IEEE Intelligent Systems
www.computer.org/intelligent

Computing Arguments and Attacks in Assumption-Based Argumentation

Dorian Gaertner and Francesca Toni

Vol. 22, No. 6
November/December 2007

Computing Arguments and Attacks in Assumption-Based Argumentation

Dorian Gaertner and Francesca Toni, *Imperial College London*

This computational framework renders explicit arguments and attacks while reducing the problem of computing arguments for and against claims to the problem of computing assumptions.

Most computational frameworks for argumentation are based on *abstract argumentation*, which determines an argument's acceptability on the basis of its ability to counterattack all arguments attacking it.[1] However, this view of argumentation doesn't address how to find arguments, identify attacks, and exploit premises shared by different arguments.

Assumption-based argumentation addresses these three issues.[2–4] It's a refinement of abstract argumentation but remains general purpose, nonetheless. Rather than considering arguments to be a primitive concept, assumption-based argumentation defines them as *backward deductions* (using sets of rules in an underlying logic) supported by sets of *assumptions*. This approach reduces the notion of an attack against an argument to that of deduction of a *contrary* of an assumption. (We describe assumptions and contraries in more detail later.)

Computational models for assumption-based argumentation[3,4] let us determine the acceptability of claims by building and exploring a dialectical structure of

- a *proponent*'s arguments in favor of the claims,
- an *opponent*'s counterarguments against these arguments,
- the proponent's arguments against the counterarguments,

and so on. In more detail, the opponent disputes the proponent's arguments by attacking any of the supporting assumptions of an argument. In turn, the proponent defends its arguments by defeating the opponent's counterarguments by means of further arguments, possibly with the aid of defending assumptions. This computation style has several advantages over other computational mechanisms for argumentation, owing mostly to the low-level granularity afforded by

- interleaving the construction of arguments and the identification of their supporting assumptions and
- exploring the dialectical structure of arguments and counterarguments.

In particular, it avoids recomputation by filtering out assumptions that have already been defended or defeated.

However, this computation style obscures the dialectical structure of proponent-opponent arguments and counterarguments. For example, a proponent and opponent can attack one another before arguments are fully built, by starting to build an argument for the contrary of any assumption identified in a partially constructed argument by the other player. Also, a proponent's arguments are all "mixed" within a single set, in the sense that this set holds assumptions supporting

1541-1672/07/$25.00 © 2007 IEEE
Published by the IEEE Computer Society

all the proponent's arguments. Moreover, the link between assumptions and the conclusions they support (and thus the argument they form) is lost in the computation. Finally, an opponent's arguments might be only partially constructed at the end of the computation, because they might have been defeated (and thus discarded) as soon as a "culprit" assumption was identified in them that the proponent could defeat.

We've developed and implemented an argumentation system combining the virtues of assumption-based argumentation and abstract argumentation and avoiding the problems mentioned earlier. As in assumption-based argumentation, we manipulate assumptions and employ filtering to promote efficiency. As in abstract argumentation, we compute, for any given claim, the dialectical structure of arguments and counterarguments leading to the claim's acceptability. This system extends our earlier CaSAPI (Credulous and Skeptical Argumentation: Prolog Implementation) system.[5] The original CaSAPI only lets us manipulate assumptions. However, it lets us compute notions of acceptability ranging from the credulous notion of admissibility considered in this article (details appear later) to "skeptical" notions, disregarding conflicting alternatives to a greater or lesser extent.

Assumption-based argumentation

Assumption-based argumentation frameworks are tuples $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$, where

- $(\mathcal{L}, \mathcal{R})$ is a deductive system with a language $\mathcal{L}$ and a set of rules $\mathcal{R}$,
- $\mathcal{A} \subseteq \mathcal{L}$ is a nonempty set of assumptions, and
- $\bar{}$ is a total mapping from $\mathcal{A}$ to the power set $\wp(\mathcal{L}) - \{\{\}\}$, where, for any $\alpha \in \mathcal{A}$, $\bar{\alpha}$ is the nonempty set of contraries of α.

(In the original framework, $\bar{}$ is a total mapping from $\mathcal{A}$ into $\mathcal{L}$; each assumption has a single contrary.[2,3] The extension we adopt here is useful, for example, to support decision making in agents.[5,6])

Rules can be domain specific or domain independent.[2] They can correspond to causal information, argument schemes,[7] inference rules and axioms in a chosen logic-based language,[2] or laws and regulations.[8] Assumptions are sentences that can be questioned and disputed (as opposed to axioms, which are beyond dispute)—for example, uncertain beliefs ("it will rain"), unsupported beliefs ("I believe X"), or decisions ("law such-and-such applies" or "perform action A"). A contrary, in general, is a reason the assumption might be undermined and thus might need to be dropped. For example, a contrary of the assumption "it will rain" might be "it will be sunny" and "it will snow," or "¬ it will rain," depending on the underlying $\mathcal{L}$ (¬ is the negation symbol). A contrary of the assumption "law such-and-such applies" might be "law such-and-such is overruled by another law taking precedence." A contrary of the assumption "perform action A" might be "perform action B" and "perform action C," where A, B, and C are mutually exclusive. Finally, a contrary of the assumption "I believe X" might be "X is not the case" or "there is evidence against X."

> We've developed and implemented an argumentation system combining the virtues of assumption-based argumentation and abstract argumentation.

The mapping $\bar{}$ need not be symmetric. For example, "X is not the case" might be a contrary of the assumption "I believe X," but the latter might not be an assumption. Moreover, an agent's preference for action B over action A might force the assumption "perform action B" to be a contrary of the assumption "perform action A," but not the other way around.

We assume that the rules in $\mathcal{R}$ have the syntax $c_0 \leftarrow c_1, \ldots, c_n$ with $n > 0$ or c_0, where $c_i \in \mathcal{L}$ for $i = 0, \ldots, n$. The left part, c_0, is the rule's *head*; $c_1, \ldots, c_n$ (if present) is its *body*. We consider the body of a rule c_0 to be empty; in this case, we often refer to c_0 as a *fact*. For brevity of presentation, we focus here on abstract, symbolic examples. We restrict our attention to *flat assumption-based frameworks*,[3,4] such that if $c \in \mathcal{A}$, no rule in $\mathcal{R}$ has head c. These frameworks are still quite general.[2]

Example 1 shows a simple assumption-based framework:

Example 1. $\mathcal{L} = \{p, a, \neg a, b, \neg b\}$, $\mathcal{R} = \{p \leftarrow a; \neg a \leftarrow b; \neg b \leftarrow a\}$, $\mathcal{A} = \{a, b\}$, *and* $\bar{a} = \{\neg a\}$, $\bar{b} = \{\neg b\}$.

