Synthesis of proof procedures for default reasoning

Phan Minh Dung[1], Robert A. Kowalski[2] and Francesca Toni[3]

[1] Asian Institute of Technology, Division of Computer Science
PO Box 2754, Bangkok 10501, Thailand
dung@cs.ait.ac.th

[2] Imperial College, Department of Computing
180 Queen's Gate, London SW7 2BZ, UK
rak@doc.ic.ac.uk

[3] National Technical University of Athens
Department of Electrical and Computing Engineering, Division of Computer Science
15773 Zographou, Athens, Greece
ftoni@softlab.ece.ntua.gr

Abstract. We apply logic program development technology to define abstract proof procedures, in the form of logic programs, for computing the admissibility semantics for default reasoning proposed in [2].
The proof procedures are derived from a formal specification. The derivation guarantees the soundness of the proof procedures. The completeness of the proof procedures is shown by employing a technique of symbolic execution of logic programs to compute (an instance of) a relation implied by the specification.

1 Introduction

In [2], we have shown that many default logics [13, 19, 14, 15] can be understood as special cases of a single abstract framework, based upon an abductive interpretation of the semantics of logic programming [7, 8] and its abstractions [4, 5, 1, 11], and extending Theorist [18]. Moreover, we have proposed a new semantics for default logics, more liberal than their standard semantics and generalising the admissibility semantics for logic programming [4], equivalent to the partial stable model semantics [20] (see [10]).

In this paper, we define two proof procedures for computing the abstract admissibility semantics. The second proof procedure is a computationally more efficient refinement of the first. Both procedures generalise and abstract a proof procedure [8] for logic programming, but are formulated as logic programs. The relationships of the proof procedures with other existing proof procedures for default reasoning and the relevance of the proof procedures in the field of default reasoning are discussed in an extended version of this paper [6]. In the present paper, we describe the technology used to define the abstract proof procedures. Both are derived from a formal specification by conventional techniques of deductive synthesis of logic programs (e.g. those described already in [12], Chapter 10, and, more recently, in [3]). The derivation guarantees the soundness of the proof procedures. The completeness of the proof procedures is shown via symbolic execution of the logic programs to compute (an instance of) a relation implied by the specification.

The logic programs are derived top-down in two stages: the top-most level is derived first, relative to lower-level predicates that can then be "developed". The top-level program is proved correct and complete, parametrically with respect to the lower-level predicates. (Generalised) logic programs computing the lower-level predicates are given in [6].

The rest of the paper has the following structure: Section 2 revises the main features of the abstract framework and the admissibility semantics; Section 3 introduces the top-level of the first abstract proof procedure to compute the admissibility semantics; Section 4 introduces the top-level of the more efficient proof procedure; Section 5 gives conclusions.

2 Argumentation-theoretic framework and admissibility semantics

An *argumentation-theoretic framework* consists of a set of sentences T, the *theory*, viewed as a given set of beliefs, a (non-empty) set of sentences Ab, viewed as *assumptions* that can be used to extend T, and a notion of *attack*, namely a (binary) relation between sets of assumptions.

Both theory and assumptions are formulated in some underlying language provided with a notion of derivability Th which is *monotonic*, in the sense that $T \subseteq T'$ implies $Th(T) \subseteq Th(T')$, and *compact*, in the sense that $\alpha \in Th(T)$ implies $\alpha \in Th(T')$ for some finite subset T' of T.

The notion of attack is *monotonic*, in the sense that, for any sets of assumptions $A, A', \Delta, \Delta' \subseteq Ab$, if A attacks Δ then:

- A' attacks Δ for any $A' \supseteq A$;
- A attacks Δ' for any $\Delta' \supseteq \Delta$.

Moreover, the notion of attack satisfies the property that no set of assumptions attacks the empty set of assumptions.

Theorist [18], circumscription [13], logic programming, default logic [19], autoepistemic logic [15] and non-monotonic modal logic [14] are all instances of the abstract argumentation-theoretic framework (see [2]).

A set of assumptions $\Delta \subseteq Ab$ is *closed* iff $\Delta = Ab \cap Th(T \cup \Delta)$.

An argumentation-theoretic framework is *flat* iff every set of assumptions is closed. The frameworks for logic programming and default logic are flat.

A set of assumptions Δ is

- *admissible* iff Δ is closed, Δ does not attack itself and
  for each closed $A \subseteq Ab$, if A attacks Δ then Δ attacks A.

