Argumentation System Allowing Suspend/Resume of an Argumentation Line

Kenichi Okuno* and Kazuko Takahashi

School of Science&Technology, Kwansei Gakuin University,
2-1, Gakuen, Sanda, 669-1337, Japan
o.kenichi@gmail.com, ktaka@kwansei.ac.jp

Abstract. This paper discusses an argumentation system that treats argumentation dynamically. We previously proposed a model for dynamic treatment of argumentation in which all lines of argumentation are executed in succession, with the change of the agent's knowledge base. This system was designed for grasping the behaviour of actual argumentation, but it has several limitations. In this paper, we propose an extended system in which these points are revised so that the model can more precisely simulate actual argumentation. In addition, we provide a simpler algorithm for judgement of given argumentation, which can be applied to make a strategy to win.

Keywords: computational model for argumentation, belief change, agent communication.

1 Introduction

Argumentation is a model that evaluates arguments. It was originally investigated in legal reasoning. Dung's work on constructing a logical framework for argumentation and showing the relationships with nonmonotomic reasoning and logic programming [8] enlarged the possibility of application area of argumentation to the field of artificial intelligence (AI). As a result, formal models of argumentation have received much attention by AI researchers [4,22]. These works include applications for defeasible logic programming [10,6,18,17,15], belief revision [9,19] and so on. Argumentation is considered to be a powerful tool to logically analyse significant phenomena that appear in multiagent systems such as negotiation, agreement and persuation [14,1], and to make a computational model for a bahavior of multiagents [12]. Generally, argumentation proceeds between two agents by giving arguments in turn that attacks the opponent's argument until one of them cannot attack any more. Finally, the loser accepts the winner's proposal. This process is usually represented in the tree form [1,10]. The root node is a proposed formula and each branch corresponds to a single argumentation line, namely, a sequence of arguments. Lots of argumentation systems have proposed so far [4,22], but they considered evaluation of a single argumentation line, and it cannot handle the dynamic properties of actual

* Currently, JSOL Corporation.

P. McBurney, I. Rahwan, and S. Parsons (Eds.): ArgMAS 2010, LNAI 6614, pp. 248–267, 2011.

argumentation. On the other hand, we have proposed a system that can treat continous evaluation of multiple argumentation lines [20,21].

Let us consider an example of argumentation. According to many argumentation systems, a proposer P makes the first argument and a defeater C makes counterarguments. We suppose a situation in which a murderer P tells a lie: "I did not commit murder". A policeman C argues that P's statement is a lie. P_i and C_i represent P's and C's i-th utterances, respectively.

P_1: "I did not commit murder! There is no evidence!"
C_1: "There is evidence. We found your license near the scene."
P_2: "It's not evidence! I had my license stolen!"
C_2: "It is you who killed the victim. Only you were near the scene at the time of the murder."
P_3: "I didn't go there. I was at facility A at that time."
C_3: "At facility A? No, that's impossible. Facility A does not allow a person to enter without a license. You said that you had your license stolen, didn't you?"

Figure 1 shows the structure of this argumentation.

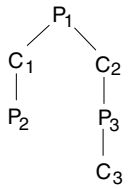


Fig. 1. Structure of argumentation

In this example, if argumentation proceeds along the left branch, and if C has no counterargument to P_2, then C continues a counterargument in the right branch which attacks P_1 from another side. Finally, C points out the contradiction between P's utterances and wins. P's utterance P_2 gives C new information and causes C to generate C_3.

To capture the behaviour in this example, we have proposed an argumentation system incorporating changes in an agent's knowledge base caused by the exchange of arguments and defined a new concept of "dynamic win" which is different from the usual concept of "win" obtained by static analysis [20,21]. The goal of this system is to dynamically grasp argumentation by providing a model for actual argumentation. However, several points exist in which this earlier system does not reflect actual argumentation.

The first limitation is on the mechanism that brings up the settled matter in some argumentation line. Consider another argumentation:

P_1: "I did not commit murder! There is no evidence!"
C_2: "It is you who killed the victim. Only you were near the scene at the time of the murder."

P_3: "I didn't go there. I was at facility A at that time."
C_1: "There is evidence. We found your license near the scene."
P_2: "That's not evidence! I had my license stolen!"
C'_3: "That's strange. Facility A does not allow a person to enter without a license. You said that you were at facility A when the murder occurred. How did you enter?"

In this case, an argumentation first proceeds along the right branch, and then continues to the left branch, P's utterance P_2 gives C new information and causes C to generate C'_3 as a counterargument to P_3. C also points out the contradiction between P's utterances, and wins. Such a phenomenon frequently occurs in actual argumentation when each argumentation line is not so long. In our earlier system, this mechanism could not be handled. In this paper, we present a revised system in which each argumentation line is considered as suspended but may be resumed afterward.

A second shortcoming is the inequivalent rights of agents. In the earlier version, the defeater could continue an argumentation with the revised knowledge base after he/she loses one argumentation line, leading him/her to ultimately win the argumentation tree. However, the proposer loses the whole argumentation tree if he/she loses one argumentation line. In the revised version, we also allow the proposer to continue an argumentation after he/she loses one argumentation line.

The third point is also related to the equivalent rights of agents. In the earlier version, a proposer could not use disclosed information whereas a defeater could. This condition is unfair and unnatural. In the revised version, we adopt *commitment store* [13], a common knowledge base to store all the disclosed information, and both agents can use this knowledge base.

We extend the earlier system by addressing these three points so that it can more precisely simulate an actual argumentation and redefine the dynamic win/lose of an argumentation tree. We show that how this extension improves the treatment of dynamic argumentation.

Moreover, we propose an algorithm for judging the result of an argumentation tree. This algorithm is simpler and easier to implement and it can be applied to formulate an argumentation strategy.

This paper is organised as follows. Section 2 provides the definitions of basic concepts such as argumentation and the argumentation tree. Section 3 proposes an extended model for argumentation incorporating changes in an agent's knowledge base and also presents an algorithm for the judgement of an argumentation tree. Section 4 provides an example of this algorithm. Section 5 outlines the major changes from our previous work and compares the proposed approach with related works. Finally, section 6 presents conclusions.

2 Argumentation

2.1 Argumentation Framework

We define an argumentation framework based on Dung [8].

Definition 1 (consistent). *Let Ψ be a set of formulas in propositional logic. If there does not exist ψ that satisfies both $\psi \in \Psi$ and $\neg\psi \in \Psi$, Ψ is said to be* consistent.