The choice of all elements of an assumption-based framework depends on an application's knowledge-representation needs. For instance, in example 1, we could have $\mathcal{L}$ also include $\neg p$ and have $\mathcal{A}$ not include a. In general, if ¬ occurs at all in $\mathcal{L}$, its role will be purely syntactic (and $\mathcal{L}$ might not be closed under negation, as in the example). Another representation of the framework in the example without using negation might be $\mathcal{L} = \{p, a, na, b, nb\}$, $\mathcal{R} = \{p \leftarrow a; na \leftarrow b; nb \leftarrow a\}$, $\mathcal{A} = \{a, b\}$, and $\bar{a} = \{na\}$, $\bar{b} = \{nb\}$.

An *argument* in favor of a sentence x in $\mathcal{L}$ supported by a set of assumptions X is a (*backward*) *deduction* from x to X, via the backward application of rules in $\mathcal{R}$ (with respect to the direction ←, from the rule's head to its body). In example 1, we can obtain an argument in favor of p supported by $\{a\}$ by applying $p \leftarrow a$ backwards. Also, given $q \leftarrow b, p$, we can obtain an argument in favor of q supported by $\{b, a\}$ by first applying $q \leftarrow b, p$ backwards and then applying $p \leftarrow a$.

From now on, whenever it's clear from the context (and with an abuse of notation), we'll represent an argument in favor of x with support X as $X \vdash x$, and we'll refer to x as the *conclusion* of the argument $X \vdash x$. This notation equates an argument with the pair consisting of its supporting assumptions and its conclusion. It ignores the deduction that links the two and, in particular, the rules used to generate the argument. It also ignores that we can arrive at the same $X \vdash x$ relationship in more than one way. However, the set of assumptions supporting an argument together with the argument's conclusion encapsulate the argument's essence. That is, the only way to attack an argument is to attack one of its assumptions by constructing an argument in favor of a contrary of that assumption:[3,4]

> An argument $X \vdash x$ attacks an argument $Y \vdash y$ if and only if x is a contrary of some assumption in Y.

In example 1, $\{b\} \vdash \neg a$ attacks $\{a\} \vdash p$ because $\bar{a} = \{\neg a\}$.

Assumption-based argumentation lets us determine whether a rational reasoner would accept a given claim. The claim could be, for example, a potential belief of the reasoner or a goal that an agent might hold or have acquired. We represent the claim simply as

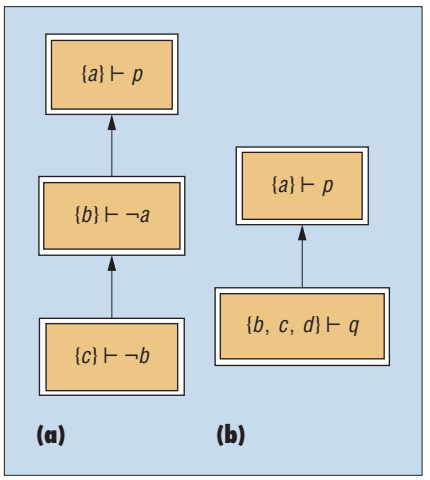


Figure 1. Dialectical structures for (a) example 2 ("core" and defending assumptions) and (b) example 3 (a potential argument and attack).

Table 1. An AB-dispute derivation for example 2. (AB stands for admissible beliefs.)

i	$\mathcal{P}_i$	$\mathcal{O}_i$	A_i	C_i	Comment
0	{p}	{}	{}	{}	Start
1	{a}	{}	{a}	{}	By using rule $p \leftarrow a$
2	{}	{{¬a}}	{a}	{}	Because $\overline{a}$ = {¬a}
3	{}	{{b}}	{a}	{}	By using rule $\neg a \leftarrow b$
4	{¬b}	{}	{a}	{b}	Because $\overline{b}$ = {¬b}
5	{c}	{}	{a, c}	{b}	By using rule $\neg b \leftarrow c$
6	{}	{{¬c}}	{a, c}	{b}	Because $\overline{c}$ = {¬c}
7	{}	{}	{a, c}	{b}	Because there is no rule with head ¬c

a sentence in $\mathcal{L}$. To determine the claim's acceptability, we need to identify a set of assumptions that's consistent and includes a core support (support for the initial claim) as well as assumptions that defend that support. Example 2 illustrates this.

Example 2. *Let* $\mathcal{L} = \{p, a, \neg a, b, \neg b, c, \neg c\}$, $\mathcal{R} = \{p \leftarrow a; \neg a \leftarrow b; \neg b \leftarrow c\}$, $\mathcal{A} = \{a, b, c\}$, *and* $\overline{a} = \{\neg a\}$, $\overline{b} = \{\neg b\}$, $\overline{c} = \{\neg c\}$. *To prove claim p, we need an acceptable support* $\{a, c\}$, *with a being the core (because* $\{a\}$ *supports an argument in favor of p) and c being the assumption that lets us defend argument* $\{a\} \vdash p$ *against the attack* $\{b\} \vdash \neg a$. *Indeed, c supports the argument* $\{c\} \vdash \neg b$ *against* $\{b\} \vdash \neg a$ *by attacking the culprit b.*

Figure 1a illustrates this example.

We can formalize this informal definition of acceptability in many ways, using a notion of attack among sets of assumptions whereby

> a set of assumptions X attacks a set of assumptions Y if and only if for some $y \in Y$ and some $x \in \overline{y}$, there's an argument in favor of x supported by (a subset of) X.

In example 2, $\{b\}$ attacks $\{a\}$.

A single attack between sets of assumptions stands for potentially many attacks between arguments. If X attacks Y as we described, then some argument supported by some subset of X attacks every argument supported by any subset of Y containing y. For example, given the rules $p \leftarrow a$, $q \leftarrow a$, and $r \leftarrow b$ and assumptions a and b, with r a contrary of a, $\{b\}$ attacks $\{a\}$, and $\{b\} \vdash r$ attacks both $\{a\} \vdash p$ and $\{a\} \vdash q$.

We use the following formalization of the notion of acceptable sets of assumptions:

> A set of assumptions is admissible if and only if it doesn't attack itself and it counterattacks every set of assumptions attacking it.

From that notion, we derive the notion of acceptable sets of arguments, in the case of admissibility:

> A set of arguments is admissible if and only if the union of all sets of assumptions supporting the arguments is admissible.

The computational model

To compute admissible sets of assumptions supporting an acceptable claim, we can use AB-dispute derivations.[4] (AB stands for admissible beliefs. AB-dispute derivations are a slight modification of the dispute derivations of Phan Minh Dung, Robert Kowalski, and Francesca Toni.[3] Dung and his colleagues also propose GB- and IB-dispute derivations, for computing skeptically grounded and ideal sets of assumptions.[4] These are ignored in this article.) AB-dispute derivations take a claim as input and then output an admissible set of assumptions. These derivations represent a kind of game, in the form of a dispute between two fictional players: a claim's proponent and an opponent trying to undermine the claim.

