An Argumentation Engine: ArgTrust

Yuqing Tang[1,2], Elizabeth Sklar[2,3], and Simon Parsons[2,3]

[1] Robotics Institute, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213, USA
[2] Dept of Computer Science, Graduate Center, City University of New York
365 Fifth Avenue, New York, NY 10016, USA
[3] Dept of Computer & Information Science, Brooklyn College,
City University of New York, 2900 Bedford Avenue,
Brooklyn, NY 11210 USA
yuqing.tang@cs.cmu.edu,sklar@sci.brooklyn.cuny.edu,
parsons@sci.brooklyn.cuny.edu

Abstract. Argumentation is a well-studied formal model for multiagent interaction that supports exchange of information about agents' beliefs and reasons why agents hold their beliefs. We describe an argumentation engine, called *ArgTrust*, that we have implemented in Java, and explain how the underlying logic formalism is translated into a computational framework.

1 Introduction

A growing number of application areas have been recognized as appropriate contexts for implementing aspects of *argumentation* in order to provide a formal, well-studied and well-structured basis for computer-supported reasoning. The key aspects we focus on include the use of argumentation to prove or disprove the correctness of a claim and to identify a minimal set of reasons (i.e., evidence) for why the claim is true or not true. Further, we address situations in which there may be numeric values (e.g., $n \in [0, 1]$) associated with each piece of evidence that indicate the strength (or weakness) of that element. In this paper, we introduce an argumentation engine, called *ArgTrust*[4], that we designed based on the theoretical framework described in [18] and implemented to perform the generalized task of reasoning about the validity of claims using weighted evidence.

ArgTrust is written in Java, and in addition to the argumentation computation that is described here, can deal with arguments that have numerical strengths attached to them. In our implementation, an underlying logical formalism is decoupled from argument construction and status computation. The formalism enables connectives ($\wedge, \vee, \neg$ and $\rightarrow$) by abstracting them as Java

[4] The name comes from the fact that ArgTrust started out as an implementation of an argumentation system that is used to reason with information that comes from sources that are trusted to different degrees.

interfaces. By using interfaces, different applications with different logical languages could include ArgTrust as a component. To date, we have implemented language interfaces to JASON [4] and the simple first order language used in [16]. We are currently working on implementing interfaces to other languages, such as a description logic for arguing about ontological knowledge.

In the literature, there are other implementations of argumentation engines, such as defeasible logic programming [12] (and its application in robotic environments [11]), IACAS [19], Argue tuProlog [7], CaSAPI [5], ArgLab [13] and others. See [6] for a comprehensive review of argumentation engines. Most of these argumentation engines are based on various forms of logic programming or are implemented using Prolog. The choice of logic programming or Prolog, on one hand, helps to rapidly prototype an argumentation engines; however, on the other hand, this choice also limits the forms of languages and argumentation strategies permitted. Our work attempts to address this issue by developing ArgTrust in a more flexible and extensible way, with the longer term goal of providing an open source framework that may better fit a wider range of specific applications and contexts.

2 Argumentation Engine

The ArgTrust engine we describe here is an implementation based on a formalism described in some of our recent related work [18]. This formalism combines argumentation with work on propagating trust through a social network, and shows how the results of this propagation can be linked to Dung-style argumentation [10], where the arguments are structured as in [12, 14]. The result is a methodology in which an agent can reason using information from other agents ("sources") that it "knows" through a social network, assigning belief values to that information depending on how much the source is "trusted". These trust values are then propagated through the inference structure of individual arguments towards their conclusions. The propagated values are then assigned to modulate the strength of these arguments. During the analysis of defeat relationships amongst arguments, the values that modulate arguments are then used to weight the arguments and adjust the strength of a defeat accordingly.

This section describes the underlying operation of the ArgTrust engine. We begin by outlining two algorithms that drive the computation, and then detail how these algorithms translate the underlying logic formalism into Java functions. Many terms are used that may or may not be known to the reader, depending on their familiarity with the argumentation literature on which these are drawn. For convenience and clarity, an Appendix (Section 4, at the end of this paper) contains definitions for all terms first mentioned, below, in **bold** font.

