Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 170 (2006) 114–159
Artificial Intelligence
www.elsevier.com/locate/artint

Dialectic proof procedures for assumption-based, admissible argumentation

P.M. Dung [a], R.A. Kowalski [b], F. Toni [b,*]

[a] *Division of Computer Science, Asian Institute of Technology, PO Box 2754, Bangkok 10501, Thailand*
[b] *Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, UK*

Received 8 February 2005; received in revised form 15 July 2005; accepted 18 July 2005

Available online 21 October 2005

Abstract

We present a family of dialectic proof procedures for the admissibility semantics of assumption-based argumentation. These proof procedures are defined for any conventional logic formulated as a collection of inference rules and show how any such logic can be extended to a dialectic argumentation system.

The proof procedures find a set of assumptions, to defend a given belief, by starting from an initial set of assumptions that supports an argument for the belief and adding defending assumptions incrementally to counter-attack all attacks.

The proof procedures share the same notion of winning strategy for a dispute and differ only in the search strategy they use for finding it. The novelty of our approach lies mainly in its use of backward reasoning to construct arguments and potential arguments, and the fact that the proponent and opponent can attack one another before an argument is completed. The definition of winning strategy can be implemented directly as a non-deterministic program, whose search strategy implements the search for defences.

Keywords: Argumentation; Admissibility semantics; Dialectic; Proof procedure; Assumption-based reasoning; Dispute

* Corresponding author.
E-mail addresses: dung@cs.ait.ac.th (P.M. Dung), rak@doc.ic.ac.uk (R.A. Kowalski), ft@doc.ic.ac.uk (F. Toni).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.07.002

1. Introduction

In conventional logic, beliefs are derived from axioms, which are held to be beyond dispute. In everyday argumentation, however, beliefs are based on assumptions, which can be questioned and disputed.

Starting perhaps with Toulmin's landmark book, *The Uses of Argument* [37], this contrast between conventional logic and argumentation has led many researchers, including Perelman [30] and Walton [39], to regard ordinary, human argumentation as being beyond the reach of formal logic. However, in recent years a number of other researchers, including Pollock [32], Nute [29], Gordon [18], Loui [26] and Prakken and Sartor [34], have shown how argumentation can be understood in formal, logical terms. This work on logical models of argumentation is surveyed by Chesnevar, Maguitman and Loui [6].

In our logical model of argumentation, we use conventional logic to construct an argument, but we focus on the assumptions that support the argument. An opponent can dispute a proponent's argument by attacking one of the argument's supporting assumptions. The proponent can defend the argument by counter-attacking the opponent's attack with other arguments, possibly with the aid of other defending assumptions.

In [3] we introduced an assumption-based, argumentation-theoretic framework and showed how it can be used for the semantics of default reasoning. We showed that the *stable semantics* of many logics for default reasoning can be understood as sanctioning a belief if the belief is the conclusion of an argument whose set of supporting assumptions can be extended to a set of assumptions that both attacks every other assumption not in the set, and does not attack itself.

We also argued that the same logics can be given an alternative semantics that has a more natural argumentation-theoretic interpretation. This semantics, the *admissibility semantics*, sanctions a belief if it is the conclusion of an argument whose set of supporting assumptions can be extended to a set of defending assumptions, which both counter-attacks every attack, and does not attack itself. In [25] we explored the application of this semantics to argumentation in legal reasoning.

The admissibility semantics is a semantics in the sense that it non-constructively specifies when a belief is admissible. In [11] we showed how to derive from the specification the top-most level of constructive proof procedures, using logic program transformation techniques [31]. This top-most level generates defences incrementally, without showing how to construct them by means of a dialogue between a proponent and an opponent. In this paper we derive our (full) proof procedures in a more informal, intuitive fashion, which clarifies their dialectic character.

The basic proof procedure is illustrated by the following simplified example.

Example 1.1. Assume that two parties to a dispute agree on a common background set of beliefs[1]:

$$p \leftarrow q,\ r \tag{1}$$

[1] Here "$\leftarrow$" should be read as "if" and "," as "and". We will explain this notation in Section 3.

$$\neg q \leftarrow s,\ v \tag{2}$$

$$\neg q \leftarrow t,\ u \tag{3}$$

$$s \leftarrow t \tag{4}$$

$$r \tag{5}$$

$$v \tag{6}$$

$$u \leftarrow w \tag{7}$$

$$w. \tag{8}$$

Suppose that the proponent wants to defend the belief p. He/she can do so by putting forward the argument:

a_1: q (by assumption)
r (by 5)
p (by 1)

with conclusion p and assumption q. The opponent can attack the argument, by attacking the assumption q with the argument:

a_2: t (by assumption)
s (by 4)
v (by 6)
$\neg q$ (by 2)

with conclusion $\neg q$ and assumption t.

The proponent can counter-attack this attack with the trivial argument:

a_3: $\neg t$ (by assumption)

based on the assumption $\neg t$. If the opponent attempts to attack a_3 with the argument:

a_4: t (by assumption)

based on the assumption t, then the proponent simply repeats the counter-argument a_3. However, the opponent can also attack a_1 with the alternative argument:

a_2': t (by assumption)
s (by 4)
w (by 8)
u (by 7)
$\neg q$ (by 3)

also based on the assumption t. But then the proponent simply repeats the counter-argument a_3.

The proponent's belief p is admissible because it is the conclusion of an argument a_1 that is supported by a set of assumptions $\{q\}$ that can be extended to a defending set of assumptions $\{q, \neg t\}$ that can counter-attack every attack (and does not attack itself).

Notice that, by the same line of reasoning, the opponent can similarly defend a belief in the contrary conclusion $\neg p$, based on the defending set of assumptions $\{\neg p, t\}$. It is for this reason—because different agents can admissibly hold contrary beliefs—that the admissibility semantics is said to be credulous, rather than sceptical.

The assumption-based approach to argumentation of [3] builds upon Dung's [9,10], which showed that many logics for default reasoning can be viewed as instances of an abstract argumentation framework in which arguments and the attack relation between arguments are defined entirely abstractly, ignoring their internal structure. In [3], we showed that the attack relation between arguments in default reasoning depends only on the assumptions on which those arguments are based. Different logics for default reasoning differ mainly in their differing notions of assumption and of the contrary of an assumption.

Starting with Vreeswijk and Prakken [38], a number of authors [5,14,19] have developed dialectic proof procedures for the abstract version of Dung's admissibility semantics. Applied to the example above, these proof procedures demonstrate the admissibility of the proponent's belief in p in terms of the abstract arguments, a_1, a_2, a'_2, a_3 and a_4:

Proponent: a_1

Opponent: a_2 attacks a_1
Proponent: a_3 attacks a_2
Opponent: a_4 attacks a_3
Proponent: a_3 attacks a_4

Opponent: a'_2 attacks a_1
Proponent: a_3 attacks a'_2

This abstract view of argumentation simplifies the proof procedures, but does not show how to find arguments and how to exploit the fact that different arguments can share the same assumptions.

Our proof procedures generate and find arguments by reasoning backwards from conclusions to assumptions. They use backward reasoning both to find an initial argument for a given belief and to find attacking and defending arguments for the contrary of an assumption.

Each step in a backward argument can be viewed as a partially completed, potential argument. Any assumption in such a potential argument can be attacked (by finding an argument for its contrary) before the argument is completed. For example, our proof procedures can attack and defeat the assumption t in the opponent's argument a'_2 before it is completed, by reusing the argument a_3.

Our proof procedures are presented at three levels of abstraction. We derive one from the other by successive refinement, thereby simplifying the proofs and clarifying the relationship between seemingly different proof procedures. Our final refinement can be viewed as a generalisation of logic programming, which uses backward reasoning to generate proofs and uses a generalisation of negation as failure to show that an assumption is admissible (because its contrary can not be shown).

The remainder of this paper has the following structure: Section 2 describes the abstract framework and the admissibility semantics. Section 3 describes the simplified class of frameworks that we use for our examples. Section 4 shows how tight arguments, which are used in the remainder of the paper, can be generated by backward reasoning. Sections 5, 6 and 7 present the successive refinements of the proof procedures, in terms of abstract

dispute trees, concrete dispute trees and dispute derivations, respectively. Section 8 discusses algorithmic issues, including implementation and complexity. Section 9 discusses related work and is followed by the conclusions.

2. Admissibility for assumption-based argumentation frameworks

In this section we briefly review the notion of assumption-based framework [3,4,22,25] and show how it applies to argumentation.

Any logic, viewed as a deductive system, can be extended to an assumption-based argumentation framework.

Definition 2.1. 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 countable set of inference rules of the form

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

$\alpha \in \mathcal{L}$ is called the *conclusion* of the inference rule, $\alpha_1, \ldots, \alpha_n \in \mathcal{L}$ are called the *premises* of the inference rule and $n \geqslant 0$.

If $n = 0$, then the inference rule represents an axiom. We do not distinguish between domain-independent axioms, which belong to the specification of the logic, and domain-dependent axioms, which represent a background theory. Similarly, we allow both domain-independent and domain-specific inference rules. For notational convenience, we write α instead of $\frac{}{\alpha}$ throughout the paper.

Definition 2.2. A *deduction of a conclusion α based on a set of premises P* is a sequence $\beta_1, \ldots, \beta_m$ of sentences in $\mathcal{L}$, where $m > 0$ and $\alpha = \beta_m$, such that, for all $i = 1, \ldots, m$,

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

If there is a deduction of a conclusion α based on a set of premises P, we write $P \vdash \alpha$. We also say that the deduction is *supported by* or *based upon* P.

Notice that a deduction can contain applications of inference rules (called *inference steps*) that are not relevant to the derivation of the conclusion. Similarly, it may contain premises that are not relevant to the rest of the deduction. In particular, if $P \vdash \alpha$, then $P' \vdash \alpha$ for any $P \subseteq P'$. This property of deductive systems is called *monotonicity*.

Deductions are the basis for the construction of arguments, but to obtain an argument from a deduction we restrict the premises of the deduction to ones that are acceptable as *assumptions*. In this paper we restrict ourselves to *flat* frameworks [3], whose assumptions do not occur as conclusions of inference rules.

To specify when one argument attacks another, we need to determine when a sentence is the *contrary* of an assumption. The notion of contrary is in general non-symmetric (see [3]).

Given a deductive system $(\mathcal{L}, \mathcal{R})$, these two notions—the notion of assumption and the notion of the contrary of an assumption—determine the framework within which arguments and counter-arguments take place.

Definition 2.3. An *assumption-based framework* is a tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ where

- $(\mathcal{L}, \mathcal{R})$ is a deductive system.
- $\mathcal{A} \subseteq \mathcal{L}$, $\mathcal{A} \neq \{\}$. $\mathcal{A}$ is the set of candidate *assumptions*.
- If $\alpha \in \mathcal{A}$, then there is no inference rule of the form $\frac{\alpha_1, \ldots, \alpha_n}{\alpha} \in \mathcal{R}$.
- $\overline{\ }$ is a (total) mapping from $\mathcal{A}$ into $\mathcal{L}$. $\overline{\alpha}$ is the *contrary* of α.

In general, a deduction can fail to be an argument, because some of its premises may be conclusions of inference rules (the framework is not "flat") or because some of its premises may not be acceptable as assumptions (they do not belong to $\mathcal{A}$):

Definition 2.4. An *argument* is a deduction whose premises are all assumptions.

In our approach to argumentation, the only way to attack an argument is to attack one of its assumptions.

Definition 2.5.

- An *argument a attacks an argument b* if and only if a attacks an assumption in the set of assumptions on which b is based.
- An *argument a attacks an assumption* α if and only if the conclusion of a is the contrary $\overline{\alpha}$ of α.

The notation $A \vdash \alpha$ encapsulates the essence of an argument, by focusing attention on its set of assumptions A and its conclusion α. Not only does this notation ignore the internal structure of the argument, namely the inference rules used to generate it, but it ignores the fact that there can be several distinct arguments that give rise to the same $A \vdash \alpha$ relationship. Nonetheless, we will use the notation $A \vdash \alpha$ to stand for an argument, as an abuse of notation, when we want to draw attention to its assumptions and conclusion, and ignore its internal structure.

Our focus on the assumptions of arguments motivates the following definition:

Definition 2.6. A *set of assumptions A attacks a set of assumptions B* if and only if there exists an argument a based upon a set of assumptions $A' \subseteq A$ which attacks an assumption in B.

In our approach, the attack relationship between arguments depends solely on sets of assumptions. In some other approaches, however, such as that of Pollock [32] and Prakken and Sartor [34], an argument can attack another argument by contradicting its conclusion.

We reduce such "rebuttal" attacks to "undermining" attacks, as described in [25] and illustrated by the following example.

Example 2.1. Consider the inference rules

$$\frac{q}{p}, \quad q, \quad \neg p.$$

These can be used to justify both the argument q, p and the attacking argument $\neg p$ which rebuts the first argument by deriving a conclusion that contradicts the first argument's conclusion.

We obtain a similar effect by using instead the inference rules

$$\frac{q, \textit{assuming-}p}{p}, \quad q, \quad \neg p$$

where $\neg p$ is the contrary of the assumption *assuming-*p, and *assuming-*p is a new sentence in the language of the underlying deductive system.[2] Here the effect of the rebuttal is obtained by undermining the assumption of the argument

$$\textit{assuming-}p, \quad q, \quad p$$

by means of the counter-argument

$$\neg p.$$

The attack relationship is the basis of the *admissibility* semantics for argumentation. Informally speaking, a belief is admissible if it is the conclusion of an argument based upon a set of assumptions which can be defended against any attack. This set of assumptions, which we call the *defence set*, has two kinds of assumptions: those that are necessary to support an argument for the belief, and those that are necessary to counter-attack all attacks. In addition, the defence set must not attack itself.

Definition 2.7.

- A *set of assumptions* A is *admissible* if and only if
  1. A attacks every set of assumptions that attacks A, and
  2. A does not attack itself.
- A *belief* α is *admissible* if and only if there exists an argument for α based on a set of assumptions A_0, and A_0 is a subset of an admissible set A.

3. Simplified frameworks for assumption-based argumentation

To illustrate our approach, we use simplified assumption-based frameworks of the form $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ where:

[2] The assumption *assuming-*p is like Mp, read "consistent p", in default logic [35].