The derivations are finite sequences of tuples $\langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i \rangle$. $\mathcal{P}_i$ and $\mathcal{O}_i$ represent the set of sentences held by the proponent and opponent at step i in the dispute. A_i is the set of assumptions the proponent generates to support the initial claim and to defend itself against the opponent. C_i is the set of assumptions in the opponent's counterarguments that the proponent has chosen as culprits to counterattack. The tuple at the start of a derivation for claim x is

$$\langle \{x\}, \{\}, \mathcal{A} \cap \{x\}, \{\} \rangle$$

Table 1 shows an AB-dispute derivation computing an admissible support $\{a, c\}$ for p, for example 2.

Using a *selection function*, AB-dispute derivations pick a sentence from the support of a potential argument to expand further (if it isn't an assumption) or to identify a possible point of attack (if it is). Such selection can be based on random choice, on whether the sentence is an assumption, or on more complex criteria.

Here, we define *structured AB-dispute derivations*, an extension of AB-dispute derivations that computes

- an admissible set of assumptions in support of an initial claim or sentence,
- the set of underlying arguments, implicitly built by the proponent and opponent, and
- the attack relationship among those arguments.

For example, this amounts to computing the derivation in table 1 and the structure in figure 1a. This computation relies on a *patient* selection function (which we explain later). The set of computed arguments is guaranteed to be admissible, because the underlying set of assumptions is admissible. (This is due to AB-dispute derivations being correct[6,7] and to the fact that structured AB-dispute derivations compute the same set of assumptions as ordinary AB-dispute derivations.)

Structured AB-dispute derivations are also sequences of tuples, but of this form:

$$\langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i, Arg_i, Rel_i \rangle$$

A_i and C_i, the defense set and the set of culprits, are exactly as in the original AB-dispute derivations. Again, $\mathcal{P}_i$ and $\mathcal{O}_i$, represent the

proponent's and opponent's state, but they're no longer sets of sentences and sets of sets of sentences, respectively. Instead, they consist of *potential arguments* and information about which arguments they *potentially attack*. Potential arguments are deductions whose support contains nonassumptions—namely, elements of $\mathcal{L} - \mathcal{A}$. There might be no deductions for these nonassumptions in the support of potential arguments. *Actual arguments* have sets of assumptions as their support. Potential attacks are attacks by potential arguments.

The two new elements, Arg_i and Rel_i, hold, respectively, the currently computed *actual arguments* and a binary attack relation between these arguments. With these new elements, the argumentation system's user can explicitly see why a certain argument was made and which other arguments it attacks and defends.

Example 3 illustrates the notions of actual and potential arguments and attacks:

EXAMPLE 3. *Consider a simple assumption-based framework* $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ *where* $\mathcal{A} = \{a, b, c, d\}$; $\mathcal{R}$ *contains two rules,* $p \leftarrow a$ *and* $q \leftarrow r, b, c$*; and the contrary of assumption* a *is* q*. The actual argument* $\{a\} \vdash p$ *supports claim* p*. The opponent can attack this argument only by supporting the contrary of some sentence in its premise. So, the opponent must find an argument supporting* q*, the contrary of* a*. Using the second rule, we can build the potential argument* $\{r, b, c\} \vdash q$*. Any actual argument obtained by a backward deduction from the nonassumptions in the support of this potential argument is an actual argument attacking* $\{a\} \vdash p$*. Indeed, imagine that we add a third rule,* $r \leftarrow d$*, to* $\mathcal{R}$*. We can use this rule to turn the potential argument* $\{r, b, c\} \vdash q$ *into the actual argument* $\{b, c, d\} \vdash q$ *attacking* $\{a\} \vdash p$*. Figure 1b shows the attack relationship between these two arguments. However, if the third rule had the form* $r \leftarrow s$*, the potential argument would have evolved to another potential argument* $\{s, b, c\} \vdash q$*. Because no rule in* $\mathcal{R}$ *can expand the sentence* s *and, furthermore,* $s \notin \mathcal{A}$*, the argument cannot be completed. So, we can find no actual attack against* $\{a\} \vdash p$*.*

Concretely, in a structured AB-dispute derivation, Arg_i consists of expressions of the form

$$id : (X \vdash x)$$

representing an actual argument labeled *id* and supported by X with conclusion x. The argument labels are needed to express the interrelationships between arguments. Indeed, Rel_i is a set of expressions of the form

$$id \rightsquigarrow id^*$$

standing for "the actual argument labeled by *id* actually attacks the actual argument labeled by *id**." For example, we can represent the arguments and attack relationship in figure 1b as $id_1 : \{a\} \vdash p$, $id_2 : \{b, c, d\} \vdash q$ (in Arg_i) and $id_2 \rightsquigarrow id_1$ (in Rel_i).

Finally, $\mathcal{P}_i$ and $\mathcal{O}_i$ are sets of expressions of the form

$$(X \vdash x) \rightsquigarrow id$$

> AB-dispute derivations take a claim as input and then output an admissible set of assumptions. These derivations represent a kind of game.

indicating a potential or actual argument $X \vdash x$ potentially or actually attacking another argument labeled *id*. In example 3, at some stage in the derivation, $\mathcal{O}_i$ might contain $(\{r, b, c\} \vdash q) \rightsquigarrow id_1$ to represent the potential argument $(\{r, b, c\} \vdash q)$ potentially attacking the argument labeled by id_1 (in this case, $\{a\} \vdash p$).

The algorithm

To guarantee the construction of actual arguments, structured AB-dispute derivations assume that the selection function is patient,[3] in that it selects nonassumptions whenever it can. We use this formal definition of a patient selection function:

DEFINITION 1. *A selection function* f *is patient if and only if for any* $X \subseteq \mathcal{L}$*, if* $X - \mathcal{A} \neq \{\}$*, then* $f(X) \in X - \mathcal{A}$*.*

To label arguments, we assume a function *newLabel*() that returns new identifiers every time it's called. In line with the original AB-dispute derivations, our derivations aim to find an admissible support for a given sentence (claim) α. We represent this here starting with $\mathcal{P}_0$ consisting of $(\{\alpha\} \vdash \alpha) \rightsquigarrow \phi$, which states that to search for an argument supporting the initial claim α, we should try to prove α. This argument won't attack any other argument, which we indicate by ϕ.

Figure 2 shows the formal definition of the structured AB-dispute derivation algorithm; figure 3 provides a high-level procedural view of one pass through the algorithm. In figure 3, each rectangle represents some action or actions, and each oval is a choice point. The algorithm makes some of these choices. Others are implicit parameters of the algorithm that any implementation needs to specify, as we discuss later. We use the passive voice in figure 2 for all the choices of this second type. In figure 3, the first choice is which player should act next and which of its potential arguments to deal with. The left subtree covers the proponent; the right subtree covers the opponent. The branches are labeled with the outcomes of the respective choice and the number of the corresponding case in the algorithm (in square brackets). The leaves focus on the Arg_i element. The tree represents only the transition from the ith to the $(i + 1)$th tuple.

