Argumentation-theoretic proof procedures for logic programming

Francesca Toni

Logic Programming Section Technical Report

Imperial College of Science, Technology and Medicine,
Department of Computing
180 Queen's Gate, London SW7 2BZ, UK

April 1997

Abstract

We instantiate the abstract, argumentation-based proof procedure for (credulous) default reasoning proposed in [4, 3] to the normal logic programming case. The proof procedure computes the partial stable model [15]/preferred extension semantics [2] for logic programming. The proof procedure is sound but incomplete. We employ the credulous proof procedure to define a sound sceptical proof procedure, adapting the results of [16]. We compare the resulting proof procedures with others, computing the same semantics, notably the ones in [5, 2, 16].

1 Introduction

In [1], we have shown that many formalisms for default reasoning [10, 14, 11, 12] can be understood as special cases of a single abstract framework, based upon Theorist [13] and an abductive interpretation of the semantics of logic programming [5]. In the framework, default reasoning in general is understood as extending a theory formulated in a monotonic logic by a defeasible set of assumptions. An assumption can be defeated (or "attacked") if its "contrary" can be proved, possibly with the aid of other conflicting assumptions. We have proposed a new semantics for default reasoning, more liberal than their standard (stable) semantics and abstracting the admissibility/preferred extension semantics for logic programming [2], which is equivalent to the partial stable model semantics [15] (see [7]) and regular model semantics [18].

In [3, 4] we have defined two proof procedures for computing the abstract admissibility semantics. The second proof procedure is a computationally more efficient refinement of the first. Both procedures are formulated as logic programs and are sound and complete with respect to the semantics. There are two obvious advantages of formulating proof procedures as logic programs: it is easier to relate to the semantics (specifications) and prove soundness; and the logic program is more abstract than conventional algorithms, so that different algorithms can be obtained by employing different control strategies.

The logic programs describing the proof procedures of [3, 4] define topmost level predicates in terms of lower-level predicates that can then be "developed", differently for different concrete instance of the framework.

In the present paper, we define the lower-level predicates for the more efficient abstract proof procedure in the logic programming case (see also [17]). The resulting procedure is in the form of a (generalised) logic program that can be almost directly compiled by any Prolog system, by using the `forall` construct. The proof procedure is sound but not complete, since the definition of the lower-level predicates is not complete.

The proof procedure can be employed to perform credulous reasoning directly, and sceptical reasoning by adapting the method of [16].

We restrict ourselves to the propositional case.

The rest of the paper has the following structure: Section 2 reviews the main features of the abstract framework and the abstract admissibility semantics Section 3 reviews the abstract more efficient proof procedure. Section 4 defines the (credulous) proof procedure for the logic programming case. Section 5 defines an abstract, sceptical proof procedure and instantiates it for logic programming by employing the proof procedure of section 4. Section 6 compares the proof procedures for logic programming given in sections 4 and 5 with the ones given in [5, 2] and [16]. Section 7 concludes.

2 Abstract 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. The notion of attack is formulated in terms of a notion of a sentence being the "contrary" of an assumption.

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 properties that no set of assumptions attacks the empty set of assumptions and that if a set of assumptions attacks the union of two sets of assumptions then the first set attacks either of the two (or both).

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

Logic programming, Theorist [13], (some cases of) circumscription [10], default logic [14], autoepistemic logic [12] and non-monotonic modal logic [11] are all instances of the abstract argumentation-theoretic framework (see [1]). In the *logic programming case*:

- T is a logic program (thought of, without loss of generality, as the set of all variable-free instances over its Herbrand universe of all its clauses), i.e. a set of clauses of the form

  $p_0 \leftarrow p_1, \ldots, p_n, not\, p_{n+1}, \ldots, not\, p_{n+m}$
  with the p_i (variable-free) atoms and $n, m \geq 0$;
- $Ab = \{not\, p \,|\, p$ belongs to the Herbrand base of $T\}$;
- $Th(T)$ is closure of T under modus ponens for $\leftarrow$;
- given two sets of assumptions A and Δ, A attacks Δ iff there exists $not\, p \in \Delta$ such that $p \in Th(T \cup A)$;
- every set of assumptions is closed and therefore Δ is admissible iff Δ does not attack itself and for each $A \subseteq Ab$, if A attacks Δ then Δ attacks A.

