A Semantics for Dynamic Argumentation Frameworks

Kazuko Takahashi and Yu Nambu

School of Science & Technology, Kwansei Gakuin University,
2-1, Gakuen, Sanda, 669-1337, Japan
ktaka@kwansei.ac.jp, jammy_jam_up@yahoo.co.jp

Abstract. This paper presents a semantics for dynamic argumentation frameworks. A dynamic argumentation system involves the concept of execution of an argumentation affecting subsequent arguments. Although such dynamic treatment is necessary to grasp the behavior of actual argumentation, semantics proposed to date can only handle the static aspects. Here, we present a new semantics that fits dynamic argumentation. We discuss what properties hold and explain how to compute changes in the set of acceptable arguments, depending on the presentation order of arguments.

1 Introduction

Argumentation is a powerful tool that enables the formal treatment of interactions, such as negotiation and agreement, among agents. There have been many studies of argumentation systems [4,21].

An argumentation framework is usually defined as a pair $\langle Args, Atts \rangle$, where $Args$ is a set of arguments, and $Atts$ is a binary relation over $Args$ that indicates an attack by one argument on another. Most argumentation systems developed to date analyze a given argumentation framework statically. They consider argumentation theory as fixed or focus on the selection of a specific argumentation theory that will result in the acceptance of a particular proposal. These systems are based on the assumption that arguing agents have a common knowledge base and can survey all possible arguments. However, knowledge bases actually differ between agents, so as each argument is presented, new information is added to modify the subsequent argumentation. We have developed a dynamic argumentation system, *"the Argumentation Procedure with Knowledge Change (APKC),"* in which argumentation theory changes depending on the execution [19], and its extended version, APKC2 [20]. Our goal was to capture the behavior of actual argumentation with greater precision. The proposed system is based on the concept of "execution" of an argument. We investigated the phenomenon in which new information is added by a presented argument, and this generates a new attack.

In APKC2, an argumentation continues over multiple argumentation lines. We demonstrated that the results may differ depending on the order of execution.

P. McBurney et al. (Eds.): ArgMAS 2011, LNAI 7543, pp. 66–85, 2012.

We also proposed a judgment algorithm, JC, which can determine which agent wins without actually simulating each execution individually [20]. Although this previous work investigated simulation and judgment in dynamic argumentation, it did not clarify the meaning of each execution and the relationships between executions. In this paper, we present a new semantics to fit the dynamic argumentation system.

A semantics for an argumentation framework is usually given with the notion of extension [11], i.e., a set of arguments that can be accepted together within a given argumentation framework. However, in dynamic argumentation, arguments and attacks change as the argumentation proceeds. Therefore, a semantics in which acceptability is defined for a static argumentation is not suitable for dynamic argumentation. In this paper, we present a separate extension for each execution as an acceptable set of arguments for that execution. An extension for a dynamic argumentation framework is defined as the set of these individual extensions. In addition, we discuss how these extensions are changed as argumentation proceeds and investigate their interrelationships and properties.

The remainder of this paper is organized as follows. In section 2, we explain our motivation. In section 3, after presenting basic concepts, such as argumentation frameworks, we present a dynamic argumentation system. In section 4, we define the semantics for dynamic argumentation frameworks, and show the rules by which the revision of extensions is computed. In section 5, we compare our approach to those used in related studies. Finally, in section 6, we present our conclusions.

2 Informal Description for Dynamic Argumentation

In general, argumentation involves two agents taking turns presenting arguments to attack their respective opponent's argument until one is no longer able to attack. Finally, the loser accepts the winner's proposal. This process is usually represented in the form of a tree [1,13]. The root node is a proposal statement, and each branch corresponds to a single argumentation line, i.e., a sequence of arguments. In a dynamic argumentation system [20], an argumentation proceeds along each branch. Once an argument is presented, the corresponding node is marked as "executed" and never reappears in the series of argumentation. If there is no executable node in the current branch, then another branch that has an executable node is selected. Finally, the agent that cannot make a counterargument loses the argumentation. An important feature of this system is the concept of a "threat." This refers to a case in which the execution of an argument results in the creation of a new counterargument to another argument. Intuitively, a threat is an argument that may provide information advantageous to the opponent. It changes the argumentation and affects the win/loss outcome.

For example, consider the argumentation tree shown in Figure 1(1). In this figure, P_i and C_i show the argument of a proposer (P) and a defeater (C),

respectively. If we execute the argumentation from the left branch, after P_0, C_1, and P_1 are executed, C_2 and P_2 are executed, and P wins. If we execute from the right branch, after P_0, C_2, and P_2 are executed, C_1 and P_1 are executed, and P also wins. Now, consider the argumentation tree shown in Figure 1(2), which has a threat from C_1 to C_2. This means that execution of C_1 causes the creation of P_2, a new counterargument to C_2. If we execute an argumentation from the left branch, after P_0, C_1, and P_1 are executed, P_2 is generated. Then, C_2 and P_2 are executed and the execution finally terminates with P winning. In contrast, if we execute the argumentation from the right branch, after P_0 and C_2 are executed, the execution terminates because C has the next turn, but no branch is available that can start with C's argument. In this case, C wins. Note that P_2 does not occur until execution of C_1. This example illustrates two important issues that must be addressed: (i) the winner of an argumentation differs depending on the order of execution of the branches, and (ii) it is not appropriate to handle a revised tree in the same way as one that consists of the same nodes and edges without a threat.

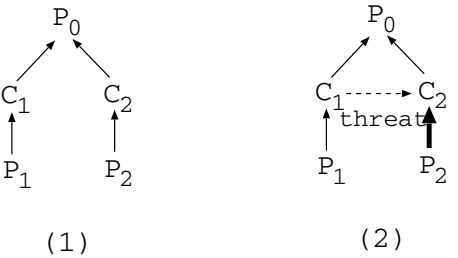

Fig. 1. Effects of a threat

3 Dynamic Argumentation System

3.1 Basic Concepts

In a dynamic argumentation, P and C have their own knowledge bases, which may have common elements. We construct a dynamic argumentation framework from given knowledge bases of agents and preference [19]. Preference is defined in advance for each formula in the knowledge base. The preference of each argument is used so that attack is possible only from an argument with a high preference to an argument with a lower preference. Here, we do not explain preference in detail, as it is beyond the scope of this paper.