The knowledge base $\mathbf{K}_\mathrm{a}$ for each agent a is a finite set of propositional formulas. Note that $\mathbf{K}_\mathrm{a}$ is not necessarily consistent and may have no deductive closure; that is, a case may exist in which $\phi, \phi \rightarrow \psi \in \mathbf{K}_\mathrm{a}$ and $\psi \notin \mathbf{K}_\mathrm{a}$ hold. An agent a participates in argumentation using elements of $\mathbf{K}_\mathrm{a}$.

Definition 2 (support). *For a nonempty set of formulas Ψ and a formula ψ, if there exist ϕ, $\phi \rightarrow \psi \in \Psi$, then Ψ is said to be a* support for ψ.

Definition 3 (argument). *Let $\mathbf{K}_\mathrm{a}$ be a knowledge base for an agent a.* An argument of a *is a pair (Ψ, ψ) where Ψ is a subset of $\mathbf{K}_\mathrm{a}$, and $\psi \in \mathbf{K}_\mathrm{a}$ such that Ψ is the empty set or a consistent support for ψ,*

For an argument $A = (\Psi, \psi)$, Ψ and ψ are said to be *grounds* and *a sentence of* A, respectively. They are denoted by $Grounds(A)$ and $Sentence(A)$, respectively. $S(A)$ denotes $Grounds(A) \cup \{Sentence(A)\}$. If $\psi \in S(A)$, it is said that *a formula ψ is contained in an argument A.*

Similar to many argumentation systems, we adopt the concept of *preference* [3,16]. Preferences are assigned to formulas depending on their strength, certainty and stability to avoid loops in the argumentation. Here, we assume that a formula is given a preference value based on some basic rules in advance regardless of the knowledge base in which it is contained, and adopt a simple definition for computating the preference of an argument. Although these definitions affect the result of argumentation, we do not discuss the definitions here, since this aspect of argumentation is out of the scope of this paper.

Definition 4 (preference). *Each formula is assigned a preference value. Let $\nu(\psi)$ be the preference for a formula ψ. Then, the preference of an argument A is defined by* $\sum_{\psi \in S(A)} \nu(\psi)$.

Definition 5 (attack). *Let $AR_{\mathbf{K}_\mathrm{a}}$ and $AR_{\mathbf{K}_\mathrm{b}}$ be sets of all possible arguments of agents a and b, respectively.*

1. *If $Sentence(A_a) \equiv \neg Sentence(A_b)$ and $\nu(A_a) \geq \nu(A_b)$, then (A_a, A_b) is said to be* a rebut *from a to b.*
2. *If $\neg Sentence(A_a) \in Grounds(A_b)$ and $\nu(A_a) \geq \nu((\emptyset, \neg Sentence(A_a)))$, then (A_a, A_b) is said to be* an undercut *from a to b.*
3. An attack *from a to b is either a rebut or an undercut from a to b.*

When (A_a, A_b) is an attack from a to b, it is said that A_a *attacks* A_b.

Based on Dung [8], in an argumentation framework between two agents, a proposer P makes the first argument and a defeater C makes counterarguments. Hereafter, $\mathbf{K}_\mathrm{P}$ and $\mathbf{K}_\mathrm{C}$ denote their knowledge bases, respectively.

Definition 6 (argumentation framework). *Let $AR_{\mathbf{K}_\mathrm{P}}$ and $AR_{\mathbf{K}_\mathrm{C}}$ be sets of all possible arguments of P and C, respectively, with preferences ν. Let $AT_{\mathbf{K}_\mathrm{P} \rightharpoonup \mathbf{K}_\mathrm{C}}$ and $AT_{\mathbf{K}_\mathrm{C} \rightharpoonup \mathbf{K}_\mathrm{P}}$ be sets of attacks from P to C and from C to P, respectively.* An argumentation framework *between P and C, $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$ is defined as a quadruple $\langle AR_{\mathbf{K}_\mathrm{P}}, AR_{\mathbf{K}_\mathrm{C}}, AT_{\mathbf{K}_\mathrm{P} \rightharpoonup \mathbf{K}_\mathrm{C}}, AT_{\mathbf{K}_\mathrm{C} \rightharpoonup \mathbf{K}_\mathrm{P}} \rangle$.*

2.2 Argumentation Tree

Definition 7 (move). A move *is a pair of a player (an agent) P/C and an argument A in which $A \in AR_{\mathbf{K}_\mathrm{P}}/AR_{\mathbf{K}_\mathrm{C}}$. If player is P/C, then it is said to be P/C's move. For a move $M = (player, argument)$, we denote player and argument by $Ply(M)$ and $Arg(M)$, respectively.*

Definition 8 (move's attack). *M is said to be* an attack to *M', if $(Arg(M), Arg(M'))$ is an attack from $Ply(M)$ to $Ply(M')$.*

Definition 9 (argumentation line, argument set). *Let P and C denote a proposer of a formula φ and its defeater, respectively. Let $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$ be an argumentation framework between P and C.* An argumentation line *D for φ on $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$ is a finite nonempty sequence of moves $[M_1, \ldots, M_n]$ $(i = 1, \ldots, n)$ that satisfies the following:*

1. *$Ply(M_1) = P$, where $Sentence(Arg(M_1)) = \varphi$.*
2. *If i is odd, then $Ply(M_i) = P$, and if i is even, then $Ply(M_i) = C$.*
3. *M_{i+1} is an attack to M_i for each i $(1 \leq i \leq n-1)$.*
4. *No attack occurs against $Arg(M_n)$.*
5. *$M_i \neq M_j$ for each pair of i, j $(1 \leq i \neq j \leq n)$.*
6. *Both $S(Arg(M_1)) \cup S(Arg(M_3)) \cup S(Arg(M_5)) \cup \ldots \cup S(Arg(M_o))$ and $S(Arg(M_2)) \cup S(Arg(M_4)) \cup S(Arg(M_6)) \cup \ldots \cup S(Arg(M_e))$ are consistent, where o and e are the largest odd number and the largest even number less than or equal to n, respectively.*

The above $S(Arg(M_1)) \cup S(Arg(M_3)) \cup S(Arg(M_5)) \cup \ldots \cup S(Arg(M_o))$ and $S(Arg(M_2)) \cup S(Arg(M_4)) \cup S(Arg(M_6)) \cup \ldots \cup S(Arg(M_e))$ are said to be P's argument set on *D and* C's argument set on *D, and they are denoted by $S_P(D)$ and $S_C(D)$, respectively.*

This definition puts the constraints of loop-freeness and consistency of each agent's arguments on an argumentation line.