Admissible sets of assumptions correspond to admissible scenaria for logic programming [4]. The standard semantics of scenaria in Theorist [18], extensions in default logic [19], stable expansions in autoepistemic logic [15], fixed points in non-monotonic modal logic [14] and stable models in logic programming [9] correspond to the less liberal notion of *stable* sets of assumptions, i.e. sets of assumptions which are admissible and attack every assumption they do not contain. [4]

The semantics of admissible and stable sets of assumptions are *credulous*, in the sense that a sentence δ is a *non-monotonic consequence* of a theory T iff δ belongs to *some* extension sanctioned by the semantics. Corresponding to every credulous semantics there is a *sceptical* semantics in which δ is a *non-monotonic consequence* of T iff δ belongs to *all* extensions sanctioned by the semantics. Many cases of circumscription [13] can be understood as the sceptical semantics corresponding to stable sets of assumptions.

In this paper we focus upon the computation of non-monotonic consequences using the (credulous) admissibility semantics. We define proof procedures for computing the admissibility semantics for any abstract argumentation-theoretic framework.

3 Proof procedure for admissibility

The procedure is defined in the form of a *metalevel logic program*, the top-level clauses of which define the predicate *adm*, whose specification is given as follows:

Definition 1. Let $\langle T, Ab, \text{attacks}\rangle$ be an argumentation-theoretic framework. For any sets of assumptions $\Delta_0, \Delta \subseteq Ab$

$adm(\Delta_0, \Delta) \leftrightarrow [\Delta_0 \subseteq \Delta \wedge \Delta \text{ is admissible}]$.

Typically, the set Δ_0 will be given, such that $T \cup \Delta_0 \vdash \alpha$ for some formula $\alpha \in \mathcal{L}$, and the problem will be to generate Δ, such that $adm(\Delta_0, \Delta)$. Consequently, $T \cup \Delta \vdash \alpha$ as well, and the set Δ provides an admissible "explanation" for the query α.

This characterisation of the predicate *adm* provides a *specification* for the proof procedure. In the remainder of this section, this specification together with the definition of admissibility given earlier will be referred to as $\mathcal{S}pec_{adm}$. The logic program providing the proof procedure will consist of top-level clauses defining *adm* and lower-level clauses, defining the predicate $defends$ given later in the section, in definition 3. The predicate *adm* takes names of sets of sentences as arguments, and is therefore a metapredicate. [5]

We focus on the top-level of the program. This part of the program will be derived from $\mathcal{S}pec_{adm}$ and from the specification (given later, in definition 3) of the lower-level predicate $defends$.

[4] Trivially, a set of assumptions $A \subseteq Ab$ attacks an assumption $\alpha \in Ab$ iff A attacks $\{\alpha\}$.

[5] Moreover, there is an additional, implicit argument T in *adm* and all predicates considered in these paper.

The following simple, but important, theorem provides an alternative characterisation of admissibility. By virtue of this theorem, the condition that an admissible set of assumptions Δ does not attack itself does not need to be checked explicitly. It can be shown to hold implicitly if, for all closed attacks A against Δ, we restrict attention to counter attacks against assumptions in $A - \Delta$. This restriction has the additional computational advantage of reducing the number of candidate counter attacks that need to be considered.

Theorem 2. *A set of assumptions $\Delta \subseteq Ab$ is admissible iff*
Δ is closed, and
for each closed $A \subseteq Ab$, if A attacks Δ then Δ attacks $A - \Delta$.

The proof of this theorem can be found in the appendix.

Definition 3. Let $\langle T, Ab, \text{attacks}\rangle$ be an argumentation-theoretic framework. For any sets of assumptions $\mathcal{D}, \Delta \subseteq Ab$,
$defends(\mathcal{D}, \Delta) \leftrightarrow \forall A \subseteq Ab\ [[A \text{ attacks } \Delta \wedge closed(A)] \rightarrow \mathcal{D} \text{ attacks } A - \Delta]$
where $closed(A)$ means "A is closed". We also say that $\mathcal{D}$ *defends* Δ.

This definition provides a specification for the predicate $defends$. This specification together with the auxiliary definitions of attackand *closed* and with definitions of set-theoretic operations and relationships will be referred to as $\mathcal{S}pec_{defends}$.

The following corollary, which follows directly from theorem 2, characterises admissibility and $\mathcal{S}pec_{adm}$ in terms of $defends$, and will be used to prove theorems 5 and 14 below.

Corollary 4.
1. *$\Delta \subseteq Ab$ is admissible iff Δ is closed and Δ defends Δ.*
2. *The specification $\mathcal{S}pec_{adm}$ can be expressed equivalently as*
   $adm(\Delta_0, \Delta) \leftrightarrow \Delta_0 \subseteq \Delta \wedge defends(\Delta, \Delta) \wedge closed(\Delta)$.