- All sentences in $\mathcal{L}$ are atoms $p, q, \ldots$ or negations of atoms $\neg p, \neg q, \ldots$ (i.e., $\mathcal{L}$ is a set of *literals*).
- The set of assumptions $\mathcal{A}$ is a subset of the set of all literals that do not occur as the conclusion of any inference rule in $\mathcal{R}$.
- The contrary of any assumption p is $\neg p$; the contrary of any assumption $\neg p$ is p.

Note that these simplified frameworks can be viewed as a generalisation of extended logic programs. As we will see later, when we discuss Example 6.1, these frameworks obtain the effect of negation as failure by means of assumptions whose contrary cannot be shown [3,15].

For notational convenience, we write inference rules

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

in the linear notation

$$\alpha \leftarrow \alpha_1, \ \ldots, \ \alpha_n.$$

We also use the same linear notation for inference rule schemata containing meta-variables that range over all elements of some domain. These schemata are a compact representation of the set of all inference rules obtained by instantiating the meta-variables. For example, $p(X) \leftarrow q(X)$, with the domain of natural numbers, stands for the infinite set of inference rules $p(0) \leftarrow q(0)$, $p(1) \leftarrow q(1)$, etc.

Example 3.1. Let $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ be the assumption-based framework:

- $\mathcal{L} = \{p, q, r, s, t, \neg p, \neg q, \neg r, \neg s, \neg t\}$
- $\mathcal{R}$ consists of

$$p \leftarrow q, \ r$$
$$r \leftarrow s$$
$$\neg q \leftarrow t.$$

- $\mathcal{A} = \{q, s, t\}$
- $\bar{q} = \neg q$, $\bar{s} = \neg s$, $\bar{t} = \neg t$.

One argument for the conclusion p is

$$s, \ r, \ q, \ p$$

based upon the set of assumptions $\{q, s\}$. Another is

$$t, \ \neg q, \ s, \ r, \ q, \ p$$

based upon the set of assumptions $\{q, s, t\}$. Both the assumption t and the application of the inference rule $\neg q \leftarrow t$ in the second argument are not relevant to the deduction of the conclusion of the argument.

Note that the same abstract representation $\{q, s\} \vdash p$ represents several different arguments, e.g. both s, r, q, p and q, s, r, p.

Also, note that the set of assumptions $\{q, s\}$, supporting the argument $\{q, s\} \vdash p$, is not admissible, since there is no counter-attack against the attack $\{t\} \vdash \neg q$. However, for the same reason, both the assumption t and therefore the conclusion $\neg q$ are admissible.

Every assumption-based framework of the simplified form introduced in this section is equivalent to a framework in which $\leftarrow$ is treated as object-level implication and the comma is treated as object-level conjunction $\wedge$. The equivalent framework has inference rule schemata for modus ponens and $\wedge$-introduction:

$$\frac{\beta \leftarrow \alpha, \alpha}{\beta}$$

$$\frac{\alpha, \beta}{\alpha \wedge \beta}.$$

In addition, wherever the simplified framework has an inference rule

$$\alpha \leftarrow \alpha_1, \ \ldots, \ \alpha_n$$

the equivalent framework has the inference rule

$$\overline{\alpha \leftarrow \alpha_1 \wedge \cdots \wedge \alpha_n}.$$

The two frameworks are equivalent in the case where assumptions and conclusions are restricted to literals, in the sense that they generate the same deductive relationship between assumptions and conclusions of arguments.

Example 3.2. Consider the framework in Example 3.1. This can be written as an equivalent framework with inference rules $\frac{}{p \leftarrow q \wedge r}$, $\frac{}{r \leftarrow s}$, $\frac{}{\neg q \leftarrow t}$, in addition to modus ponens and $\wedge$-introduction. The argument

$$q, \ s, \ r, \ p$$

in Example 3.1 is equivalent to the argument

$$q, \ s, \ r \leftarrow s, \ r, \ q \wedge r, \ p \leftarrow q \wedge r, \ p$$

in this framework. Both arguments have the same abstract form $\{q, s\} \vdash p$.

4. Tight arguments

The admissibility semantics is a semantics in the sense that it is a non-constructive specification of admissibility. A practical proof procedure, however, needs to be both constructive and efficient.

A major source of the non-constructivity and inefficiency of the specification is the monotonicity of deductive systems. Monotonicity has the consequence that for every superset A' of the set of assumptions A that supports an argument a attacking another argument b, there exists an argument a' supported by A' that also attacks b. Thus, in general, there can be infinitely many arguments against another argument b, differing only in the superset of assumptions A' on which those arguments are based. Moreover, for each

such attack, the proponent may need to search among infinitely many candidate counter-attacks to find one that is successful.

The problem stems from our definition of deduction, which suggests that arguments are constructed by reasoning forwards from assumptions to conclusions. Forward reasoning makes it hard to ensure that inference steps and assumptions that are used early in an argument will be relevant to the final conclusion of the argument. By comparison, backward reasoning from conclusions to assumptions automatically restricts the search for arguments to those whose individual inference steps are all relevant to the conclusion.

Perhaps the most natural way to represent the links between the assumptions and the conclusion of an argument is in the form of a proof tree: the root of the tree is labelled by the conclusion and the terminal nodes are labelled by the assumptions supporting the argument. For every non-terminal node in the tree, there is an inference rule whose conclusion matches the sentence labelling the node. The children of the node are labelled by the premises of the inference rule. Each sentence in such a proof tree is relevant to the argument, either because it is the conclusion itself, or because it is a premise of an inference rule whose conclusion is relevant.

Backward reasoning can be seen as generating such proof trees top-down, from the root to the terminal nodes. A *backward argument* is a sequence of frontiers $S_1, \ldots, S_m$ of the proof tree. The individual steps S_i in a backward argument can be represented by multi-sets, in which the same sentence can have several occurrences, if it is generated more than once as a premise of different inference steps.[3] We could replace multi-sets by sets, but this would implicitly force proof procedures always to check whether a newly introduced premise is already duplicated in S_i. This check can increase efficiency in cases where such duplications occur frequently, but it can decrease efficiency in cases where duplications occur only infrequently.

Backward arguments are a generalisation of SLD resolution, which is the basis of proof procedures for logic programming. As in SLD, if there is a backward argument using one strategy for selecting occurrences of sentences in S_i, then there is a backward argument using any other selection strategy. Thus, different selection strategies are simply different but equivalent ways of generating the same implicit proof tree. The selection strategy for backward arguments can be formalised by means of a *selection function*, as in the formalisation of SLD.[4]

Definition 4.1. Given a selection function, a *backward argument* of a conclusion α based on (or supported by) a set of assumptions A is a sequence of multi-sets $S_1, \ldots, S_m$, where $S_1 = \{\alpha\}$, $S_m = A$, and for every $1 \leqslant i < m$, where σ is the selected sentence occurrence in S_i:

[3] Multi-sets of sentences are equivalent to nodes labelled by sentences. The fact that a sentence can have several occurrences in a multi-set is equivalent to the fact that several nodes in a proof tree can be labelled by the same sentence.

[4] A selection function, in this context, takes as input a sequence of multi-sets S_i and returns as output a sentence occurrence in S_i.

1. If σ is a non-assumption sentence then $S_{i+1} = S_i - \{\sigma\} \cup S$ for some inference rule of the form $\frac{S}{\sigma} \in \mathcal{R}$.[5]
2. If σ is an assumption then $S_{i+1} = S_i$.[6]

The multi-sets S_i are called *steps* of the backward argument.

Terminology 4.1. To more clearly distinguish between ordinary arguments, as defined in Section 2, and backward arguments as defined here, we also call ordinary arguments *forward arguments*.

There exists a forward argument for a conclusion α supported by a set of assumptions A if and only if there exists a backward argument for α from a subset of A:

Theorem 4.1.

1. *For every backward argument of a conclusion α supported by a set of assumptions A there exists a forward argument of α supported by A.*
2. *For every forward argument for a conclusion α supported by a set of assumptions A and for every selection function, there exists a backward argument of α supported by some subset $A' \subseteq A$.*

The proof of this theorem and of all other results in the paper is given in Appendix A.

Example 4.1. Let $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ be the assumption-based framework where $\mathcal{R}$ consists of

$$u \leftarrow p,\ s$$
$$p \leftarrow q,\ r$$
$$r \leftarrow s$$
$$\neg q \leftarrow t$$

and $\mathcal{A} = \{q, s, t\}$. A backward argument for u supported by $\{q, s\}$ is (selected sentences are underlined):

$$\{\underline{u}\},\ \{\underline{p},\ s\},\ \{\underline{q},\ r,\ s\},\ \{q,\ \underline{r},\ s\},\ \{q,\ \underline{s},\ s\},\ \{q,\ s,\ \underline{s}\},\ \{q,\ s,\ s\}.$$

A corresponding forward argument, supported by the same set of assumptions, is

$$s,\ r,\ q,\ p,\ u.$$

Another forward argument for u, supported by the larger set of assumptions $\{q, s, t\}$, is

$$s,\ r,\ t,\ q,\ p,\ u.$$

There is no backward argument supported by $\{q, s, t\}$.

[5] We use the same symbols for multi-set membership, union, intersection and subtraction as we use for ordinary sets.

[6] Note that we need to restrict the selection function so that if σ is selected in S_i then it will not be selected again in any later S_j, $j > i$.

Note that every initial segment of a forward argument is an argument whose conclusion is the last sentence of the segment. By contrast, an initial segment of a backward argument is only a *potential argument*, in that the last step of the segment typically contains premises that are not assumptions. A potential argument might not lead to a full argument, if it contains premises that cannot be reduced to assumptions. For example, given the assumption-based framework with inference rules

$$p \leftarrow q,\ r$$

and assumptions $\{q\}$, then

$$\{p\},\ \{q, r\}$$

is a potential argument for p, supported by the set of premises $\{q, r\}$, which cannot be extended to a full argument.

Terminology 4.2. Because all steps in a backward argument are relevant to the conclusion by construction, we also call backward arguments *tight arguments*.

To show that a set of assumptions A is admissible, it suffices to consider only tight attacks against A and tight counter-attacks supported by assumptions in A:

Theorem 4.2. *A set of assumptions A is* admissible *if and only if*

1. *for every tight argument a that attacks A there exists a tight argument supported by $A' \subseteq A$ that counter-attacks a, and*
2. *no $A' \subseteq A$ supports a tight argument that attacks an assumption in A.*

This theorem is the basis of the abstract dialectic proof procedure in the next section. To exploit the theorem for this purpose, we need to show how admissible sets of assumptions can be generated incrementally, in defence of a given, desired conclusion. The proof procedure, which does this, can be seen as generating a winning strategy for a *proponent* to win a dispute against an ideal *opponent*. The proponent starts by putting forward an initial, tight argument for the desired conclusion, and then the proponent and the opponent alternate in attacking each other's previously presented arguments. The proponent wins if it has a counter-attack against every attacking argument by the opponent.

Terminology 4.3. In the remainder of the paper, we will limit our attention to tight arguments and refer to them simply as *arguments*, unless we need to emphasise their tightness.

5. Abstract dispute trees

In this section we represent the incremental construction of an admissible set of assumptions in defence of a given, desired conclusion by an abstract dispute tree, whose nodes are

labelled[7] by arguments. Every node is assigned the status of *proponent node* or *opponent node*, depending upon whether the argument at that node is put forward by the proponent or by the opponent.

Each branch of the tree represents a *winning dispute* for the proponent. The root of the tree, at which the proponent puts forward an initial argument, is the starting point of the dispute. (The argument at) every opponent node in the branch attacks (the argument at) the immediately preceding proponent node, and similarly every proponent node counter-attacks the immediately preceding opponent node.

A branch may be finite or infinite. A finite branch represents a winning dispute that ends with an argument by the proponent that the opponent is unable to attack. An infinite branch represents a winning dispute in which the proponent counter-attacks every attack of the opponent, ad infinitum.

A tree as a whole represents a *winning argumentation strategy* for the proponent. For every proponent node, there is a set (possibly empty) of children, which are opponent nodes labelled by all the attacks against the proponent node. For every such attacking, opponent node, there exists a single child, which is a proponent node, labelled by a single counter-attack against the opponent node.

Thus, an abstract dispute tree is an incremental construction, starting from an initial argument in favour of a given, desired conclusion, of a collection of proponent arguments that together counter-attack every possible attacking argument that might be put forward by the opponent.

An abstract dispute tree is an abstraction of a winning strategy for a dispute, because it does not show the construction of arguments and counter-arguments. We will show how this is done, by applying inference rules backwards, in the next section.

An abstract dispute tree is an abstraction of a winning strategy also because, although it contains all possible attacks by the opponent, it contains only one successful counter-attack by the proponent for each such attack. The tree does not show the search for counter-attacks.

An abstract dispute tree can be viewed as an and-tree ("and" because it includes *all* attacks by the opponent against *all* proponent arguments in the tree). The search space can be viewed as an and-or-tree ("or" because it includes *all* the alternative counter-attacks by the proponent against the opponent's attacks). Whereas the and-tree contains only one winning strategy for the proponent, the and-or tree contains *all* winning strategies, as well as *all* failed attempts by the proponent to counter-attack the opponent.

To obtain a proof procedure, we need to specify a strategy for searching the and-or tree to find an abstract dispute tree. Many such search strategies are possible, ranging from depth-first to breadth-first, including everything in between.

Definition 5.1. An *abstract dispute tree* for an initial argument a is a (possibly infinite) tree $\mathcal{T}$ such that

[7] We distinguish between nodes and their labels, because the same argument can label different nodes.

1. Every node of $\mathcal{T}$ is labelled by an argument and is assigned the status of *proponent* node or *opponent* node, but not both.
2. The root is a proponent node labelled by a.
3. For every proponent node N labelled by an argument b, and for every argument c that attacks b, there exists a child of N, which is an opponent node labelled by c.
4. For every opponent node N labelled by an argument b, there exists exactly one child of N which is a proponent node labelled by an argument which attacks some assumption α in the set supporting b. α is said to be the *culprit* in b.
5. There are no other nodes in $\mathcal{T}$ except those given by 1–4 above.

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

In the remainder of this section, we will refer to abstract dispute trees simply as "dispute trees". However, in the next section, where we introduce the notion of "concrete dispute tree", we will use the qualifications "abstract" and "concrete" when we need to distinguish between them.

Note that, in 3 above, for every proponent node N labelled by an argument b, if there are no attacks against b, then N is a terminal node. In particular, N is a terminal node if the set of assumptions supporting b is empty.