2.1 Argumentation Formalism

Given a query q, an **agent** (Definition 2) can construct an **argument** (Definition 5) for q using the knowledge in its information base, Σ, and a domain-specific

rule base, Δ, using the algorithms proposed in our previous work [18]. That work describes a backward-chaining inference mechanism that constructs **all possible arguments**, ARG (Definition 9), from Σ and Δ. Shared inferences are cached in a hash table which maps each conclusion to a list of *inference nodes.* An inference node is either a fact in Σ or an instantiation of an inference rule in Δ with premises being connected to the inference nodes with these premises as conclusions. In the hash table, if it is a "fact" inference node, then the hash key of the node is the fact itself; if it is a "rule" inference node, then the hash key is the conclusion of the instantiated rule. In this way, the constructed proofs are cached in the hash table and can be looked up by their conclusions. As a result, we can avoid the re-construction of the same inference repeatedly, and different arguments can share inferences.

In addition, for the conclusions that cannot be proven, if Σ and Δ, are static, then we can also cache the unproven conclusions to a special proof network, which we refer to as NO-PROOF. This will help avoid repeated proof failure. As shown in [18], assuming that all inferences re fully cached, this mechanism guarantees that the complexity of constructing the argumentation graph of a query is bounded by:

$$O(\ (|C_\Delta| + |\Sigma|) \times |P_\Delta|\ )$$

where $|C_\Delta|$ and $|P_\Delta|$ are the numbers of distinct conclusions and distinct premises, respectively, of all possible instantiated rules in Δ. Of course this result does not mean that inference can be completed in polynomial time since the number of all possible instantiated rules can be exponential in terms of the arity of predicates in the language. In practical applications, however, we can limit the arity of the predicates in the language.

Key to the notion of argumentation is the idea of conflict between arguments, and so detecting and handling conflicts is a key aspect of ArgTrust. The conflicts that can be picked up by the types of **defeat** described in Definition 10 depend on the capability of the argument construction mechanism to construct negated premises (*premise-undercut*), negated intermediate conclusions (*intermediate-undercut*), and negated material implications of the inference rules (*inference-undercut*). The more knowledge and inference rules that are input to ArgTrust, the more arguments will be constructed. This can result not only in more arguments that can potentially support the conclusions we are interested in, but also in more arguments that can potentially undermine the arguments and can defeat previously constructed arguments. These defeats can both defeat the arguments that support the favored conclusions, as well as defeat the arguments that undermine those favored arguments. The possible positive and negative effects of argument construction lead to a reasoning mechanism that forces agents to put forth information and construct arguments cautiously. In turn, the reasoning mechanism forces the argumentation-reasoning process to search the set of acceptable arguments as early as possible in the process, assuming the agents are rational and are capable of predicting the effects of putting forth information and constructing new arguments.

Ideally, we would like the reasoning mechanism to identify any conflict between arguments that corresponds to a disagreement between statements in the application domain. If this is the case, then any **conflict-free** (Definition 14) set of arguments will correspond to situations in the application domain in which there are no disagreements. To make it easier to identify conflicts between arguments, we provide what we call *defeat rules*. These are pairs of formulas in the underlying **predicate language**, $\mathcal{L}$ (Definition 1):

$$(\text{defeater}, \text{defeatee})$$

where the defeater is the conclusion of a defeating argument and the defeatee is the conclusion of a defeated argument. For example, the following expression

$$(\ \text{At}(now, here), \text{At}(now, there)\ )$$

picks up a conflict that can not be established by inference without the following rule:

$$\frac{\text{At}(now, here)}{\neg\text{At}(now, there)}$$

We call this form of defeat *customized defeat*.

2.2 Argumentation Semantics

Many argumentation semantics have been proposed in the literature (e.g., [3, 9, 10]). We have chosen to follow the argumentation semantics of Dung [10], who provides a specific meaning for the system of argumentation we have described above. First, it captures the principle "the one who says the last word wins" found in human argumentation. Second, there is a correspondence between Dung's semantics and the equilibrium of n-person games that allows us to relate argumentation-based and the more traditional game-theoretical approaches to dialogue [10]. A third reason for choosing Dung's semantics is for its abstraction away from the internal representation of an argument. In ArgTrust, this allows us to implement the argumentation semantics without having to commit to a specific language in which arguments are constructed. This, in turn, means that any language which includes the connectives $\wedge, \vee, \neg, \rightarrow$ can be imported into our implementation.