The proof procedure is given by the logic program

$$Prog_{adm}: \boxed{\begin{array}{l} adm(\Delta, \Delta) \leftarrow defends(\Delta, \Delta), closed(\Delta) \\ adm(\Delta, \Delta') \leftarrow defends(\mathcal{D}, \Delta), closed(\Delta \cup \mathcal{D}), adm(\Delta \cup \mathcal{D}, \Delta') \end{array}}$$

Note that, in the case of flat argumentation-theoretic frameworks, every set of assumptions is closed. Therefore, in this case, the conditions $closed(\Delta)$ and $closed(\Delta \cup \mathcal{D})$ in $Prog_{adm}$ can be omitted.

The soundness of $Prog_{adm}$ is expressed by corollary 6 below, which is a direct consequence of the following theorem:

Theorem 5. $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models Prog_{adm}$.

Proof : We prove the theorem by deriving the program $Prog_{adm}$ from the specification. By letting $\Delta_0 = \Delta$ in $\mathcal{S}pec_{adm}$, as formulated in corollary 4.2

$adm(\Delta_0, \Delta) \leftrightarrow \Delta_0 \subseteq \Delta \wedge defends(\Delta, \Delta) \wedge closed(\Delta)$

we immediately obtain the first clause of the program.

To obtain the second clause, we let $\Delta_0 = \Delta_0' \cup \mathcal{D}$ in the only-if half of $\mathcal{S}pec_{adm}$, as formulated in corollary 4.2, and observe that $\Delta_0' \cup \mathcal{D} \subseteq \Delta$ implies $\Delta_0' \subseteq \Delta$, obtaining

$adm(\Delta_0' \cup \mathcal{D}, \Delta) \rightarrow [\Delta_0' \subseteq \Delta \wedge defends(\Delta, \Delta) \wedge closed(\Delta)]$.

Then, by applying the if half of $\mathcal{S}pec_{adm}$, by transitivity of $\rightarrow$, we obtain

$adm(\Delta_0' \cup \mathcal{D}, \Delta) \rightarrow adm(\Delta_0', \Delta)$

which implies

$adm(\Delta_0' \cup \mathcal{D}, \Delta) \wedge closed(\Delta_0' \cup \mathcal{D}) \wedge defends(\mathcal{D}, \Delta_0') \rightarrow adm(\Delta_0', \Delta)$.

By renaming Δ_0' to Δ and Δ to Δ', we obtain the second clause of the program. □

Note that the derivation of the program $Prog_{adm}$ from the specifications $\mathcal{S}pec_{adm}$ and $\mathcal{S}pec_{defends}$ is achieved by simple deductive steps (e.g. transitivity of $\rightarrow$ and or introduction) possibly exploiting properties of the relations involved (e.g., of $\subseteq$).

Corollary 6. *For all $\Delta_0, \Delta \subseteq Ab$,*

if $Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$,

then $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$.

Namely, if, for some given $\Delta_0 \subseteq Ab$, the goal $\leftarrow adm(\Delta_0, X)$ succeeds for $X = \Delta$, with respect to $Prog_{adm}$ and assuming $\mathcal{S}pec_{defends}$, then Δ is an admissible superset of Δ_0. As a consequence, the procedure $Prog_{adm}$ is sound. The procedure $Prog_{adm}$ is also complete in the following sense:

Theorem 7. *For all $\Delta_0, \Delta \subseteq Ab$,*

if $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$

then $Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$.

Proof : Assume $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$. Then,

$\mathcal{S}pec_{defends} \models \Delta_0 \subseteq \Delta \wedge defends(\Delta, \Delta) \wedge closed(\Delta)$.

Then, by the first clause of $Prog_{adm}$

$Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta, \Delta)$.

There are two cases: (1) $\Delta_0 = \Delta$ and (2) $\Delta_0 \subset \Delta$.

In the first case, $Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$ immediately.

In the second case, since, trivially, any defence of a set Δ also defends any subset of Δ, i.e. for any sets of assumptions $D, \Delta, \Delta' \subseteq Ab$

$\mathcal{S}pec_{defends} \models defends(D, \Delta) \wedge \Delta' \subseteq \Delta \rightarrow defends(D, \Delta')$

then

$\mathcal{S}pec_{defends} \models defends(\Delta, \Delta_0) \wedge closed(\Delta)$.

Then, $Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta, \Delta) \wedge defends(\Delta, \Delta_0) \wedge closed(\Delta)$.

But $\Delta = \Delta \cup \Delta_0$. Therefore,

$Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta \cup \Delta_0, \Delta) \wedge defends(\Delta, \Delta_0) \wedge closed(\Delta \cup \Delta_0)$.

But then, by the second clause of $Prog_{adm}$,

$Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$. □

Namely, if Δ is an admissible superset of a given set of assumptions Δ_0, then the program $Prog_{adm}$, assuming $\mathcal{S}pec_{defends}$, successfully computes $X = \Delta$, given the goal $\leftarrow adm(\Delta_0, X)$. Note that the proof of completeness is achieved by symbolic execution of the program $Prog_{adm}$, and by appropriately choosing defences satisfying $\mathcal{S}pec_{defends}$.