Example 5.1. Consider the assumption-based framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ with $\mathcal{R}$ consisting of:

$$\neg s \leftarrow q$$
$$\neg q \leftarrow r,\ s$$
$$\neg q \leftarrow u,\ v$$
$$\neg r$$
$$\neg u$$

where $\mathcal{A} = \{q, r, s, u\}$, $\mathcal{L} = \mathcal{A} \cup \{\neg s, \neg q, \neg r, \neg u\}$, $\overline{\alpha} = \neg\alpha$ for all $\alpha \in \mathcal{A}$.

Suppose the problem is to show that $\neg s$ is an admissible belief in the framework. To do so, we construct a dispute tree (given in Fig. 1) whose root is a proponent node labelled by the argument $\{q\} \vdash \neg s$. The root has a single child, which is an opponent node, labelled by the argument $\{r, s\} \vdash \neg q$. This opponent node also has a single child, which is a proponent node, labelled by the argument $\{\} \vdash \neg r$, which attacks the culprit r in $\{r, s\} \vdash \neg q$. This proponent node is also a terminal node, because there is no attack against an empty (support) set.

Note that the and-or tree search space for the problem contains, in addition to the finite branch of Fig. 1, an infinite branch representing an alternative winning strategy for the proponent. On this infinite branch, the proponent attacks the alternative culprit s in $\{r, s\}$ with the argument $\{q\} \vdash \neg s$. The infinite branch represents a winning strategy in which the proponent uses the same counter-attack $\{q\} \vdash \neg s$ against each of the opponent's repeated attacks $\{r, s\} \vdash \neg q$. In this example, the search space contains no failed attempts by the proponent to win the dispute.

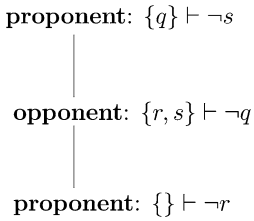

Fig. 1. Abstract dispute tree for $\{q\} \vdash \neg s$ in Example 5.1.

The definition of dispute tree incorporates the requirement that the proponent must counter-attack every attack, but it does not incorporate the further requirement that the proponent does not attack itself. This further requirement is incorporated in the definition of *admissible* dispute tree:

Definition 5.2. An abstract dispute tree $\mathcal{T}$ is *admissible* if and only if no culprit in the argument of an opponent node belongs to the defence set of $\mathcal{T}$.

Note that the tree in Fig. 1 is admissible. Note also that the definition of admissibility does not require that proponent nodes and opponent nodes have no assumptions in common. This is because the opponent can use the proponent's own assumptions against the proponent. The opponent can also use assumptions that are neutral, in the sense that they are neither in the defence set of the proponent nor in the culprit set of the opponent.

If the opponent can attack the proponent using only the proponent's own assumptions, then the proponent loses the dispute, because then the proponent attacks itself. However, to win the dispute, the proponent needs to identify and counter-attack in every attack of the opponent some culprit that does not belong to the proponent's own defence.

The admissibility requirement does not necessarily imply that a proof procedure that searches for admissible dispute trees needs to incorporate an explicit admissibility check. As we will see in Theorem 5.2, finite dispute trees are guaranteed to be admissible even without such a check.

The first part of the following theorem is a soundness result, which guarantees that the defence set of an admissible dispute tree is admissible. The second part is a strong form of completeness, which guarantees that, for any initial argument a whose support set is contained in an admissible set A of assumptions, there exists a dispute tree for a whose defence set A' is contained in A. Whereas the admissible set A may contain assumptions that are irrelevant to the defence of a, A' contains only relevant assumptions, associated with tight arguments.

Theorem 5.1.

(i) *If $\mathcal{T}$ is an admissible abstract dispute tree for an argument a and if A is the defence set of $\mathcal{T}$, then A is an admissible set of assumptions.*
(ii) *If a is an argument supported by a set of assumptions A_0 and A is an admissible set of assumptions such that $A_0 \subseteq A$, then there exists an admissible dispute tree for a with defence set A' and $A_0 \subseteq A' \subseteq A$ and A' is admissible.*

Admissible dispute trees bring us closer to a proof procedure, because they show how to extend an initial set of assumptions incrementally to an admissible set of assumptions. However, they are still non-constructive, because they can be both infinite in breadth and infinite in depth: They can be infinite in breadth, because there can be infinitely many tight attacks against a single assumption. They can be infinite in depth, because there can be infinitely long branches of alternating attack and counter-attacks. The following examples illustrate these two possibilities.

Example 5.2. Consider the assumption-based framework with inference rules consisting of all instances over the infinite domain

$$Dom = \{succ^i(0) \mid i \geqslant 0\} = \{0, succ(0), succ(succ(0)) \ldots\}$$

of the inference rule schemata

$$p \leftarrow q$$
$$\neg q \leftarrow r(X),\ s(X)$$
$$r(succ(X)) \leftarrow r(X)$$
$$r(0)$$
$$\neg s(X)$$

with set of assumptions $\{q\} \cup \{s(X) \mid X \in Dom\}$. There are infinitely many (finite) attacks against the argument $\{q\} \vdash p$, i.e.

$$\{s(0)\} \vdash \neg q$$
$$\{s(succ(0))\} \vdash \neg q$$
$$\{s(succ(succ(0)))\} \vdash \neg q$$
etc.

Therefore, every dispute tree for $\{q\} \vdash p$ is infinite in width. Note that every argument of the form $\{s(succ^i(0))\} \vdash \neg q$ is counter-attacked by an argument of the form $\{\} \vdash \neg s(succ^i(0))$ and thus there exists an admissible dispute tree for $\{q\} \vdash p$ with defence set $\{q\}$. Therefore, by Theorem 5.1, part (i), $\{q\}$ is admissible, and as a consequence the belief p is admissible.

Example 5.3. Consider the assumption-based framework with inference rules consisting of all instances over the infinite domain *Dom* in Example 5.2 of the inference rule schemata

$$\neg p(X) \leftarrow q(succ(X))$$
$$\neg q(X) \leftarrow p(succ(X))$$

and set of assumptions $\{q(X) \mid X \in Dom\} \cup \{p(X) \mid X \in Dom\}$. Consider the argument $\{q(succ(0))\} \vdash \neg p(0)$. There exists an admissible dispute tree for this argument, which consists of a single infinite branch of alternating proponent and opponent nodes. For every proponent node labelled by an argument of the form $\{q(succ^{2i+1}(0))\} \vdash \neg p(succ^{2i}(0))$, there is a single child node, which is an opponent node labelled by $\{p(succ^{2i+2}(0))\} \vdash$

$\neg q(succ^{2i+1}(0))$, Similarly, for every opponent node labelled by $\{p(succ^{2i+2}(0))\} \vdash \neg q(succ^{2i+1}(0))$, there is a single child node, which is a proponent node labelled by $\{q(succ^{2i+3}(0))\} \vdash \neg p(succ^{2i+2}(0))$. The defence set

$$\left\{q\left(succ^{2i+1}(0)\right) \mid i \geqslant 0\right\}$$

is admissible.

In the special case of dispute trees that are finite in depth, the admissibility check of Definition 5.2 is unnecessary[8]:

Theorem 5.2. *Any dispute tree that has no infinitely long branches is an admissible dispute tree.*

For a large class of frameworks, generalising the class of stratified logic programs [1], all branches of all abstract dispute trees are finite in depth.

Definition 5.3. A framework is *stratified* if and only if there exists no infinite sequence of arguments $a_1, \ldots, a_n, \ldots$, where for every $n \geqslant 1$, a_{n+1} attacks a_n.

The assumption-based framework of Example 5.1 is stratified (and the dispute tree in Fig. 1 is indeed finite). However, the framework of Example 5.3 is not stratified. An even simpler example of a non-stratified assumption-based framework is:

Example 5.4. The assumption-based framework with inference rules

$$\neg p \leftarrow q$$
$$\neg q \leftarrow p$$

and set of assumptions $\{p, q\}$ is not stratified, because there exists an infinite sequence of arguments of the form:

$$\{q\} \vdash \neg p,\ \{p\} \vdash \neg q,\ \{q\} \vdash \neg p,\ \ldots.$$

There exists an admissible dispute tree for $\neg p$, but this is infinite in depth.

6. Concrete dispute trees

In this section, we expand abstract dispute trees, to incorporate the incremental construction of (tight) arguments. We call such expanded dispute trees *concrete dispute trees*. To minimise the use of new terminology, when there is no possibility of confusion, we use the same terminology, *dispute tree*, for both abstract and concrete dispute trees.

[8] However, as we will see later in Section 7, although an explicit admissibility check is unnecessary for finite trees, it can nevertheless decrease the size of dispute trees and therefore increase the efficiency of proof procedures that search for them.

Whereas in abstract dispute trees individual nodes are labelled by complete arguments, in concrete dispute trees they are labelled by steps of *potential arguments*. However, as in abstract dispute trees, every branch of a concrete dispute tree represents a sequence of alternating attacks by the opponent and counter-attacks by the proponent. However, these attacks and counter-attacks are against assumptions in *potential*, rather than full, arguments. A concrete dispute tree represents a winning strategy, in which the proponent counter-attacks with a *full* attack every *potential* attack by the opponent.

In concrete dispute trees, there is an important difference between the potential arguments of the proponent and of the opponent. For the proponent, all potential arguments must eventually be completed. However, for the opponent, potential arguments can be presented only partially, up to the point where they fail. As we will see later in this section, there are two ways such a potential argument can fail—either because some non-assumption sentence is selected and can not be proved (namely there is no complete argument extending the potential argument); or because some assumption is selected as culprit of the incomplete, potential argument and is defeated by a counter-argument of the proponent (in which case it doesn't matter whether there is a complete argument extending the potential argument).

The partial presentation and counter-attacking of the opponent's potential arguments has both a good point and a bad point. The good point, already illustrated in Section 1, is that, when a potential argument is successfully counter-attacked, all ways of completing the potential argument are also counter-attacked. So one counter-attack by the proponent can successfully defeat many complete attacks by the opponent, without the need to generate the complete attacks in full. The bad point, illustrated in Example 6.1 below, is that some potential arguments cannot be extended into complete arguments. Counter-attacking such unextendable potential arguments is a waste of time.

There is also an important difference in the role of the selection function for proponent and opponent. For both of them, when the selection function chooses a non-assumption sentence, it does so to expand the potential argument constructed so far into a more complete argument. The order in which such sentences are selected does not matter.

Similarly, when the selection function selects a proponent's assumption, it does so to determine an order in which attacks against the proponent's argument are considered. Since all such attacks must be counter-attacked, it does not matter in which order they are selected.

However, when the selection function selects an opponent's assumption, the assumption becomes a potential culprit for counter-attack by the proponent. Since the dispute tree represents a single winning strategy, which contains only one counter-attack against every attack, either the selected assumption is ignored (if it is not the actual culprit in the attack) or else it is counter-attacked. In the definition of dispute tree, these two possibilities are represented by a non-deterministic choice.

Thus the selection of the opponent's assumptions is linked to the search strategy for finding counter-attacks. The dispute tree itself displays only one successful counter-attack against every attack, and "ignores" all of the other winning or failing alternatives, which are present in the search space.

Definition 6.1. Given a selection function, a *concrete dispute tree* for a sentence α is a (possibly infinite) tree $\mathcal{T}$ such that

1. Every node of $\mathcal{T}$ is labelled by a multi-set of sentences (representing a potential argument) and is assigned the status of *proponent* node or *opponent* node, but not both.
2. The root of $\mathcal{T}$ is a proponent node labelled by $\{\alpha\}$.
3. Let N be a proponent node labelled by $\mathcal{P}$. If $\mathcal{P}$ is empty, then N is a terminal node. Otherwise, $\mathcal{P}$ is not empty, and there exists some selected occurrence of a sentence δ in $\mathcal{P}$.
   (i) If δ is an assumption, then there exists one child of N, which is an opponent node labelled by $\{\overline{\delta}\}$ and a second child of N that is a proponent node labelled by $\mathcal{P} - \{\delta\}$ (*to consider all attacks against* $\mathcal{P}$).
   (ii) If δ 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 proponent node labelled by $\mathcal{P} - \{\delta\} \cup S$.
4. Let N be an opponent node labelled by $\mathcal{O}$. Then $\mathcal{O}$ is not empty, and there exists some selected occurrence of a sentence σ in $\mathcal{O}$.
   (i) If σ is an assumption, then
      (a) either σ is *ignored* and there exists exactly one child of N, which is an opponent node labelled by $\mathcal{O} - \{\sigma\}$,
      (b) or σ is a *culprit*, and there exists exactly one child of N, which is a proponent node labelled by $\{\overline{\sigma}\}$.
   (ii) If σ is not an assumption and there exists no inference rule $\frac{S}{\sigma} \in \mathcal{R}$, then N is a terminal node (*and the potential attack* $\mathcal{O}$ *fails of its own accord*). Otherwise, for every $\frac{S}{\sigma} \in \mathcal{R}$, there exists a child of N, which is an opponent node labelled by the multi-set of sentences $\mathcal{O} - \{\sigma\} \cup S$.
5. There is no infinite sequence of consecutive nodes all of which are proponent nodes.
6. There are no other nodes in $\mathcal{T}$ except those given by 1–4 above.

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

Notice that the label $\mathcal{O}$ of an opponent node is never empty. Empty multi-sets $\mathcal{O}$, however, can occur in failed parts of the search space, either because in step 4(i) all assumptions in $\mathcal{O}$ are ignored, or because in step 4(ii) $\mathcal{O} = \{\sigma\}$ and there exists an inference rule $\frac{S}{\sigma} \in \mathcal{R}$ with $S = \{\}$, giving an attack for which no counter-attack is possible.

Definition 6.2. A concrete dispute tree $\mathcal{T}$ for a sentence α is *admissible* if and only if no culprit at an opponent node belongs to the defence set of $\mathcal{T}$.