The binary **defeat** relation, DFT (Definition 10) is particularly important when conflicts arise during argumentation. This is because DFT allows the resolution of these conflicts by determining which other arguments can be used to **defend** (Definition 12) conflicting arguments, or arguments being attacked. The notation of defend captures a concept of collective re-establishment of beliefs in which a defeated argument can be re-established if there are other arguments which can defeat its defeaters.

For example, consider the following defeat relation, illustrated in Figure 1:

$$\mathsf{DFT} = \{(A_1, A_2), (A_2, A_3)\}$$

Although argument A_3 is defeated by A_2, A_2 is in turn defeated by A_1 so as to re-establish the acceptability of A_3. Consequently, the argument set A_1 defends argument A_3.

Fig. 1. $\mathsf{DFT} = \{(A_1, A_2), (A_2, A_3)\}$: Arrows represent defeats.

Now, if we extend the defeat relation to contain more defeats:

$$\mathsf{DFT} = \{(A_1, A_2), (A_2, A_3), (A_4, A_3), (A_5, A_4)\}$$

then $\{A_1\}$ alone is not enough to defend argument A_3, since A_1 cannot defend A_3 against A_4. So in this case, we need a larger set, $\{A_1, A_5\}$, to defend A_3. This example is illustrated in Figure 2.

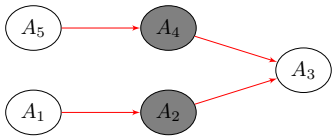

Fig. 2. $\mathsf{DFT} = \{(A_1, A_2), (A_2, A_3), (A_4, A_3), (A_5, A_4)\}$: Arrows represent defeats.

In complex dialogues, long chains of disputes among mutually attacking arguments are bound to happen, so the argumentation semantics must provide a means to compute the argument that ultimately wins the dispute. For this, Dung defines a method to resolve conflicting arguments through a **fixed point** (Definition 13) computation. For example, given a linear defeat relation:

$$\mathsf{DFT} = \{(A_1, A_2), (A_2, A_3), (A_3, A_4)\}$$

As shown in Figure 3, computing the least fixed point starts by initializing the set of **acceptable arguments**, Acc (Definition 13), to the empty set: $Acc^0 = \emptyset$. In the next step, since A_1 is not defeated by any argument, A_1 can be defended by Acc^0, resulting in a larger set of acceptable arguments $Acc^1 = \{A_1\}$. Then, A_2 is the only defeater of A_3 and A_2 is defeated by A_1, therefore Acc^1 can defend A_3, resulting in a larger set of acceptable arguments: $Acc^2 = \{A_1, A_3\}$. At this point, applying the defend function $\mathcal{F}_{\mathsf{DFT}}$ on Acc^2 can not defend additional arguments, namely $\mathcal{F}_{\mathsf{DFT}}(Acc^2) = Acc^2$ where a least fixed point is reached. The set of acceptable arguments with respect to DFT is thus $\{A_1, A_3\}$.

During reasoning, it is not always the case that a (non-empty) fixed point of the argumentation framework exists. There can be multiple conflict-free sets of

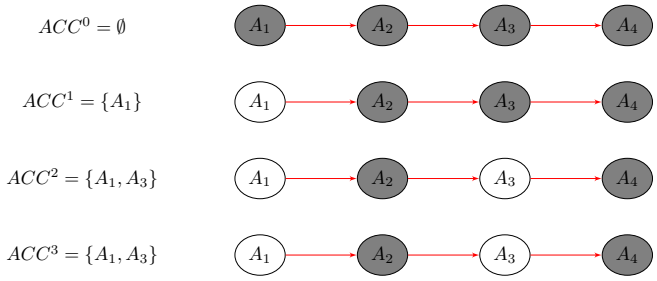

Fig. 3. The computation of the least fixed point of $\mathsf{DFT} = \{(A_1, A_2), (A_2, A_3), (A_3, A_4)\}$.