In the logic programming instance, admissible sets of assumptions correspond to admissible scenaria [2], and maximally admissible sets of assumptions correspond to preferred extensions [2] and partial stable models [15]. The standard semantics of scenaria in Theorist [13], extensions in default logic [14], stable expansions in autoepistemic logic [12], fixed points in non-monotonic modal logic [11] and stable models in logic programming [6] 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. [1]

The semantics of admissible (and stable) sets of assumptions is *credulous*, in the sense that a sentence δ is a *non-monotonic consequence* of a theory T iff δ belongs to *some* "extension" [2] 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 [10] can be understood as the sceptical semantics corresponding to stable sets of assumptions.

3 Abstract proof procedure

In [3, 4] we define two proof procedures, in the form of logic programs, for the computation of credulous non-monotonic consequences, for any abstract argumentation-theoretic framework, using the admissibility semantics. The top-most level of the more efficient proof procedure is

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

Intuitively, $adm^e(\Delta, \mathcal{D}, \Delta')$ holds iff assuming that Δ is given (is the set of assumptions "accumulated so far") and $\mathcal{D}$ has been computed to "defend" Δ, checking $\mathcal{D}$ will possibly add further "defences" to return an admissible Δ', containing both Δ and $\mathcal{D}$. More precisely, the lower level predicates *closed* and $defends^e$ are defined by the formulas

$\forall \Delta \subseteq Ab\ [closed(\Delta) \leftrightarrow \Delta = Th(T \cup \Delta) \cap Ab]$

$\forall \mathcal{D}', \Delta, \mathcal{D} \subseteq Ab\ [defends^e(\mathcal{D}', \Delta, \mathcal{D}) \leftrightarrow$

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

Such formulas act as specifications for (possibly logic) programs computing *closed* and $defends^e$.

The proof procedure $Prog_{adm^e}$ is sound with respect to the credulous version of the admissibility semantics in the following sense: if, for some given set of assumptions Δ_0, the goal $adm^e(\emptyset, \Delta_0, X) \wedge closed(\Delta_0)$ follows, for $X = \Delta$, from $Prog_{adm^e}$ and the specification for *closed* and $defends^e$, then Δ is an admissible superset of Δ_0. Typically, Δ_0 is a set of assumptions such that $\alpha \in Th(T \cup \Delta_0)$ for some formula (query) α.

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

[2] An *extension* of T is a theory $Th(T \cup \Delta)$ for some set of assumptions $\Delta \subseteq Ab$.

Moreover, $Prog_{adm^e}$ is complete in the following sense: if, for some given set of assumptions Δ, Δ_0, Δ is an admissible superset of Δ_0, then the goal $adm^e(\emptyset, \Delta_0, \Delta)$ follows from $Prog_{adm^e}$ and the specification for *closed* and $defends^e$.

The full proof procedure is obtained by providing programs computing the lower-level predicates *closed* and $defends^e$. The specification of $defends^e(\mathcal{D}', \Delta, \mathcal{D})$ requires $\mathcal{D}' \cup \Delta$ to attack *all* (closed) attacks A against $\mathcal{D}$. In the general case, the number of such attacks A can be infinite, because any superset of an attack is also an attack. [3] proves that it is possible to restrict considerably the number of attacks A that need to be considered, by limiting attention to the closure of minimal attacks, and the following is an equivalent specification for $defends^e$:

$$\begin{aligned} &\forall \mathcal{D}', \Delta, \mathcal{D} \subseteq Ab\ [defends^e(\mathcal{D}', \Delta, \mathcal{D}) \leftrightarrow \\ &\quad \forall A \subseteq Ab \quad [\textit{min-closed-attacks}(A, \mathcal{D}) \rightarrow \\ &\quad\quad \mathcal{D}' \cup \Delta \cup \mathcal{D} attacks\ A - (\Delta \cup \mathcal{D})]] \end{aligned}$$

where the predicate *min-closed-attacks* is defined by the following formula (specification):

$$\begin{aligned} &\forall A, \Delta \subseteq Ab\ [\textit{min-closed-attacks}(A, \Delta) \leftrightarrow \\ &closed(A) \wedge A attacks\ \Delta \wedge\ \nexists A' \subseteq Ab[closed(A') \wedge A'\, attacks\ \Delta \wedge A' \subset A]]. \end{aligned}$$