Definition 1 (argument). *Let Arg_a be a knowledge base for an agent a.* An argument of a *is a pair (Ψ, ψ) where Ψ is a subset of Arg_a, and $\psi \in Arg_a$ such that Ψ is the empty set or $\Psi = \{\phi, \phi \Rightarrow \psi\}$. Ψ and ψ are said to be* the grounds *and* the statement *of the argument, respectively.*

Definition 2 (argumentation framework). *An argumentation framework is defined as a triple* $\langle Arg_P, Arg_C, Atts \rangle$*, where* Arg_P *and* Arg_C *are sets of P's arguments and C's arguments, respectively, Atts is a binary relation called* attack *over* $Arg_P \cup Arg_C$*, where for each* $(A, B) \in Atts$*, either* $A \in Arg_P, B \in Arg_C$*, or* $A \in Arg_C, B \in Arg_P$ *holds. For each pair of arguments* A, B*, both* (A, B) *and* (B, A) *are never contained in Atts at the same time.*

Definition 3 (argumentation tree). *Let* φ *be a proposal statement, and let P and C be a proposer and a defeater of* φ*, respectively. Let AF be an argumentation framework* $\langle Arg_P, Arg_C, Atts \rangle$*. Then, an argumentation tree for AF on* φ *is defined as follows [1].*

- *This is a finite directed tree, the root node of which corresponds to an argument of which the statement is* φ [1]
- *Every node corresponds to an argument in* $Arg_P \cup Arg_C$*.*
- *Every edge from node N to M corresponds to an attack from an argument corresponding to N to that corresponding to M.*

Here, we call a path from the root node to a leaf node *a branch*. P's argument and C's argument appear in turn in each branch. The same arguments may be present in different branches; hence, it follows that each node has a unique parent node. There is no loop in each branch due to the preference constraint.

Definition 4 (win of a branch). *If the leaf of a branch D is P's argument, then P* wins *D; otherwise, P* loses *D.*

Definition 5 (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 argument in the original tree and selects all child nodes for each node corresponding to P's argument.*

Definition 6 (solution subtree). A solution subtree *is a candidate subtree in which P wins all of the branches in the tree.*

Example 1. *For an argumentation tree shown in Figure 2(1), Figure 2(2) and Figure 2(3) are its candidate subtrees and Figure 2(2) is a solution subtree.*

In most argumentation systems, the win/loss of an argumentation is defined by handling each branch independently. However, in a dynamic argumentation system, another branch may continue to be executed after all arguments of one branch are executed. In this case, arguments disclosed so far in one line affect arguments in another line. This may create a new argument and change the

[1] In general, there may exist multiple arguments of which the sentence is φ with different grounds in Arg_P. Therefore, the root is considered an empty argument, and the arguments to support φ should be considered its child nodes [19]. To simplify, we consider a simple version by assuming that there exists only one such argument and taking it as the root node.

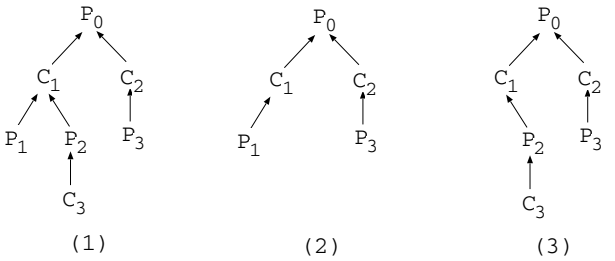

Fig. 2. Candidate subtrees

winner of the argumentation. This is the most characteristic feature of dynamic argumentation systems.

If $\phi \Rightarrow \psi$ and ψ are contained in P's knowledge base, while ϕ is not, then P cannot make an argument $(\{\phi, \phi \Rightarrow \psi\}, \psi)$. However, if ϕ is contained in C's knowledge base and once it is disclosed, P can use ϕ to generate this argument, which may be a new counterargument to C. We define such a case as *a threat*.

Definition 7 (threat). *Let A and A' both be arguments in Arg_P or in Arg_C. If A generates more than one new argument that attacks A', then it is said that there is* a threat from A to A', *and Arg_P/Arg_C* contains a threat. *A and A' are* a threat resource *and* a threat destination, *respectively, which is denoted by $threat(A, A')$.*

Intuitively, a threat is an argument that may provide information advantageous to the opponent. An argument may be a threat to another argument in the same branch.

3.2 Execution of an Argumentation

Here, we present a dynamic argumentation system.

Both agents have their own knowledge bases. A set of all of the formulas contained in all of the arguments given so far is stored in a commitment store [15].

First, for a given argumentation framework, we construct an initial argumentation tree in which all nodes are unmarked. An argumentation starts by selecting a branch of an initial argumentation tree. It proceeds along the branch with marking of the nodes, and when the execution reaches the leaf node, the branch is suspended. At that time, the nodes in the branch are added to the commitment store.

Next, another branch is selected. The branch containing unmarked nodes can be selected. The suspended branch may be resumed if a new unmarked node is added to it. Upon selection of a branch, the utterance turns should be kept. This means that if one branch is suspended at the node that corresponds to one

agent's argument, then the next branch should start with the node that corresponds to the other agent's argument. Agents can make new arguments using the commitment store in addition to their own knowledge bases. Therefore, the numbers of arguments and attacks increase in accordance with the execution of each branch. New nodes are added to the argumentation tree if new arguments are generated.

We show formal definitions in Figure 3 and Figure 4.

Definition 8 (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, $M_i, \ldots, M_n$ are unmarked, and M_i is t's argument, then the node M_i is* executable.

Definition 9 (suspend/resume). *After the execution of all nodes in a branch, D is* 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* resumed.

Execution of a branch from a specific node M_i ($1 \leq i \leq n$)

Let $D = [M_1, \ldots, M_n]$ be a branch and **K** be the commitment store.

1. Mark $M_i, \ldots, M_n$.
2. Update **K** by adding all of the formulas contained in arguments $M_i, \ldots, M_n$.
3. **if** M_n is P's argument,
   then set the current turn to C.
   if M_n is C's argument,
   then set the current turn to P.

Fig. 3. Execution of a branch

In APKC2, both agents present arguments in turn, and the agent that cannot give a counterargument loses the argumentation. An execution is determined on a certain order of selecting branches.

Proposition 1. *[19] (1) Any execution of APKC2 terminates in a finite time, and its winner is decidable.*

(2) The number of executions for an argumentation tree is finite.