The full proof procedure is obtained by adding to $Prog_{adm}$ a program $Prog_{defends}$ for computing *defends*, for checking *closed* and for computing the set-theoretic constructs, $\cup, \subseteq$, etc. This program may or may not be in the form of a logic program. If such a program is sound with respect to the specification $\mathcal{S}pec_{defends}$, then $Prog_{adm} \wedge Prog_{defends}$ is also sound, with respect to $\mathcal{S}pec_{adm}$ and $\mathcal{S}pec_{defends}$:

Theorem 8. *Given $Prog_{defends}$ such that, for all $\Delta, \mathcal{D} \subseteq Ab$,*
if $Prog_{defends} \models defends(\mathcal{D}, \Delta)$ then $\mathcal{S}pec_{defends} \models defends(\mathcal{D}, \Delta)$, and
if $Prog_{defends} \models closed(\Delta)$ then $\mathcal{S}pec_{defends} \models closed(\Delta)$,
then, for all $\Delta_0, \Delta \subseteq Ab$,
if $Prog_{adm} \wedge Prog_{defends} \models adm(\Delta_0, \Delta)$
then $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$.

The proof of this and the following theorem can be found in the appendix.

Moreover, if a given program $Prog_{defends}$ is complete with respect to the specification $\mathcal{S}pec_{defends}$, then $Prog_{adm} \wedge Prog_{defends}$ is also complete, with respect to $\mathcal{S}pec_{adm}$ and $\mathcal{S}pec_{defends}$. More precisely:

Theorem 9. *Given $Prog_{defends}$ such that, for all $\Delta, \mathcal{D} \subseteq Ab$,*
if $\mathcal{S}pec_{defends} \models defends(\mathcal{D}, \Delta)$ then $Prog_{defends} \models defends(\mathcal{D}, \Delta)$, and
if $\mathcal{S}pec_{defends} \models closed(\Delta)$ then $Prog_{defends} \models closed(\Delta)$
then, for all $\Delta_0, \Delta \subseteq Ab$,
if $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$
then $Prog_{adm} \wedge Prog_{defends} \models adm(\Delta_0, \Delta)$.

4 More efficient proof procedure

The proof procedure given by the program $Prog_{adm}$ performs a great deal of redundant computation. When a defence for the currently accumulated set of assumptions is generated, it is added to the accumulated set, without distinguishing between old assumptions that have already been defended and new assumptions that still have to be defended. As a consequence, defences for the old assumptions are recomputed redundantly when generating a defence for the new set. Moreover, when re-defending assumptions, new defences for such assumptions might be selected, different from the ones generated before, and these may need to be defended in turn. To avoid these redundancies, it suffices to distinguish in the currently accumulated set of assumptions, $\Delta \cup \mathcal{D}$, between those assumptions Δ that are already "defended" by $\Delta \cup \mathcal{D}$ itself and those assumptions $\mathcal{D}$ that have just been added to $\Delta \cup \mathcal{D}$ and require further defence. For this purpose, we employ a variant $adm^e(\Delta_0, \mathcal{D}, \Delta)$ of the predicate $adm(\Delta_0, \Delta)$.

Definition 10. Let $\langle T, Ab, \text{attacks}\rangle$ be an argumentation-theoretic framework. For any sets of assumptions $\Delta_0, \mathcal{D}, \Delta \subseteq Ab$,

$$adm^e(\Delta_0, \mathcal{D}, \Delta) \leftrightarrow \Delta_0 \cup \mathcal{D} \subseteq \Delta \ \wedge$$
$$[[defends(\Delta_0 \cup \mathcal{D}, \Delta_0) \wedge closed(\Delta_0 \cup \mathcal{D})] \rightarrow \Delta \text{ is admissible}].$$

We refer to this definition, together with that of admissibility, as $\mathcal{S}pec_{adm^e}$.

The relationship between adm and adm^e is given by the following lemma, whose proof can be found in the appendix.

Lemma 11. *For all sets of assumptions Δ_0 and Δ,*

1. *if $\mathcal{S}pec_{adm^e} \wedge \mathcal{S}pec_{defends} \models adm^e(\emptyset, \Delta_0, \Delta) \wedge closed(\Delta_0)$ then $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$;*
2. *if $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$ then $\mathcal{S}pec_{adm^e} \wedge \mathcal{S}pec_{defends} \models adm^e(\emptyset, \Delta_0, \Delta)$*