arguments. Each conflict-free set of arguments might correspond to a possible coherent solution in the application domain, but the unfolded information is not enough to determine which extension is more acceptable than others. According to [10], we can employ the **preferred extension** semantics (Definition 14). For example, given a defeat relation:

$$\mathsf{DFT} = \{(A, B), (B, A), (A, C), (C, D), (C, E)\}$$

As illustrated in Figure 4, no arguments can be defended by the empty set. In this case, the fixed point of DFT is the empty set. However, there are preferred extensions—two maximally admissible sets of arguments: $\{A, D, E\}$ and $\{B, C\}$. Each of these two extensions can defend against the outside defeaters and conflict-free within themselves. If we can make a decision on accepting A or B, we will be able to choose between these two extensions. We can either generate more arguments so that a conflict-free set of arguments can be reached or introduce strength measurement (in probabilities or expected utilities) over the arguments to choose one of these conflicting arguments so that an extension with stronger arguments can survive.

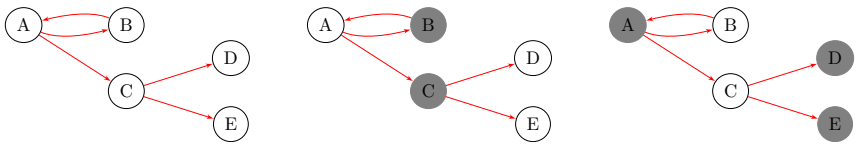

Fig. 4. Defeat relation: $\{(A, B), (B, A), (A, C), (C, D), (C, E)\}$.

3 Computing the Argumentation Semantics

Two primary functions drive the computation in ArgTrust:

– The function *getStatus*(DFT, A) determines the *argumentation status* of argument A in an iterative manner, until the status of all arguments have converged.

– The function *computeStatus*(DFT, A) is called by *getStatus*() to compute the argumentation status.

Pseudo-code for these functions are given by Algorithm 1 and Algorithm 2, respectively. The final output is a *status label* for a given argument, A, and defeat relation, DFT.

We extend the approach outlined in [8], which defines three possible labels that indicate the status of an argument: IN (acceptable), OUT (unacceptable) or $UNDEC$ (undecided). Our extension defines two additional possible labels: $UNDEC'$ and $UNDEC''$. Thus, in ArgTrust, the status of an argument, A, given a defeat relation, DFT, is one of the following values:

$$Status = \mathsf{ARG} \rightarrow \{IN, OUT, UNDEC, UNDEC', UNDEC''\}$$

where IN means the argument is accepted, OUT means the argument is rejected, $UNDEC$ means the argument is undecided, and $UNDEC'$ and $UNDEC''$ are two special temporary status variables assigned to the argument. $UNDEC'$ means that an attempt has been made to compute the status of the argument but it is still undecided; $UNDEC''$ means that in the previous round of computation the status of the argument is undecided. The status label computation is performed using a hash table, as described earlier, to avoid repeated re-construction of the same inferences and repeated proof failure.

Algorithm 1: *getStatus*(DFT, A): Get the status of argument A in DFT

Input: DFT: a defeat relation; A: the argument of interest
repeat
  status ← *computeStatus*(DFT, A);
  Change all $UNDEC'$ status in *StatusCache* to $UNDEC$ for the arguments whose status are determined final;
  Change all $UNDEC'$ status in *StatusCache* to $UNDEC''$;
until *no arguments have their status changed to IN or OUT* ;
return *status*;

The correctness of Algorithm 1 follows from the result in [8], which states that grounded extensions are equivalent to the labelings assigned by Algorithm 2 with the maximal number of arguments labeled with status $UNDEC$. The function *computeStatus*() initially sets the status of every argument encountered to be $UNDEC$. Thereafter, Algorithms 1 and 2 iteratively refine the status of the arguments encountered until no $UNDEC$ arguments can be changed to IN or OUT.