Informally, when the implementation of the algorithm chooses a proponent's potential argument and that argument's premises contain no more nonassumptions (case 1(i)), the argument is an actual one and can be added to Arg_i. Otherwise (case 1(ii)), the algorithm selects a nonassumption and applies a step of backward deduction to it with some rule in $\mathcal{R}$. (Backward deduction here aims to turn the potential argument into an actual one.) In this case, if the resulting argument is an actual one with empty support, the algorithm adds this to Arg_i. (This would occur, for example, if the argument was $\{p\} \vdash q$ and the algorithm chose a fact $p \in \mathcal{R}$, thus giving $\{\} \vdash q$.)

If, instead, the implementation of the algorithm chooses one of the opponent's potential arguments (case 2), the algorithm selects a premise in the argument. If this premise isn't an assumption (case 2(ii)), the algorithm replaces the argument with all potential arguments obtained by the application of all applicable rules in $\mathcal{R}$ (that is, those with the premise as their head). This action aims to turn the potential argument into all possible actual ones. If the selected premise is an assumption (case 2(i)), the

DEFINITION 2. *Let* $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ *be an assumption-based argumentation framework. Given a patient selection function, a structured AB-dispute derivation of a defense set A and of a dialectical structure* (*Arg*, *Rel*) *for a sentence* α *is a finite sequence of tuples*

$\langle \mathcal{P}_0, \mathcal{O}_0, A_0, C_0, Arg_0, Rel_0 \rangle$,
...
$\langle \mathcal{P}_i, \mathcal{O}_i, A_i, C_i, Arg_i, Rel_i \rangle$,
...
$\langle \mathcal{P}_n, \mathcal{O}_n, A_n, C_n, Arg_n, Rel_n \rangle$

Initially,

$\mathcal{P}_0 = \{(\{\alpha\} \vdash \alpha) \rightsquigarrow \phi\}$
$A_0 = \mathcal{A} \cap \{\alpha\}$
$\mathcal{O}_0 = C_0 = Arg_0 = Rel_0 = \{\}$

Upon termination,

$\mathcal{P}_n = \mathcal{O}_n = \{\}$
$A = A_n$
$Arg = Arg_n$
$Rel = Rel_n$

For every $0 \le i < n$, *only one potential argument* $(S \vdash c) \rightsquigarrow id$ *in* $\mathcal{P}_i$ *or in* $\mathcal{O}_i$ *is chosen, and*

1. *If* $(S \vdash c) \rightsquigarrow id$ *is chosen from* $\mathcal{P}_i$, *then*
   (i) *If the selection function picks an assumption in S (this means that* $S \subseteq \mathcal{A}$, *because the selection function is patient), then*

$$\mathcal{P}_{i+1} = \mathcal{P}_i - \{(S \vdash c) \rightsquigarrow id\}$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i \cup \{(\{x\} \vdash x) \rightsquigarrow new \mid x \in \bar{\sigma} \text{ and } \sigma \in S - A_i \text{ and } new = newLabel()\}$$
$$A_{i+1} = A_i \cup S$$
$$C_{i+1} = C_i$$
$$Arg_{i+1} = Arg_i \cup \{new : (S \vdash c)\}$$
$$Rel_{i+1} = Rel_i \cup \{new \rightsquigarrow id \mid (\{x\} \vdash x) \rightsquigarrow new \in \mathcal{O}_{i+1} - \mathcal{O}_i\} \cup \{old \rightsquigarrow new \mid old : (Z \vdash z) \in Arg_i \text{ and } z \in \bar{\sigma} \text{ and } \sigma \in S \cap A_i\}$$

   (ii) *If the selection function picks nonassumption* σ *from S and there exists some rule of the form* $\sigma \leftarrow R \in \mathcal{R}$ *such that* $C_i \cap R = \{\}$, *then*

$$\mathcal{P}_{i+1} = \mathcal{P}_i - \{(S \vdash c) \rightsquigarrow id\} \cup \{((S - \{\sigma\}) \cup R \vdash c) \rightsquigarrow id\}$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i$$
$$A_{i+1} = A_i$$
$$C_{i+1} = C_i$$

$$Arg_{i+1} = \begin{cases} Arg_i \cup \{new : (\{\} \vdash c)\} & \text{if } (S - \{\sigma\}) \cup R = \{\} \text{ and } new = newLabel() \\ Arg_i & \text{otherwise} \end{cases}$$

$$Rel_{i+1} = \begin{cases} Rel_i \cup \{new \rightsquigarrow id\} & \text{if } (S - \{\sigma\}) \cup R = \{\} \\ Rel_i & \text{otherwise} \end{cases}$$

Figure 2. The formal definition of structured AB-dispute derivations.

implementation of the algorithm can choose to ignore it (case 2(i)a) or not. In the latter case, the assumption shouldn't already have been chosen as a defense (that is, it shouldn't already be in A_i) so that it can be chosen as a culprit. If that assumption has already been chosen as a culprit (case 2(i)b), the algorithm can use an existing argument in Arg_i to counterattack it. Otherwise (case 2(i)c), the proponent adds a potential argument attacking the culprit to its arguments to be defended.

To support the ignoring step (2(i)a), we assume the following marking strategy. Assumptions supporting potential arguments in $\mathcal{O}_i$ are marked after the selection function has chosen them. All assumptions that support a newly created potential argument in $\mathcal{O}_i$ are unmarked. Then, the selection function always operates on those unmarked assumptions. When a potential argument in $\mathcal{O}_i$ becomes an actual argument and is added to Arg_i, the marking is dropped. The marking is implicit in the algorithm definition in figure 2.

Table 2 shows structured AB-dispute derivations for example 2. The construction of such derivations involves five types of choices that need to be made in any implementation of the algorithm (we'll describe later the concrete choices we made for CaSAPI 3.0):

- *Choice of player*. The proponent and opponent take turns, not necessarily alternating. During the proponent's turn, case 1 of the algorithm executes; during the opponent's turn, case 2 runs.