The top-most level of a procedure which computes the predicate adm^e is given by the logic program

$$Prog_{adm^e}: \boxed{\begin{array}{l} adm^e(\Delta, \emptyset, \Delta) \\ adm^e(\Delta, \mathcal{D}, \Delta') \leftarrow defends^e(\mathcal{D}', \Delta, \mathcal{D}), \\ \qquad\qquad\qquad\quad closed(\Delta \cup \mathcal{D} \cup \mathcal{D}'), \\ \qquad\qquad\qquad\quad adm^e(\Delta \cup \mathcal{D}, \mathcal{D}' - (\Delta \cup \mathcal{D}), \Delta') \end{array}}$$

where $defends^e$ is the variant of the predicate $defends$ specified as follows:

Definition 12. Let $\langle T, Ab, \text{attacks}\rangle$ be an argumentation-theoretic framework. For any sets of assumptions $\Delta, \mathcal{D}, \Delta' \subseteq Ab$,

$$defends^e(\mathcal{D}', \Delta, \mathcal{D}) \leftrightarrow$$
$$\forall A \subseteq Ab\, [[A\, attacks\, \mathcal{D} \wedge closed(A)] \rightarrow \mathcal{D}' \cup \Delta \cup \mathcal{D}\, attacks\, A - (\Delta \cup \mathcal{D})].$$

We will refer to this specification together with the definitions of attack, *closed* and the set-theoretic constructs as $\mathcal{S}pec_{defends^e}$.

The following corollary, which follows directly from theorem 2, characterises admissibility and $\mathcal{S}pec_{adm}$ in terms of $defends^e$, and will be used to prove theorem 14.

Corollary 13.
1. *Δ is admissible iff $defends^e(\Delta, \emptyset, \Delta)$ and $closed(\Delta)$.*
2. *$\mathcal{S}pec_{adm^e}$ is equivalent to*
$$adm^e(\Delta_0, \mathcal{D}, \Delta) \leftrightarrow \Delta_0 \cup \mathcal{D} \subseteq \Delta \ \wedge$$
$$[[defends^e(\mathcal{D}, \Delta_0, \Delta_0) \wedge closed(\Delta_0 \cup \mathcal{D})] \rightarrow \Delta \textit{ is admissible}].$$

The soundness of $Prog_{adm^e}$ is given by corollary 16 below, which follows directly from lemma 11 and from the following theorem:

Theorem 14. $\mathcal{S}pec_{adm^e} \wedge \mathcal{S}pec_{defends^e} \models Prog_{adm^e}$.

Proof : We prove the theorem by deriving the program $Prog_{adm^e}$ from the specification. By letting $\mathcal{D} = \emptyset$ and $\Delta = \Delta_0$ in $\mathcal{S}pec_{adm^e}$

$$adm^e(\Delta_0, \mathcal{D}, \Delta) \leftrightarrow \Delta_0 \cup \mathcal{D} \subseteq \Delta \ \wedge \ [defends(\Delta_0 \cup \mathcal{D}, \Delta_0) \wedge closed(\Delta_0 \cup \mathcal{D}) \rightarrow \Delta \text{ is admissible}]$$

we obtain

$$adm^e(\Delta, \emptyset, \Delta) \leftrightarrow \Delta \subseteq \Delta \wedge [defends(\Delta, \Delta) \wedge closed(\Delta) \rightarrow \Delta \text{ is admissible}]$$

equivalent to the first clause of $Prog_{adm^e}$ because of corollary 4.1.
To obtain the second clause, first replace the predicate adm^e in the second clause of the program by the equivalent specification in terms of $defends^e$ given by corollary 13.2, obtaining

$$\begin{array}{l}[\Delta \cup \mathcal{D} \subseteq \Delta' \wedge [[defends^e(\mathcal{D}, \Delta, \Delta) \wedge closed(\Delta \cup \mathcal{D})] \rightarrow \Delta' \text{ is admissible}]] \leftarrow \\ \quad [defends^e(\mathcal{D}', \Delta, \mathcal{D}) \wedge closed(\Delta \cup \mathcal{D} \cup \mathcal{D}') \wedge \Delta \cup \mathcal{D} \cup \mathcal{D}' \subseteq \Delta' \wedge \\ \quad [[defends^e(\mathcal{D}' - (\Delta \cup \mathcal{D}), \Delta \cup \mathcal{D}, \Delta \cup \mathcal{D}) \wedge closed(\Delta \cup \mathcal{D} \cup \mathcal{D}')] \rightarrow \\ \hfill \Delta' \text{ is admissible}]].\end{array}$$