Definition 10 (win of an argumentation line). *If the last element of an argumentation line D is P's move, then it is said that P* wins *D; otherwise, P* loses *D.*

Definition 11 (argumentation tree). An argumentation tree for *φ* on *$AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$ is a tree in which the root node at depth 0 is empty and all the branches*[1] *starting from the node of depth 1 are different argumentation lines for φ on $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$.*

[1] Here, a branch is a path from the designated node to a leaf node.

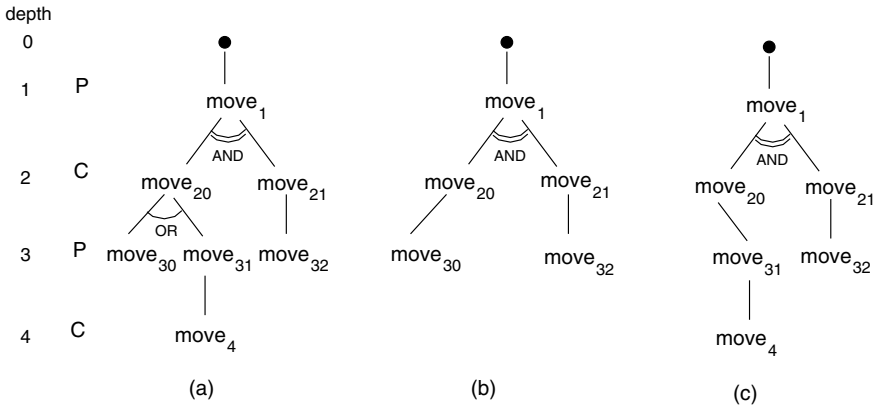


Fig. 2. An argumentation tree and its candidate subtrees

Definition 12 (candidate subtree). A candidate subtree *is a subtree of an argumentation tree that selects only one child node for each node corresponding to C's move in the original tree, and selects all child nodes for each node corresponding to P's move.*

Definition 13 (solution subtree). A solution subtree *is a candidate subtree in which P* wins *all of the argumentation lines in the tree.*

Each candidate subtree corresponds to P's selection of an argument, and the solution subtree indicates the case in which P takes a winning strategy. In Figure 2, (a) is an argumentation tree, (b) and (c) are its candidate subtrees, and (b) is the solution subtree.

In general, judgement of an argumentation tree is defined as follows.

Definition 14 (static win of an argumentation tree). *If an argumentation tree has a solution subtree, then P* statically wins *the argumentation tree; otherwise, P* statically loses *it.*

3 Argumentation with Changes in the Knowledge Base

3.1 Execution of Argumentation

We propose a dynamic argumentation system that considers the successive executions of all possible argumentation lines, whilst the usual ones consider only a single argumentation line. In a dynamic argumentation, we have to consider the interaction of argumentation lines.

We introduce the commitment store and a hitory. The commitment store is a set of all the formulas contained in all arguments given so far. History $\mathbf{H}_a$ is prepared for each agent a to preserve the coherence of each agent's arguments. $\mathbf{H}_a$ is a set of all the formulas contained in a's arguments in the argumentation lines in which a wins. We, however, ignore the coherence of the loser's side. This

is based on the idea that the winner should be responsible for his/her arguments, but the loser can make an attack from a different side.

A dynamic argumentation line is defined by extending a static argumentation line with history.

Definition 15 (dynamic argumentation line). *Let P,C denote a proposer of a formula φ and its defeater. Let $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$ be an argumentation framework between P and C.* A dynamic argumentation line $D = [M_1, \ldots, M_n]$ *for φ on $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$ with histories $\mathbf{H}_P$ and $\mathbf{H}_C$ is defined as the extension of the (static) argumentation line by adding the following additional condition.*

7. $\mathbf{H}_{Ply(M_i)} \cup S(Arg(M_i))$ is consistent for each i $(1 \leq i \leq n)$.

If no misleading is involved, *a dynamic argumentation line* for φ on $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$ with a history $\mathbf{H}_P$ and $\mathbf{H}_C$, is said to be just *an argumentation line* on $AF(\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{C}, \nu)$.

Here, we present a dynamic argumentation model. We consider the execution of an argumentation as selecting a branch, updating the commitment store and agents' histories and modifying a tree.

An argumentation starts by selecting a branch of an initial argumentation tree. It proceeds along the branch and when the execution reaches the leaf node, the branch is suspended. At that time, the commitment store is updated and agents can make new arguments using the commitment store in addition to their own knowledge bases. New nodes are added to the argumentation tree if new arguments are generated due to this change of knowledge base. Next, another branch is selected.

On the execution procedure, the executed node is marked and the branch containing unmarked nodes can be selected. The suspended branch may be resumed if a new unmarked node is added to it. On the selection of a branch, the turn of an utterance should be kept. This means that if one branch is suspended at the node that corresponds to one agent's argument, then a next branch should start with the node that corresponds to the other agent's argument.

Definition 16 (executable node). *For a node M_i $(1 \leq i \leq n)$ in a branch $D = [M_1, \ldots, M_n]$ and a current turn t, if $M_1, \ldots, M_{i-1}$ are marked and $M_i, \ldots, M_n$ are unmarked, and $Ply(M_i) = t$, then the node M_i is said to be* executable.

Definition 17 (execution of a branch). *For a branch $D = [M_1, \ldots, M_n]$, histories $\mathbf{H}_P, \mathbf{H}_C$ and the commitment store $\mathbf{K}$, execution of D from i $(1 \leq i \leq n)$ with $\mathbf{H}_P$ and $\mathbf{H}_C$ is defined as follows.*

1. Mark $M_i, \ldots, M_n$.
2. Set $\mathbf{K} = \mathbf{K} \cup \bigcup_{k=i}^{n} S(Arg(M_i))$.
3. **if** *$Ply(M_n) = P$,*
  then *set the current turn to C and $\mathbf{H}_P = \mathbf{H}_P \cup S_P(D)$.*
  if *$Ply(M_n) = C$,*
  then *set the current turn to P and $\mathbf{H}_C = \mathbf{H}_C \cup S_C(D)$.*

Definition 18 (suspend/resume). *After the execution of all nodes in a branch, D is said to be* suspended. *For a suspended branch D, if an executable node is added to its leaf on the modification of a tree, and D is selected, then D is said to be* resumed.

This Argumentation Procedure with Knowledge Change is formalised as follows.