Algorithm 2: *computeStatus*(DFT, A): Compute the status of argument A in DFT

Input: DFT: a defeat relation; A: the argument of interest
if *StatusCache has an entry for A, and StatusCache*$[A]$ *is* IN, OUT, $UNDEC$, *or* $UNDEC'$ **then**
  return *StatusCache*$[A]$;
end
if *A has no defeaters* **then**
  StatusCache$[A] \leftarrow IN$;
  return IN;
else
  StatusCache$[A] \leftarrow UNDEC'$;
end
for *each* B *such that* $(B, A) \in \mathsf{DFT}$ **do**
  $defeaterStatus[B] \leftarrow$ *computeStatus*(DFT, B);
end
if *all defeaters' status are* OUT **then**
  result $\leftarrow IN$;
else if *there is one defeater's status that is* IN **then**
  result $\leftarrow OUT$;
else if *there is one defeater's status that is* $UNDEC$ **then**
  result $\leftarrow UNDEC$;
else if *there is one defeater's status that is* $UNDEC'$ **then**
  result $\leftarrow UNDEC'$;
end
StatusCache$[A] \leftarrow$ *result*;
return *result*;

Note that if there are no loops in the defeat relation DFT (so we don't, for example, have A_1 which defeats A_2 which defeats A_3 which defeats A_1), then one call to *computeStatus*(DFT, A) will have all the arguments' status converged. No $UNDEC$ status will be assumed, in this case the IN arguments are in the grounded extension as well as the preferred extension of the argumentation framework. If there are loops in the argumentation framework, then the computation will be guaranteed to stop after N rounds of computation, where N is the diameter of the argumentation graph.

This can be proved by induction: after N rounds of computation, the arguments whose distances to their known status defeaters is N can be determined about their status. At round 0, the known status arguments $\emptyset$. At round 1, the undefeated arguments' status are determined; their distances to $\emptyset$ in the argumentation graph is 1. Let Det_{N-1} be the set of arguments whose status have been determined in round $N-1$. After executing *computeStatus*(DFT, A) for another round, all the arguments with N distance to $\emptyset$ will not change in the subsequent computation because all its defeaters are of distance less than or equal to $N-1$ to $\emptyset$, therefore their status will be change again. As *computeStatus*(DFT, A) is with the hash cached mechanism, in each round of computation it will explore every node of the whole argumentation graph at most once, the complex-

ity is bounded by the size of argumentation graph, namely $O(|\mathsf{DFT}| + |\mathsf{ARG}|)$. The diameter of DFT is at most $|\mathsf{ARG}|$, therefore the complexity is bounded by $O(|\mathsf{ARG}| \times (|\mathsf{DFT}| + |\mathsf{ARG}|))$. Note that the number of arguments are bounded by $O((|C_\Delta + |\Sigma|) \times |P_\Delta|)$ (see the earlier discussion and our previous work [18]). This means that the complexity of this specific implementation is polynomial in terms of the number of all possible instantiations of the rules in Δ and facts in Σ. The number of all possible instantiations of rules and facts can be limited in polynomial size of the knowledge base and rule base if we limit the arity of the predicates to be a small enough number.

We observe that the arguments whose status will be changed in Det_N of round N are totally determined by the arguments Det_{N-1}. This can help us improve the space usage of the hash cache $StatusCache$ by removing all the arguments in Det_{N-1} except the arguments which immediately defeat an argument with status $UNDEC$.

This completes our discussion of the way that ArgTrust computes the status of arguments, and hence how it implments the Dungian argumentation semantics.

4 Summary

We have introduced an argumentation engine, ArgTrust, and described how it has been implemented in Java to compute the acceptability, non-acceptability or undecidability of an argument based on an information base and rule base. Current work involves deployment of the ArgTrust engine in two different application areas: for reasoning about rules that control a firewall in a network security context [2], and for reasoning about joint activity in a human-robot dialogue.

Note that in addition to the engine described here, we have developed a tool for the display of the arguments generated by ArgTrust (these take the form of the hypergraphs described in the Appendix), and another line of current work is to identify how complex sets of arguments can best be presented to the user.

Appendix: Definitions

This Appendix contains definitions that are used extensively in this paper. The definitions are adapted from [1, 17], with two extensions. First, the information base is expanded to include domain inference rules. Second, the concept of argument is enriched to include a reasoning structure.

Definition 1 (Predicate Language). *A* predicate language*, $\mathcal{L}$, is based on a set of* predicates *(i.e., symbols), $p \in \mathcal{P}$, with standard connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$. Any term belonging to a predicate in $\mathcal{P}$ is finite, and no functional symbols are allowed for any term belonging to a predicate in $\mathcal{P}$.*

We assume that standard semantics apply. In this way, we have a finite set of grounded predicates.