2. *If* $(S \vdash c) \rightsquigarrow id$ *is chosen in* $\mathcal{O}_i$ *and the selection function picks* σ *in* S*, then*
   (i) *If* σ *is an assumption (this means that* $S \subseteq \mathcal{A}$*, because the selection function is patient), then*

      (a) *Either* σ *is ignored; that is,*

$$\mathcal{P}_{i+1} = \mathcal{P}_i$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i \text{ (The selected } \sigma \text{ will be marked here and not selected again.)}$$
$$A_{i+1} = A_i$$
$$C_{i+1} = C_i$$
$$Arg_{i+1} = Arg_i$$
$$Rel_{i+1} = Rel_i$$

      (b) *Or* $\sigma \notin A_i$ *and* $\sigma \in C_i$*. In this case, let counter* $: (S' \vdash \omega) \in Arg_i$ *such that* $\omega \in \overline{\sigma}$*. (This argument is guaranteed to exist because* $\sigma \in C_i$*.) Then, given new = newLabel():*

$$\mathcal{P}_{i+1} = \mathcal{P}_i$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i - \{(S \vdash c) \rightsquigarrow id\}$$
$$A_{i+1} = A_i$$
$$C_{i+1} = C_i$$
$$Arg_{i+1} = Arg_i \cup \{new : (S \vdash c)\}$$
$$Rel_{i+1} = Rel_i \cup \{new \rightsquigarrow id\} \cup \{counter \rightsquigarrow new\}$$

      (c) *Or* $\sigma \notin A_i$ *and* $\sigma \notin C_i$ *and, choosing some* $x \in \overline{\sigma}$ *and new = newLabel()*:

$$\mathcal{P}_{i+1} = \mathcal{P}_i \cup \{(\{x\} \vdash x) \rightsquigarrow new\}$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i - \{(S \vdash c) \rightsquigarrow id\}$$
$$A_{i+1} = A_i$$
$$C_{i+1} = C_i \cup \{\sigma\}$$
$$Arg_{i+1} = Arg_i \cup \{new : (S \vdash c)\}$$
$$Rel_{i+1} = Rel_i \cup \{new \rightsquigarrow id\}$$

   (ii) *If* σ *isn't an assumption, then*

$$\mathcal{P}_{i+1} = \mathcal{P}_i$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i - \{(S \vdash c) \rightsquigarrow id\} \cup \{((S - \{\sigma\}) \cup R \vdash c) \rightsquigarrow id \mid \sigma \leftarrow R \in \mathcal{R} \text{ and } R \cap C_i = \{\}\}$$
$$A_{i+1} = A_i$$
$$C_{i+1} = C_i$$

$$Arg_{i+1} = \begin{cases} Arg_i \cup \{new : (\{\} \vdash c)\} & \text{if } (S - \{\sigma\}) \cup R = \{\} \text{ and } new = newLabel() \\ Arg_i & \text{otherwise} \end{cases}$$

$Rel_{i+1} = Rel_i \cup \{old(R) \rightsquigarrow id \mid \sigma \leftarrow R \in \mathcal{R} \text{ and } R \cap C_i \neq \{\}\} \cup X$ *where, for each* R*,* $old(R)$ *is* $(old(R) : (Z \vdash z)) \in Arg_i$*, with* $z \in \overline{\delta}$ *and* $\delta \in R \cap C_i$ *and*

$$X = \begin{cases} \{new \rightsquigarrow id\} & \text{if } (S - \{\sigma\}) \cup R = \{\} \\ \{\} & \text{otherwise} \end{cases}$$

- *Choice of argument*. Once the implementation has decided which player will take the next turn, that player picks one potential argument that it currently holds.
- *Selection function*. Once a player has chosen a potential argument, the selection function picks a sentence from that argument's premises. Depending on whether this sentence is an assumption, different parts of the algorithm execute.
- *Choice of rule*. If the proponent is selected and the selection function returns an element that isn't an assumption, the proponent expands the element by applying an appropriate rule in $\mathcal{R}$. If multiple rules are applicable, the proponent selects one of them. The choice of rule applies only to the proponent. If the opponent is expanding a nonassumption element, the opponent uses all applicable rules to create as many potential attacks as possible.
- *Choice to ignore*. If the opponent is selected and the selection function returns an element that's an assumption, the proponent can either counterattack this element (choosing it as the culprit in the argument) or ignore it to select another element during the proponent's next turn.

Ordinary AB-dispute derivations require all these choices except the choice of argument, because arguments aren't explicit in AB-dispute derivations. The choice to ignore is necessary for completeness only as in the case of ordinary AB-dispute derivations.[3]

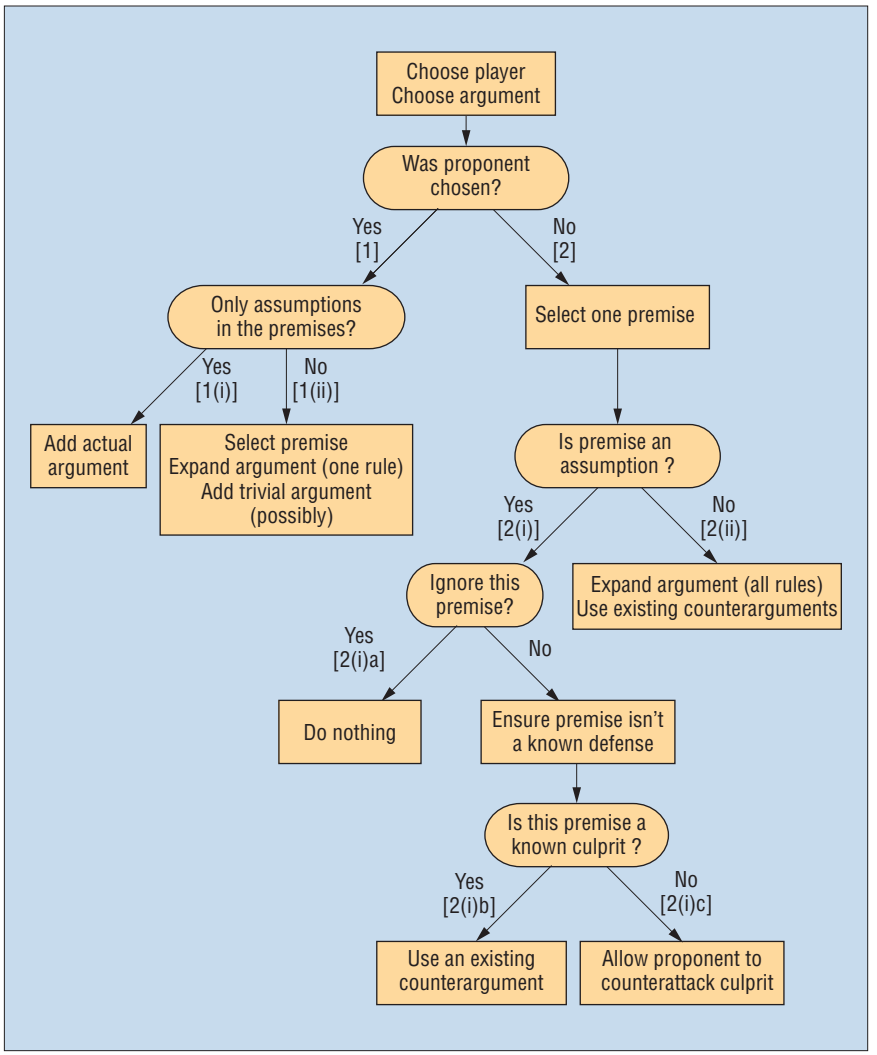


Figure 3. A high-level decision tree representation of one pass through the structured AB-dispute derivation algorithm (see figure 2). A premise here is an element of the support of the chosen argument. The numbers in square brackets indicate the corresponding case in the algorithm.