Example 6.1. Consider the assumption-based framework of Example 5.1, with inference rules:

$$\neg s \leftarrow q$$
$$\neg q \leftarrow r,\ s$$
$$\neg q \leftarrow u,\ v$$

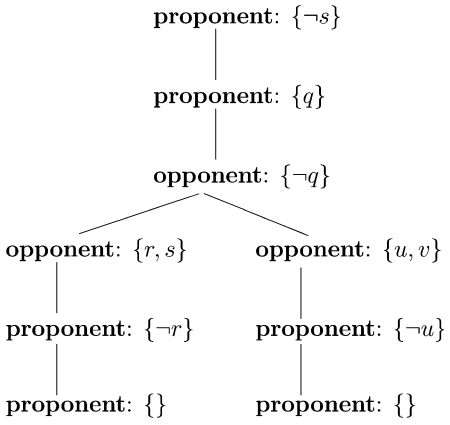

Fig. 2. Concrete dispute tree for $\neg s$ in Example 6.1.

$\neg r$

$\neg u$

and set of assumptions $\{q, r, s, u\}$. Again, the problem is to show that $\neg s$ is admissible. A concrete dispute tree for $\neg s$ is given in Fig. 2. Notice that the concrete dispute tree contains a branch that does not correspond to any branch in the abstract dispute tree of Fig. 1. This is because the branch unnecessarily contains a defence against the potential attack $\{u, v\}$, which cannot be developed into a complete attack.

The example can be used to illustrate how our proof procedures generalise negation as failure in logic programming: Rename the literals $\neg s$, $\neg q$, $\neg r$, $\neg u$ by atoms s', q', r', u' and the assumptions s, q, r, u by the negation as failure literals $not\, s', not\, q', not\, r', not\, u'$, respectively. Then showing that all attacks against the assumption q can successfully be counter-attacked is similar to showing that $not\, q'$ holds because q' fails to hold. Our proof procedures are more general than negation as failure, not only because literals can be positive and negative as in extended logic programming, but more importantly because negation as failure completely explores the proof of a negation as failure literal in isolation of other sentences in an argument. Our argumentation proof procedures, on the other hand, allow the counterattacking of all attacks against an assumption to be interleaved with the other parts of the search for a dispute tree.

The correspondence between abstract and concrete dispute trees is most obvious in the special case of selection functions that always choose non-assumption sentences in preference to assumptions. We call such selection functions *patient*, because they wait until a complete argument has been constructed before beginning to attack it.

For every admissible concrete dispute tree constructed by means of a patient selection function, there exists a corresponding admissible abstract dispute tree with the same defence set. Conversely, for every admissible abstract dispute tree and for every patient selection function, there exists a corresponding admissible concrete dispute tree with the same defence set. However, in addition to nodes that simply add concrete inference steps,

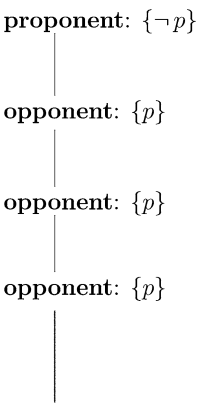

Fig. 3. Concrete dispute tree for $\neg p$ for the assumption-based framework with $p \leftarrow p$.

a concrete dispute tree may contain extra branches that represent potential arguments by the opponent that fail, not because they are counter-attacked by the proponent, but because they contain some non-assumption sentence that cannot be proved (as we have seen in Example 6.1).

For example, given the assumption-based framework with the single inference rule $p \leftarrow p$ and set of assumptions $\{\neg p\}$, there exists an admissible infinite concrete dispute tree for the sentence $\neg p$, as shown in Fig. 3. The corresponding abstract dispute tree contains only the root node $\{\} \vdash \neg p$.

However, whether the selection function is patient or not, a concrete dispute tree must expand all of the proponent's potential arguments into complete arguments. It is for this reason that a concrete dispute tree may not contain any infinite sequences of consecutive inference steps by the proponent (case 5).

For example, given the assumption-based framework with the single inference rule $p \leftarrow p$ and set of assumptions $\{\neg p\}$, there exists no dispute tree for the sentence p.

As in the case of abstract dispute trees, to obtain a proof procedure, we need to specify a strategy for searching the implicit and-or tree associated with the definition of concrete dispute tree. One part of this strategy is to find a culprit in every attack. The other part is to find a way of counter-attacking the culprit.

Finding a culprit is dealt with in case 4(i) of Definition 6.1, by choosing whether to ignore or to counter-attack the selected assumption in an opponent node. Finding a way of attacking the culprit is dealt with by the search strategy.

Our definition leaves the search strategy undetermined. As a consequence, the definition can be implemented, directly as it stands, in a non-deterministic programming language, such as Prolog. 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 implementation of the definition.

The following theorem expresses the relationship between abstract and concrete dispute trees.

Theorem 6.1.

(i) *For every admissible abstract dispute tree $\mathcal{T}$ for a conclusion α, and for every selection function, there exists an admissible concrete dispute tree $\mathcal{T}'$ for α such that the defence set of $\mathcal{T}'$ is a subset of the defence set of $\mathcal{T}$.*
(ii) *For every admissible concrete dispute tree $\mathcal{T}'$ for a sentence α, there exists an admissible abstract dispute tree $\mathcal{T}$ for a tight argument a of the conclusion α such that the defence set of $\mathcal{T}$ is a subset of the defence set of $\mathcal{T}'$.*

The following corollary is a direct consequence of this theorem and Theorem 5.1 for abstract dispute trees:

Corollary 6.1.

(i) *If $\mathcal{T}$ is an admissible concrete dispute tree and A is the defence set of $\mathcal{T}$, then A is an admissible set of assumptions.*
(ii) *If there is an argument for a conclusion α supported by a set of assumptions A_0 and A is an admissible set of assumptions such that $A_0 \subseteq A$, then for every selection function there exists an admissible concrete dispute tree for α with defence set A' and $A_0 \subseteq A' \subseteq A$ and A' is admissible.*

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

Theorem 6.2. *Any concrete dispute tree that has no infinitely long branches is an admissible concrete dispute tree.*

For a large class of frameworks, generalising the class of acyclic logic programs [2], all branches of all concrete dispute trees are finite.

Definition 6.3. 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 conclusion of the inference rule.

The assumption-based framework of Example 5.1 is acyclic (and the tree in Fig. 1 is indeed finite). On the other hand, any assumption-based framework with the inference rule $p \leftarrow p$ is an example of a non-acyclic framework.

7. Dispute derivations

Dispute derivations are to dispute trees what backward arguments are to proof trees. In both cases, the top-down generation of a tree is represented by a sequence of frontiers of

the tree. Dispute derivations generalise the abductive derivations for logic programming of [15].

The frontier of a dispute tree is a set of proponent and opponent nodes labelled by multi-sets of sentences, representing steps of potential arguments. A *dispute derivation* represents the current state of this frontier, together with the set of defence assumptions A_i and culprits C_i generated so far, as a quadruple: $\langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i, \rangle$. The sets A_i and C_i are used to filter arguments, as we will explain later.

$\mathcal{O}_i$ is a multi-set corresponding directly to the set of opponent nodes in the frontier—i.e., its members are multi-sets of sentences representing the state of all of the opponent's potential arguments against the proponent.

The multi-set $\mathcal{P}_i$ is a flattened version of the proponent's potential arguments—i.e., its members are occurrences of sentences belonging to any of the proponent's potential arguments.

The first step of a dispute derivation represents the root of the dispute tree. Each transition in the dispute derivation represents the selection of a node in the frontier of the dispute tree and its replacement by its children. Any node in the frontier 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 dispute tree.

The selection of nodes in the dispute tree is different from the search for counter-attacks against attacks. The search for counter-attacks requires a search strategy, which turns the definition of derivation into a proof procedure for finding derivations.[9]

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, because of filtering by defences, the corresponding dispute trees can sometimes be infinite.

The set of defence assumptions A_i is used both to filter proponent assumptions in $\mathcal{P}_i$, so they are not considered redundantly, more than once, and to filter potential culprit assumptions in $\mathcal{O}_i$, so that the final defence set A constructed by the derivation does not attack itself. The set of culprits C_i is similarly used both to filter potential culprit assumptions in $\mathcal{O}_i$, so they are not counter-attacked redundantly more than once, and to filter proponent assumptions in $\mathcal{P}_i$, so that A does not attack itself.

Definition 7.1. Given a selection function, a *dispute derivation of a defence set A* for a sentence α is a finite sequence of quadruples

$$\langle \mathcal{P}_0, \mathcal{O}_0, A_0, C_0 \rangle, \ \ldots, \ \langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i \rangle, \ \ldots, \ \langle \mathcal{P}_n, \mathcal{O}_n, A_n, C_n \rangle$$

where

$$\mathcal{P}_0 = \{\alpha\},$$
$$A_0 = \mathcal{A} \cap \{\alpha\},$$
$$\mathcal{O}_0 = C_0 = \{\},$$

[9] Here the selection function is an extension of the selection function for concrete dispute trees. In addition to selecting sentence occurrences in potential arguments, here the selection function also decides whether to focus attention on proponent assumptions in $\mathcal{P}_i$ or on potential opponent arguments in $\mathcal{O}_i$.

$\mathcal{P}_n = \mathcal{O}_n = \{\}$,

$A = A_n$,

and for every $0 \leqslant i < n$, only one σ in $\mathcal{P}_i$ or one S in $\mathcal{O}_i$ is selected, and:

1. If $\sigma \in \mathcal{P}_i$ is selected then
   (i) if σ is an assumption, then

$$\mathcal{P}_{i+1} = \mathcal{P}_i - \{\sigma\},$$
$$A_{i+1} = A_i,$$
$$C_{i+1} = C_i,$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i \cup \big\{\{\overline{\sigma}\}\big\};$$

   (ii) if σ is not an assumption, then there exists some inference rule $\frac{R}{\sigma} \in \mathcal{R}$ and

$$C_i \cap R = \{\} \quad \text{(filtering of defence assumptions by culprits)},$$
$$\mathcal{P}_{i+1} = \mathcal{P}_i - \{\sigma\} \cup (R - A_i) \quad \text{(filtering of defence assumptions by defences)},$$
$$A_{i+1} = A_i \cup (\mathcal{A} \cap R),$$
$$C_{i+1} = C_i,$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i.$$

2. If S is selected in $\mathcal{O}_i$ and σ is selected in S then
   (i) if σ is an assumption, then
      (a) either σ is ignored, i.e.

$$\mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} \cup \big\{S - \{\sigma\}\big\},$$
$$\mathcal{P}_{i+1} = \mathcal{P}_i,$$
$$A_{i+1} = A_i,$$
$$C_{i+1} = C_i;$$

      (b) or $\sigma \notin A_i$ (filtering of culprits by defences) and

$$\mathcal{O}_{i+1} = \mathcal{O}_i - \{S\},$$
$$\mathcal{P}_{i+1} = \mathcal{P}_i \cup \{\overline{\sigma}\},$$
$$A_{i+1} = A_i,$$
$$C_{i+1} = C_i \cup \{\sigma\};$$

   (ii) if σ is not an assumption, then

$$\mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} \cup \left\{S - \{\sigma\} \cup R \;\middle|\; \frac{R}{\sigma} \in \mathcal{R}, \text{ and } R \cap C_i = \{\}\right\}$$

   (filtering of culprits by culprits),

$$\mathcal{P}_{i+1} = \mathcal{P}_i,$$

$$A_{i+1} = A_i,$$
$$C_{i+1} = C_i.$$

Note that, in case 2(ii), if there is no inference rule whose conclusion matches the selected sentence σ then the potential attack S cannot be extended to a complete attack. In such a case, the set $\{S - \{\sigma\} \cup R \mid \frac{R}{\sigma} \in \mathcal{R}, \text{ and } R \cap C_i = \{\}\}$ is empty and therefore $\mathcal{O}_{i+1} = \mathcal{O}_i - \{S\}$. In other words, the potential attack fails of its own accord and is removed from the set of potential attacks that need to be counter-attacked.

Example 7.1. Consider the assumption-based framework of Examples 5.1 and 6.1, with inference rules:

$$\neg s \leftarrow q$$
$$\neg q \leftarrow r,\ s$$
$$\neg q \leftarrow u,\ v$$
$$\neg r$$
$$\neg u$$

and set of assumptions $\{q, r, s, u\}$. There exists a derivation for the sentence $\neg s$ of the defence set $\{q\}$, as follows (with the output of the selection function underlined):

$$\langle \{\underline{\neg s}\}, \{\}, \{\}, \{\} \rangle,$$
$$\langle \{\underline{q}\}, \{\}, \{\}, \{\} \rangle,$$
$$\langle \{\}, \{\{\underline{\neg q}\}\}, \{q\}, \{\} \rangle,$$
$$\langle \{\}, \{\{\underline{r}, s\}, \{u, v\}\}, \{q\}, \{\} \rangle,$$
$$\langle \{\underline{\neg r}\}, \{\{u, v\}\}, \{q\}, \{r\} \rangle,$$
$$\langle \{\}, \{\{\underline{u}, v\}\}, \{q\}, \{r\} \rangle,$$
$$\langle \{\underline{\neg u}\}, \{\}, \{q\}, \{r, u\} \rangle,$$
$$\langle \{\}, \{\}, \{q\}, \{r, u\} \rangle.$$

The following example illustrates the need for ignoring assumptions selected in case 2(i)(a) of the definition of dispute derivation.

Example 7.2. Consider the assumption-based framework with inference rules:

$$\neg s \leftarrow q$$
$$\neg q \leftarrow r,\ t$$
$$\neg t$$

and set of assumptions $\{q, r, t\}$. There exists a derivation for the sentence $\neg s$ of defence set $\{q\}$ in which one of the outputs of the selection function is ignored:

$\langle\{\underline{\neg s}\}, \{\}, \{\}, \{\}\rangle$,
$\langle\{\underline{q}\}, \{\}, \{\}, \{\}\rangle$,
$\langle\{\}, \{\{\underline{\neg q}\}\}, \{q\}, \{\}\rangle$,
$\langle\{\}, \{\{\underline{r}, t\}\}, \{q\}, \{\}\rangle$,
$\langle\{\}, \{\{\underline{t}\}\}, \{q\}, \{\}\rangle$ (r is ignored),
$\langle\{\underline{\neg t}\}, \{\}, \{q\}, \{t\}\rangle$,
$\langle\{\}, \{\}, \{q\}, \{t\}\rangle$.

The filtering of defence assumptions by defences can turn an infinite dispute tree into a finite derivation, as illustrated by the following example.

Example 7.3. Consider the assumption-based framework of Example 5.4 with inference rules

$$\neg p \leftarrow q$$
$$\neg q \leftarrow p$$

and set of assumptions $\{p, q\}$. Consider the sentence p.