Argumentation Procedure with Knowledge Change (*APKC2*)

Let $AF(\mathbf{K}_{\mathrm{P}}, \mathbf{K}_{\mathrm{C}}, \nu)$ be an argumentation framework, and φ be a proposed formula.

[STEP 1(initialisation)]
Set $\mathbf{K} = \emptyset$, $\mathbf{H}_P = \emptyset$, $\mathbf{H}_C = \emptyset$, $turn = P$. Construct an initial argumentation tree for $AF(\mathbf{K}_{\mathrm{P}}, \mathbf{K}_{\mathrm{C}}, \nu)$ on φ with $\mathbf{H}_P, \mathbf{H}_C$ with all the nodes unmarked.
[STEP 2(execution of an argumentation)]
if no branch has an executable node,
  if *turn*=P, **then** terminate with P's lose.
  else *turn*=C, **then** terminate with P's win.
else select a branch and execute it from the executable node.
[STEP 3(modification of a tree)]
Reconstruct an argumentation tree for $AF(\mathbf{K}_{\mathrm{P}} \cup \mathbf{K}, \mathbf{K}_{\mathrm{C}} \cup \mathbf{K}, \nu)$ on φ with $\mathbf{H}_P, \mathbf{H}_C$.
if for any pair of node N and M in the tree
    where $Ply(N) = Ply(M)$ and $Arg(N) = Arg(M)$,
    N is marked whilst M is unmarked,
  then mark M.
go to STEP 2.

The elements of $\mathbf{K}$ are included either by $\mathbf{K}_{\mathrm{P}}$ or $\mathbf{K}_{\mathrm{C}}$, which are both finite sets. It follows that finite kinds of moves can be generated. Therefore, *APKC2* terminates.

In the modification of a tree in *APKC2*, a new node may be added. An idea of *threat* is introduced to explain this situation.

Definition 19 (threat). *Let M and M' be moves in an argumentation tree T on $AF(\mathbf{K}_{\mathrm{P}}, \mathbf{K}_{\mathrm{C}}, \nu)$. If $S(Arg(M))$ generates more than one new move that attacks M', then it is said that M is* a threat to M', *and that T* contains a threat. *M and M' are said to be* a threat resource *and* a threat destination, *respectively.*

Intuitively, a threat is a move that may provide information advantageous to the opponent. A move may be a threat to a move in the same branch.

Definition 20 (continuous candidate subtree). *For a candidate subtree CT, if at least one candidate subtree is generated by the addition of nodes, then these subtrees are said to be* continuous candidate subtrees of CT.

Note that nondeterminism is involved in the selection of a branch in *APKC2*, and finally obtained trees are different depending on the selection.

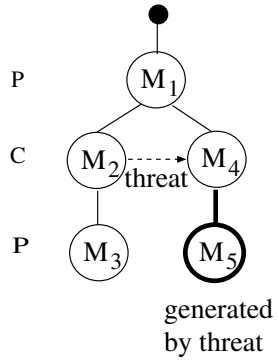


Fig. 3. Argumentation affected on the execution order of branches

Consider an argumentation tree in Figure 3. In this figure, M_2 and M_4 are a threat resource and a threat destination, respectively, and M_5 is a newly generated node by this threat. If we execute from the left branch, then *APKC2* proceeds by executing M_1, M_2, M_3, M_4, M_5, and teminates with P's win. On the other hand, if we execute from the right branch, then *APKC2* proceeds by executing M_1, M_4 and suspends. The next turn is P. If there exists no branch in the other candidate trees that starts with P and ends with P, the suspended branch never resumes, and *APKC2* terminates with P's lose.

We define a dynamic win/lose of an argumentation tree according to *APKC2*.

Definition 21 (dynamic solution subtree). *Let CT be a candidate subtree of an initial argumentation tree. For any execution order of branches of CT, if* APKC2 *terminates with P's win or CT has a continuous subtree such that P wins, then CT is said to be* a dynamic solution subtree.

Definition 22 (dynamic win of an argumentation tree). *If an argumentation tree has a dynamic solution subtree, then P* dynamically wins *the argumentation tree; otherwise, P* dynamically loses *it.*

3.2 Judgement of Dynamic Win/Lose

APKC2 gives an execution model for an argumentation procedure. If we only want to judge the result of an argumentation and not simulate the procedure, then there exists a simpler algorithm.

Definition 23 (consistent candidate subtree). *Let CT be a candidate subtree. If there does not exist moves M, M' and a formula ψ that satisfy $Ply(M) = Ply(M') = P$, $\psi \in S(Arg(M))$ and $\neg\psi \in S(Arg(M'))$, then CT is said to be* a consistent candidate subtree.

Let CT be a candidate subtree of an argumentation tree of $AF(\mathbf{K}_{\mathrm{P}}, \mathbf{K}_{\mathrm{C}}, \nu)$. Then we can judge a proposer P's win/lose of CT by the following algorithm. Hereafter, $D \in T$ denotes that a branch D in a tree T.

Judgement of Win/Lose of a Candidate Subtree (*JC*)

[STEP 1]
if CT is not consistent, **then** terminate with P's lose.
else if there exists a leaf corresponding C's move in CT,
then terminate with P's lose.
else set $\mathcal{K} = \bigcup_{D \in CT} S_P(D) \ \cup \ \bigcup_{D \in CT} S_C(D)$.
[STEP 2]
Reconstruct CT on $AF(\mathbf{K}_{\mathrm{P}} \cup \mathcal{K}, \mathbf{K}_{\mathrm{C}} \cup \mathcal{K}, \nu)$, and let the resultant tree be CT'.
[STEP 3]
if $CT' = CT$, **then** terminate with P's win.
else select a new continuous candidate subtree and go to STEP 1.

The algorithm *JC* terminates by the same reason as that for termination of *APKC2*.

We show the relationship of dynamic win of an argumentation tree and the judgement by *JC*.

First, we show that P dynamically wins an argumentation tree T if there exists a candidate subtree in T for which *JC* terminates with P's win.

Theorem 1. *Let T be an argumentation tree which includes no threat over different candidate subtrees. P dynamically wins T if there exists a candidate subtree in T for which* JC *terminates with P's win.*

Proof

Let CT be a candidate subtree of an argumentation tree of $AF(\mathbf{K}_{\mathrm{P}}, \mathbf{K}_{\mathrm{C}}, \nu)$ for which *JC* terminates with P's win. We show that for any execution order of branches APKC2 terminates with P's win.