Definition 10 (execution tree). *For an argumentation framework, a subtree of the tree finally obtained as a result of APKC2 along an execution exec, which consists of the marked nodes and the edges between them is called* an execution tree for *exec.*

Example 2. *Consider the argumentation tree shown in Figure 5(1), where $threat(C_1, C_2)$ exists. Let $exec_1$ and $exec_2$ be executions in which the left or right branch is executed first, respectively. Then, the execution trees for $exec_1$ and $exec_2$ are shown in Figure 5(2) and Figure 5(3), respectively.*

Argumentation Procedure with Knowledge Change (*APKC2*)

Let $AF = \langle Arg_P, Arg_C, Atts \rangle$ be an argumentation framework, φ be a proposed statement, and $\mathbf{K}$ be the commitment store.

[STEP 1 (initialization)]
Set $\mathbf{K} = \emptyset$.Construct an initial argumentation tree for AF on φ with all nodes unmarked.

[STEP 2 (execution of an argumentation)]
if no branch has an executable node,
  if the current turn is P, **then** terminate with P's loss.
  else the current turn is C, **then** terminate with P's win.
else select a branch and execute it from the executable node to the leaf node.

[STEP 3 (modification of a tree)]
For a pair of arguments $A, A' \in Arg_P/Arg_C$ such that $threat(A, A')$ holds,
if A is marked,
  then add a new argument B to Arg_C/Arg_P, respectively,
    and add a new attack (B, A') to $Atts$ and modify the tree.
if the nodes N and M are identical, and N is marked while M is unmarked,
  then mark M.
go to STEP 2.

Fig. 4. Argumentation Procedure with Knowledge Base (APKC2)

Definition 11 (win/loss execution). *If APKC2 along an execution terminates with P's win/loss, then* P wins/loses the execution.

Example 3. *(Cont'd) P wins $exec_1$ and loses $exec_2$ in Example 2.*

Definition 12 (continuous candidate subtree). *For a candidate subtree CT, if more than one candidate subtree is generated by the addition of nodes, then these subtrees are said to be continuous candidate subtrees of CT.*

Definition 13 (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 candidate subtree such that P wins, then CT is* a dynamic solution subtree.

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

Example 4. *(Cont'd) P loses the argumentation shown in Figure 5(1).*

For an argumentation framework $\langle Arg_P, Arg_C, Atts \rangle$, let T_{init} be an initial argumentation tree, and let T_{exec} be an execution tree for an execution $exec$. If there is no threat in Arg_P and Arg_C, then for any execution $exec$, $T_{exec} \subseteq T_{init}$.

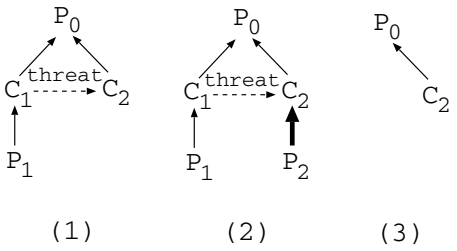

Fig. 5. An argumentation tree (1) and the execution trees (2)(3)

4 Semantics

4.1 Extensions

Following the definition set out by Dung [11], we can define the following concepts related to arguments.

Definition 15 (conflict-free, admissible). *For an argumentation framework* $AF = \langle Arg_P, Arg_C, Atts \rangle$*, let* $A \in Arg_P \cup Arg_C$ *and* $\mathcal{S} \subseteq Arg_P \cup Arg_C$*.*
(1) $\mathcal{S}$ *is* conflict-free *iff there are no elements* $A, B \in \mathcal{S}$ *such that* A *attacks* B*.*
(2) $\mathcal{S}$ defends A *iff* $\mathcal{S}$ *attacks each argument that attacks* A*. The set of arguments that* $\mathcal{S}$ *defends is denoted by* $\mathcal{F}(\mathcal{S})$*.* $\mathcal{F}$ *is called* the characteristic function of an argumentation framework $\langle Arg_P, Arg_C, Atts \rangle$*.*
(3) $\mathcal{S}$ *is* admissible *iff* $\mathcal{S}$ *is conflict-free and defends all of the elements.*

There are several definitions of acceptability, and different extensions exist for each acceptability.

Definition 16 (extensions). *Let* $\mathcal{E} \subseteq Arg_P \cup Arg_C$*.*
(1) $\mathcal{E}$ *is* a preferred extension *iff* $\mathcal{E}$ *is maximal w.r.t.* $\subseteq$ *admissible set.*
(2) $\mathcal{E}$ *is* a grounded extension *iff* $\mathcal{E}$ *is the least fixed point w.r.t.* $\subseteq$ *of the characteristic function* $\mathcal{F}$*.*
(3) $\mathcal{E}$ *is* a stable extension *iff* $\mathcal{E}$ *is conflict-free and attacks each argument that is not included in* $\mathcal{E}$*.*

The following relations hold among these extensions.

Proposition 2. *[11,10] (1) There is at least one preferred extension, always a unique grounded extension, and there may be zero, one, or many stable extensions.*
(2) If there is no cyclic structure in an argumentation framework, then there is a unique stable extension, and the three extensions coincide.

4.2 Dynamic Extension

For an argumentation framework, let T_{exec} be an execution tree for an execution *exec*. Let Arg'_P and Arg'_C be a set of P's and C's arguments in T_{exec}, respectively,

and $Atts'$ be a set of attacks between these arguments. Then, T_{exec} is an argumentation tree for an argumentation framework $AF_{exec} = \langle Arg'_P, Arg'_C, Atts' \rangle$. We call such AF_{exec} *an argumentation framework for exec.*

Definition 17 (dynamic extension). *For an argumentation framework AF and its execution exec, let AF_{exec} be an argumentation framework for exec. Then, the preferred extension for AF_{exec} is* dynamic extension for *exec* of *AF, and a set of all of the dynamic extensions for executions of AF is* the dynamic extension for *AF.*

For a given execution *exec*, we can construct a dynamic extension $\mathcal{E}_{exec}$ for *exec* from the corresponding execution tree. For each node, we determine whether it is included in a dynamic extension by exploring the execution tree from the leaf nodes in a bottom-up manner using the following rule (Figure 6).

Judgment for inclusion of each node by $\mathcal{E}_{exec}$
(1) A leaf node is in $\mathcal{E}_{exec}$.
(2) The node of which all child nodes are not in $\mathcal{E}_{exec}$ is in $\mathcal{E}_{exec}$.
(3) The node of which child nodes include at least one node that is in $\mathcal{E}_{exec}$ is not in $\mathcal{E}_{exec}$.