There exists a five-step derivation for p of the defence set $\{p\}$:

$\langle\{\underline{p}\}, \{\}, \{p\}, \{\}\rangle$,
$\langle\{\}, \{\{\underline{\neg p}\}\}, \{p\}, \{\}\rangle$,
$\langle\{\}, \{\{\underline{q}\}\}, \{p\}, \{\}\rangle$,
$\langle\{\underline{\neg q}\}, \{\}, \{p\}, \{q\}\rangle$,
$\langle\{\}, \{\}, \{p\}, \{q\}\rangle$ (filtering of defence assumptions by defence assumption p).

Without such filtering of and by defence assumptions, it would be necessary to generate an infinite sequence of alternating proponent arguments $\{p\} \vdash \neg q$ and opponent arguments $\{q\} \vdash \neg p$, and the derivation would never terminate. Thus, filtering defence assumptions allows finite derivations to be constructed in some cases where the corresponding dispute tree is infinite. Example 5.3, however, is a case of an infinite dispute tree that does not have a corresponding finite dispute derivation.

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

Example 7.4. Consider the assumption-based framework with inference rules

$$r \leftarrow p,\ q$$
$$\neg p \leftarrow q$$
$$\neg q \leftarrow p$$

and set of assumptions $\{p, q\}$. Consider now the sentence r.

With filtering of defence assumptions, but without filtering of culprits by defences, there would exist a derivation:

$\langle \{\underline{r}\}, \{\}, \{\}, \{\} \rangle$,

$\langle \{\underline{p}, q\}, \{\}, \{p, q\}, \{\} \rangle$,

$\langle \{q\}, \{\{\underline{\neg p}\}\}, \{p, q\}, \{\} \rangle$,

$\langle \{q\}, \{\{\underline{q}\}\}, \{p, q\}, \{\} \rangle$,

$\langle \{q, \underline{\neg q}\}, \{\}, \{p, q\}, \{q\} \rangle$ (no filtering of culprit q),

$\langle \{\underline{q}\}, \{\}, \{p, q\}, \{q\} \rangle$ (filtering of defence assumption p),

$\langle \{\}, \{\{\underline{\neg q}\}\}, \{p, q\}, \{q\} \rangle$,

$\langle \{\}, \{\{\underline{p}\}\}, \{p, q\}, \{q\} \rangle$ (no filtering of culprit p),

$\langle \{\underline{\neg p}\}, \{\}, \{p, q\}, \{q, p\} \rangle$,

$\langle \{\}, \{\}, \{p, q\}, \{q, p\} \rangle$ (filtering of defence assumption q).

This derivation corresponds to an infinite dispute tree that is not admissible.

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

Example 7.5. Consider the assumption-based framework with only one inference rule $p \leftarrow \neg p$ and only one assumption $\neg p$. Consider the sentence $\neg p$.

The only partial derivation that can be generated by any proof procedure terminates in failure after only two steps:

$\langle \{\neg p\}, \{\}, \{\neg p\}, \{\} \rangle$,

$\langle \{\}, \{\{p\}\}, \{\neg p\}, \{\} \rangle$,

$\langle \{\}, \{\{\neg p\}\}, \{\neg p\}, \{\} \rangle$ (filtering of potential culprit $\neg p$ by defence assumption $\neg p$).

Without filtering of culprits by defences, a proof procedure would attempt to generate an infinite sequence of alternating defending arguments $\{\neg p\} \vdash p$ and attacking arguments $\{\neg p\} \vdash p$ and would never terminate.

The defence set derived by a dispute derivation is admissible, because there is a corresponding admissible dispute tree with the same defence set:

Theorem 7.1. *For every dispute derivation of a defence set A for a sentence α, there exists a* (*possibly infinite*) *admissible dispute tree for α with defence set A.*

It follows immediately that:

Corollary 7.1. *For every dispute derivation of a defence set A for a sentence α, the defence set A is admissible, and there exists some $A' \subseteq A$ that supports an argument for α.*

Completeness holds for finite derivations, in the sense that for every finite dispute tree there exists a corresponding finite dispute derivation. However, because of filtering, a dispute derivation can contain only one way of defending a defence assumption and only one way of counter-attacking a culprit assumption. But a dispute tree, without filtering, can have different defences for different occurrences of a defence assumption and different counter-attacks against different occurrences of the same culprit. It is for this reason that the defence set of a dispute derivation can be strictly contained in the defence set of the corresponding dispute tree:

Theorem 7.2. *For every finite dispute tree for a sentence α with defence set A, there exists a dispute derivation for α of a defence set $A' \subseteq A$.*

8. Algorithmic issues

The definition of dispute derivation specifies the structure of a winning argumentation strategy. However, it does not determine the search strategy for finding winning strategies. In this respect, it resembles a non-deterministic, rather than a deterministic program. In fact, the definition can be rewritten directly as a non-deterministic program.

The non-deterministic program below is expressed in procedural form, but could also be expressed declaratively as a Prolog program, in which case it would be virtually identical to its definition. In fact, we have implemented the definition as a Prolog program, to test it on a variety of simple examples. The implementation inherits Prolog's depth-first search strategy for finding defences. It also inherits Prolog's unification for matching sentences with the conclusions of inference rule schemata.

In the procedural version of the program below, we represent destructive assignment by $:=$.

Algorithm 8.1. To find a defence set A for a sentence α, let

$$\mathcal{P} := \{\alpha\},$$
$$\mathcal{O} := \{\},$$
$$A := \mathcal{A} \cap \{\alpha\},$$
$$C := \{\}.$$

While $\mathcal{P} \neq \{\}$ and $\mathcal{O} \neq \{\}$, select $\sigma \in \mathcal{P}$ or $S \in \mathcal{O}$.

1. If $\sigma \in \mathcal{P}$ is selected, then
   (i) if σ is an assumption, then

$$\mathcal{P} := \mathcal{P} - \{\sigma\},$$
$$\mathcal{O} := \mathcal{O} \cup \{\{\overline{\sigma}\}\};$$

(ii) if σ is not an assumption,
then find some inference rule $\frac{R}{\sigma} \in \mathcal{R}$ such that $C \cap R = \{\}$.

$$\mathcal{P} := \mathcal{P} - \{\sigma\} \cup (R - A),$$
$$A := A \cup (\mathcal{A} \cap R).$$

2. If $S \in \mathcal{O}$ is selected and $\sigma \in S$ is selected in S, then
   (i) if σ is an assumption,
   then choose one of the following two alternative options:
   (a) either σ is ignored, i.e.

$$\mathcal{O} := \mathcal{O} - \{S\} \cup \big\{S - \{\sigma\}\big\};$$

   (b) or $\sigma \notin A$ and

$$\mathcal{O} := \mathcal{O} - \{S\},$$
$$\mathcal{P} := \mathcal{P} \cup \{\overline{\sigma}\},$$
$$C := C \cup \{\sigma\};$$

   (ii) if σ is not an assumption, then

$$\mathcal{O} := \mathcal{O} - \{S\} \cup \left\{ S - \{\sigma\} \cup R \;\middle|\; \frac{R}{\sigma} \in \mathcal{R} \text{ and } R \cap C_i = \{\} \right\}.$$

Return A.

Because our Prolog implementation inherits Prolog's unification, it can represent inference rule schemata finitely. As a result, it is possible to represent infinitely many potential attacks by a single schematic potential attack, as illustrated by the following example.

Example 8.1. Consider the sentence $\neg p$ and the assumption-based framework with inference rules:

$$p \leftarrow q(X),\ \neg r\big(succ(X)\big)$$
$$q(0)$$
$$r\big(succ(0)\big)$$
$$r\big(succ(X)\big) \leftarrow r(X)$$

and $\mathcal{A} = \{\neg p\} \cup \{\neg r(X) \mid X \in \{succ^i(0) \mid i \geqslant 0\}\}$.

The only way to attack the sentence is by using instances of the inference rule schema $p \leftarrow q(X),\ \neg r(succ(X))$. Without unification, a proof procedure would have to consider separately the infinitely many potential attacks $\{q(succ^i(0)), \neg r(succ^{i+1}(0))\}$, where $i \geqslant 0$, associated with the infinitely many instances of the inference rule schema.

However, by using unification instead of instantiation, it is possible to consider instead only the single schematic potential attack $\{q(X), \neg r(succ(X))\}$. Then, if the selection rule chooses $q(X)$ in this schematic potential attack, with unification, only the attack $\{\neg r(succ(0))\}$ needs to be considered. This is trivially counter-attacked by the empty set

of assumptions, thus leading to a successful defence for the admissible belief $\neg p$. The following trace shows the state of the variables during the execution of Algorithm 8.1.

$\mathcal{P}$	$\mathcal{O}$	A	C
$\{\underline{\neg p}\}$	$\{\}$	$\{\neg p\}$	$\{\}$
$\{\}$	$\{\{\underline{p}\}\}$	$\{\neg p\}$	$\{\}$
$\{\}$	$\{\{\underline{q(X)}, \neg r(succ(X))\}\}$	$\{\neg p\}$	$\{\}$
$\{\}$	$\{\{\underline{\neg r(succ(0))}\}\}$	$\{\neg p\}$	$\{\}$
$\{\underline{r(succ(0))}\}$	$\{\}$	$\{\neg p\}$	$\{\neg r(succ(0))\}$
$\{\}$	$\{\}$	$\{\neg p\}$	$\{\neg r(succ(0))\}$

The algorithm terminates, returning defence set $\{\neg p\}$.

Because of unification, the algorithm can also represent infinitely counter-attacks against infinitely many attacks, as illustrated by the following example.

Example 8.2. Consider again sentence p and the assumption-based framework in Example 5.2:

$$p \leftarrow q$$
$$\neg q \leftarrow r(X),\ s(X)$$
$$r\big(succ(X)\big) \leftarrow r(X)$$
$$r(0)$$
$$\neg s(X)$$

with set of assumptions $\{q\} \cup \{s(X) \mid X \in \{succ^i(0) \mid i \geqslant 0\}\}$.

The following trace shows the state of the variables during the execution of Algorithm 8.1.

$\mathcal{P}$	$\mathcal{O}$	A	C
$\{\underline{p}\}$	$\{\}$	$\{\}$	$\{\}$
$\{\underline{q}\}$	$\{\}$	$\{q\}$	$\{\}$
$\{\}$	$\{\{\underline{\neg q}\}\}$	$\{q\}$	$\{\}$
$\{\}$	$\{\{r(X), \underline{s(X)}\}\}$	$\{q\}$	$\{\}$
$\{\underline{\neg s(X)}\}$	$\{\}$	$\{q\}$	$\{s(X)\}$
$\{\}$	$\{\}$	$\{q\}$	$\{s(X)\}$

The algorithm terminates, returning defence set $\{q\}$.

The computational complexity of determining whether a set of assumptions is admissible has been studied by Dimopoulos, Nebel and Toni [7]. They investigate the concrete instance of the assumption-based framework for logic programming [10,15], default logic [35], autoepistemic logic [28], Theorist [33] and Circumscription [27]. Because we have restricted ourselves to *flat* assumption-based frameworks, only their results for logic programming and default logic are relevant to this paper.

In [7], the problem of determining whether a set of assumptions is admissible has been shown to be NP-complete for logic programming, and Σ_2^P-complete for default logic. These complexity bounds are identical to the ones for the analogous problem under the more conventional stability semantics for logic programming [17] and default logic [35]. However, as [7] notes, these worse-case results do not take into account the fact that reasoning under the admissibility semantics can be much simpler than reasoning under the stability semantics, due to the "locality" of the former and the "globality" of the latter.

9. Related work

The proof procedures presented in this paper are based upon the admissibility semantics for default reasoning and argumentation developed in [9] and extended in [3]. An earlier version of the top-most level of these proof procedures was presented in [11].

The proof procedures of [11] are expressed in the form of metalogic programs and are derived from specifications in logic, using logic program transformation techniques [31]. In applying these techniques, we discovered that their formal character seemed to obscure the intuitive, dialectic nature of the proof procedures. As a consequence, we decided to develop the less formal, but more intuitive dispute tree approach in this paper.

Kakas and Toni [22,36] also developed argumentation-theoretic proof procedures for the admissibility semantics and (by suitably varying parameters) for the well-founded semantics [16], the weak stability semantics [23] and the acceptability semantics [24]. Their proof procedures are 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.

Although the proof procedures of Kakas and Toni [22,36] employ a form of argumentation tree, they are, in fact, based on a form of dispute derivation, rather than on dispute trees as we have defined them. Therefore, although their proof procedures are related to our dispute derivations, they are not based on dispute trees in our sense. Moreover, their use of tree terminology arguably obscures the dialectic nature of their proof procedures.

The proof procedures of [11,22,36], like those of this paper, generalise the abductive proof procedure for logic programming of [15]. The abductive proof procedure uses recursion to implement a Prolog-like depth-first selection strategy for finding dispute derivations. It interleaves two types of computation. The first type, called the abductive phase, corresponds to arguments by the proponent. The second type, called the consistency phase, corresponds to arguments by the opponent. The argumentation-theoretic nature of the abductive proof procedure was pointed out in [20,21] and developed also by Dung in [10].

Whereas the proof procedures of this paper and of the earlier papers [11,22,36] are based on the assumption-based framework of [3], the two-party immediate response disputes (or TPI-disputes) of Vreeswijk and Prakken [38] are based on the abstract framework of [9]. These TPI disputes are formalised and their efficiency is analysed by Dunne and Bench-Capon in [14]. TPI-disputes are similar to our dispute derivations, but differ in their incorporation of a specific, depth-first search strategy for finding defences.

For this reason, our proof procedures are closer to the argument games of Cayrol, Doutre and Mengin [5], which similarly focus on the definition of winning argumentation strategy, leaving open the search strategy for finding them. The argument games of [5] are based on

the dialectic framework of Jakobovits and Vermeir [19], which in turn is also based on the abstract framework of [9].

Like us, [5] also present their proof procedures both in the form of trees and in the form of derivations. In particular, their ϕ_1-proofs are similar to our dispute derivations (restricted to abstract arguments); and their ϕ_2-winning strategies (extended to the case involving possibly infinitely many arguments) are similar to our abstract dispute trees.

The definition of ϕ_1-proof imposes two intuitively appealing restrictions. One is that the opponent should not put forward an argument if it can be attacked by an argument already deployed by the proponent. The other is that the proponent should not reuse an already deployed argument. While these restrictions may be useful in a setting where arguments are considered only abstractly, they would be very expensive to implement in a framework, like ours, which requires that concrete arguments be constructed explicitly.