Let M_0 be a node nearest to the root node of CT that corresponds to P's move. Let $\mathcal{B}$ be a set of sequences each of which consists of nodes except for M_0 in each branch of CT. Let $\mathcal{N}$ be a set of sequences each of which consists of nodes added on STEP2 of *JC*. And let $Nodes = \mathcal{B} \cup \mathcal{N}$ (Figure 4). Every element of $Nodes$ is a sequence of nodes $M_1 \ldots, M_h$ where $Ply(M_1) = C, Ply(M_h) = P$ and M_{i+1} attacks M_i $(1 \leq i \leq h-1)$. Then, any execution order of branches can be represented as a finite sequence of elements of $Nodes$ following M_0. For example, $M_0 \rightarrow B_1 \rightarrow B_2 \rightarrow B_5$ in Figure 4 is such a sequece. Then, its final node is P's move. Moreover, consistency of $\mathbf{H}_P$ and $\mathbf{H}_C$ in APKC2 are preserved since all reconstructed candidate trees in *JC* are consistent. Therefore, for any execution order of branches APKC2 terminates with P's win. □

Next, we show the opposite direction of this theorem, that is, there exists a candidate subtree in T for which *JC* terminates with P's win, if P dynamically wins T. First, we prove it for a simple case, then for a general case.

Lemma 1. *Let T be an argumentation tree which includes no threat over different candidate subtrees. Assume that all the branches selected in APKC2 belong to a single candidate subtree. There exists a candidate subtree in T for which* JC *terminates with P's win if P dynamically wins T.*

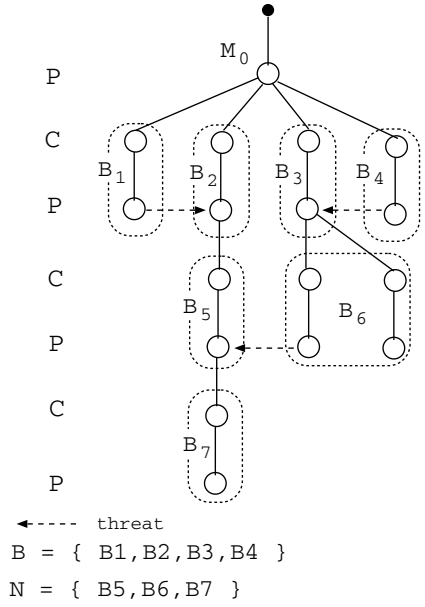


Fig. 4. Illustration of the proof for Theorem 1

Proof

In this case, we show that P dynamically loses T if JC terminates with P's lose for all candidate subtrees in T.

For a candidate subtree CT, if JC terminates with P's lose for CT, there exists a branch whose leaf node is C's move in some step of JC procedure.

Assume that there exists such a branch at an initial state. *APKC2* terminates with P's lose immediately if this branch is selected, since there is no branch beginning from P's move and there is no executable node.

Let $CT_0, \ldots, CT_k$ be a sequence of reconstructed candidate subtrees of CT in JC procedure. Assume that all the leaves are P's moves in CT_i $(0 \leq i \leq k-1)$ and there exists a branch D whose leaf node is C's move in CT_{i+1}. The leaf node M of D in CT_i is P's move (Figure 5). There exists threat resource in the nodes in CT_i whose threat destination is M. It follows that new nodes are added to D in CT_{i+1}. Let D' be the branch that contains the threat resource. (Note that D' may be D.) If D' and D are executed in this order in *APKC2*, *APKC2* terminates with C's move after executing all the nodes including new nodes. Then, *APKC2* terminates with P's lose since there is no branch beginning from P's move and there is no executable node.

Therefore, P dynamically loses T. □

In lemma 1, we assume that all the branches selected in *APKC2* belong to a single candidate subtree. However, branches in multiple candidate subtrees may be selected, since *APKC2* allows selection of any branch. Generally, there should

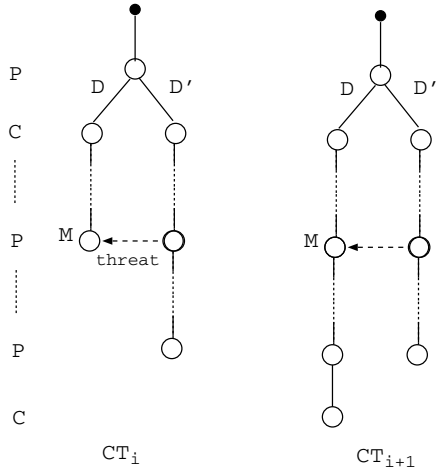


Fig. 5. Illustration of the proof for Lemma 1

exist a (static) solution subtree that is included by a set of all executed branches. We show this by the following two lemmas.

Lemma 2. *Let T_f be a finally obtained tree when* APKC2 *terminates with P's win. For a subtree T whose root node M is C's move in T_f, let $M_{P_1}, \ldots, M_{P_n}$ be M's child nodes, and let $T_1, \ldots, T_n$ be subtrees whose root nodes are $M_{P_1}, \ldots, M_{P_n}$, respectively. If $T_1, \ldots, T_n$ are all candidate subtrees, then there exists a (static) solution subtree T_i ($1 \leq i \leq n$).*

Assume that T_i is not a solution subtree for some i. Then, T_i includes C's move as a leaf node. Let D be a branch of T_f that contains this node. There should exist another branch as the successive execution of D, since *APKC2* terminates with P's move. On the other hand, when the leaf node of D has been executed, the unmarked nodes nearest to the root node of T_f in every branch of T_i that includes unmarked nodes are C's moves. They are unexecutable. Therefore, a branch in subtrees other than T_i should be selected as D's successive execution. If none of $T_1, \ldots, T_n$ is a solution subtree, it is impossible to terminate *APKC2* with P's move. Hence, one of them should be a solution subtree. □

We can take such T_i as T's candidate subtree, and obtain the following lemma.