This can be rewritten in the logically equivalent form

$$\begin{array}{l}[\Delta \cup \mathcal{D} \subseteq \Delta' \ \ \wedge \Delta' \text{ is admissible}] \leftarrow \\ \quad [defends^e(\mathcal{D}, \Delta, \Delta) \wedge closed(\Delta \cup \mathcal{D}) \wedge \\ \quad defends^e(\mathcal{D}', \Delta, \mathcal{D}) \wedge closed(\Delta \cup \mathcal{D} \cup \mathcal{D}') \wedge \Delta \cup \mathcal{D} \cup \mathcal{D}' \subseteq \Delta' \wedge \\ \quad [[defends^e(\mathcal{D}' - (\Delta \cup \mathcal{D}), \Delta \cup \mathcal{D}, \Delta \cup \mathcal{D}) \wedge closed(\Delta \cup \mathcal{D} \cup \mathcal{D}') \rightarrow \\ \hfill \Delta' \text{ is admissible}]].\end{array}$$

which follows immediately from the fact that

$$\Delta \cup \mathcal{D} \subseteq \Delta' \leftarrow \Delta \cup \mathcal{D} \cup \mathcal{D}' \subseteq \Delta'$$

and from the following lemma, whose proof can be found in the appendix. □

Lemma 15.
$defends^e(\mathcal{D}, \Delta, \Delta) \wedge defends^e(\mathcal{D}', \Delta, \mathcal{D}) \rightarrow defends^e(\mathcal{D}' - (\Delta \cup \mathcal{D}), \Delta \cup \mathcal{D}, \Delta \cup \mathcal{D})$.

As for $Prog_{adm}$ given in section 3, the derivation of $Prog_{adm^e}$ from the specifications $\mathcal{S}pec_{adm^e}$ and $\mathcal{S}pec_{defends^e}$ consists of simple deductive steps (here presented in a backward fashion), possibly exploiting properties of the relations involved (e.g. $\subseteq$ and $defends$, as expressed by lemma 15).

Corollary 16. *For all* $\Delta_0, \Delta \subseteq Ab$,
if $Prog_{adm^e} \wedge \mathcal{S}pec_{defends^e} \models adm^e(\emptyset, \Delta_0, \Delta) \wedge closed(\Delta_0)$,
then $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$.

Namely, if, for some given set of assumptions Δ_0, the goal $adm^e(\emptyset, \Delta_0, X)$ succeeds for $X = \Delta$, with respect to $Prog_{adm^e}$ and assuming $\mathcal{S}pec_{defends^e}$, then Δ is an admissible superset of Δ_0. As a consequence, the proof procedure $Prog_{adm^e}$ is sound. $Prog_{adm^e}$ is also complete in the following sense:

Theorem 17. *For all* $\Delta_0, \Delta \subseteq Ab$,
if $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$,
then $Prog_{adm^e} \wedge \mathcal{S}pec_{defends^e} \models adm^e(\emptyset, \Delta_0, \Delta)$.

Proof : Assume $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$. Then

$$\mathcal{S}pec_{defends} \models \Delta_0 \subseteq \Delta \wedge defends(\Delta, \Delta) \wedge closed(\Delta), \text{ and}$$

$\mathcal{S}pec_{defends^e} \models \Delta_0 \subseteq \Delta \wedge defends^e(\Delta, \emptyset, \Delta) \wedge closed(\Delta)$.
Moreover, it is easy to see that
$\mathcal{S}pec_{defends^e} \models [[\Delta_0 \subseteq \Delta \wedge defends^e(\Delta, \emptyset, \Delta)] \rightarrow defends^e(\Delta, \emptyset, \Delta_0)]$.
Therefore, (i) $\mathcal{S}pec_{defends^e} \models defends^e(\Delta, \emptyset, \Delta_0)$.
Similarly,
$\mathcal{S}pec_{defends^e} \models [\Delta_0 \subseteq \Delta \wedge defends^e(\Delta, \emptyset, \Delta) \rightarrow defends^e(\Delta, \Delta_0, \Delta - \Delta_0)]$.
Therefore, (ii) $\mathcal{S}pec_{defends^e} \models defends^e(\Delta, \Delta_0, \Delta - \Delta_0)$.
To show $Prog_{adm^e} \wedge \mathcal{S}pec_{defends^e} \models adm^e(\emptyset, \Delta_0, \Delta)$, use the following instance of the second clause of the program:

$$\begin{aligned} adm^e(\emptyset, \Delta_0, \Delta) \leftarrow & defends^e(\mathcal{D}', \emptyset, \Delta_0), \\ & closed(\Delta_0 \cup \mathcal{D}'), \\ & adm^e(\Delta_0, \mathcal{D}' - \Delta_0, \Delta) \end{aligned}$$

Let $\mathcal{D}' = \Delta$. Then, the first condition is provable from $\mathcal{S}pec_{defends^e}$ by (i), and the second condition is provable from $\mathcal{S}pec_{defends^e}$ since $\Delta_0 \subseteq \Delta$ and Δ is closed. To prove the third condition, use the following instance of the second clause of the program:

$$\begin{aligned} adm^e(\Delta_0, \Delta - \Delta_0, \Delta) \leftarrow & defends^e(\mathcal{D}', \Delta_0, \Delta - \Delta_0), \\ & closed(\Delta \cup \mathcal{D}'), \\ & adm^e(\Delta, \mathcal{D}' - \Delta, \Delta) \end{aligned}$$