In our concrete approach, we approximate the restrictions on ϕ_1-proofs, by filtering with defence and culprit assumptions. In this way, although we can not prevent the proponent from redeploying some already deployed argument, filtering using defence assumptions ensures that the proponent does not have to defend the same defence assumption more than once. Filtering using culprit assumptions similarly ensures that an opponent argument that has already been counter-attacked by some proponent argument will not be selected for counter-attack again.

In addition to this difference between our approach and that of [5], our assumption-based approach has the following features, which distinguish it from all the abstract approaches, as we have already remarked earlier in the paper:

- tight arguments are generated by reasoning backwards from conclusions to assumptions,
- partially constructed, potential arguments can be attacked before they are completed,
- the same counter-argument can be used to attack different arguments sharing the same assumption.

Our proof procedures implement the credulous, admissibility semantics. Dung, Mancarella and Toni in [12] have shown that any proof procedure for the credulous admissibility semantics can be used to compute the sceptical admissibility semantics. Thus, following the approach of [12], the proof procedures in this paper could be used for the sceptical semantics. On the other hand, [5,14,38] present somewhat simpler proof procedures for the sceptical admissibility semantics. The resulting sceptical procedures are weaker than the ones in [12], in the sense that they are proven to be sound and complete only for *coherent* frameworks [9], i.e., frameworks for which the preferred and stable semantics coincide. Instead, the construction of [12] is sound for *any* framework, and complete for any framework in which the procedures for the credulous semantics are complete. However, due to the results of [7,13], it seems likely that the proof procedures for sceptical argumentation of both [5,14,38] and [12] are too inefficient to be used for general frameworks in practise. In particular, [13] shows that sceptical reasoning is Σ_2^p-complete for generic, non-coherent argument systems, and [7] shows that it is Π_2^p-complete for logic programming and Π_3^p-complete for default logic, both of which are flat instances of the assumption-based frameworks considered in this paper.

Dung [8] showed that the SLDNF proof procedure for logic programming can be viewed as an argument game. It can be shown that this argument game is an instance of our finite concrete dispute trees applied to this sceptical semantics.

10. Conclusions

We have presented three, successive refinements of dialectic proof procedures for the admissibility semantics of assumption-based frameworks. The proof procedures search for a winning strategy for a proponent, who argues to establish the admissibility of a belief, against an opponent, who attacks in every possible way the initial and defending arguments of the proponent.

The proof procedures are abstract in the sense that they can be defined for any conventional logic formulated as a collection of inference rules. They show, therefore, how any logic can be extended to a dialectic argumentation system.

The first refinement of the proof procedures is the most abstract, in that it focuses only on the assumptions and conclusions of arguments and ignores their internal structure. The second refinement incorporates the internal structure of tight arguments, by reasoning backwards from conclusions to assumptions.

The first two refinements both represent the proponent's winning strategy in the form of a dispute tree. The third refinement represents winning strategies in the form of dispute derivations, which construct concrete dispute trees by successively expanding their frontiers. It also incorporates filtering of and by defence and culprit assumptions.

Our proof procedures generalise the abductive proof procedure for logic programming of [15], which in turn is an extension of Prolog and of the SLDNF proof procedure. On the one hand, they contribute to the special case of logic programming, because they allow a more general and more flexible way of executing negation as failure. On the other hand, they also show how any logic can be turned into a logic programming style language with the potential efficiency of logic programming style proof procedures.

Among the desirable features of our proof procedure are the generality and flexibility provided by separating out both the search strategy for finding defences and the selection strategy for deciding what part of a dispute tree to investigate next. This flexibility can be exploited by such heuristics as focusing on subgoals with fewer possible solutions before subgoals with a greater number of possible solutions.

Our dispute derivations expand the frontiers of concrete dispute trees one step at a time. It might be useful to investigate other refinements of concrete dispute trees that expand frontiers several nodes at a time, in parallel.

Our proof procedures can be used for default reasoning, either by using the credulous, admissibility semantics directly, or by using it as a basis for computing other sceptical semantics and even certain cases of the stable semantics, as shown for example in [12].

It would be useful to investigate further the applications of our proof procedures in such areas as legal reasoning and negotiation. The assumption-based nature of our approach to argumentation might be especially useful for these application because it shows how a difference of opinion can be reduced to a difference of assumptions. In particular, the defence and culprit sets generated by our proof procedures can be understood as identifying

the critical differences of opinion on which conflicting beliefs are based. Moreover, by separating the search space from the search strategy, our approach might help to identify different ways in which a proponent might be able to win an argument, while minimising the conflict (culprit set) with the opponent.

Acknowledgements

This research was partially funded by the IST programme of the EC, FET under the IST-2001-32530 SOCS project, within the Global Computing proactive initiative, by the Italian MIUR programme "Rientro dei cervelli", and by a research grant from the Royal Thai Government. The authors would like to thank Paul Krause and the anonymous referees for their comments on an earlier version of this paper.

Appendix A. Proofs

A.1. Proof of Theorem 4.1

A.1.1. Preliminaries

Given a backward deduction $S_1, \ldots, S_m$, based on a set of assumptions A, we write

$$S_i \xrightarrow{\frac{S}{\sigma}} S_{i+1}$$

if S_{i+1} is obtained from S_i by the application of an inference rule $\frac{S}{\sigma}$, and we write

$$S_i \xrightarrow{\sigma} S_{i+1}$$

where σ is an assumption (and $S_{i+1} = S_i$).

A.1.2. Proof

1. Let $S_1, \ldots, S_m$ be a backward argument of α supported by a set of assumptions A. It is not difficult to see that the sequence $\sigma_{m-1}, \ldots, \sigma_1$, where $\sigma_1 = \alpha$ and, for each $2 \leqslant i \leqslant m-1$, either $S_i \xrightarrow{\frac{S}{\sigma_i}} S_{i+1}$ or $S_i \xrightarrow{\sigma_i} S_{i+1}$ holds, is a forward deduction of α.
2. Let $\sigma_1, \ldots, \sigma_n$ be a forward deduction of α (i.e., $\alpha = \sigma_n$). For each σ_i, let $rank(i) = \min_{j=1,\ldots,n}\{j \mid \sigma_j = \sigma_i\}$. Hence $\sigma_{rank(i)} = \sigma_i$ and for each $1 \leqslant j < rank(i)$, $\sigma_j \neq \sigma_i$. If σ_i is not an assumption, let R be the rule that is used to derive $\sigma_{rank(i)}$, i.e., R is of the form $\frac{S}{\sigma_{rank(i)}}$ such that $S \subseteq \{\sigma_{rank(i)-1}, \ldots, \sigma_1\}$. Define $Pre(\sigma_i) = S$. It is clear that for each $\sigma \in Pre(\sigma_i)$: $rank(\sigma) < rank(\sigma_i)$.
   Let *sl* be a selection rule. Define inductively a sequence $S_1, \ldots, S_m$ as follows:
   $S_1 = \{\alpha\}$.
   Suppose S_i has been constructed. Let σ be the sentence selected by *sl* from S_i.[10] If σ is an assumption then $S_{i+1} = S_i$. If σ is not an assumption then $S_{i+1} = S_i - \{\sigma\} \cup Pre(\sigma)$.

[10] Note that no assumption should be selected twice by *sl*.

It remains to show that the above construction is finite.
For each set $S \subseteq \{\sigma_1, \ldots, \sigma_n\}$, define $rank(S) = \max\{rank(\alpha) \mid \alpha \in S\}$. Further, for each i, define $S|_i = \{\alpha \in S \mid rank(\alpha) = i\}$.
Define an order between sets of sentences from $\sigma_1, \ldots, \sigma_n$ as follows: $S \sqsupset R$ iff $rank(S) \geqslant rank(R)$ and there exists $k \leqslant rank(S)$ such that the following conditions hold:

- for each $k < i \leqslant rank(S)$: $S|_i = R|_i$,
- $S|_k \supset R|_k$.

It is clear that there is no infinite decreasing sequence wrt $\sqsupset$. It is also easy to see that $S_i \sqsupset S_{i+1}$. It follows immediately that the constructed sequence is a backward deduction.

A.2. Proof of Theorem 4.2

Obvious.

A.3. Proof of Theorem 5.1

A.3.1. Preliminaries

For each node N in a abstract dispute tree, let $arg(N)$ denote the argument labelling N while $supp(N)$ denotes the set of assumptions on which $arg(N)$ is based. If N is an opponent node and assumption σ is the selected culprit in N, then σ is denoted by $culprit(N)$. For each argument a, $supp(a)$ denotes the set of assumptions on which a is based.

A.3.2. Proof

i) We prove that 1) A attacks every tight attack against it and 2) A does not attack itself.
   1) Assume that a is a tight argument attacking some assumption $\delta \in A$. From the definition of A, there is a proponent node N such that $\delta \in supp(N)$. Therefore there is a child M of N which is an opponent node labelled by a. Let K be the proponent node that is the only child of M. It is clear that $supp(K) \subseteq A$. This implies that A attacks a.
   2) From the definition of admissible dispute tree, it follows that for each opponent node N in $\mathcal{T}$, $culprit(N) \notin A$.
   Assume that A attacks itself. Let $\delta \in A$ such that A attacks δ. Because A is the union of the support of arguments at proponent nodes, there is a proponent node N such that $\delta \in supp(N)$. Since A attacks δ, A attacks $supp(N)$. Therefore, there is a tight argument t that attacks δ and $supp(t) \subseteq A$ (Theorem 4.1). Hence there is a child M of N that is labelled by t. Since $culprit(M) \in supp(t)$ and $supp(t) \subseteq A$, it follows that $culprit(M) \in A$: contradiction to the admissibility of $\mathcal{T}$.

ii) Construct a dispute tree as follows:
   (a) The root is labelled by a. This node is of rank 0.
   (b) Suppose we have constructed all the nodes of rank $\leqslant i$ where $i = 2k$, $k \geqslant 0$. Note that nodes of rank i are proponent nodes whose supports consist of assumptions from A.

- For each node N of rank i, for each tight attack t against $arg(N)$ there is exactly one child of N that is an opponent node and labelled with t. The rank of these new nodes is $i+1$.
- As each node M of rank $i+1$ represents an attack against A, there is an assumption $\delta \in supp(M)$ such that A attacks δ. Thus, there is a tight argument t_M whose support consist of assumptions from A that attacks δ. Expand the tree as follows:
  For each node M of rank $i+1$, create exactly one child that is a proponent node, and label it with t_M. The new nodes created in this step are of rank $i+2$.
  It is clear that the supports of the new nodes of rank $i+2$ consist only of assumptions from A.
- If the set of nodes of rank $i+2$ is empty then stop. We have successfully constructed a finite abstract dispute tree. Otherwise, set i to $i+2$ and goto (b).

If the above algorithm does not terminate, we have constructed an infinite abstract dispute tree. It is obvious that the constructed tree is an admissible dispute tree whose defence set A' is a subset of A and a superset of $A_0 = supp(a)$. Finally, by part (i) of the theorem, A is admissible.

A.4. Proof of Theorem 5.2

A.4.1. Preliminaries

A pair of nodes (P, O) is said to be conflicting if P is a proponent node and O is an opponent node such that the culprit of O belongs to the support of P. Define $(P, O) \sqsubset (P', O')$ if (P, O) and (P', O') are conflicting pairs of nodes, P' is a child of O, and O' is a child of P.

It is clear that the existence of an infinite sequence $(P_0, O_0) \sqsubset (P_1, O_1) \sqsubset \cdots$ implies that the dispute tree is infinite.

A.4.2. Proof

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

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

Proof of claim. Let α be the culprit at O. Let P' be the only child of O and t be the argument labelling P'. It is clear that P' is a proponent node and t attacks α.

Because α belongs to $supp(P)$ (defined in the proof of Theorem 5.1) there is a child O' of P that is labelled by t. Because O' is an opponent node, $supp(O') \neq \{\}$. Hence it is clear that the culprit at O' belongs to $supp(t)$ whereas $supp(t) = supp(P')$. This implies that (P', O') is a conflicting pair of nodes and $(P, O) \sqsubset (P', O')$. □

A.5. Proof of Theorem 6.1

A.5.1. Preliminaries

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

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

Lemma A.1. *Let α be a non-assumption sentence.*

1. *Let $\mathcal{B}$ be a representative set of support of α. Then for each rule of the form $\frac{S}{\alpha}$, there is at least one $\beta \in S$ such that $\mathcal{B}$ is also a representative set of supports of β.*
2. *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*

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

is a representative set of supports of α.

Proof of Lemma A.1.

1. Let $\mathcal{B}$ be a representative set of supports of α. Let r be a rule of the form $\frac{S}{\alpha}$. We want to show 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_β such that R_β supports β and $\mathcal{B} \cap R_\beta = \{\}$. Therefore, $R = \bigcup\{R_\beta \mid \beta \in S\}$ is a set of assumptions that supports α and $R \cap \mathcal{B} = \{\}$. This is contrary to the fact that $\mathcal{B}$ is a representative set of supports of α. Therefore, there is $\beta \in S$ such that $\mathcal{B}$ is a representative set of supports for β.
2. Let R be a set of assumption that supports α. 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 that $R \cap B_r \neq \{\}$. Hence $\mathcal{B} \cap R \neq \{\}$. Thus, we have proved that $\mathcal{B}$ is a representative set of supports of α. □

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

A.5.2. Support tree

A support tree of a sentence α (wrt a selection function *sl*) is defined as follows:

1. The root is a proponent node labelled by $\{\alpha\}$.
2. Let N be a proponent node labelled P. If P is empty, then N is a terminal node. If P is not empty and the selected sentence (wrt *sl*) in P is an assumption σ, then there exist exactly two children of N: one is an opponent node that is a terminal node labelled by $\{\overline{\sigma}\}$ and the other is a proponent node labelled by $P - \{\sigma\}$. If σ is not an assumption,

then there exists some inference rule $\frac{S}{\sigma} \in \mathcal{R}$ and there exists exactly one child of N labelled by $P - \{\sigma\} \cup S$.

It is easy to see that finite support trees of α correspond to backward deductions of α.