A sound (but incomplete, see [3]) program computing $defends^e$ is given by the (generalised, because of the presence of universal quantifiers and implications in the body of clauses) logic program

$$Prog_{defends^e}: \boxed{\begin{aligned} &defends^e(\mathcal{D}', \Delta, \mathcal{D}) \leftarrow \\ &\quad \forall A \subseteq Ab \quad [attacks^*(A, \mathcal{D}) \rightarrow \\ &\quad\quad attacks^*(\mathcal{D}' \cup \Delta \cup \mathcal{D}, A - (\Delta \cup \mathcal{D}))] \end{aligned}}$$

for any predicate $attacks^*$ such that

$$\forall A, \Delta \subseteq Ab\ [\textit{min-closed-attacks}(A, \Delta) \rightarrow attacks^*(A, \Delta)]$$
$$\forall A, \Delta \subseteq Ab\ [attacks^*(A, \Delta) \rightarrow A\, attacks\ \Delta].$$

Namely, $attacks^*$ is intermediate between *min-closed-attacks* and *attacks*. In particular, $attacks^*$ can be chosen to be *min-closed-attacks*. However, in general it might be too costly to check for minimality of attacks, and some weaker notion of $attacks^*$ might be appropriate (see section 4).

The proof procedure $Prog_{adm^e} \wedge Prog_{defends^e}$ needs to be extended with programs defining $attacks^*$ and *closed*. Such a program will be different for different instances of the abstract argumentation-theoretic framework. In the next section we define such programs for the logic programming case. Because in the logic programming case all sets of assumptions are automatically closed, the atoms in the predicate *closed* can be omitted from $Prog_{adm^e}$ and from the specification for *min-closed-attacks*. In such case, minimal closed attacks are minimal (with respect to subset inclusion) attacks.

4 Proof procedure for logic programming

The predicate *attacks** is computed by the following

$$Prog_{attacks^*}: \boxed{\begin{array}{ll} attacks^*(A,\Delta) \leftarrow & not\,P \in \Delta, \\ & demo(T \cup A, P), \\ & A \subseteq Ab \end{array}}$$

where *demo* is specified by:

$\forall \Delta \subseteq Ab\,[demo(T \cup \Delta, \alpha) \leftrightarrow \alpha \in Th(T \cup \Delta)]$.

It follows directly from the abstract soundness results summarised in the previous section 3 (and from theorem 4.2 below) that $Prog_{adm^e} \wedge Prog_{defends^e}$ $\wedge$ $Prog_{attacks^*}$ is a sound proof procedure to compute the admissibility semantics for logic programming, in the following sense:

Theorem 4.1 *If, for some given set of assumptions (variable-free negative literals) Δ_0, the goal $adm^e(\emptyset, \Delta_0, X)$ follows, for $X = \Delta$, from $Prog_{adm^e}$ $\wedge$ $Prog_{defends^e} \wedge Prog_{attacks^*}$ and the specification for demo, then Δ is an admissible superset of Δ_0.*

Theorem 4.2 *Given sets of assumptions $A, \Delta \subseteq Ab$,*

- *if min-closed-attacks(A,Δ) holds with respect to the instance for logic programming of the specification of min-closed-attacks in section 3 then the goal attacks$^*(A,\Delta)$ follows from $Prog_{attacks^*}$ and the specification for demo;*

- *if the goal attacks$^*(A,\Delta)$ follows from $Prog_{attacks^*}$ and the specification for demo then A attacks Δ.*

The proof of this theorem is straightforward.

It is easy to see that, by definition of Th for logic programming, $\alpha \in Th(T \cup \Delta)$ holds iff there is an SLD derivation of α from $T \cup \Delta$, interpreting negative literals $not\,P \in \Delta$ as (positive) facts (see [5]). Therefore, following [9], the predicate *demo* can be computed by the following logic program: [3]

$demo(T', P) \leftarrow demo(T', P \leftarrow Body), demo(T', Body)$
$demo(T', L \wedge Rest) \leftarrow demo(T', L), demo(T', Rest)$
$demo(T', true)$
$demo(T', not\,P) \leftarrow not\,P \in T'$

Note that the last clause formalises the treatment of negative literals as positive facts mentioned earlier on.

This program for *demo* is just one of the various alternatives to define the *demo* predicate so that it satisfies the given specification. Other alternatives are discussed in [9].

[3] We assume that, for any $T' = T \cup \Delta$, for any logic program T and set of assumptions Δ, $demo(T', C)$ holds for every clause $C \in T$, that conjunctions of literals always have $true$ as the last conjunct, that facts $P \in T$ are clauses of the form $P \leftarrow true$.