Let $\mathcal{D}' = \Delta$. Then, the first condition is provable from $\mathcal{S}pec_{defends^e}$ by (ii), and the second condition is provable from $\mathcal{S}pec_{defends^e}$ since Δ is closed. Moreover, the third condition is provable by the first clause of $Prog_{adm^e}$. Therefore,
$Prog_{adm^e} \wedge \mathcal{S}pec_{defends^e} \models adm^e(\emptyset, \Delta_0, \Delta)$. □

As in section 3, the proof of completeness is achieved by symbolic execution of the procedure $Prog_{adm^e}$, with two calls to the specification $\mathcal{S}pec_{defends^e}$.

The full proof procedure is obtained by adding to $Prog_{adm^e}$ a program $Prog_{defends^e}$ for computing $defends^e$, for checking *closed* and for computing the set-theoretic constructs, $\cup, \subseteq$, etc. In [6] we give the top-most level of a (generalised) logic program for computing $defends^e$, which provides a sound but incomplete proof procedure.

5 Conclusion

We have used logic program development technology to define two proof procedures for the admissibility semantics for the abstract, argumentation-theoretic framework presented in [2].

Rather than develop new methods, we have employed existing techniques of deductive synthesis [3] to derive two small but non-trivial programs and to prove them sound, and techniques of symbolic execution to prove them complete.

The second program is an improvement of the first, obtained by adding an argument, $\mathcal{D}$, to the predicate *adm*, thus obtaining a predicate adm^e. The new argument plays the role of an accumulator, and gives rise to a more efficient proof procedure $Prog_{adm^e}$. This is re-synthesised from scratch from a new specification for adm^e. As a subject for future work, it would be interesting to explore the possibility of deriving $Prog_{adm^e}$ from the initial, inefficient proof procedure, $Prog_{adm}$, using standard techniques of logic program transformation (fold, unfold and so on, see [17]) and/or techniques borrowed from functional programming (e.g., see [16]).

Acknowledgements

This research was supported by the EEC activity KIT011-LPKRR. The third author was partially supported by EEC HCM Project no CHRX-CT93-00414, "Logic Program Synthesis and Transformation". The authors are grateful to the LOPSTR'96 participants for helpful comments and suggestions.

References

1. A. Bondarenko, F. Toni, R. A. Kowalski, An assumption-based framework for non-monotonic reasoning. *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, Lisbon, Portugal (1993), MIT Press (L. M. Pereira and A. Nerode, eds) 171–189
2. A. Bondarenko, P. M. Dung, R. A. Kowalski, F. Toni, An abstract, argumentation-theoretic framework for default reasoning. To appear in *Artificial Intelligence*, Elsevier.
3. Y. Deville, K.-K. Lau, Logic program synthesis. *Journal of Logic Programming* 19/20 (1994), Elsevier, 321–350
4. P. M. Dung, Negation as hypothesis: an abductive foundation for logic programming. *Proceedings of the 8th International Conference on Logic Programming*, Paris, France (1991), MIT Press (K. Furukawa, ed.) 3–17
5. P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France (1993), Morgan Kaufmann (R. Bajcsy, ed.) 852–857
6. P. M. Dung, R. A. Kowalski, F. Toni, Argumentation-theoretic proof procedures for non-monotonic reasoning. Logic Programming Section Technical Report, Department of Computing, Imperial College, London (1996)
7. K. Eshghi, R.A. Kowalski, Abduction through deduction. Logic Programming Section Technical Report, Department of Computing, Imperial College, London (1988)
8. K. Eshghi, R. A. Kowalski, Abduction compared with negation as failure. *Proceedings of the 6th International Conference on Logic Programming*, Lisbon, Portugal (1989), MIT Press (G. Levi and M. Martelli, eds) 234–254
9. M. Gelfond, V. Lifschitz, The stable model semantics for logic programming. *Proceedings of the 5th International Conference on Logic Programming*, Washington, Seattle (1988), MIT Press (K. Bowen and R. A. Kowalski, eds) 1070–1080
10. A. C. Kakas, P. Mancarella. Preferred extensions are partial stable models. *Journal of Logic Programming* 14(3,4) (1993), Elsevier, 341–348
11. A. C. Kakas, P. Mancarella, P.M. Dung, The Acceptability Semantics for Logic Programs. *Proceedings of the 11th International Conference on Logic Programming*, Santa Margherita Ligure, Italy (1994), MIT Press (P. van Hentenryck, ed.) 504–519