CaSAPI 3.0

We developed CaSAPI 3.0 to implement structured AB-dispute derivations.

How to use CaSAPI

After invoking a Prolog process and loading the CaSAPI program, users must load the input assumption-based framework and the claim to be proved. These are best specified in a separate file, before invoking Prolog. For instance, we can express example 1, using `not( )` to represent ¬, as

```
myRule(p,[a]).
myRule(not(a),[b]).
myRule(not(b),[a]).
myAsm([a,b]).
contrary(a,not(a)).
contrary(b,not(b)).
toBeProved([p]).
```

The users can then control the amount of output to the screen: either silent (`s`), displaying only the output of derivations, or noisy (`n`), displaying the full derivations. They can also control the number of supports computed: either one (`1`) or all that can be computed (`all`). Users specify these choices as arguments to the command `run/3`, upon which CaSAPI will begin the argumentation process in a manner dictated by the users' choices. The first argument is always `ab` but is expressed as a parameter to allow the future integration of other forms of dispute derivations.

For example, to run structured AB-dispute derivations in silent mode for only one computed support, the user specifies `run(ab,s,1)`. Figure 4 compares the CaSAPI 3.0 output with the CaSAPI 2.0 output for example 1, for a noisy run for one computed support

Table 2. A structured AB-dispute derivation for example 2.

i	$\mathcal{P}_i$	$\mathcal{O}_i$	A_i	C_i	Arg_i	Rel_i	Case
0	$\{(\{p\} \vdash p) \rightsquigarrow \phi\}$	{}	{}	{}	{}	{}	Start
1	$\{(\{a\} \vdash p) \rightsquigarrow \phi\}$	—*	—	—	—	—	1(ii)
2	{}	$\{(\{\neg a\} \vdash \neg a) \rightsquigarrow id_1\}$	$\{a\}$	—	$\{id_1 : \{a\} \vdash p\}$	$\{id_1 \rightsquigarrow \phi\}$	1(i)
3	—	$\{(\{b\} \vdash \neg a) \rightsquigarrow id_1\}$	—	—	—	—	2(ii)
4	$\{(\{\neg b\} \vdash \neg b) \rightsquigarrow id_2\}$	{}	—	$\{b\}$	$\{id_1 : \{a\} \vdash p,$ $id_2 : \{b\} \vdash \neg a\}$	$\{id_1 \rightsquigarrow \phi,$ $id_2 \rightsquigarrow id_1\}$	2(i)c
5	$\{(\{c\} \vdash \neg b) \rightsquigarrow id_2\}$	—	—	—	—	—	1(ii)
6	{}	$\{(\{\neg c\} \vdash \neg c) \rightsquigarrow id_3\}$	$\{a, c\}$	—	$\{id_1 : \{a\} \vdash p,$ $id_2 : \{b\} \vdash \neg a,$ $id_3 : \{c\} \vdash \neg b\}$	$\{id_1 \rightsquigarrow \phi,$ $id_2 \rightsquigarrow id_1,$ $id_3 \rightsquigarrow id_2\}$	1(i)
7	—	{}	—	—	—	—	2(ii)

* "—" means unchanged from the previous step.

(run(ab,n,1)). (CaSAPI 2.0 implements ordinary AB-dispute derivations, as well as GB- and IB-dispute derivations.) Clearly, the two systems mirror one another as far as their common components are concerned. Both systems, when run in noisy mode, indicate explicitly the appropriate cases of the algorithm being applied, for debugging purposes. Step i corresponds to the *i*th tuple in the derivations being computed.

CaSAPI 3.0 uses a concrete representation of the potential and actual arguments and attacks. For example, at step 0, infoTerm(p,localGoal(p),attacking(nothing)) represents $(\{p\} \vdash p) \rightsquigarrow \phi$. This representation stands for "we are trying to show p to find an argument supporting the local goal p. If we succeed, this argument will attack the argument labeled with nothing." At step 2, argument(1,[a],p) represents the actual argument $id_1 : \{a\} \vdash p$. At step 4, attacks(2,1) represents that argument 2 is actually attacking argument 1.

Design choices

We picked Sicstus Prolog as the implementation language because we intend to employ some of its constraint-solving features in future versions of CaSAPI. Version 3.0 doesn't use any Sicstus-specific code, so it runs on most standard Prolog engines, subject to minor modifications. A further design decision, which we hope will allow for interesting experimental research, is that most concrete choices for the choice points we described earlier aren't hardwired into CaSAPI. Instead, we've implemented them through modular, easily replaceable pieces of code.

For example, for the choice of player, the default behavior favors the proponent in that it selects the proponent whenever the proponent has a potential argument to make. We could easily adjust this function to favor the opponent or to alternate between players.

For the choice of rule, the default behavior chooses the rules in the order in which they appear in the input file. However, we can modify the Prolog predicate that handles rule choice to, for instance, prefer the rule with the fewest elements or assumptions in the support. The final result remains unchanged, but intelligent rule selection might speed up the derivation in some cases.

The choice of argument can pick one argument randomly or treat all the arguments sequentially. The behavior we've implemented is the preference for arguments that still have nonassumptions. We do this in the spirit of the patient selection function, which also favors nonassumptions.