Fig. 6. Judgment for inclusion of each node

Example 5. *(Cont'd) In Figure 5, the argumentation framework for $exec_1$ is $AF_{exec_1} = \langle \{P_0, C_1, P_1, C_2, P_2\}, \{(C_1, P_0), (P_1, C_1), (C_2, P_0), (P_2, C_2)\} \rangle$, and the dynamic extension for $exec_1$ is $\mathcal{E}_{exec_1} = \{P_0, P_1, P_2\}$. Those for $exec_2$ are $AF_{exec_2} = \langle \{P_0, C_2\}, \{(C_2, P_0)\} \rangle$ and $\mathcal{E}_{exec_2} = \{C_2\}$, respectively. The dynamic extension for AF is $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2\}$.*

Proposition 3. *Let T_1 and T_2 be execution trees for executions $exec_1$ and $exec_2$ in AF, respectively, and $\mathcal{E}_1$ and $\mathcal{E}_2$ be dynamic extensions for $exec_1$ and $exec_2$, respectively. If T_1 is a subtree of T_2 such that $T_1 \neq T_2$, then $\mathcal{E}_1 \subset \mathcal{E}_2$.*

Proof) Let D_1 and D_2 be branches in an argumentation tree for AF. Also, let $exec_1$ be an execution in which branches are executed in the order of $D_1 D_2$, and let $exec_2$ be an execution in the order of $D_2 D_1$. Assume that the number of nodes included in D_1 except for the root node is even. Then, the leaf node of D_1 is P's argument. Therefore, after D_1 is executed, D_2 should be executed. In this case, T_1 should not be a subtree of T_2. Then, the number of nodes included in D_1 is odd. Therefore, $\mathcal{E}_1$ does not include the root node. Moreover, for any node N in D_1 other than the root node, it is obvious that if $N \in \mathcal{E}_1$, then $N \in \mathcal{E}_2$ holds. Thus, $\mathcal{E}_1 \subset \mathcal{E}_2$.

Definition 18 (minimal dynamic extension). *Let $\mathcal{E}_1 \ldots, \mathcal{E}_n$ be dynamic extensions for executions of AF. If $\mathcal{E}_i$ such that $\mathcal{E}_i \subset \mathcal{E}_j$ $(i \neq j)$ does not exist, then $\mathcal{E}_j$ is* a minimal dynamic extension for *AF.*

The characteristics of dynamic extensions depend on which agent is in the leaf node, which agent has a threat, and/or which part of a branch a threat appears. We first discuss the characteristics of extensions in cases without a threat, and then investigate how they change with the effect of a threat.

4.3 Case in Which No Threat Exists

First, we explain the case in which both Arg_P and Arg_C contain no threats.

Let $AF = \langle Arg_P, Arg_C, Atts \rangle$ be an argumentation framework and T be an initial argumentation tree for AF.

Let $\mathcal{D}_P$ and $\mathcal{D}_C$ be sets of branches in which the leaf nodes of T are P's arguments and C's arguments, respectively. Let $|\mathcal{D}_P| = n$ and $|\mathcal{D}_C| = m$. APKC2 proceeds by selecting a branch with an executable node from $\mathcal{D}_P \cup \mathcal{D}_C$ in an arbitrary order.

One Candidate Subtree. When an argumentation tree has one candidate subtree, the result is rather simple.

Considering that APKC2 proceeds by turn of P and C, we can classify argumentation trees into three types by focusing on the leaf nodes.

(1) All leaf nodes are P's arguments.

In this case, all branches $D_P^1, \ldots, D_P^n$ in $\mathcal{D}_P$ can be executed in an arbitrary order. Then, dynamic extensions for all executions consist of all of P's nodes appearing in T, and they coincide with each other. They include the root node. Therefore, a dynamic extension for AF is a singleton.

(2) All leaf nodes are C's arguments.

In this case, only one branch, D_C^j $(1 \leq j \leq m)$ in $\mathcal{D}_C$, can be executed. Then, a dynamic extension for each execution $\mathcal{E}_j$ consists of all of C's nodes in D_C^j. Therefore, a dynamic extension for AF is $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$. Each $\mathcal{E}_j$ contains only C's nodes and is a minimal dynamic extension. Moreover, their intersection is an empty set.

(3) Leaf nodes consist of both P's argument and C's arguments.

In this case, a branch D_C^j $(1 \leq j \leq m)$ in $\mathcal{D}_C$ is executed after executing several branches $D_P^1, \ldots, D_P^k$ $(1 \leq k \leq n)$ in $\mathcal{D}_P$, or executing no other branches. Then, a dynamic extension $\mathcal{E}_{kj}$ for each execution consists of C's nodes in D_C^j and all of P's nodes in $D_P^1 \cup \ldots \cup D_P^k$ that are not in D_C^j, irrespective of the execution order of $D_P^1, \ldots, D_P^k$.

Let N be $\sum_{k=0}^{n} {}_nC_k$, i.e., the sum of the number of all possible combinations of selecting an arbitrary number of elements from $\mathcal{D}_P$. Then, a dynamic extension for AF is $\mathcal{E} = \cup_{0 \leq i \leq N, 1 \leq j \leq m} \{\mathcal{E}_{ij}\}$.

Proposition 4. *For the above three cases, the number of minimal dynamic extensions can be defined as follows.*
(1) There exists a unique minimal dynamic extension.
(2) There exist $|\mathcal{D}_C|$ minimal dynamic extensions.
(3) There exist $|\mathcal{D}_C|$ minimal dynamic extensions.

Moreover, because P only wins in case (1), the following property holds.

Proposition 5. *For an argumentation framework, if there is no threat, if the argumentation tree has only one candidate tree, and if all of its leaf nodes are P's arguments, then there never occurs a case in which P wins in one execution and loses in another execution.*

Multiple Candidate Subtrees. When an argumentation tree has multiple candidate subtrees, the result is rather complicated.

(1) All leaf nodes are P's arguments.

In this case, all branches in $\mathcal{D}_P$ that belong to a single candidate subtree CT can be executed in an arbitrary order. Then, dynamic extensions for all executions consist of all of P's nodes appearing in CT, and they coincide with each other. Therefore, a dynamic extension for an argumentation framework is a set of these extensions. All of them include the root node.

(2) All leaf nodes are C's arguments.

In this case, only one branch in $\mathcal{D}_C$ of each candidate subtree can be executed. A dynamic extension for each execution consists of only C's nodes. All of them are disjoint.

(3) Leaf nodes consist of both P's argument and C's arguments.

In this case, several branches both in $\mathcal{D}_P$ and in $\mathcal{D}_C$ can be executed, as long as the turn is kept. A dynamic extension for each execution may include P's node and C's node.