Lemma 3. *Let T_f be a finally obtained tree when* APKC2 *terminates with P's win. T_f includes a (static) solution subtree.*

Proof

For a subtree whose root node M is C's move in T_f, let $M_{P_1}, \ldots, M_{P_n}$ be M's child nodes, and let $T_1, \ldots, T_n$ be subtrees whose root nodes are $M_{P_1}, \ldots, M_{P_n}$, respectively. For each i ($1 \leq i \leq n$), if T_i is not a candidate subtree, then replace it by its candidate subtree T_i' from lemma 2; otherwise, set T_i' be T_i. There

exists a solution subtree T_i' $(1 \leq i \leq n)$, since all of them are candidate subtrees. Repeating this procedure, it is proved that T_f includes a solution subtree. □

Theorem 2. *Let T be an argumentation tree which includes no threat over different candidate subtrees. There exists a candidate subtree in T for which* JC *terminates with P's win if P dynamically wins T.*

Proof

Let CT' is a finally obtained tree for a candidate subtree CT in *JC*. From lemma 3, the finally obtained tree T_f in *APKC2* includes a (static) solution subtree. There exists CT that contains this solution subtree, since both threat resource and threat desitination are in the same candidate subtree from the condition. Moreover, $\bigcup_{D \in T_f} S_P(D)$ is consistent because of the constraints on $\mathbf{H}_P$. Therefore, there exists a candidate subtree for which *JC* terminates with P's win. □

4 An Example

Consider the example shown in Section 1. We illustrate various properties of *APKC2* and *JC* using this example.

4.1 Formalisation

The knowledge bases of a proposer P and a defeater C are shown below. The number attached to each formula shows its preference. We assume that the facts and rules are all represented in the knowledge base and the agents have no other knowledge.

$$\mathbf{K}_{\mathrm{P}} = \left\{ \begin{array}{l} \neg m[1],\ \neg e[2],\ (\neg e \rightarrow \neg m)[1],\ \neg(la \rightarrow e)[1], \\ ls[1],\ (ls \rightarrow \neg(ls \rightarrow e))[1],\ \neg n[1],\ a[2], \\ (a \rightarrow \neg n)[1] \end{array} \right\}$$

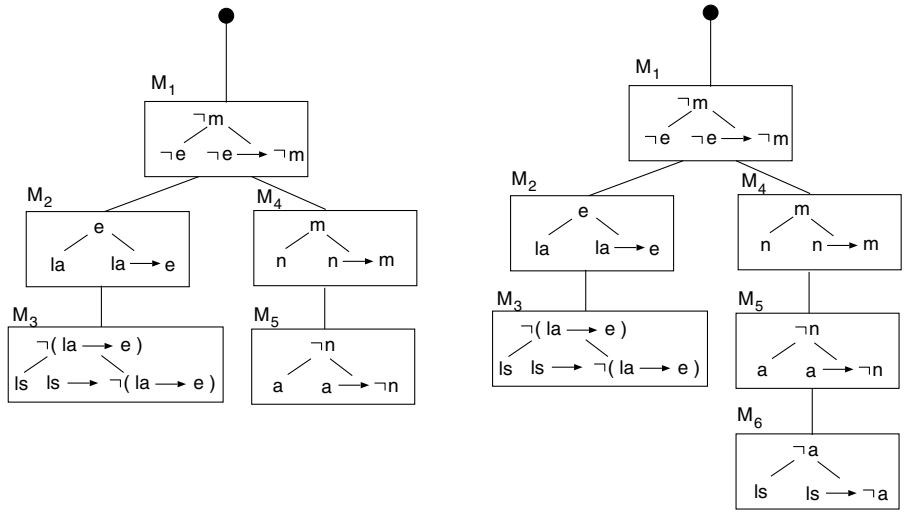
$$\mathbf{K}_{\mathrm{C}} = \left\{ \begin{array}{l} e[1],\ la[1],\ (la \rightarrow e)[2],\ m[2],\ n[2], \\ (n \rightarrow m)[1],\ \neg a[1],\ (ls \rightarrow \neg a)[1] \end{array} \right\}$$

The propositions have the following meanings:
m: P commits murder.
e: there is evidence.
la: P's license was left at the scene of the murder.
ls: P's license was stolen.
n: P was near the scene when the murder was committed.
a: P was at facility A when the murder was committed.

4.2 The Case of Changing from Static Win to Dynamic Lose

Figure 6 shows a relevant part of an initial argumentation tree and a final argumentation tree in *APKC2*. The argumentation starts with the murderer's utterance. The nodes M_1, M_2, M_3, M_4, M_5, and M_6 correspond to the utterances P_1,C_1,P_2,C_2,P_3, and C_3, respectively.

This example shows the case in which a proposer statically wins but dynamically loses the argumentation tree.



(a) Initial argumentation tree (b) Final argumentation tree

Fig. 6. The argumentation trees starting from the murderer

4.3 Behaviour of Suspend/Resume

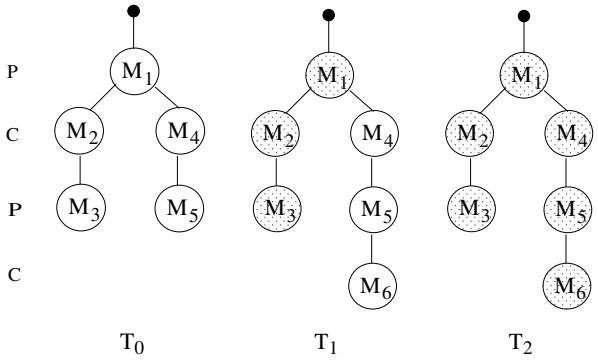
Figure 7(a) shows the trees at each step of *APKC2* procedure in case the left branch of the tree in Figure 6(a) is selected first. T_0 is the initial argumentation tree, and T_1 is the modified tree based on the knowledge bases obtained after the execution of the left branch. In these trees, the hatched nodes are marked. T_2 is the tree modified based on the knowledge bases after the execution of the right branch afterward. No more attacks to the leaf nodes exist. No unmarked node exists in T_2, which indicates the absence of a counterargument. Then, the procedure terminates. The winner is C, who gives the final argument.

Figure 7(b) shows the trees at each step in case the right branch of the tree in Figure 6(a) is selected first. T_1' is the modified tree based on the knowledge bases obtained after the execution of the right branch. The right branch is suspended. T_2' is the modified tree based on the knowledge bases obtained after the execution of the left branch afterward. In this case, a new node M_6, which corresponds to the utterance C_3', is added, and it is the only node that is unmarked in T_2'. To execute this node, the right branch is resumed. T_3' is the modified tree based on the knowledge bases obtained after this execution. No unmarked node exists in T_3'. Then, the procedure terminates. The winner is C, who gives the final argument.