Therefore, the overall proof procedure for a given logic program T is given by the following (generalised) metalogic program

$$
Proc: \quad \boxed{\begin{array}{l}
adm^e(\Delta, \emptyset, \Delta) \\
adm^e(\Delta, \mathcal{D}, \Delta') \leftarrow defends^e(\mathcal{D}', \Delta, \mathcal{D}), \\
\qquad\qquad\qquad\qquad adm^e(\Delta \cup \mathcal{D}, \mathcal{D}' - (\Delta \cup \mathcal{D}), \Delta') \\
defends^e(\mathcal{D}', \Delta, \mathcal{D}) \leftarrow \\
\quad \forall A \subseteq Ab \quad [attacks^*(A, \mathcal{D}) \rightarrow \\
\qquad\qquad attacks^*(\mathcal{D}' \cup \Delta \cup \mathcal{D}, A - (\Delta \cup \mathcal{D}))] \\
attacks^*(A, \Delta) \leftarrow \quad not\, P \in \Delta, \\
\qquad\qquad\qquad\quad demo(T \cup A, P) \\
\qquad\qquad\qquad\quad A \subseteq Ab \\
demo(T', P) \leftarrow demo(T', P \leftarrow Body), demo(T', Body) \\
demo(T', L \wedge Rest) \leftarrow demo(T', L), demo(T', Rest) \\
demo(T', true) \\
demo(T', not\, P) \leftarrow not\, P \in T'
\end{array}}
$$

Directly from corollary 4.1 and by the soundness of the program computing *demo*, *Proc* is sound, i.e. if, given some set of assumptions (negative literals) Δ_0, the goal $adm^e(\emptyset, \Delta_0, X)$ follows, for $X = \Delta$, from *Proc*, then Δ is an admissible superset of Δ_0. Note that the program defining *demo* can be used to compute Δ_0 such that $\alpha \in Th(T \cup \Delta_0)$, for some query α, by returning $X = \Delta_0$ for the goal

$demo(T \cup X, \alpha) \wedge X \subseteq Ab$.

Proc is not complete, as illustrated in [3] for its abstract version.

Proc can be almost directly implemented in any Prolog system, by using the `forall` construct. Such direct implementation would not be efficient, due to the generate and test style of *Proc*: for example, in $Prog_{attacks^*}$, a set A is generated first, and then it is checked that A consists of assumptions only. Effective implementations should remove such inefficiencies.

Many different control strategies can be employed to obtain an actual algorithm to compute admissible sets of assumptions (see algorithm = logic + control [8]). The control strategy of Prolog might not be always appropriate, even if the obvious inefficiencies are eliminated from *Proc*, as illustrated by the following example.

Example 4.1 Consider the logic program T

$p \leftarrow not\, q, p$

q

and the query $not\, p$. Execution of $demo(T \cup \Delta_0, not\, p) \wedge \Delta_0 \subseteq Ab$ returns $\Delta_0 = \{not\, p\}$. There are no attacks against $\{not\, p\}$, therefore Δ_0 is admissible. However, by applying the control strategy of Prolog, the goal $adm^e(\emptyset, \Delta_0, X)$ does not terminate, since the derived subgoal $attacks^*(A, \Delta_0)$ does not terminate. A better control strategy would be to "freeze" the resolution of the subgoal $attacks^*(A, \Delta_0)$ once $not\, q$ is identified as a member of a "possible attack" A, since $not\, q$ is attacked by $\emptyset$ and the procedure can terminate. In section 6 we will describe such control strategy, which is the one needed to simulate the proof procedure of [5] using our metalogic program *Proc*.

5 Sceptical proof procedure for logic programming

Since the empty set of assumptions is admissible in the logic programming case, whatever the logic program T might be, the sceptical version of the admissibility semantics is not meaningful. In this section we consider the sceptical version of the *preferred extension* semantics, where a set of assumptions is *preferred* iff it is maximally admissible.

The following theorem states that to check that a sentence is a sceptical non-monotonic consequence of a theory, with respect to the preferred extension semantics, it is not needed to enumerate all preferred sets of assumptions, but it is sufficient to guarantee that the sentence holds in at least one admissible extension, and then restrict attention to admissible extensions attacking the given one. It is a version of theorem 16 in [16], as we will discuss in section 6 in some depth.