Definition 2 (Agent, Information Base, Rule Base). *An* agent, Ag, *has an* information base, $\Sigma \subseteq \mathcal{L}$, *and a* rule base, Δ. *The rule base is a set of inference rules,* $\delta_i \in \Delta$ *(see Definition 6), where each rule* δ *specifies a conclusion,* c, *that can be drawn from a set of predicates,* $\{p_1, \ldots p_n\}$. *Such a rule is written as follows:*

$$\delta = \frac{\{p_1, \ldots p_n\}}{c}$$

where every predicate, $p_i(\delta)$, *and the conclusion,* $c(\delta)$, *are members of* $\mathcal{L}$.

Complex arguments can be very difficult to analyze, and as a consequence, various graphical forms of argument representation have been developed and found useful by the community, such as the ARAUCARIA software by [15]. We adapt a graphical form to write these rules as directed hyper-edges $\langle \{p_i\}, \{c\} \rangle$, as illustrated in Figure 5 and defined formally below.

Definition 3 (Rule Network). *A* rule network, $\mathcal{R} = \langle V^r, E^r \rangle$, *connects premises and conclusions of rules in the form of a directed hypergraph where:*

1. *the set of vertices* V^r *are elements of* $\mathcal{L}$ *(i.e., predicates,* $p \in \mathcal{P}$*);*
2. *the set of hyper-edges* E^r *are inference rules (i.e.,* $\delta \in \Delta$*);*
3. *the initial vertices of an edge* $e \in E^r$ *are the premises of the corresponding rule* δ*; and*
4. *the terminal node of that edge is the corresponding conclusion,* c.

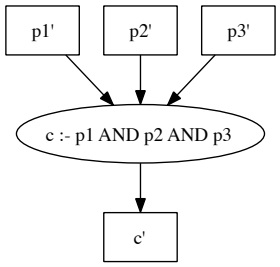

Fig. 5. A rule network. Rectangular nodes denote instantiated premises ($p1', p2', p3'$) and conclusion (c'), and the oval hyperedge denotes an inference rule in the general form "conclusion :- premises" ($c \leftarrow p1 \wedge p2 \wedge p3$).

Under certain circumstances, a rule network captures a proof that is constructed using the rules and premises in a particular Σ, defined formally below.

Definition 4 (Proof Network). *For a given information base,* Σ, *and a rule base,* Δ, *a rule network,* $\mathcal{R} = \langle V^r, E^r \rangle$, *is a* proof network *if and only if every*

premise of each $\delta \in E^r$ is either a member of Σ or the conclusion of some $\delta' \in E^r$.

To be a proof network, the rule network has to be constructed from the contents of some knowledge-base—a rule cannot be in the proof network unless its premises are either in the information base or are derived by applying rules to premises that are in the information base. We say that a proof network is *for a conclusion* c if c is a leaf of the network. For example, Figure 5 is a proof network for conclusion c' if $\{p1', p2', p3'\} \subseteq \Sigma$ and $c \leftarrow \{p1 \wedge p2 \wedge p3\} \subseteq \Delta$, where $p1', p2', p3'$ and c' are instantiated versions of $p1, p2, p3$ and c, respectively.

Some proof networks correspond to *arguments*, defined below:

Definition 5 (Argument, Support, Conclusion). *An* argument*, A, is a pair $\langle H, h\rangle$, where $H = \langle V^r, E^r\rangle$ is a proof network (the* support*) for h (the* conclusion*), and h is the only leaf of H. Constraints on h and H with respect to Σ and Δ follow from Definitions 3 and 4.*

Definition 6 (Inference Rules). *The set of* inference rules*, $R(H)$, is the set of rules, $\delta \in E^r$, that have been instantiated in H. It follows that $R(H) \subseteq \Delta$.*

Definition 7 (Pure Premises). *The set of* pure premises*, $P(H)$, is the set of nodes in H with no incoming edges:*

$$P(H) = \{v \in V^r \mid \text{ there exists no } \langle x, v\rangle \in E^r, x \in \mathcal{L}\}$$

Note that $h \notin P(H)$, because by definition, h is a leaf of H and must be the result of an inference rule, and thus must have an incoming edge.