A.5.3. Representative tree

A representative tree (wrt selection function *sl*) of a sentence α is defined as follows:

1. The root is an opponent node labelled by $\{\alpha\}$.
2. Let N be an opponent node labelled by O. Then $O \neq \{\}$. Let σ be the sentence selected (wrt *sl*) in O.
   If σ is an assumption, then N has exactly one child that is either a terminal and proponent node labelled by $\{\overline{\sigma}\}$ (σ is a *culprit*) or an opponent node labelled by $O - \{\sigma\}$ (σ is *ignored*).
   If σ is not an assumption and there exists no inference rule $\frac{S}{\sigma} \in \mathcal{R}$, then N is a terminal node. Otherwise, the children of N are a set of opponent nodes labelled by the sets of sentences $O - \{\sigma\} \cup S$, where $\frac{S}{\sigma} \in \mathcal{R}$. (There being one such child for each such inference rule.)

Let α be a sentence and *sl* be a selection function, and $\mathcal{T}$ be a representative tree of α wrt *sl*. Further, let $B = S_0, \ldots, S_m$ be a backward deduction of α wrt *sl*. We say that *a path $N_0, \ldots, N_k$ from the root N_0 of $\mathcal{T}$ to N_i corresponds to a prefix of B* iff (1) for each $0 \leqslant i \leqslant k$, $label(N_i) = S_i - A_i$ where A_i is the set of assumptions selected on the path from N_0 to N_i, and (2) the same sentence is selected at N_i and S_i. It is easy to see (by induction) that the following property holds:

Lemma A.2. *There is a terminal node N in $\mathcal{T}$ that is also a proponent node such that the path from the root to the parent node of N corresponds to a prefix of B.*

Furthermore:

Lemma A.3.

1. *Let $\mathcal{B}$ be the set of assumptions selected in a representative tree $\mathcal{T}$ of α wrt sl. Then $\mathcal{B}$ is a representative set of supports of α.*
2. *Let $\mathcal{B}$ be a representative set of supports of α. Then, there is a representative tree $\mathcal{T}$ of α wrt sl such that the set of culprit assumptions selected in the tree is a subset of $\mathcal{B}$.*

Proof of Lemma A.3.

1. Let A be a set of assumptions supporting α. We want to show that $A \cap \mathcal{B} \neq \{\}$. From Theorem 4.1, there is a backward deduction $D = S_0, \ldots, S_m$ of α wrt *sl* such that $S_m \subseteq A$ and $S_0 = \{\alpha\}$. From Lemma A.2, there exists a terminal node N in $\mathcal{T}$ such that the path from the root to the parent node of N corresponds to a prefix of D and N

is a proponent node. It follows that the sentence selected at the parent of N belongs to S_m. That means $A \cap \mathcal{B} \neq \{\}$.

2. $\mathcal{T}$ is defined as follows:
   (a) The root is an opponent node labelled by $\{\alpha\}$.
   (b) Let N be an opponent node labelled by B.
   If $B \neq \{\}$, select a sentence $\sigma \in B$ following *sl*.
   If σ is an assumption not belonging to $\mathcal{B}$, then N has exactly one child that is an opponent node labelled by $B - \{\sigma\}$.
   If σ is an assumption belonging to $\mathcal{B}$, then N has exactly one child that is a proponent node labelled by $\{\overline{\sigma}\}$.
   If σ is not an assumption and there exists no inference rule $\frac{S}{\sigma} \in \mathcal{R}$, then N is a terminal node. Otherwise, the children of N are a set of opponent nodes labelled by the sets of sentences $B - \{\sigma\} \cup S$, where $\frac{S}{\sigma} \in \mathcal{R}$. (There being one such child for each such inference rule.)

It remains to show that $\mathcal{T}$ is a representative tree of α. Suppose the contrary. Hence there is an opponent node N in $\mathcal{T}$ that is labelled by an empty set. It follows that the branch from the root to N represents a backward deduction of α supported by a set of assumptions disjoint to $\mathcal{B}$: contradiction to the assumption that $\mathcal{B}$ is a representative set of supports of α. □

A.5.4. Proof

We prove Theorem 6.1 by transforming concrete dispute trees into abstract dispute trees and vice versa.

A.5.5. Transformation from concrete dispute trees into abstract dispute trees

Let $\mathcal{T}$ be a concrete dispute tree (wrt selection function *sl*) and N be a node in $\mathcal{T}$. Define $\mathcal{T}(N)$ as follows:

- Let $\mathcal{T}_1$ be the subtree of $\mathcal{T}$ rooted at N.
- Delete every node M in $\mathcal{T}_1$ such that the type of the parent of M is different to the type of N. The resulting tree is $\mathcal{T}(N)$.[11]

It is easy to see that if N is an opponent node labelled by $\{\alpha\}$ then $\mathcal{T}(N)$ is a representative tree of α. Further if N is a proponent node labelled by $\{\alpha\}$ then $\mathcal{T}(N)$ is a support tree of α. In the latter case, let $arg(N)$ denote the tight argument corresponding to $\mathcal{T}(N)$.

In the following, we construct inductively a sequence of trees $(T_0, f_0), \ldots, (T_n, f_n), \ldots$ where:

- Nodes in T_i are either proponent or opponent nodes labelled with tight arguments. For an opponent node, an assumption from the set of assumptions supporting the argument labelling it is also given as the culprit at this node.
- The frontier of T_i consists only of proponent nodes.

[11] The type of a node determines whether it is an proponent or opponent node. Note also that if a node is deleted then all of its successors are deleted as well.

- f_i are functions mapping the proponent nodes in the frontier of T_i into the proponent nodes of $\mathcal{T}$ such that each node N in the frontier of T_i is labelled with $arg(\mathcal{T}(f_i(N)))$.

1. T_0 consists of exactly one node that is a proponent node labelled by $arg(N_0)$ where N_0 is the root of $\mathcal{T}$. f_0 maps the root of T_0 into N_0.
2. For each proponent node N in the frontier of T_i with $f_i(N) = M$ where M is a proponent node in $\mathcal{T}$ and a is the argument labelling N, expand T_i as follows:
   For each tight argument b attacking a, add a child N_b of N to T_i that is an opponent node labelled by b.
   Let σ be the assumption in a such that b is a tight argument of $\overline{\sigma}$. From the assumption hypothesis, $a = arg(\mathcal{T}(M))$. It is clear that σ is selected at some node M' in $\mathcal{T}(M)$.
   From Theorem 4.1, there is a tight argument c wrt the selection function *sl* whose support is a subset of $supp(b)$.
   Let H be the child of M' in $\mathcal{T}$ that is labelled with $\{\overline{\sigma}\}$. H is hence an opponent node. Therefore, there is a branch in $\mathcal{T}(H)$ corresponding to c (Lemma A.2). Let δ be the assumption selected on this branch and let K be the terminal proponent node labelled by $\{\overline{\delta}\}$ in $\mathcal{T}(H)$. It is clear that δ belongs to the set of assumptions of b. Select δ as the culprit at N_b and add to N_b in T_i exactly one child N' that is a proponent node labelled by $arg(\mathcal{T}(K)$.
   Define $f_{i+1}(N') = K$.
3. The obtained tree is T_{i+1}.
4. Define $reduct(\mathcal{T})$ to be the limit of $T_0, T_1, \ldots, T_i, \ldots$.

It follows immediately that $reduct(\mathcal{T})$ is an abstract dispute tree of α such that the defence set of $reduct(\mathcal{T})$ is a subset of the defence set of $\mathcal{T}$.
Due to the fact the culprits in $reduct(\mathcal{T})$ are also the culprits in $\mathcal{T}$, it follows that, if $\mathcal{T}$ is admissible, then $reduct(\mathcal{T})$ is also admissible.

A.5.6. Transformations from abstract dispute trees into concrete dispute trees

Let $\mathcal{T}$ be an abstract dispute tree for a sentence α. Let D, C be the defence set and the set of culprits of $\mathcal{T}$, respectively.

For each $\sigma \in C$, let $arg(\overline{\sigma})$ be an argument with conclusion $\overline{\sigma}$ labelling some node in $\mathcal{T}$.

For each $\sigma \in D$, let B_σ be the set of culprits in the arguments attacking σ in $\mathcal{T}$. It is clear that B_σ is a representative set of supports for $\overline{\sigma}$.

In the following, we construct inductively a sequence of trees $T_0, \ldots, T_n, \ldots$ as follows:

1. T_0 is a support tree of α corresponding to the argument labelling the root of $\mathcal{T}$.
2. Let $i = 2k$ such that the non-terminal nodes in the frontier of T_i are opponent nodes labelled by set of sentences of the form $\{\overline{\sigma}\}$, where $\sigma \in D$.
   Expand each such node by a representative tree (wrt *sl*) of $\overline{\sigma}$ wrt B_σ (the existence of such tree is guaranteed by Lemma A.3). The obtained tree is T_{i+1}.
3. Let $i = 2k + 1$ such that the non-terminal nodes in the frontier of T_i are proponent nodes labelled by sets of sentences of the form $\{\overline{\sigma}\}$, where $\sigma \in C$.
   Expand each such node by a support tree of $\overline{\sigma}$ (wrt *sl*) corresponding to $arg(\sigma)$. The obtained tree is T_{i+1}.

4. Define $expand(\mathcal{T})$ to be the limit of T_i.

It follows immediately that $expand(\mathcal{T})$ is a concrete dispute tree of α such that the defence set of $expand(\mathcal{T})$ is a subset of the defence set of $\mathcal{T}$.

Due to the fact the culprits in $expand(\mathcal{T})$ are also the culprits in $\mathcal{T}$, it follows that, if $\mathcal{T}$ is admissible, then $expand(\mathcal{T})$ is also admissible.

A.6. Proof of Corollary 6.1

i) Let *sl* be the selection function of $\mathcal{T}$. We prove that 1) A attacks every attack against it, and 2) A does not attack itself.
   1) Assume that a is a tight argument attacking some assumption $\sigma \in A$. From Theorem 4.1, there exists a tight argument b wrt *sl* such that $supp(b) \subseteq supp(a)$ and both a and b support the same conclusion. (See the proof of Theorem 5.1 for the definition of *supp*.)
   From the definition of A, there is an opponent node N labelled by $\{\overline{\sigma}\}$. Therefore there is a branch in the representative tree $\mathcal{T}(N)$ corresponding to b (Lemma A.2). Hence, the culprit selected on this branch is attacked by A. Therefore A attacks b, and hence it also attacks a.
   2) Assume A attacks itself. Then, there exists an argument a attacking some assumption $\sigma \in A$ and such that the support of a is a subset of A.
   From the definition of A, there is an opponent node N labelled by $\{\overline{\sigma}\}$. Therefore there is a branch in the representative tree $\mathcal{T}(N)$ corresponding to b (Lemma A.2). Hence, the culprit selected on this branch belongs to A. This is impossible due to the admissibility of $\mathcal{T}$.

ii) From Theorem 5.1, part (ii), there exists an admissible abstract dispute tree $\mathcal{T}$ for a such that the defence set D of $\mathcal{T}$ is a subset of A. From Theorem 6.1, there exists an admissible concrete dispute tree $\mathcal{T}'$ of α such that the defence set A' of $\mathcal{T}'$ is a subset of D. Hence, A' is a subset of A and a superset of $A_0 = supp(a)$. Finally, by part (i) of the corollary, A is admissible.

A.7. Proof of Theorem 6.2

A.7.1. Preliminaries

A pair of nodes (P, O) are said to be conflicting if P is a proponent node and O is an opponent node such that both of them are labelled by the same set $\{\overline{\alpha}\}$ where α is an assumption. Define $(P, O) \sqsubset (P', O')$ if (P, O) and (P', O') are conflicting pairs of nodes, P' is a successor of O, and O' is a successor of P.

It is clear that the existence of an infinite sequence $(P_0, O_0) \sqsubset (P_1, O_1) \sqsubset \cdots$ implies that the concrete dispute tree is infinite.

A.7.2. Proof

To prove the theorem, we prove that any non-admissible concrete dispute tree is infinite. Let $\mathcal{T}$ be a non-admissible concrete dispute tree. It is obvious that there exists a conflicting pair in $\mathcal{T}$. The theorem follows directly from the following:

Claim. *Let (P, O) be a conflicting pair of nodes in a concrete dispute tree $\mathcal{T}$ There is another conflicting pair (P', O') in $\mathcal{T}$ such that $(P, O) \sqsubset (P', O')$.*

Proof of claim. Let α be an assumption such that $\{\overline{\alpha}\}$ labels both P and O. Let D be the backward deduction represented by the support tree $\mathcal{T}(P)$. From Lemma A.2, there is a path from the root O to a terminal node P' in the tree $\mathcal{T}(O)$ that corresponds to D. Let β be the assumption such that $\{\overline{\beta}\}$ labels P'. It follows that there is also a terminal node O' in $\mathcal{T}(P)$ also labelled by $\{\overline{\beta}\}$. It is clear that P' is a proponent node and O' is an opponent node and $(P, O) \sqsubset (P', O')$. □

A.8. Proof of Theorem 7.1

A.8.1. Preliminaries

A partial (concrete) dispute tree is a tree obtained from a concrete dispute tree by deleting some nodes together with all of their successors and siblings and the successors of the siblings.

A terminal node in a partial dispute tree is a terminal node N such that (1) N is a proponent labelled by the empty set, or (2) N is an opponent node and there is no rule whose conclusion coincides with the selected sentence at N.

A.8.2. Proof

Let $\mathcal{D} = \langle \mathcal{P}_0, \mathcal{O}_0, A_0, C_0\rangle, \ldots, \langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i\rangle, \ldots, \langle \mathcal{P}_n, \mathcal{O}_n, A_n, C_n\rangle$ be a dispute derivation of a sentence α.

In the following, we construct inductively a sequence $T_0, \ldots, T_n$ of partial (concrete) dispute trees satisfying the following properties:

1. The union of all sentences belonging to the labels of proponents nodes in T_i is equal to $\mathcal{P}_0 \cup \cdots \cup \mathcal{P}_i$.
2. The multiset of all non-assumption sentences appearing in the frontier proponent nodes in T_i is equal to the multiset of non-assumption sentences in $\mathcal{P}_i$.
3. The set of assumptions in $\mathcal{P}_i$ is a subset of the set of assumptions appearing at the frontier proponent nodes.
4. There is one-to-one mapping mp_i from $\mathcal{O}_i$ into the set of frontier non-terminal opponent nodes such that for each $S \in \mathcal{O}_i$, S labels $mp_i(S)$. Moreover, if a non-terminal frontier opponent node N in T_i does not belong to the range of mp_i, then N satisfies two properties:
   (a) N is labelled by $\{\overline{\alpha}\}$ for some assumption $\alpha \in A_i$.
   (b) The parent of N is a proponent node.

