) Let $N$ be an node labelled by $B$. Then $B \neq \{\}$. If the sentence selected in $B$ is an assumption $\alpha$, then$N$is a leaf. If the selected sentence $\alpha$ is not an assumption and there exists no inference rule $\frac{S}{\alpha} \in \mathcal{R}$, then $N$ is a terminal node. Otherwise, $\mathcal{B}$ is a representative set of support of $\alpha$ and the children of $N$ are a set of attack nodes labelled by the sets of sentences $A - \{\alpha\} \cup S$, where $\frac{S}{\alpha} \in \mathcal{R}$. (There being one such child for each such inference rule.)

   The existence of such a tree $\mathcal{T}$ is guaranteed by lemma C.2. Because $\mathcal{B}$ is a representative set of supports of $\alpha$, from lemma C.2, it follows that if a assumption subgoal is selected, it belongs to $\mathcal{B}$. It follows immediately that the set of assumptions selected at the leaves of $\mathcal{T}$ is a subset of $\mathcal{B}$.

## C.4 Transformation from Concrete Proof Trees into Abstract Proof Trees

Let $\mathcal{T}$ be a concrete proof tree. Define $\mathcal{T}(N)$ as the maximal subtree of $\mathcal{T}$ rooted at $N$ all of whose nodes have the same attack/defence node status as $N$. Intuitively, $\mathcal{T}(N)$ isolates the monotonic part of the subtree of $\mathcal{T}$ rooted at $N$. Equivalently, $\mathcal{T}(N)$ can be obtained from $\mathcal{T}$ as follows:

- Let $\mathcal{T}_1$ be the subtree of $\mathcal{T}$ rooted at $N$.

- Delete all the nodes in $\mathcal{T}_1$ which have a different attack/defence node status than $N$ or which are successors of a node that has different attack/defence node status than $N$. The resulting tree is $\mathcal{T}(N)$.

It is easy to see that if N is an attack node labelled by $\overline{\delta}$ for some assumption $\delta$ then $\mathcal{T}(N)$ is a representative tree of $\overline{\delta}$. Further if N is an defence node labelled by $\overline{\delta}$ for some assumption $\delta$ then $\mathcal{T}(N)$ is a support tree of $\overline{\delta}$.

Let $assumptions(\mathcal{T}(N))$ be the set of assumptions selected in $\mathcal{T}(N)$. If $N$ is a defence node, then these are all the assumptions in $\mathcal{T}(N)$. If $N$ is an attack node, then these are all culprits, which are counter-attacked at a lower level in $\mathcal{T}$.

We construct an abstract proof tree $reduct(\mathcal{T})$, corresponding to $\mathcal{T}$, such that $reduct(\mathcal{T})$ and $\mathcal{T}$ have the same defence assumptions and culprits:

**Definition C.1** 1. The root of $reduct(\mathcal{T})$ is a defence node labelled by $assumptions(\mathcal{T}(N_0))$ where $N_0$ is the root of $\mathcal{T}$.

2. Suppose that $N$ is a defence node and the ancestors of $N$ in $reduct(\mathcal{T})$ are already defined. Let $D$ be the set of assumptions labelling $N$. For each assumption $\delta$ belonging to $D$, let $N_{\overline{\delta}}$ be an attack node in $\mathcal{T}$ labelled by $\{\overline{\delta}\}$. Since $\mathcal{T}(N_{\overline{\delta}})$ is a representative tree for $\overline{\delta}$, $B_\delta = assumptions(\mathcal{T}(N_{\overline{\delta}}))$ is a representative set of supports for $\overline{\delta}$ (lemma C.4).

   Let $\mathcal{C}_\delta$ be an arbitrary but fixed covering set of attacks against $\delta$. For each set $A \in \mathcal{C}_\delta$, let $culprit(A)$ be an arbitrary but fixed assumption from $A \cap B_\delta$ (From lemma C.1, it is clear that $A \cap B_\delta$ is not empty).

   The children of $N$ are a set of attack nodes labelled by $\bigcup\{\mathcal{C}_\delta \mid \delta \in D\}$. For each label $A$ of one of these children, $culprit(A)$ is the selected culprit in $A$.

3. Suppose that $N$ is an attack node and the ancestors of $N$ in $reduct(\mathcal{T})$ are already defined. Let $A$ be the set of assumptions labelling $N$, and let $\alpha = culprit(A)$ be the selected culprit in $A$. Let $N_{\overline{\alpha}}$ be an defence node in $\mathcal{T}$ labelled by $\{\overline{\alpha}\}$.

   Then $N$ has a single child labelled by $assumptions(\mathcal{T}(N_{\overline{\alpha}}))$.

   Note that since $\mathcal{T}(N_{\overline{\delta}})$ is a support tree for $\overline{\delta}$, $assumptions(\mathcal{T}(N_{\overline{\delta}}))$ is a support for $\overline{\delta}$ (lemma C.3).

It is easy to see that, by letting $\mathcal{T}' = reduct(\mathcal{T})$, we obtain theorem 8.1, part i).

# D Transformations from Abstract Proof Trees into Concrete Proof Trees

Let $\mathcal{T}$ be an abstract proof tree for a set of assumptions $\Delta_0$. For each assumption $\delta$ in the defence set or culprit set of $\mathcal{T}$ let $N_\delta$ be an arbitrary but fixed node such that $\delta$ belongs to the label of $N_\delta$. Let $\mathcal{D}$ be the defence set of $\mathcal{T}$.

For each assumption $\delta \in \mathcal{D}$, $\mathcal{C}_\delta$ consists of the labels of the children of $N_\delta$ that are attacks against $\delta$. It is not difficult to see that $\mathcal{C}_\delta$ is covering set of attacks against $\delta$. Let $B_\delta$ be the set of culprits in $\mathcal{C}_\delta$. It is obvious that $B_\delta$ is a representative set of supports for $\overline{\delta}$.

For each assumption $\delta$ that is the culprit at some attack node $N$ labelled by $A$, let $D_\delta$ denote the label of the only child of $N$. It is clear that $D_\delta$ is a support of $\delta$.

Before defining a concrete proof tree $\mathcal{T}'$, we define the following tree $\mathcal{T}_1$.

1. The root of $\mathcal{T}_1$ is a defence node labelled by $\Delta_0$.

2. Let N be a defence node labelled by $D$. If $D$ is empty, then $N$ is a terminal node. If $D$ is not empty and the selected sentence in $D$ is an assumption $\delta$, then there are two children of $N$: one is an attack node labelled by $\{\overline{\delta}\}$ and the other is a defence node labelled by $D$ - $\{\delta\}$.

Now we recursively expand $\mathcal{T}_1$ into a concrete proof tree as follows:

1. Expand each attack node $N$ labelled by $\{\overline{\delta}\}$ that is a child of a defence node by a representative tree T whose set of selected assumptions is a subset of $B_\delta$. (The existence of such tree is guaranteed by lemma C.4). Further, expand each terminal node in T whose selected sentence is an assumption $\delta$ by a child that is a defence node and labelled with $\overline{\delta}$.

2. For each defence node $N$ labelled by $\{\overline{\delta}\}$ that is a child of a attack node, expand N by a support tree T whose set of selected assumptions is a subset of $D_\delta$. (The existence of such tree is guaranteed by lemma C.3). For each node in T whose selected sentence is an assumption $\delta$, expand this node by a child that is a attack node and labelled with $\overline{\delta}$.

Theorem 8.1, part ii), follows immediately from the lemmas C.3, C.4.