Definition 8 (Intermediate Conclusions). *The set of* intermediate conclusions*, $C(H)$, of H is the set of all conclusions of rules $\delta \in E^r$ other than h (the ultimate conclusion of the argument H itself):*

$$C(H) = V^r - \{h\} - P(H)$$

The set of pure premises of H can be expressed similarly, as the set of premises, $\delta \in E^r$, that are not (intermediate) conclusions of H:

$$P(H) = V^r - \{h\} - C(H)$$

We can remove the internal structure from our definition of argument (given in Definition 5) and keep only the pure premises and ultimate conclusion, resulting in $A = \langle h, P(H)\rangle$, which is the same definition of argument as described in [1].

Definition 9 (All Possible Arguments). *The set of* all possible arguments*, ARG, is the set of all arguments, $A = \langle H, h\rangle$, that can be constructed from an agent's information base, Σ, and rule base, Δ.*

A key notion in argumentation is that arguments *defeat* one another. That is, one argument casts doubt on another by, for example, contradicting the conclusion of the second argument. Arguments may defeat one another in a number of ways, and we differentiate different forms of defeats as follows:

Definition 10 (Defeat, Rebut, Undercut). *An argument* $\langle H_1, h_1 \rangle$ defeats *another argument* $\langle H_2, h_2 \rangle$ *if it* rebuts*,* premise-undercuts*,* intermediate-undercuts*, or* inference-undercuts *it, where:*

- *An argument* $\langle H_1, h_1 \rangle$ rebuts *another argument* $\langle H_2, h_2 \rangle$ *iff* $h_1 \equiv \neg h_2$*.*
- *An argument* $\langle H_1, h_1 \rangle$ premise-undercuts *another argument* $\langle H_2, h_2 \rangle$ *iff there is a premise* $p \in P(H_2)$ *such that* $h_1 \equiv \neg p$*.*
- *An argument* $\langle H_1, h_1 \rangle$ intermediate-undercuts *another argument* $\langle H_2, h_2 \rangle$ *iff there is an intermediate conclusion* $c \in C(H_2)$ *such that* $c \neq h_2$ *and* $h_1 \equiv \neg c$*.*
- *An argument* $\langle H_1, h_1 \rangle$ inference-undercuts *another argument* $\langle H_2, h_2 \rangle$ *iff there is an inference rule* $\delta \in \Delta(H_2)$ *such that*

$$\delta = \frac{p_1, \ldots, p_n}{c}$$

and $h_1 \equiv \neg(p_1 \wedge \ldots \wedge p_n \rightarrow c)$*.*

When an argument $\langle H_1, h_1 \rangle$ defeats another argument $\langle H_2, h_2 \rangle$, $\langle H_1, h_1 \rangle$ is said to be a **defeater** of $\langle H_2, h_2 \rangle$, and $\langle H_2, h_2 \rangle$ is said to be the **defeatee**. The relation defeat collects all pairs $(\langle H_1, h_1 \rangle, \langle H_2, h_2 \rangle)$ such that $\langle H_1, h_1 \rangle$ defeats $\langle H_2, h_2 \rangle$. We denote the set of all possible defeats as DFT.

Definition 11 (Argumentation framework). *An* argumentation framework*,* AF*, is a pair,* $\mathsf{AF} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$*, where* ARG *is a set of arguments, and* DFT *is the binary relation* defeat *over the arguments.*

Definition 12 (Defend). *Let* $\mathsf{AF} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *be an argumentation framework and* $S \subseteq \mathsf{ARG}$ *be a set of arguments. An argument,* $A1$*, is defended by a set of arguments,* S*, iff* $\forall A2 \in \mathsf{ARG}$*: if* $(A2, A1) \in \mathsf{DFT}$ *then* $\exists A3 \in S$ *such that* $(A3, A2) \in \mathsf{DFT}$*.*

Definition 13 (Characteristic Function, Acceptable Arguments, Fixed Point). *Let* $\mathsf{AF} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *be an argumentation framework and* $S \subseteq \mathsf{ARG}$ *be a set of arguments.*

- *A* characteristic function *is:*

$$\mathcal{F}_{\mathsf{DFT}}(S) = \{A \in \mathsf{ARG} \mid A \text{ is defended by } S \text{ with respect to } \mathsf{DFT}\}$$