- T_0 is a tree consisting of exactly one proponent node labelled $\{\alpha\}$.
- Suppose T_i has been constructed. T_i is expanded into T_{i+1} as follows:
  - If $\sigma \in \mathcal{P}_i$ is selected at step i in the derivation then proceed as follows: Let N be a frontier proponent node in T_i such that σ appears at N.
    * If σ is a non-assumption then expand T_i into T_i' by adding a child to N that is a proponent node P_0 labelled by $S - \{\sigma\} \cup R$ where S labels N and $\frac{R}{\sigma}$ is

the rule selected to proceed from $\langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i \rangle$ to $\langle \mathcal{P}_{i+1}, \mathcal{O}_{i+1}, A_{i+1}, C_{i+1} \rangle$. Let $R \cap A_i = \{\alpha_1, \ldots, \alpha_k\}$. Construct a sequence of trees $T_{i,0}, T_{i,1}, \ldots, T_{i,k}$ as follows:

- $T_{i,0} = T'_i$.
- Suppose $T_{i,j}$ has been constructed. Expand $T_{i,j}$ into $T_{i,j+1}$ by adding two children to node P_j: one is a proponent node P_{j+1} labelled by $S - \{\alpha_j\}$, where S is the multiset labelling P_j, the other is an opponent node labelled by $\{\overline{\alpha_j}\}$.
- $T_{i+1} = T_{i,k}$.

* If σ is an assumption then expand T_i into T_{i+1} by adding two children to N: one is a proponent node labelled by $S - \{\sigma\}$, where S is the multiset labelling N, the other is an opponent node labelled by $\{\overline{\sigma}\}$.

– If $S \in \mathcal{O}_i$ and $\sigma \in S$ are selected at step i in the derivation then proceed as follows: Let $N = mp_i(S)$. N is hence a frontier opponent node in T_i such that S labels N and σ appears in S.

* If σ is a non-assumption then expand T_i into T_{i+1} by: 1) adding for each rule $\frac{R}{\sigma}$ a child M to N that is an opponent node labelled by $S - \{\sigma\} \cup R$. Note that if no such child exists, then N is a terminal node in T_{i+1}; 2) if R contains an assumption σ from C_i then σ is the culprit at N and add exactly one child to M that is a proponent node labelled by $\{\overline{\sigma}\}$.
* If σ is an assumption then expand T_i into T_{i+1} by adding exactly one child to N which is
  - an opponent node labelled by $S - \{\sigma\}$ if σ is ignored in the derivation, or
  - a proponent node labelled by $\{\overline{\sigma}\}$, otherwise.

It is not difficult to show by induction that all trees T_i satisfy the properties 1–4 listed above. Since $\mathcal{D}$ is a dispute derivation, the following lemma holds obviously:

Lemma A.4.

1. *The frontier opponent* (*resp. proponent*) *nodes of T_n are either terminal nodes or labelled with a set of the form $\{\overline{\sigma}\}$ where σ is an assumption such that there is a unique non-frontier opponent* (*resp. proponent*) *node N_σ in T_n that is also labelled by $\{\overline{\sigma}\}$.*
2. *The set of assumptions appearing at the proponent nodes in T_n is equal to A_n.*
3. *The set of assumptions α such that $\{\overline{\alpha}\}$ labels a proponent node is equal to C_n.*

For each assumption σ appearing at a proponent node in T_n, let T_σ be the subtree of T_n whose root is N_σ.

Construct a new sequence of trees $T'_0, T'_1, \ldots$ as follows:

- $T'_0 = T_n$.
- Suppose T'_i has been constructed. Expand T'_i into T'_{i+1} by adding simultaneously to all non-terminal frontier nodes in T'_i that are labelled by $\{\overline{\sigma}\}$, where σ is an assumption, the tree $T_{\overline{\sigma}}$.

Let $\mathcal{T}$ be the limit of $T'_0, T'_1, \ldots$. It is not difficult to see that $\mathcal{T}$ is a concrete dispute tree whose defence set coincides with the defence set of T_n and whose culprit set coincides with the culprit set of T_n. From Lemma A.4, it follows immediately that $\mathcal{T}$ is an admissible concrete dispute tree whose defence set coincides with A_n.

A.9. Proof of Theorem 7.2

Let T be a finite admissible concrete dispute tree of α. Let $N_0, \ldots, N_n$ be a listing of nodes in T according to the leftmost depth-first order.

A node N in T is said to be an *attack node* if N is labelled by $\{\overline{\sigma}\}$ where σ is an assumption and the parent of N is a node of the opposite type.[12] Because T is admissible, attack nodes with the same label are of the same type.

Let N_k be the first attack node in $N_0, \ldots, N_n$ such that there exists another attack node N_i, $i > k$, with the same label.

The tree $simpl(T)$ is obtained by deleting all nodes except N_k that are attack nodes with the same label as N_k, together with their successors.

Because T is finite, there exists i such that $simpl^i(T) = simpl^{i+1}(T)$. Let $T' = simpl^i(T)$.

It is clear that the listing of the nodes in T' according to the leftmost depth-first search is a subsequence of the list $N_0, \ldots, N_n$.

It is obvious that following property holds for T':

Property. *Let σ be an assumption.*

If σ appears in some proponent nodes of T', then there is exactly one attack opponent node in T' that is labelled by $\{\overline{\sigma}\}$.

If σ is the culprit in some opponent nodes of T' then there is exactly one attack proponent node in T' that is labelled by $\{\overline{\sigma}\}$.

Let N be a proponent node (labelled by S) in T' such that the selected sentence at N is an assumption σ and N has exactly one child M in T' that is a proponent node (labelled by $S - \{\sigma\}$). Simplify T' as follows: Collapse N, M together and remove σ from the labels of the all proponent nodes that are ancestors of N such that there is no opponent nodes on the path from them to N. Repeat this step until it can not carried out anymore. Let the obtained tree be T''.

Let $K_0, \ldots, K_m$ be a listing of the nodes in T'' according to the leftmost depth-first order. An opponent node K_i is said to be redundant in T'' if the label of K_i contains assumption σ and there is a proponent node K_j, $j < i$, that is labelled by $\{\overline{\sigma}\}$.

Continue to simplify T'' by removing all redundant nodes in T'' together with their successors. Let the obtained tree be $\mathcal{T}$. It is easy to see that the listing $M_0, \ldots, M_k$ of the nodes in $\mathcal{T}$ according to the leftmost depth-first search is a subsequence of the list $N_0, \ldots, N_n$.

[12] Note that a opponent (resp. proponent) node is said to be of opposite type of a proponent (resp. opponent) node.

Let T_i be the tree obtained from $\mathcal{T}$ be deleting all nodes M_j, $j > i$, except those that are siblings of the nodes $M_0, \ldots, M_i$.

For each $0 \leqslant i \leqslant k$, define

- $\mathcal{P}_i$ is the multiset of sentences appearing in the frontier proponent nodes of T_i,
- $\mathcal{O}_i$ is the multiset of the labels of the non-terminal frontier opponent nodes in T_i,
- A_i is the set of assumptions appearing in the proponent nodes in T_i,
- C_i is the set of assumptions σ such that $\{\overline{\sigma}\}$ labels some proponent node in T_i whose parent is an opponent node.

From the definition of T_i, it is not difficult to see that

$$\mathcal{D} = \langle \mathcal{P}_0, \mathcal{O}_0, A_0, C_0 \rangle,\ \ldots,\ \langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i \rangle,\ \ldots,\ \langle \mathcal{P}_n, \mathcal{O}_n, A_n, C_n \rangle$$

is a dispute derivation.

References

[1] K.R. Apt, H. Blair, A. Walker, Towards a theory of declarative knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, San Mateo, CA, 1988.
[2] K.R. Apt, R. Bol, Logic programming and negation: A survey, J. Logic Programming 19–20 (1994) 9–71.
[3] A. Bondarenko, P.M. Dung, R.A. Kowalski, F. Toni, An abstract, argumentation-theoretic framework for default reasoning, Artificial Intelligence 93 (1–2) (1997) 63–101.
[4] A. Bondarenko, F. Toni, R.A. Kowalski, An assumption-based framework for non-monotonic reasoning, in: A. Nerode, L. Pereira, et al. (Eds.), Proc. 2nd International Workshop on Logic Programming and Non-monotonic Reasoning, MIT Press, Cambridge, MA, 1993, pp. 171–189.
[5] C. Cayrol, S. Doutre, J. Mengin, On decision problems related to preferred semantics a of argumentation frameworks, J. Logic Comput. 13 (3) (2003) 377–403.
[6] C. Chesnevar, A. Maguitman, R. Loui, Logical models of argument, ACM Comput. Surv. 32 (4) (2000) 337–383.
[7] Y. Dimopoulos, B. Nebel, F. Toni, On the computational complexity of assumption-based argumentation for default reasoning, Artificial Intelligence 141 (2002) 57–78.
[8] P.M. Dung, Logic programming as dialog-game, Technical report, AIT, 1993.
[9] 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.
[10] P.M. Dung, An argumentation theoretic foundation of logic programming, J. Logic Programming 22 (1995) 151–177.
[11] P.M. Dung, R.A. Kowalski, F. Toni, Synthesis of proof procedures for default reasoning, in: J. Gallagher (Ed.), Proc. LOPSTR, in: Lecture Notes Comput. Sci., vol. 1207, Springer-Verlag, Berlin, 1996, pp. 313–324.
[12] P.M. Dung, P. Mancarella, F. Toni, Argumentation-based proof procedures for credulous and sceptical non-monotonic reasoning, in: A.C. Kakas, F. Sadri (Eds.), Computational Logic: Logic Programming and Beyond—Essays in Honour of Robert A. Kowalski, in: Lecture Notes Artif. Intell., vol. 2408, Springer-Verlag, Berlin, 2002, pp. 289–310.
[13] P. Dunne, T. Bench-Capon, Coherence in finite argument systems, Artificial Intelligence 141 (2002) 187–203.
[14] P.E. Dunne, T.J.M. Bench-Capon, Two party immediate response disputes: Properties and efficiency, Artificial Intelligence 149 (2003) 221–250.
[15] K. Eshghi, R.A. Kowalski, Abduction compared with negation as failure, in: Proc. ICLP, MIT Press, Cambridge, MA, 1989.

[16] A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 620–650.
[17] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, K.A. Bowen (Eds.), Proceedings of the 5th International Conf. on Logic Programming, MIT Press, 1988, pp. 1070–1080.
[18] T.F. Gordon, The Pleadings Game. An Artificial Intelligence Model of Procedural Justice, Kluwer, Dordrecht, 1995.
[19] H. Jakobovits, D. Vermeir, Dialectic semantics for argumentation frameworks, in: Proc. ICAIL, ACM Press, New York, 1999, pp. 53–62.
[20] A.C. Kakas, R.A. Kowalski, F. Toni, Abductive logic programming, J. Logic Comput. 2 (6) (1993) 719–770.
[21] A.C. Kakas, R.A. Kowalski, F. Toni, The role of abduction in logic programming, in: D.M. Gabbay, C.J. Hogger, J.A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 5, Oxford University Press, Oxford, 1998, pp. 235–324.
[22] A.C. Kakas, F. Toni, Computing argumentation in logic programming, J. Logic Comput. 9 (1999) 515–562.
[23] A.C. Kakas, P. Mancarella, Stable theories for logic programs, in: Proc. ISLP, MIT Press, Cambridge, MA, 1991.
[24] A.C. Kakas, P. Mancarella, P.M. Dung, The acceptability semantics for logic programs, in: P. Van Hentenryck, et al. (Eds.), Proc. ICLP, MIT Press, Cambridge, MA, 1994, pp. 504–519.
[25] R.A. Kowalski, F. Toni, Abstract argumentation, J. Artificial Intelligence and Law 4 (3–4) (1996) 275–296, special issue on Logical Models of Argumentation.
[26] R.P. Loui, J. Norman, Rationales and argument moves, Artificial Intelligence and Law 3 (1995) 158–189.
[27] J. McCarthy, Circumscription—a form of non-monotonic reasoning, Artificial Intelligence 13 (1980) 27–39.
[28] R. Moore, Semantical considerations on non-monotonic logic, Artificial Intelligence 25 (1985).
[29] D. Nute, Defeasible reasoning, in: J.H. Fetzer (Ed.), Aspects of Artificial Intelligence, Kluwer Academic Publishers, Dordrecht, 1987, pp. 251–288.
[30] C. Perelman, Justice, Law and Argument, Reidel, Dordrecht, 1980.
[31] A. Pettorossi, M. Proietti, Transformation of logic programs: Foundations and techniques, J. Logic Programming 19–20 (1994) 261–320.
[32] J. Pollock, Defeasible reasoning, Cognitive Science 11 (4) (1987) 481–518.
[33] D. Poole, A logical framework for default reasoning, Artificial Intelligence 36 (1) (1988) 27–47.
[34] H. Prakken, G. Sartor, The role of logic in computational models of legal argument: A critical survey, in: A.C. Kakas, F. Sadri (Eds.), Computational Logic: Logic Programming and Beyond—Essays in Honour of Robert A. Kowalski, in: Lecture Notes Artif. Intell., vol. 2408, Springer-Verlag, Berlin, 2002, pp. 342–381.
[35] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980).
[36] F. Toni, A.C. Kakas, Computing the acceptability semantics, in: V.W. Marek, A. Nerode, M. Truszczynski (Eds.), Proc. 3rd International Workshop on Logic Programming and Non-monotonic Reasoning, in: Lecture Notes Artif. Intell., vol. 928, Springer-Verlag, Berlin, 1995, pp. 401–415.
[37] S.E. Toulmin, The Uses of Arguments, Cambridge University Press, Cambridge, 1958.
[38] G. Vreeswijk, H. Prakken, Credulous and sceptical argument games for preferred semantics, in: Proc. JELIA, in: Lecture Notes Comput. Sci., vol. 1919, Springer-Verlag, Berlin, 2000, pp. 224–238.
[39] D.N. Walton, What is reasoning? What is an argument?, J. Philosophy 87 (8) (1990) 399–419.