Theorem 5.1 *Given an argumentation-theoretic framework $\langle T, Ab, attacks\rangle$ and a sentence α in its language,*

α is a sceptical non-monotonic consequence of T with respect to the preferred extension semantics, i.e. $\alpha \in Th(T \cup \Delta)$ for all preferred $\Delta \subseteq Ab$, iff

1. *$\alpha \in Th(T \cup \Delta_0)$, for some admissible set of assumptions $\Delta_0 \subseteq Ab$, and*
2. *for every set of assumptions $\Delta \subseteq Ab$,*
   if Δ is admissible and Δ attacks Δ_0,
   then $\alpha \in Th(T \cup \Delta')$ for every set of assumptions $\Delta' \subseteq Ab$ such that
   (a) *$\Delta' \supseteq \Delta$, and*
   (b) *Δ' is admissible.*

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

The following corollary reformulates the above theorem in terms of the predicates used in *Proc*.

Corollary 5.1 Given an argumentation-theoretic framework $\langle T, Ab, \text{attacks}\rangle$ and a sentence α in its language,

α is a sceptical non-monotonic consequence of T with respect to the preferred extension semantics, i.e. $\alpha \in Th(T \cup \Delta)$ for all preferred $\Delta \subseteq Ab$, if

$$\begin{array}{l} demo(T \cup \Delta_0^*, \alpha) \wedge closed(\Delta_0^*) \wedge adm^e(\emptyset, \Delta_0^*, \Delta_0) \wedge \\ \forall \Delta^* \subseteq Ab\,[\; attacks^*(\Delta^*, \Delta_0) \wedge closed(\Delta^*) \wedge adm^e(\emptyset, \Delta^*, \Delta) \rightarrow \\ \qquad\qquad \forall \Delta' \subseteq Ab\,[adm^e(\emptyset, \Delta, \Delta') \rightarrow demo(T \cup \Delta', \alpha)]]. \end{array}$$

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

This result can be used to define a sceptical proof procedure for logic programming with respect to the preferred extension semantics, by employing $Proc$ in section 4 for resolving the predicates adm^e, $attacks^*$ and $demo$. (In the logic programming case, all atoms in the predicate $closed$ can be ignored.) More precisely, let

$$Proc': \boxed{\begin{array}{l} check\text{-}all(\Delta_0, \alpha) \leftarrow \\ \forall \Delta^* \subseteq Ab\,[\; attacks^*(\Delta^*, \Delta_0) \wedge adm^e(\emptyset, \Delta^*, \Delta) \rightarrow \\ \qquad\qquad \forall \Delta' \subseteq Ab\,[adm^e(\emptyset, \Delta, \Delta') \rightarrow \\ \qquad\qquad\qquad\qquad\qquad demo(T \cup \Delta', \alpha)]]. \end{array}}$$

$Proc'$ is a (generalised) logic program.

Soundness of $Proc' \wedge Proc$ with respect to the (sceptical) preferred extension semantics follows directly from corollary 5.1: if the goal

$demo(T \cup \Delta_0^*, \alpha) \wedge closed(\Delta_0^*) \wedge adm^e(\emptyset, \Delta_0^*, \Delta_0) \wedge check\text{-}all(\Delta_0, \alpha)$

follows from $Proc' \wedge Proc$ then α is a sceptical non-monotonic consequence of T with respect to the preferred extension semantics.

$Proc' \wedge Proc$ is incomplete since $Proc$ is and since $attacks^*$ only approximates the notion of attack.

6 Comparisons

[5, 2] propose a proof procedure for computing preferred extensions. This is based on interleaving abductive derivations, for the generation of negative literals to "derive" goals, and consistency derivations, to check consistency of negative literals with atoms "derivable" from the program. The behaviour of this proof procedure can be obtained by executing the metaprogram $Proc$ in a special way. To show this, first note that, roughly speaking, consistency derivations in [5] correspond to finding all A such that $attacks^*(A, \mathcal{D})$ holds, in the definition of $defends^e$, and abductive derivations in [5] correspond to finding sets $\mathcal{D}'$ such that $attacks^*(\mathcal{D}' \cup \Delta \cup \mathcal{D}, A - (\Delta \cup \mathcal{D}))$ in the definition of $defends^e$. The definition of adm^e reflects the recursive/iterative nature of the proof procedure. The correspondence is not exact under a Prolog-like execution of $Proc$ since

- in $Proc$ (a superset of) the set of all minimal attacks (against the given set of assumptions) is constructed before they are counter attacked, while in the procedure of [5] assumptions in each "potential" attack (against the given set of assumptions) are counter attacked as soon as they are generated, and

- in $Proc$ assumptions are added to the current version of the computed set of assumptions in bulk, while in the procedure of [5] assumptions are added one by one, as soon as they are identified.