- *The set of* acceptable arguments*, denoted by* $Acc^{\mathcal{F}_{\mathsf{DFT}}}$*, is the least* fixed point *of the function* $\mathcal{F}_{\mathsf{DFT}}$ *with respect to set inclusion.*

Definition 14 (Conflict-free, Admissible Set, Preferred Extension). *Let* $\mathsf{AF} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *be an argumentation framework and* $S \subseteq \mathsf{ARG}$ *be a set of arguments.*

– *S is* conflict-free *if there are no two arguments* $A1, A2 \in \mathsf{ARG}$ *such that* $(A1, A2) \in \mathsf{DFT}$.

– *A conflict-free set S is* admissible *iff for any argument* $A1 \in \mathsf{ARG}$ *where* $(A1, A2) \in \mathsf{DFT}$ *and* $A2 \in S$*, then there exists an argument* $A3 \in S$ *such that* $(A3, A1) \in \mathsf{DFT}$.

– *A* preferred extension *of* AF *is the maximum admissible set* $S \subseteq \mathsf{ARG}$ *with respect to set inclusion.*

Definition 15 (Complete Extension, Grounded Extension). *An admissible set S is a* complete extension *iff all arguments defended by S are also in S.*
A conflict-free set S is a grounded extension *if it is the minimal (with respect to set inclusion) complete extension.*

Acknowledgments

This research was partially funded by the National Science Foundation, under grant CNS 1117761.

References

1. L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue, and negotiation. In W. Horn, editor, *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence*, pages 338–342, Berlin, Germany, August 20-25 2000.
2. A. Applebaum, Z. Li, A. R. Syed, K. Levitt, S. Parsons, J. Rowe, and E. Sklar. Firewall configuration: An application of multiagent metalevel argumentation. In *Proceedings of the 9th Workshop on Argumentation in Multiagent Systems*, 2012.
3. T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003.
4. R. Bordini, J. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley series in agent technology. J. Wiley, 2007.
5. D. Bryant and P. Krause. An implementation of a lightweight argumentation engine for agent applications. In M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa, editors, *Logics in Artificial Intelligence*, volume 4160 of *Lecture Notes in Computer Science*, pages 469–472. Springer Berlin / Heidelberg, 2006.
6. D. Bryant and P. Krause. A review of current defeasible reasoning implementations. *Knowl. Eng. Rev.*, 23:227–260, September 2008.
7. D. Bryant, P. J. Krause, and G. A. W. Vreeswijk. Argue tuProlog: A lightweight argumentation engine for agent applications. In *Proceedings of the 2006 conference on Computational Models of Argument: Proceedings of COMMA 2006*, pages 27–32, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.
8. M. Caminada and D. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2):109–145, Dec. 2009.
9. C. Cayrol and M.-C. Lagasquie-Schiex. Graduality in argumentation. *Journal of Artificial Intelligence Research*, 23:245–297, 2005.

10. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77:321–357, September 1995.
11. E. Ferretti, M. Errecalde, A. J. García, and G. R. Simari. An application of defeasible logic programming to decision making in a robotic environment. In *LPNMR*, pages 297–302, 2007.
12. A. J. Garcia and G. R. Simari. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming*, 4(2):95–138, 2004.
13. M. Podlaszewski, M. Caminada, and G. Pigozzi. An implementation of basic argumentation components. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, pages 1307–1308, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.
14. H. Prakken. An abstract framework for argumentation with structured arguments. *Argument and Computation*, 1(2):93–124, 2011.
15. C. Reed and G. Rowe. Araucaria: Software for argument analysis, diagramming and representation. *International Journal on Artificial Intelligence Tools*, 13(4):983–, 2004.
16. S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3nd Edition)*. Prentice Hall, December 2009.
17. G. R. Simari and R. P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2-3):125–157, 1992.
18. Y. Tang, K. Cai, E. Sklar, P. McBurney, and S. Parsons. Using argumentation to reason about trust and belief. *Journal of Logic and Computation*, 2011. (to appear).
19. G. Vreeswijk. *IACAS: an interactive argumentation system : user manual version 1.0*. Technical reports in computer science. University of Limburg, Department of Computer Science, 1994.