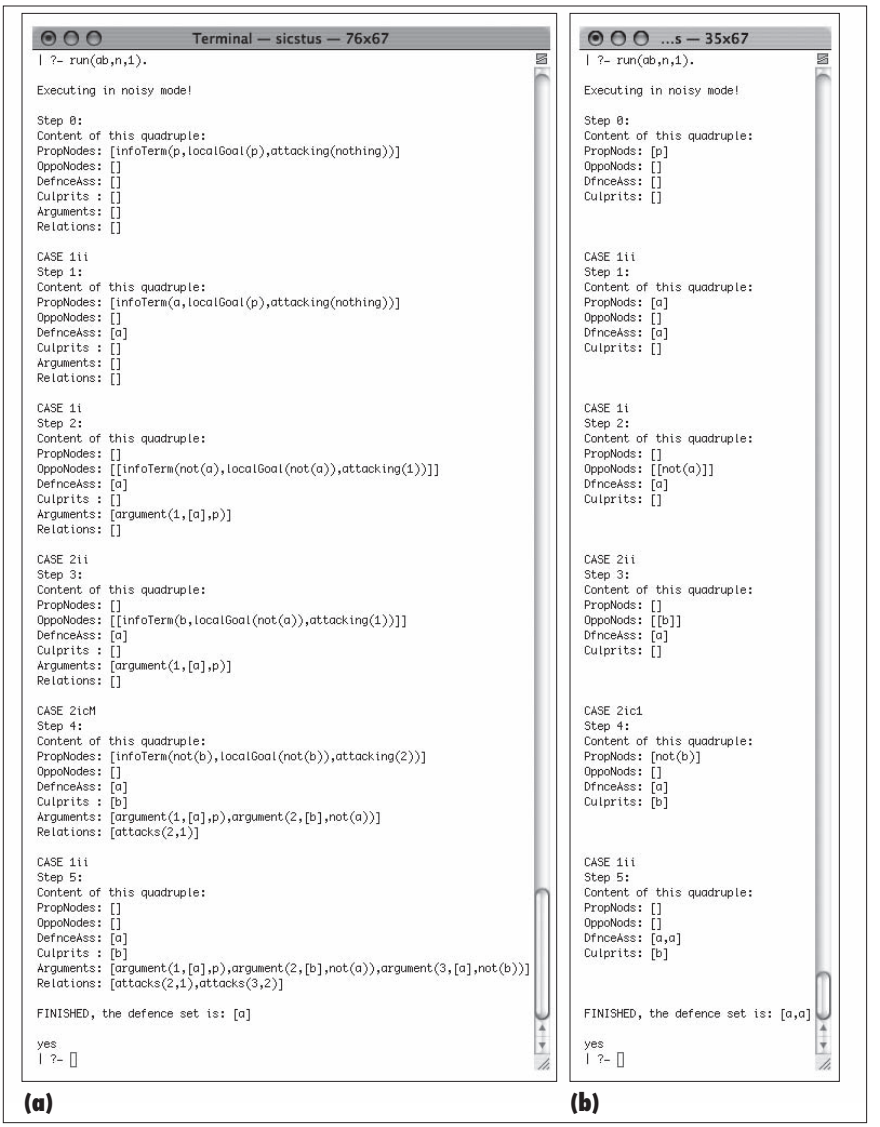


Figure 4. Comparing the (a) CaSAPI (Credulous and Skeptical Argumentation: Prolog Implementation) 3.0 output to the (b) CaSAPI 2.0 output, for example 1.

Applying a real-world example

We provide here a simple but realistic example of our system's functionalities, demonstrating its real-world applicability while showing the link to the algorithm's formal description in figure 2. The example simulates the reasoning of an impartial referee (our system) who must resolve a disagreement between a software house and a customer refusing to pay for a product that the software house developed for her. The referee uses information that both parties agree about and tries to find support for the software house's claim that the customer should pay it. So, within the dispute derivation conducted by the referee, the software house is the proponent and the customer, who is trying to dispute the claim for payment, is the opponent.

Both parties agree on the following general information: An incomplete or untimely job doesn't constitute a good job, and a delivered product that doesn't conform to the specification (consisting of requirements A and B) constitutes an incomplete job. The client and the software house are also both aware of an indisputable fact: the product was indeed delivered. Figure 5 shows this information as input to CaSAPI.

After feeding the input program into

```
myRule(payment,[goodJob]).
myRule(badJob,[tooLateJob]).
myRule(badJob,[incompleteJob]).
myRule(incompleteJob,[delivered,not(accordingToSpec)]).
myRule(accordingToSpec,[reqA,reqB]).
myRule(delivered,[]).

myAsm([goodJob,not(accordingToSpec),reqA,reqB]).

contrary(goodJob,badJob).
contrary(not(accordingToSpec),accordingToSpec).
contrary(reqA,not(reqA)).
contrary(reqB,not(reqB)).

toBeProved([payment]).
```

Figure 5. Input to CaSAPI indicating general information on which the proponent (a software house) and opponent (a customer) agree.

```
Step 3:
- PropNodes:   []
- OppoNodes:   [infoTerm(tooLateJob,localGoal(badJob),attacking(1)),
                infoTerm(incompleteJob,localGoal(badJob),attacking(1))]
- DefnceAss:   [goodJob]
- Culprits :   []
- Arguments:   [argument(1,[goodJob],payment)]
- Relations:   []
```

Figure 6. The execution's state at step 3.

```
Arguments:  [argument(1,[goodJob],payment),
             argument(2,[not(accordingToSpec)],badJob),
             argument(3,[reqA,reqB],accordingToSpec)]

Relations:  [attacks(2,1),attacks(3,2)]

FINISHED, the defense set is: [goodJob,reqA,reqB]
```

Figure 7. The arguments and attacks that CaSAPI 3.0 computed for a software house's claim for payment from a customer.

CaSAPI, the user must choose the execution options. Assume the user chooses **run(ab,n,1)**. In the first step, case 1(ii) applies and the choice of rule selects the only viable rule, with the head **payment**. So, the software house now must show **goodJob** to show the local goal **payment**. But **goodJob** is an assumption, so the algorithm adds it to the defense set.

Now, the choice of player selects the customer, who must show **badJob** to disprove the software house's assumption in the defense set. The customer can do this in two ways (via the second or third rule): by proving the job was either too late or incomplete. The customer places both these potential arguments on her agenda, even though just one would be enough for the software house's argument to fail. Figure 6 shows the execution's state at step 3.

This first attempt to prove **badJob** fails because the sentence **tooLateJob** can neither be proved nor assumed; it isn't one of the possible assumptions. The second attempt expands **incompleteJob** using the fourth rule, to leave the customer with two things to prove: **delivered** and **not(accordingToSpec)**. The former holds because both players share this belief, and the latter is an assumption.

So, if the customer can assume **not(accordingToSpec)**, she can prove **badJob**, which attacks the software house's argument for **payment** because it attacks the software house's assumption **goodJob**. Now, it's the software house's turn to defend itself. The software house can counterattack the customer's vital assumption **not(accordingToSpec)** by proving the opposite—that is, **accordingToSpec**. To do this, it needs to show that both requirements A and B are fulfilled. Because both are in the set of assumptions, the software house simply assumes them.

Finally, the customer fails to dispute that requirement A has been achieved, because no information that it has not been achieved exists (she cannot provide any). Furthermore, she cannot prove to the software house that requirement B hasn't been completed. So, the customer cannot dispute the software house's argument that it made the product according to specifications.

After both sides argue their point, the referee finds that the claim **payment** is supported by the assumptions **goodJob,reqA,reqB**. The customer failed on both occasions to prove that the software house made a bad job and in the process failed to disprove the assumptions that both requirements were fulfilled. So, the software house's claim for payment prevails. Figure 7 shows the arguments and attacks that CaSAPI 3.0 computed.