However, by interleaving the execution of different subgoals in $Proc$, we can simulate the proof procedure of [5] exactly. Roughly, as soon as an assumption possibly belonging to A is found during the execution of $attacks^*(A, \mathcal{D})$, the subgoal $attacks^*(\mathcal{D}' \cup \Delta \cup \mathcal{D}, A - (\Delta \cup \mathcal{D}))$ should immediately be activated. Similarly, as soon as an assumption possibly belonging to $\mathcal{D}'$ is found during the execution of $attacks^*(\mathcal{D}' \cup \Delta \cup \mathcal{D}, A - (\Delta \cup \mathcal{D}))$, the subgoal $attacks^*(A, \mathcal{D}')$ should immediately be activated.

Therefore, the proof procedure in [5] is a special instance of our more declaratively formulated $Proc$.

A version of theorem 5.1 has been proved in [16] (theorem 16). Whereas [16] uses the notion of "conflict-free set of arguments" (which is an atomic, abstract notion), we use the notion of admissible set of assumptions. Moreover, theorem 16 in [16] replaces the condition in part 2 of theorem 5.1 "Δ' attacks Δ_0" by the (equivalent) condition corresponding to "$\Delta' \cup \Delta_0$ attacks itself".

To see the correspondence between our theorem 5.1 and theorem 16 in [16], note that, in the logic programming case [4]

1. (α, Δ) is an "argument" in [16] iff $\alpha \in Th(T \cup \Delta)$,

2. a conflict-free set of arguments Arg is conflict-free [16] iff $\bigcup_{(\alpha,\Delta) \in Arg} \Delta$ is admissible.

(See the appendix for the proofs.)

The second result proves the guess, in [16], that the semantics of maximally conflict-free sets of arguments coincides with the regular model semantics for logic programming[18], in turn coinciding with the partial stable model semantics [15] and the preferred extension semantics [2] (see [18] for the equivalence between regular models and preferred extensions/partial stable models).

The proof procedure defined in [16] is given in imperative style as follows, for some variable-free literal α and set of variable-free negative literals $\mathcal{D}$ ($\mathcal{D}$ is initialised to $\emptyset$):

1. choose Δ_0 such that $find\_argument(\mathcal{D}, \alpha)$ returns Δ; if no such Δ exists then failure; otherwise, go to step 2;

2. find Ω returned by $complete\_conflicting(\mathcal{D}, \Delta)$; while $\Omega \neq \emptyset$, do choose $A \in \Omega$, set Ω to $\Omega - \{A\}$, and go to step 1 instantiating the parameter $\mathcal{D}$ to $\mathcal{D} \cup A$; success

[4] The abstract notions of "arguments" and "conflict-free sets of arguments" are given as atomic in [16], and therefore cannot be compared with our "derived" abstract notions of "extensions" and admissible sets of assumptions. The comparison is feasible in the logic programming case, where the notions of "arguments" and "conflict-free sets of arguments" are made concrete.

where, the functions $find\_argument$ and $complete\_conflicting$ correspond to the predicates of $Proc$ in the following sense:

- If $find\_argument(\mathcal{D}, \alpha)$ returns Δ, then $demo(T \cup \Delta_0, \alpha) \wedge \Delta_0 \subseteq Ab \wedge adm^e(\emptyset, \mathcal{D} \cup \Delta_0, \Delta)$ follows from $Prog_{adm^e}$ and the specification of $defends^e$ and $demo$;

  If $demo(T \cup \Delta_0, \alpha) \wedge \Delta_0 \subseteq Ab \wedge adm^e(\emptyset, \mathcal{D} \cup \Delta_0, \Delta)$ follows from $Proc$, then $find\_argument(\mathcal{D}, \alpha)$ returns Δ.

The converse of the first implication does not hold because $find\_arguments$ only computes negative literals encountered in the resolution of α. The converse of the second implication does not hold because $find\_argument$ incorporates some loop checking, while the definition of *demo* in *proc* does not. Note that the definition of *demo* could be modified to incorporate loop checking.

- $complete\_conflicting(\mathcal{D}, \Delta)$ returns Ω iff $\Omega = \{A^* \subseteq Ab \,|\, attacks^*(A \cup \mathcal{D}, \Delta) \,\wedge\, adm^e(\emptyset, A \cup \mathcal{D}, A^*)\}$.

Our sceptical proof procedure in section 5 seems to show similar behaviours to the proof procedure defined in [16]. However, due to the lack of loop checking, $Proc$ might not terminate in some cases where [16] does.