4.4 The Effect of a Threat from P's Argument to P's Argument

For an argumentation framework $AF = \langle Arg_P, Arg_C, Atts \rangle$, if at least one of Arg_P and Arg_C contains a threat, the threat affects the execution of an argumentation. We can explore the effect in detail by investigating how the dynamic extensions of argumentations with and without a threat differ in each pattern of the initial argumentation tree.

For simplicity, we discuss only the case in which an argumentation tree has only one candidate subtree that has a threat. We also assume that an initial argumentation tree has only two branches: D_1, which includes a threat resource, and D_2, which includes a threat destination. However, the procedure shown here is applicable to an arbitrary argumentation tree insofar as it has no threat over multiple candidate subtrees.

First, we focus on the case in which a threat from P's argument to P's argument is contained in AF.

Let P_r and P_d be a threat resource and a threat destination, respectively, and let C' be a new node generated by this threat. C' is added either to the leaf node or a mid-node of a branch D_2 by a threat. Let $\mathcal{A}_i$ be a maximal admissible set for AF of which each element is in D_i. It is revised by using the above rule of *judgment for inclusion of each node by* $\mathcal{E}_{exec}$. $UPDATE(D_i)$ shows its result.

Hereafter, we use the following notation.

T_0: an initial argumentation tree for AF
$exec_1$: execution along the order D_1D_2
$exec_2$: execution along the order D_2D_1
T_i: execution tree for $exec_i$ without a threat
$\mathcal{E}$: the dynamic extension for AF without a threat
$\mathcal{A}_i$: the maximal admissible set for AF each element of which is in D_i
T'_i: execution tree for $exec_i$
$\mathcal{E}'_i$: dynamic extension for $exec_i$
$\mathcal{E}'$: the dynamic extension for AF
N_C: the lowest node belonging to both D_1 and D_2
$uppereq(N)$: a set of nodes higher than or equal to N
$lower_i(N)$: a set of nodes lower than N in D_i

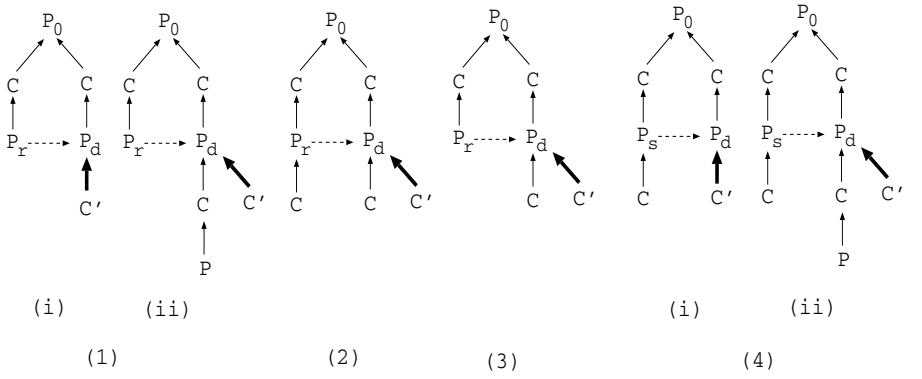

Fig. 7. The effect of P's threat

We can derive $\mathcal{E}'$ from T_0 and $threat(P_r, P_d)$. We compare execution trees with and without a threat, and discuss how their dynamic extensions change.

(P1) All leaf nodes in T_0 are P's arguments (Figure 7(1))

In this case, T_1 and T_2 are equivalent.

If T_0 has a threat, the execution trees are changed.

There may be two cases depending on the position of P_d.
(i) P_d is a leaf node.
C' is added as a leaf of D_2

In execution $exec_1$, D_1 is executed first, C' is added, then D_2 is executed. In execution $exec_2$, D_2 is executed first, it suspends at P_d, then D_1 is executed. Subsequently, C' is added, and D_2 is resumed. Finally, the execution trees of $exec_1$ and $exec_2$ are equivalent.

$T'_1 = T_1 \cup \{C'\}$. $T'_2 = T_2 \cup \{C'\}$.

The addition of a new node C' causes a change in the extensions.

$\mathcal{E}'_1 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_2) \cup \{C'\}$. $\mathcal{E}'_2 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_2) \cup \{C'\}$.

$\mathcal{E}' = \{\mathcal{E}'_1\}$.

Example 6. *Figure 8 shows the case of (P1)(i).*

Figure 8(1) shows an initial argumentation tree T_0. D_1 *and* D_2 *denote the left branch and the right branch.* $\mathcal{A}_1 = \{P_0, P_r\}$. $\mathcal{A}_2 = \{P_0, P_d\}$. *Figure 8(2) shows the execution tree* T_1 $(= T_2)$ *for execution without a threat.*

In contrast, Figure 8(3) shows the execution tree T'_1 $(= T'_2)$ *for execution with a threat from* P_r *to* P_d *to generate a new node* C'. $\mathcal{E}'_1$ *is obtained by updating* D_2. *Since* $UPDATE(D_2) = \{C_2\}$, $\mathcal{E}'_1 = \mathcal{E}'_2 = \{C', C_2, P_r\}$. *Therefore, the dynamic extension is* $\mathcal{E} = \{\mathcal{E}'_1\}$.

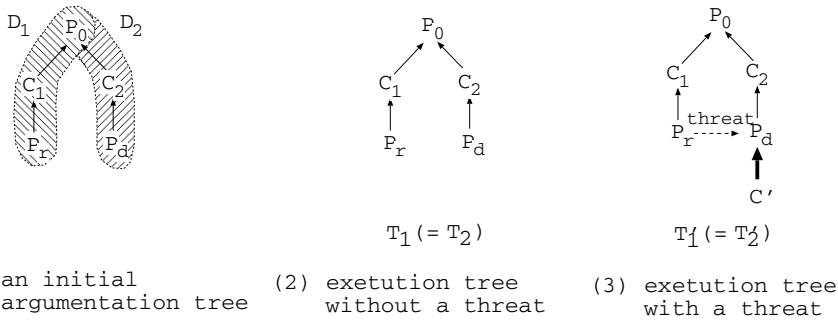

Fig. 8. Change of trees and extensions: (P1)(i)

(ii) P_d is a mid-node.
C' is added as a child node of P_d to generate a new branch D_3. In this case, three executions are possible: the orders of which are $D_1D_2D_3$, $D_2D_1D_3$, and D_1D_3. A new execution $exec_3$ is generated.

$T'_1 = T_1 \cup \{C'\}$. $T'_2 = T_2 \cup \{C'\}$. $T'_3 = T_1 \setminus lower_2(P_d) \cup \{C'\}$.