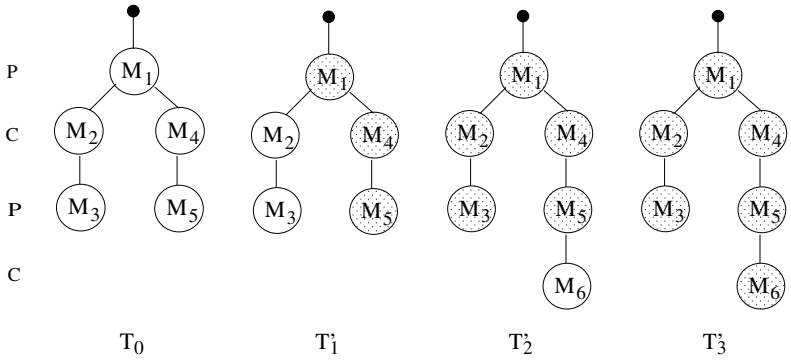
This example shows the procedures with different branch selection orders, and illustrates how suspend/resume occurs.

4.4 The Case of Changing from Static Lose to Dynamic Win

Next, we show an argumentation that starts with the policeman's utterance C_0 in the first example:



(a) APKC2 with left branch selected first



(b) APKC2 with right branch selected first

Fig. 7. Comparison of procedures on the order of selecting branches

C_0: "You committed the murder."

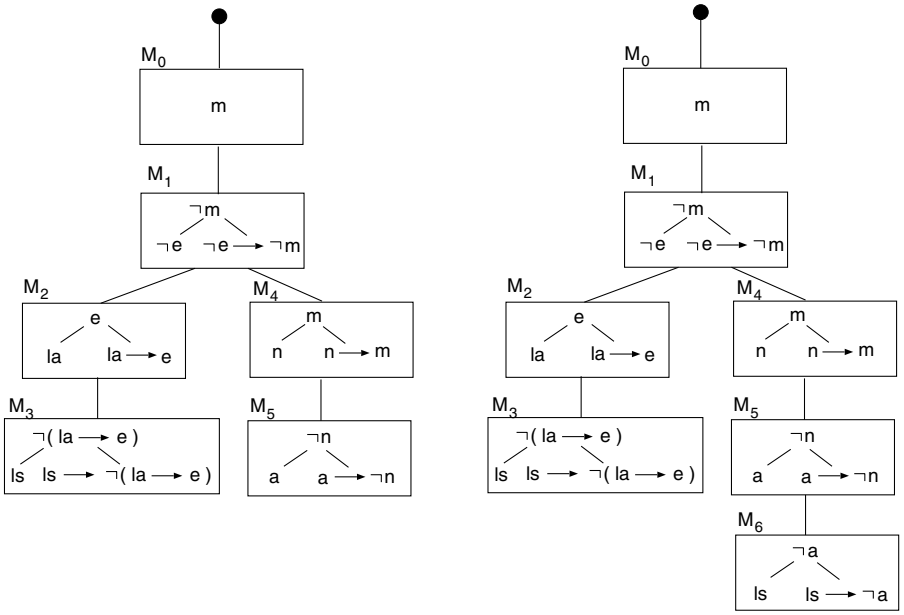
and continues to P_1, C_2, P_3, C_1, P_2, similar to the first example. The argumentation trees are shown in Figure 8. M_0 is a node corresponding to C_0.

The trees can be regarded as C's argumentation trees because the roles of P and C are switched from the first example. C statically loses, since all the leaf nodes in the initial argumentation tree shown in Figure 8(a) are P's move, but dynamically wins, since the final argumentation tree shown in Figure 8(b) is obtained by *APKC2*.

This example shows the case in which a proposer statically loses but dynamically wins the argumentation tree.

4.5 Judgement of Dynamic Win/Lose

Here, we apply an algorithm *JC* to the first example, starting from the murderer's utterance.



(a) Initial argumentation tree (b) Final argumentation tree

Fig. 8. The argumentation trees starting from the policeman

The initial argumentation tree is shown in Figure 6(a). It includes only one candidate subtree[2] and no threats over different candidate subtrees.

Figure 9 shows how *JC* works.

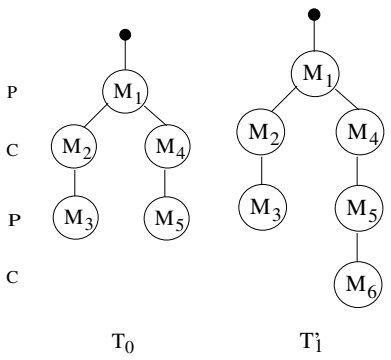


Fig. 9. The argumentation trees for judgement

[2] This figure shows only the relevant part, and it actually contains more candidate subtrees. Although we ignore them to make a description simple, the result is the same.

We take T_0 as a candidate subtree, which is consistent.

First, we obtain $\mathcal{K}$, a set of all formulas in T_0.

$$\mathcal{K} = \left\{ \begin{array}{l} \neg m,\ \neg e,\ (\neg e \rightarrow \neg m),\ \neg(la \rightarrow e), \\ ls,\ (ls \rightarrow \neg(ls \rightarrow e)),\ \neg n,\ a, \\ (a \rightarrow \neg n) \\ e,\ la,\ (la \rightarrow e),\ m,\ n, \\ (n \rightarrow m) \end{array} \right\}$$

Reconstruct the tree, then a new node M_6 is added to obtain the tree T_1', which is consistent. Since one leaf node in T_1' is C's move, P loses this candidate subtree.

Since no other candidate subtrees exist, P dynamically loses the argumentation tree.

5 Discussion

5.1 Improvements on the Earlier Version

Three significant points distinguish the argumentation system proposed in this paper from the earlier version.

First, suspend/resume of a branch is enabled, allowing for the resumption of a settled matter. We mark the executed node instead of deleting it, and make it possible to add a new node to already executed ones. We also provide a simpler judgement algorithm of win/lose for a given candidate subtree. The method of selecting a candidate tree to win can contribute to argumentation strategy.

Second, both agents can continue an argumentation after he/she loses one argumentation line, whilst only the defeater could do so in the earlier version. This makes it possible to handle the case in which a proposer statically loses but dynamically wins.

Third, both P and C can use disclosed knowledge, whereas only C could do so in the earlier version. We prepare the commitment store for this purpose.

Due to these improvements, *APKC2* provides a more natural model for actual argumentations.

In addition, in the earlier version, we had to reconstruct an argumentation tree every time a branch was executed, since some formulas might be deleted from C's knowledge base. However, in the revised version, we do not need to reconstruct a tree, only add nodes to the existing tree, since the usable knowledge is monotonically increasing. This makes the implementation of *APKC2* easier.