CaSAPI correctly finds a support for the given input, and the final dialectical structure shows that argument 2 attacks the software house's claim for payment. This argument is counterattacked by argument 3, which cannot be attacked. So, argument 3 remains undefeated; because it attacks argument 2, we can deduce that argument 2's attack on argument 1 fails. Arguments 1 and 3 are hence admissible. In this simple application, the dialectical structure that CaSAPI 3.0 computes provides great support; the defense set alone (computed by CaSAPI 2.0) doesn't provide enough information to resolve the dispute.

This system is still a prototype; more experimentation is needed to address issues such as scalability. However, its applicability is promising. For example, Maxime Morge and Paolo Mancarella have developed a decision-support system for agents that employs CaSAPI to make decisions abductively, while taking preferences into account.[6] Furthermore, we have shown CaSAPI's applicability for agent reason-

ing and particularly for normative conflict resolution.[9]

We believe that assumption-based argumentation—and particularly a tool such as the one we presented here—will be important for developers of many different kinds of intelligent systems. One way to aid the system's applicability would be to render it dynamic—for example, by allowing real proponents and opponents to exchange information and arguments during a dispute. This would be useful for the real-world example we described in the article. Moreover, a user-friendly interface will aid the system's adoption.

The research we've presented here opens several lines of future research:

- extend CaSAPI 3.0 to compute the acceptability of claims under other, skeptical semantics (as CaSAPI 2.0 already does),
- research the system's efficiency,
- study the treatment of variables and the selection function's safety, and
- consider nonpatient selection functions.

Other argumentation systems exist. For example, Gorgias handles credulous argumentation in frameworks with preferences among defeasible rules.[10] The ASPIC (Argument Service Platform with Integrated Components) system deals with quantitative uncertainty.[11] The DeLP system handles defeasible-logic programming.[12] Daniel Bryant and Paul Krause have developed a system for agent implementations.[13] These systems are defined for different argumentation frameworks from ours and cannot be directly compared. It would be interesting to provide a mapping from their frameworks onto (possibly extended) assumption-based argumentation to fully compare them. ■

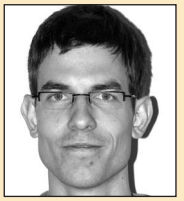
Acknowledgments

The European Commission Sixth Framework Information Society Technologies program partially funded this work, under the 035200 ARGUGRID project. Francesca Toni has also been supported by a UK Royal Academy of Engineering/Leverhulme Trust Senior Research Fellowship. We thank the reviewers of an earlier version of this article for useful suggestions. We also thank Paolo Mancarella and Phan Minh Dung for inspiring discussions about the real-world example we used in this article.

References

1. P.M. Dung, "The Acceptability of Arguments and Its Fundamental Role in Non-monotonic Reasoning and Logic Programming and N-Person Game," *Artificial Intelligence*, vol. 77, no. 2, 1995, pp. 321–357.
2. A. Bondarenko et al., "An Abstract, Argumentation-Theoretic Framework for Default Reasoning," *Artificial Intelligence*, vol. 93, nos. 1–2, 1997, pp. 63–101.
3. P.M. Dung, R.A. Kowalski, and F. Toni, "Dialectic Proof Procedures for Assumption-Based, Admissible Argumentation," *Artificial Intelligence*, vol. 170, no. 2, 2006, pp. 114–159.
4. P.M. Dung, P. Mancarella, and F. Toni, "Computing Ideal Sceptical Argumentation," *Artificial Intelligence*, vol. 171, nos. 10–15, 2007, pp. 642–674.
5. D. Gaertner and F. Toni, "CaSAPI: A System for Credulous and Sceptical Argumentation," *Proc. ArgNMR, LPNMR Workshop Argumentation and Non-monotonic Reasoning*, 2007, pp. 80–95; http://lia.deis.unibo.it/confs/ArgNMR/proceedings/ArgNMR-proceedings.pdf.
6. M. Morge and P. Mancarella, "The Hedgehog and the Fox: An Argumentation-Based Decision Support System," *Proc. 4th Int'l Workshop Argumentation in Multi-Agent Systems* (ArgMAS 07), 2007; http://homepages.inf.ed.ac.uk/irahwan/argmas/argmas07/argmas2007proceedings.pdf.
7. C. Reed and D. Walton, "Towards a Formal and Implemented Model of Argumentation Schemes in Agent Communication," *Proc. 1st Int'l Workshop Argumentation in Multi-Agent Systems* (ArgMAS 04), Springer, 2004, pp. 19–30; http://homepages.inf.ed.ac.uk/irahwan/argmas/argmas04/papers/08.pdf.
8. H. Prakken and G. Sartor, "On the Relation between Legal Language and Legal Argument: Assumptions, Applicability and Dynamic Priorities," *Proc. 5th Int'l Conf. Artificial Intelligence and the Law* (ICAIL 95), ACM Press, 1995, pp. 1–10.
9. D. Gaertner and F. Toni, "Conflict-Free Normative Agents Using Assumption-Based Argumentation," *Proc. 4th Int'l Workshop Argumentation in Multi-Agent Systems* (ArgMAS 07), 2007; http://homepages.inf.ed.ac.uk/irahwan/argmas/argmas07/argmas2007proceedings.pdf.
10. N. Demetriou and A.C. Kakas, "Argumentation with Abduction," *Proc. 4th Panhellenic Symp. Logic*, 2003.
11. M. Caminada et al., "Implementations of Argument-Based Inference," *Rev. of Argumentation Technology*, 2004, pp. 2–13.
12. A. Garcia and G. Simari, "Defeasible Logic Programming: An Argumentative Approach," *J. Theory and Practice of Logic Programming*, vol. 4, nos. 1–2, 2004, pp. 95–138.
13. D. Bryant and P. Krause, "An Implementation of a Lightweight Argumentation Engine for Agent Applications," *Logics in Artificial Intelligence: Proc. 10th European Conf.* (JELIA 06), LNAI 4160, Springer, 2006, pp. 469–472.

The Authors



Dorian Gaertner is a PhD student in the Department of Computing at Imperial College London. His research interests focus on AI, including multiagent-system coordination, normative reasoning, argumentation, and knowledge representation. He received his MEng in computing and AI from Imperial College London, along with the Departmental Prize for Excellence. Contact him at Imperial College London, Dept. of Computing, London, SW72AZ, UK; dg00@doc.ic.ac.uk.

Francesca Toni is a senior lecturer in the Department of Computing at Imperial College London and the coordinator of the EU-funded ARGUGRID project. Her research interests include argumentation, negotiation, logic-based multiagent systems, abduction, logic programming for knowledge representation and automated reasoning, and nonmonotonic and default reasoning. She received her PhD in computing from Imperial College London. Contact her at Imperial College London, Dept. of Computing, London, SW72AZ, UK; ft@doc.ic.ac.uk.