$\mathcal{E}'_1 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_2) \cup \{C'\}$. $\mathcal{E}'_2 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_2) \cup \{C'\}$. $\mathcal{E}'_3 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_3)$.

$\mathcal{E}' = \{\mathcal{E}'_1, \mathcal{E}'_3\}$.

Example 7. *Figure 9 shows the case of (P1)(ii).*

Figure 9(1) shows an initial argumentation tree T_0. D_1 *and* D_2 *denote the left branch and the right branch.* $\mathcal{A}_1 = \{P_0, P_r\}$. $\mathcal{A}_2 = \{P_0, P_d, P_1\}$. *Figure 9(2) shows the execution tree* T_1 $(= T_2)$ *for execution without a threat.*

In contrast, Figure 9(3) shows the execution trees T'_1 $(= T'_2)$ *and* T'_3 *for executions with a threat from* P_r *to* P_d *to generate a new node* C'. *In* T'_1 *(=*T_2*), since* $UPDATE(D_2) = \{P_1, C_2\}$, $\mathcal{E}'_1 = \mathcal{E}'_2 = \{P_r, P_1, C_2, C'\}$, *In* T'_3, *since* $UPDATE(D_3) = \{C', C_2\}$, $\mathcal{E}'_3 = \{P_r, C', C_2\}$.

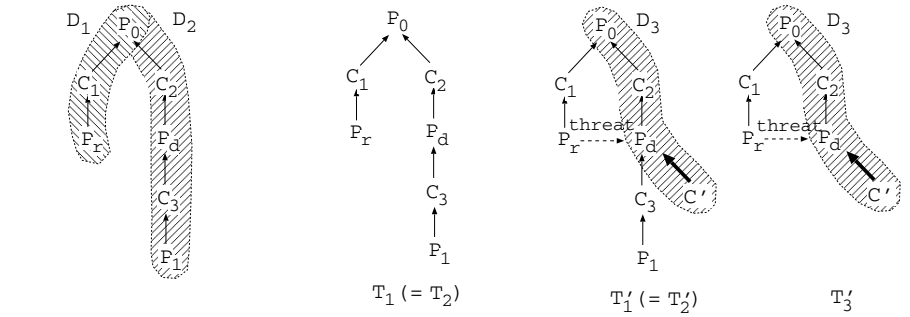

Fig. 9. Change of trees and extensions: (P1)(ii)

(P2) All leaf nodes in T_0 are C's arguments (Figure 7(2))

C' is added as a child node of P_d, and a new branch D_3 is added. However, C' is never executed in any execution because of the constraint of turn keeping. As a result, the dynamic extension does not change, $\mathcal{E}' = \mathcal{E}$.

(P3) D_1's leaf node is P's argument, D_2's leaf node is C's argument (Figure 7(3))

C' is added as a child node of P_d, and a new branch D_3 is added. A new execution $exec_3$ is generated.

$T'_1 = T_1$. $T'_2 = T_2$. $T'_3 = T_1 \setminus lower_2(P_d) \cup \{C'\}$.

In this case, the dynamic extensions are as follows.

$\mathcal{E}'_1 = \mathcal{A}_1 \cup \mathcal{A}_2$. $\mathcal{E}'_2 = \mathcal{A}_2$. $\mathcal{E}'_3 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_3)$.

$\mathcal{E}' = \{\mathcal{E}'_1, \mathcal{E}'_2, \mathcal{E}'_3\}$.

Note that the selected branch must be executed as far as possible, and a node in the other branch cannot be executed at an arbitrary time.

Example 8. *Figure 10 shows the case of (P3).*

Figure 10(1) shows an initial argumentation tree T_0. D_1 *and* D_2 *denote the left branch and the right branch.* $\mathcal{A}_1 = \{P_r\}$. $\mathcal{A}_2 = \{C_3, C_2\}$. *Figure 10(2) shows execution trees* T_1 *and* T_2 *without a threat.*

In contrast, Figure 10(3) shows execution trees T_1', T_2' and T_3' with a threat from P_r to P_d to generate a new node C'. Note that a new execution $exec_3$ is generated. $\mathcal{E}_1' = \mathcal{A}_1 \cup \mathcal{A}_2 = \{P_r, C_3, C_2\}$. $\mathcal{E}_2' = \mathcal{A}_2 = \{C_3, C_2\}$. $\mathcal{E}_3'$ is obtained by updating D_3. Since $UPDATE(D_3) = \{C', C_2\}$, $\mathcal{E}_3' = \{P_r, C', C_2\}$. Therefore, the dynamic extension is $\mathcal{E} = \{\mathcal{E}_1', \mathcal{E}_2', \mathcal{E}_3'\}$.

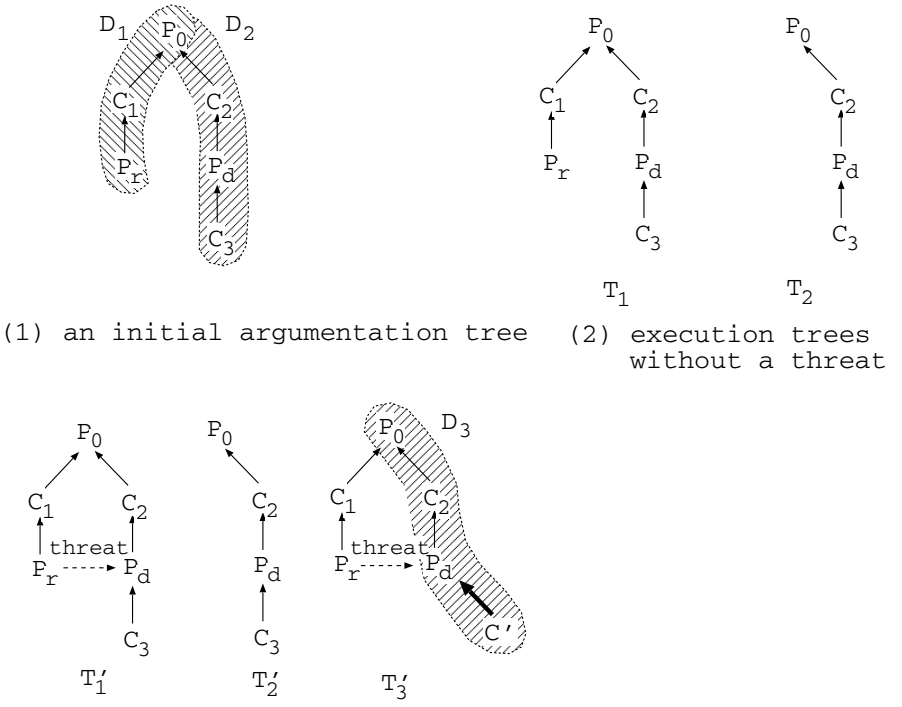

Fig. 10. Change of trees and extensions: (P3)

(P4) D_1's leaf node is C's argument, D_2's leaf node is P's argument (Figure 7(4))

There are two possible cases, depending on the position of P_d: (i) P_d is a leaf node and (ii) P_d is a mid-node.