7 Conclusions

We have defined a sound, credulous proof procedure for computing preferred extensions/partial stable models/ regular models in logic programming, by instantiating the abstract proof procedure to compute admissible sets of assumptions given in [3, 4].

We have defined an abstract proof procedure for computing the sceptical version of the same semantics, and employed the credulous proof procedure for logic programming to instantiate the abstract sceptical proof procedure in the logic programming case.

Both (credulous and sceptical) proof procedures are defined as logic programs, and are "synthesised" from formal specifications (see [4]). As a consequence, their proof of soundness is straightforward.

We have compared the proposed proof procedures with the ones proposed in [5, 2, 16].

Acknowledgements

The authors are grateful to Bob Kowalski for helpful suggestions and to Michael Thielscher for helpful discussions about his work.

References

[1] A. Bondarenko, P. M. Dung, R. A. Kowalski, F. Toni, An abstract, argumentation-theoretic framework for default reasoning. To appear in *Artificial Intelligence*, Elsevier.

[2] 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

[3] 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) `http://laotzu.doc.ic.ac.uk/UserPages/staff/ft/ft.html`

[4] P. M. Dung, R. A. Kowalski, F. Toni, Synthesis of proof procedures for default reasoning, *Proc. LOPSTR'96, International Workshop on Logic Program Synthesis and Transformation*, J. Gallagher ed., LNCS ?, Springer Verlag (1996) pages ?–?

[5] 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

[6] 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

[7] A. C. Kakas, P. Mancarella. Preferred extensions are partial stable models. *Journal of Logic Programming* 14(3,4) (1993), Elsevier, 341–348

[8] R.A. Kowalski. *Logic for problem solving.* Elsevier, New York (1979)

[9] R.A. Kowalski. Problems and promises of computational logic. In J.W. Lloyd, editor, *Proc. Symposium on Computational Logic.* Springer Verlag, 1990. Lecture Notes in Computer Science.

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

[11] D. McDermott, Nonmonotonic logic II: non-monotonic modal theories. *Journal of ACM* 29(1) (1982) 33–57

[12] R. Moore, Semantical considerations on non-monotonic logic. *Artificial Intelligence* 25 (1985), Elsevier, 75–94

[13] D. Poole, A logical framework for default reasoning. *Artificial Intelligence* 36 (1988), Elsevier, 27–47

[14] R. Reiter, A logic for default reasoning. *Artificial Intelligence* 13 (1980), Elsevier, 81–132

[15] 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

[16] M. Thielscher, A nonmonotonic disputation-based semantics and proof procedure for logic programs. *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, Bonn, Germany (1996), MIT Press (M. Maher ed.) 483–497

[17] F. Toni, *Abductive logic programming.* PhD Thesis, Imperial College, University of London (1995)

[18] J.-H. You, L.Y. Yuan, On the equivalence of semantics for normal logic programs. Journal of Logic Programming 22(3):211-221 (1995)

Appendix

Proof of theorem 5.1

The only if half is trivial. The if half is proved by contradiction. Suppose there exists a set of assumptions Δ^* such that Δ^* is preferred and $\alpha \notin Th(T \cup \Delta^*)$. Suppose Δ_0 is the set of assumptions provided in part 1. If $\Delta_0 = \emptyset$ then $\alpha \in Th(T)$ and therefore $\alpha \in Th(T \cup \Delta^*)$, thus contradicting the hypothesis. Therefore, $\Delta_0 \neq \emptyset$. Either (i) $\Delta^* \cup \Delta_0$ attacks itself or (ii) $\Delta^* \cup \Delta_0$ does not attack itself. Case (ii) implies that $\Delta^* \cup \Delta_0$ is admissible, thus contradicting the hypothesis that Δ^* is preferred (and therefore maximally admissible). Case (i) implies that either $\Delta^* \cup \Delta_0$ attacks Δ^* or $\Delta^* \cup \Delta_0$ attacks Δ_0. In the first subcase, Δ^* attacks $\Delta^* \cup \Delta_0$ (since Δ^* is admissible) and therefore (since Δ^* does not attack itself) Δ^* attacks Δ_0, and, by part 2, $\alpha \in Th(T \cup \Delta^*)$, thus contradicting the hypothesis. In the second subcase, Δ_0 attacks $\Delta^* \cup \Delta_0$ (since Δ_0 is admissible) and therefore (since Δ_0 does not attack itself) Δ_0 attacks Δ^* and therefore (since Δ^* is admissible) Δ^* attacks Δ_0 and, by part 2, $\alpha \in Th(T \cup \Delta^*)$, thus contradicting the hypothesis.