5.2 Related Works

García et al. formalized argumentation based on Defeasible Logic Programming (DeLP) [11]. In DeLP, agent's knowledge base consists of two kinds of rules: strict rules and defeasible rules. The result of argumentation is different depending on which defeasible rules are used. Afterwards, Moguillansky discussed revision of

the knowledge base [17]. In his method, after constructing the initial argumentation tree called dialectical tree, knowledge base is changed by extracting defeasible rules and the tree is altered. The goal is to construct undefeated argumentation by selecting suitable defeasible rules. They presented an algorithm for this alteration of the tree and considered a strategy to get the undefeated argumentation. In a series of studies, they formalized several properties in argumentation based on this approach [15]. Similar to our approach, they consider multiple argumentation lines altogether. The different point is that they investigate the effect of the change of knowledge base not considering the change caused by the execution of argumentation, while we focus on the effect of the execution.

Argumentation-based approach is applied to formalize processes appeared in agents communication such as negotiation, persuasion, agreement and so on [14,1,19]. Considering the effect of the execution of arguments, agents communication are rather related issue, since belief of each agent is updated on receiving information from the other agent. Amgoud proposed the protocol that handles arguments and formalized the case in accepting/rejecting new information [1]. She also presented a general framework for argumentation-based negotiation in which agent has a theory and it evolves during a dialogue [2]. She considered the knowledge base for each agent separately, as well as its revision by exchanging arguments. The significant difference between her work and ours is that in her approach only a single argumentation line is considered, so only threats to the same branch are taken into account, whereas in our approach all argumentation lines are considered successively, so threats to the other branches are examined. Dunne proposed a "dispute tree" on which successive execution of all argumentation lines are considered [7]. However, the revision of agents' knowledge base, allowing executed moves to add new information to the opponent's knowledge base, is not considered.

Cayrol studied how acceptable arguments are changed when a new argument is added to an argumentation system based on Dung's framework [5]. The aim of her research is a formal analysis on changes to argumentation, and the contents of the additional arguments and reasons for the addition are beyond its scope. In contrast, we focus specifically on the effect of knowledge gained by executing argumentation.

6 Conclusion

We have proposed an argumentation system *APKC2*, which is an extension of our earlier argumentation system *APKC*. *APKC* is a system in which multiple argumentation lines are executed in succession, and an agent's knowledge base can change during argumentation. We have extended *APKC* so that the suspend/resume of an argumentation line can be processed, both agents can continue an argumentation after he/she loses one argumentation line and both can use information given in previous arguments. These extensions provide a more natural model of actual argumentation. In addition, we proposed a simpler

algorithm for the judgement of the win/lose result of an argumentation tree, and showed that its result is equivalent to that of *APKC2*.

In future, we are considering an extension of *APKC2* that can not only directly use new information, but also derive new facts from the new knowledge. We are also considering a strategy to win an argumentation.

References

1. Amgoud, L., Parsons, S., Maudet, N.: Arguments, dialogue, and negotiation. In: ECAI 2000, pp. 338–342 (2000)
2. Amgoud, L., Dimopoulos, Y., Moraitis, P.: A general framework for argumentation-based negotiation. In: Rahwan, I., Parsons, S., Reed, C. (eds.) Argumentation in Multi-Agent Systems. LNCS (LNAI), vol. 4946, pp. 1–17. Springer, Heidelberg (2008)
3. Amgoud, L., Vesic, S.: Repairing preference-based argumentation frameworks. In: IJCAI 2009, pp. 665–670 (2009)
4. Bench-Capon, T., Dunne, P.: Argumentation in artificial intelligence. Artificial Intelligence 171, 619–641 (2007)
5. Cayrol, C., de St-Cyr, F.D., Lagasquie-Shiex, M.-C.: Revision of an argumentation system. In: KR 2008, pp. 124–134 (2008)
6. Chesñevar, C.I., Maguitman, A., Loui, R.: Logical models of argument. ACM Computing Surveys 32(4), 337–383 (2005)
7. Dunne, P.E., Bench-Capon, T.J.M.: Two party immediate response disputes: properties and efficiency. Artificial Intelligence 149(2), 221–250 (2003)
8. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence 77, 321–357 (1995)
9. Falappa, M., Kern-Isberner, G., Simari, G.R.: Explanations, belief revision and defeasible reasoning. Artificial Intelligence 141(1-2), 1–28 (2002)
10. García, A., Simari, G.: Defeasible logic programming: an argumentative approach. Theory and practice of logic programming 4(1), 95–138 (2004)
11. García, A., Chesevar, C., Rotstein, N., Simari, G.: An abstract presentation of dialectical explanations in defeasible argumentation. In: ArgNMR 2007, pp. 17–32 (2007)
12. Joseph, S., Prakken, H.: Coherence-driven argumentation to norm consensus. In: ICAIL 2009, pp. 58–67 (2009)
13. Hamblin, C.: Fallacies. Methuen (1970)
14. Kraus, S., Sycara, K., Evenchik, A.: Reaching agreements through argumentation: a logical model and implementation. Artificial Intelligence 104(1-2), 1–69 (1998)
15. Lucero, M.J.G., Chesñever, C.I., Simari, G.R.: On the accrual of arguments in defeasible logic programming. In: IJCAI 2009, pp. 804–809 (2009)
16. Modgil, S.: Reasoning about preferences in argumentation frameworks. Artificial Intelligence 173(9-10), 901–1040 (2009)
17. Moguillansky, M.O., et al.: Argument theory change applied to defeasible logic programming. In: AAAI 2008, pp. 132–137 (2008)
18. Prakken, H.: Combining skeptical epistemic reasoning with credulous practical reasoning. In: COMMA 2006, pp. 311–322 (2006)

19. Paglieri, F., Castelfranchi, C.: Revising beliefs through arguments: Bridging the gap between argumentation and belief revision in MAS. In: Rahwan, I., Moraïtis, P., Reed, C. (eds.) ArgMAS 2004. LNCS (LNAI), vol. 3366, pp. 78–94. Springer, Heidelberg (2005)
20. Okuno, K., Takahashi, K.: Argumentation with a revision of knowledge base. In: EUMAS 2008, CD-ROM (December 2008)
21. Okuno, K., Takahashi, K.: Argumentation system with changes of an agent's knowledge base. In: IJCAI 2009, pp. 226–232 (2009)
22. Rahwan, I., Simari, G. (eds.): Argumentation in Artificial Intelligence. Springer, Heidelberg (2009)