However, C' is never executed in any execution because of the constraint of turn keeping. As a result, the dynamic extension does not change, $\mathcal{E}' = \mathcal{E}$ in either case.

4.5 The Effect of a Threat from C's Argument to C's Argument

Next, we focus on the case in which a threat from C to C is contained in AF.

Let C_r and C_d be a threat resource and threat destination, respectively, and let P' be a new node generated by this threat.

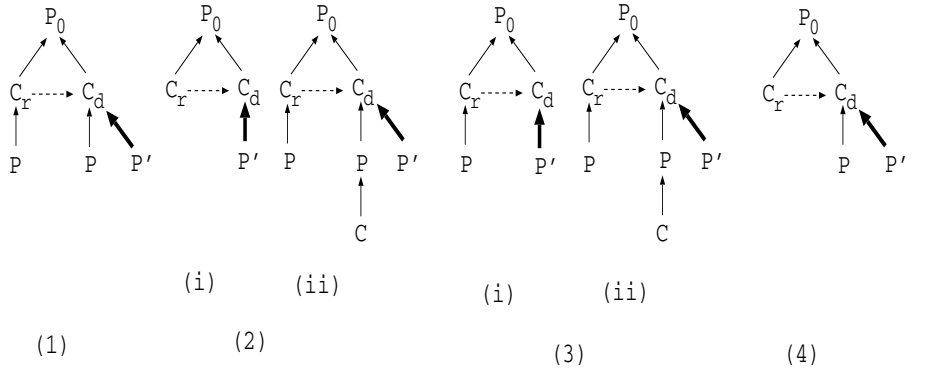

Fig. 11. The effect of C's threat

(C1) All leaf nodes in T_0 are P's arguments (Figure 11(1))

P' is added as a child node of C_d to generate a new branch D_3.

$T_1' = T_1$. $T_2' = T_2$. $T_3' = T_1 \setminus lower_2(C_d) \cup \{C'\}$.

In this case, a new execution $exec_3$ is generated. Three executions are possible: D_1D_2, D_2D_1 and D_1D_3. Dynamic extensions for these executions are as follows.

$\mathcal{E}_1' = \mathcal{E}_2' = \mathcal{A}_1 \cup \mathcal{A}_2$. $\mathcal{E}_3' = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_3)$.

$\mathcal{E}' = \{\mathcal{E}_1', \mathcal{E}_3'\}$

(C2) All leaf nodes in T_0 are C's arguments (Figure 11(2))

There are two possible cases, depending on the position of P_d: (i) C_d is a leaf node, and (ii) C_d is a mid-node.

With regard to the dynamic extension, $\mathcal{E}' = \mathcal{E}$ in either case.

(C3) D_1's leaf node is P's argument, D_2's leaf node is C's argument (Figure 11(3))

There are two possible cases, depending on the position of P_d.

(i) C_d is a leaf node.

P' is added as a leaf of D_2

$T_1' = T_1 \cup \{P'\}$. $T_2' = T_2$.

$\mathcal{E}_1' = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_2)$. $\mathcal{E}_2' = \mathcal{A}_2$.

$\mathcal{E}' = \{\mathcal{E}_1', \mathcal{E}_2'\}$.

(ii) C_d is a mid-node.

P' is added as a child node of C_d to generate a new branch D_3.

$T_1' = T_1 \cup \{P'\}$. $T_2' = T_2$. $T_3' = T_1 \setminus lower_2(C_d) \cup \{P'\}$.

A new execution $exec_3$ is generated. Three executions are possible: in execution $exec_1$, D_1 is executed first, C' is added, D_2 is executed, and D_3 is executed. In execution $exec_2$, D_2 is executed first and terminates because of the constraint of turn keeping. In execution $exec_3$, D_1 is executed first, C' is added, then D_3 is executed.

The dynamic extensions for these executions are as follows:
$\mathcal{E}'_1 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_2) \cup \{P'\}$. $\mathcal{E}'_2 = \mathcal{A}_2$.
$\mathcal{E}'_3 = \mathcal{A}_1 \setminus uppereq(N_C) \cup UPDATE(D_3)$.
$\mathcal{E}' = \{\mathcal{E}'_1, \mathcal{E}'_2, \mathcal{E}'_3\}$.

(C4) D_1's leaf node is C's argument, D_2's leaf node is P's argument (Figure 11(4))
P' is added as a child node of C_d, and a new branch D_3 is added.
With regard to the dynamic extension, $\mathcal{E}' = \mathcal{E}$.

4.6 Properties

It is not sufficient simply to consider updating each branch when changes in extensions are considered. It is interesting to note that even if a new node is added by a threat, it does not always affect the extension. This is due to the constraint of turn keeping and the fact that a new branch is not executed until all of the executable nodes in the current branch are executed.

The following relation holds between a dynamic extension and the win/loss of an argumentation.

Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be dynamic extensions for executions for an argumentation framework AF and $\mathcal{E}$ be a dynamic extension for AF.

1. If each $\mathcal{E}_i$ consists of only P's arguments, P dynamically wins. In this case, $\mathcal{E}_1, \ldots, \mathcal{E}_n$ coincide and include the root node.
2. If each $\mathcal{E}_i$ consists of only C's arguments, every one of P's arguments in an argumentation framework is attacked in any execution.
3. If each $\mathcal{E}_i$ consists of both P's and C's arguments, P loses the argumentation. In this case, each $\mathcal{E}_i$ does not contain the root node, and a minimal dynamic extension that consists of all of C's arguments exists.

5 Related Works

The abstract argumentation framework proposed by Dung [11] does not put orders of arguments and not include the idea of win/loss of an argumentation. It is represented as a graph structure in which nodes and edges correspond to arguments and attacks, respectively. On the other hand, in several studies on dialogue or dialect, argumentation has been represented in a tree form that identifies the proposal statement as the root node, gives an order to arguments, and defines the concept of win/loss of an argumentation. Amgoud et al. considered an argumentation a dialogue game that could be represented as an AND/OR tree and gave a semantics to indicate whether the argument corresponding to the root node was accepted [1]. They defined a win as a situation where a solution subtree exists in which all of the leaves are P's arguments. Dunne proposed a "dispute tree" on which subsequent execution of all branches is considered [10].