12. R.A. Kowalski. *Logic for problem solving.* Elsevier, New York (1979)
13. J. McCarthy, Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence* 13 (1980), Elsevier, 27–39
14. D. McDermott, Nonmonotonic logic II: non-monotonic modal theories. *Journal of ACM* 29(1) (1982) 33–57
15. R. Moore, Semantical considerations on non-monotonic logic. *Artificial Intelligence* 25 (1985), Elsevier, 75–94
16. R. Paige, S. Koenig, Finite differencing of computable expressions. *ACM Transactions on Programming Languages Systems* 4(3) (1982), ACM Press, 402–454
17. A. Pettorossi, M. Proietti, Transformation of logic programs. *Journal of Logic Programming* 19/20 (1994), Elsevier, 261–320
18. D. Poole, A logical framework for default reasoning. *Artificial Intelligence* 36 (1988), Elsevier, 27–47
19. R. Reiter, A logic for default reasoning. *Artificial Intelligence* 13 (1980), Elsevier, 81–132
20. D. Saccà, C. Zaniolo, Stable model semantics and non-determinism for logic programs with negation. *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Nashville, Tennessee (1990) ACM Press, 205–217

Appendix

Proof of theorem 2

$\Rightarrow$ Given a closed attack A against Δ, we need to prove only that Δ attacks $A - \Delta$. Since Δ is admissible, Δ attacks A. But, if Δ attacks $A \cap \Delta$, then Δ attacks itself, contradicting the hypothesis that Δ is admissible.

$\Leftarrow$ We need to prove only that Δ does not attack itself. Suppose that Δ attacks itself. Then, Δ attacks $\Delta - \Delta = \emptyset$. But, by definition of attack, no set can attack $\emptyset$.

Proof of theorem 8

Assume $Prog_{adm} \wedge Prog_{defends} \models adm(\Delta_0, \Delta)$. Then, since $Prog_{defends}$ is sound with respect to $\mathcal{S}pec_{defends}$,

$Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$.

Then, directly from corollary 6,

$\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$.

Proof of theorem 9

Assume $\mathcal{S}pec_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$. Then, directly from theorem 7,

$Prog_{adm} \wedge \mathcal{S}pec_{defends} \models adm(\Delta_0, \Delta)$. By completeness of $Prog_{defends}$,

$Prog_{adm} \wedge Prog_{defends} \models adm(\Delta_0, \Delta)$.

Proof of lemma 11

1. First, note that, $adm^e(\emptyset, \Delta_0, \Delta) \wedge closed(\Delta_0)$ implies
   $\Delta_0 \subseteq \Delta \wedge [[defends(\Delta_0, \emptyset) \wedge closed(\Delta_0)] \rightarrow \Delta \text{ is admissible}] \wedge closed(\Delta_0)$.
   But $\mathcal{S}pec_{defends}$ trivially implies $defends(\Delta_0, \emptyset)$. Therefore

$\Delta_0 \subseteq \Delta \;\; \wedge [closed(\Delta_0) \rightarrow \Delta$ is admissible$] \wedge closed(\Delta_0)$
which, in $\mathcal{S}pec_{adm}$, implies $adm(\Delta_0, \Delta)$.

2. $adm(\Delta_0, \Delta)$ implies $\Delta_0 \subseteq \Delta \;\; \wedge \Delta$ is admissible.
   This trivially implies
   $\Delta_0 \subseteq \Delta \;\; \wedge [[closed(\Delta_0) \wedge defends(\Delta, \emptyset)] \rightarrow \Delta$ is admissible$]$
   which, in $\mathcal{S}pec_{adm^e}$ implies $adm^e(\emptyset, \Delta_0, \Delta)$.

Proof of lemma 15 : Assume
(i) $defends^e(\mathcal{D}, \Delta, \Delta)$, and
(ii) $defends^e(\mathcal{D}', \Delta, \mathcal{D})$.
Assume $A \subseteq Ab$ attacks $\Delta \cup \mathcal{D}$. We need to show that $\mathcal{D}' \cup \Delta \cup \mathcal{D}$ attacks $A - (\Delta \cup \mathcal{D})$.

- If A attacks $\mathcal{D}$ then, by (ii), $\mathcal{D}' \cup \Delta$ attacks $A - (\Delta \cup \mathcal{D})$, and thus $\mathcal{D}' \cup \Delta \cup \mathcal{D}$ attacks $A - (\Delta \cup \mathcal{D})$.
- If A attacks Δ then, by (i), $\mathcal{D} \cup \Delta$ attacks $A - \Delta$. It suffices to show that $\mathcal{D} \cup \Delta$ does not attack $\mathcal{D}$. Suppose, on the contrary, that $\mathcal{D} \cup \Delta$ attacks $\mathcal{D}$. Then, by (ii), $\mathcal{D}' \cup \Delta$ attacks $(\mathcal{D} \cup \Delta) - (\mathcal{D} \cup \Delta) = \emptyset$. But this is not possible, because, by definition of attack, there are no attacks against $\emptyset$.