Proof of corollary 5.1

First note that the conditions of theorem 5.1 can be equivalently expressed as:

(1) $\alpha \in Th(T \cup \Delta_0) \ \wedge \ \Delta_0$ is admissible $\wedge$

(2) $\forall \Delta \subseteq Ab\,[\ \Delta$ is admissible $\wedge\ \Delta$ attacks $\Delta_0\ \rightarrow$
(3) $\quad \forall \Delta' \subseteq Ab\,[\Delta'$ is admissible $\wedge\ \Delta' \supseteq \Delta]\ \rightarrow$
(4) $\quad \alpha \in Th(T \cup \Delta')]]$.
By definition of *demo*, (4) is equivalent to $demo(T \cup \Delta', \alpha)$.

By soundness and completeness of $Prog_{adm^e}$ (see section 3), (3) is equivalent to
$(3')\ \forall \Delta' \subseteq Ab\,[adm^e(\emptyset, \Delta, \Delta') \wedge closed(\Delta)]$.

By monotonicity of Th, (1) is equivalent to
$\alpha \in Th(T \cup \Delta_0^*)\ \wedge\ \Delta_0^* \subseteq \Delta_0\ \wedge\ \Delta_0$ is admissible
which in turn, by definition of *demo* and by soundness and completeness of $Prog_{adm^e}$ (see section 3), is equivalent to
$demo(T \cup \Delta_0^*, \alpha)\ \wedge\ closed(\Delta_0^*)\ \wedge\ adm^e(\emptyset, \Delta_0^*, \Delta_0)$.

By monotonicity of Th, (2) is equivalent to
$\forall \Delta^* \subseteq Ab\,[\Delta^*$ attacks $\Delta_0\ \wedge\ \Delta^* \subseteq \Delta\ \wedge\ \Delta$ is admissible$]$
which in turn, by soundness and completeness of $Prog_{adm^e}$ (see section 3), is equivalent to
$\forall \Delta^* \subseteq Ab\,[\Delta^*$ attacks $\Delta_0\ \wedge\ closed(\Delta^*)\ \wedge\ adm^e(\emptyset, \Delta^*, \Delta)]$.
Since $\forall A, \Delta \subseteq Ab\ [attacks^*(A, \Delta) \rightarrow A\, attacks\ \Delta]$, the corollary is proved.

Relationship with [16]

1. (α, Δ) is an argument in [16] iff $\alpha \in Th(T \cup \Delta)$.
Proof: (α, Δ) is an argument
iff (by definition of logic programming argument)
either α is a (variable-free) negative literal and $\alpha \in \Delta$ or α is a (variable-free) atom and there exists a sequence of atoms $a_0, a_1, \ldots, a_n = \alpha$ such that for each $i = 0, \ldots, n$ there exists a clause $a_i \leftarrow p_1, \ldots, p_m, not\, p_{m+1}, \ldots, p_{m+k}$ in (the set of all variable-free instances of) T such that $p_1, \ldots, p_m \in \{a_0, a_1, \ldots, a_{i-1}\}$ and $not\, p_{m+1}, \ldots, p_{m+k} \in \Delta$
iff (by definition of Th for logic programming)
$\alpha \in Th(T \cup \Delta)$.

2. a set of arguments Arg is conflict-free [16] iff $\bigcup_{(\alpha,\Delta)\in Arg} \Delta$ is admissible.
Proof: Arg is conflict-free
iff (by definition of conflict-free set of argument)
$\bigcup_{(\alpha,\Delta)\in Arg} \Delta$ does not contradict itself, and for every set of negative literals A contradicting $\bigcup_{(\alpha,\Delta)\in Arg} \Delta$, $\bigcup_{(\alpha,\Delta)\in Arg} \Delta$ contradicts A
iff (by definition of "contradiction" and by our definition of attack)
$\bigcup_{(\alpha,\Delta)\in Arg} \Delta$ does not attack itself and for every set of negative literals A, if A attacks $\bigcup_{(\alpha,\Delta)\in Arg} \Delta$, then $\bigcup_{(\alpha,\Delta)\in Arg} \Delta$ attacks A
iff (by definition of admissible set of assumptions, and since every set of

assumptions is closed for logic programming)
$\bigcup_{(\alpha,\Delta)\in Arg} \Delta$ is admissible.