However, the revision of an agent's knowledge base was not considered there, allowing presented arguments to add new information to the opponent's knowledge base. García et al. also represented an argumentation framework as a tree, called a dialectical tree [13]. There, an argumentation formalism was given based on defeasible logic programming (DeLP) to decide between contradictory goals. They presented an algorithm to determine whether an argument corresponding to the root node is self-defendable. Such an argument is called "warranted." The win in argumentation in APKC2 is identical to the concept of "warranted." Later, Modgil proposed the Extended Argumentation Framework (EAF), an extension of an argumentation framework that introduced the concept of a meta-attack, that is, an attack to an attack, and discussed its semantics [16].

Moguillansky et al. considered the treatment of DeLP by an argumentation framework [17]. Their treatment made belief change theory suitable for an argumentation system based on DeLP. They presented an algorithm for judging the rules of which are selected from a given set of defeasible rules such that an argument corresponding to the root node is warranted. Their work can be considered as one handling argument theory change because an argumentation framework is changed depending on the set of rules that are selected. However, the aim of their work was to construct an argumentation framework that makes the root node warranted, not to consider the effect(s) of the execution of an algorithm. For this reason, they did not consider the timing of applying the addition/deletion of rules. In contrast, in our dynamic argumentation framework, we introduce the concept of an execution tree and insist that the execution creates a new argument.

While in the approaches based on DeLP new arguments and attacks are determined by formulas included in the rules, Cayrol et al. investigated argument theory change at a more abstract level by treating only the addition of nodes in an argumentation graph [5]. They investigated how acceptable arguments are changed when an argument is added. The aim of their research was to provide a formal analysis of changes to argumentation; the contents of the additional arguments and the reasons for their addition were beyond the scope of their study. Cobo et al. proposed an argumentation framework in which available arguments change depending on time interval [8]. In their work, these intervals were given in advance, and they did not consider the mechanism by which an argument causes the generation of a new argument. In contrast, we focused specifically on the effect of knowledge gained from presented arguments, which is essential in actual argumentation.

Several studies have been conducted on argumentation semantics. Dung provided a semantics for a given abstract argumentation framework based on acceptability [11]. He defined several acceptable sets, depending on the range of strength against an attack. Coste-Morquis et al. argued that it is controversial to include both agents' arguments in an extension because this would indicate an indirect attack [9]. They defined a new semantics, called "prudent semantics," which does not allow such cases, and compared this to Dung's semantics.

Other semantics have also been proposed, such as ideal semantics [12], semi-stable semantics [6], and others. Boroni et al. compared these types of semantics from the viewpoint of skepticism [3].

All of these semantics involved argumentation systems from a static viewpoint, whereas our proposed semantics is suitable for a dynamic argumentation system.

6 Conclusion

In this paper, we defined a new semantics that can fit a dynamic argumentation framework. In this framework, arguments and attacks are dynamically altered by a threat as the argumentation proceeds. We defined a dynamic extension for each execution of an argumentation and defined the dynamic extension for an argumentation framework as a set of these extensions. In addition, we discussed how these extensions are changed by the effect of a threat and investigated their relationships and properties. Interestingly, a threat does not always affect the outcome of the extension it changes. Although we restricted our analysis to the case in which a threat exists in only a single candidate subtree, it should be straightforward to extend the semantics to include cases in which a threat occurs over multiple candidate subtrees. We are currently formalizing this extended version.

We are also investigating the relationship of this system to the JC algorithm that we proposed previously [20], which is an algorithm for judging the win/loss of an argumentation.

References

1. Amgoud, L., Parsons, S., Maudet, N.: Arguments, dialogue, and negotiation. In: ECAI 2000, pp. 338–342 (2000)
2. Amgoud, L., Vesic, S.: Repairing preference-based argumentation frameworks. In: IJCAI 2009, pp. 665–670 (2009)
3. Baroni, P., Giacomin, M.: Comparing Argumentation Semantics with Respect to Skepticism. In: Mellouli, K. (ed.) ECSQARU 2007. LNCS (LNAI), vol. 4724, pp. 210–221. Springer, Heidelberg (2007)
4. Bench-Capon, T.J.M., Dunne, P.: Argumentation in artificial intelligence. Artificial Intelligence 171, 619–641 (2007)
5. Cayrol, C., de St-Cyr, F.D., Lagasquie-Shiex, M.-C.: Change in Abstract Argumentation Frameworks: Adding an Argument. Journal of Artificial Intelligence Research 38, 49–84 (2010)
6. Caminada, M.: Semi-stable semantics. In: COMMA 2006, pp. 121–130 (2006)
7. Chesnevar, C.I., Maguitman, A., Loui, R.: Logical models of argument. ACM Computing Surveys 32(4), 337–383 (2005)
8. Cobo, M.L., Martinez, D.C., Simari, G.R.: An approach to timed abstract argumentation. In: NMR 2010, Workshop on Argument, Dialog and Decision (2010)
9. Coste-Marquis, S., Devred, C., Marquis, P.: Prudent semantics for argumentation frameworks. In: ICTAI 2005, pp. 568–572 (2005)
10. Dunne, P.E., Bench-Capon, T.J.M.: Coherence in finite argument system. Artificial Intelligence 141(1-2), 187–203 (2002)

11. 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)
12. Dung, P.M., Mancarella, P., Toni, F.: A dialectic procedure for sceptical, assumption-based argumentation. In: COMMA 2006, pp. 145–156 (2006)
13. García, A., Simari, G.: Defeasible logic programming: an argumentative approach. Theory and Practice of Logic Programming 4(1), 95–138 (2004)
14. García, A., Chesnevar, C., Rotstein, N., Simari, G.: An abstract presentation of dialectical explanations in defeasible argumentation. In: ArgNMR 2007, pp. 17–32 (2007)
15. Hamblin, C.: Fallacies, Methuen (1970)
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. Okuno, K., Takahashi, K.: Argumentation system with changes of an agent's knowledge base. In: IJCAI 2009, pp. 226–232 (2009)
20. Okuno, K., Takahashi, K.: Argumentation System Allowing Suspend/Resume of an Argumentation Line. In: McBurney, P., Rahwan, I., Parsons, S. (eds.) ArgMAS 2010. LNCS, vol. 6614, pp. 248–267. Springer, Heidelberg (2011)
21. Rahwan, I., Simari, G. (eds.): Argumentation in Artificial Intelligence. Springer (2009)