Expressing the Characteristics of an Argumentation Framework

Sosuke MORIGUCHI [a,1] and Kazuko TAKAHASHI [a]
[a] *Kwansei Gakuin University, Japan*

Abstract. We propose a new semantics of Dung's argumentation frameworks characterized by three-valued logical expressions. Semantics for frameworks is defined based on an extension and labeling. An extension is a subset of arguments that denotes acceptable arguments, while labeling assigns one of three labels *in*, *out*, *undec* to each argument, and arguments labeled *in* are acceptable. Both extension and labeling assign basic notions to semantics, such as complete, grounded, preferred, and stable. Moreover, labeling shows that an argument might be labeled differently in other complete labelings. However, a label does not clarify that the argument is acceptable in other complete labelings. Allocation assigns three-valued logical expressions to each argument. Since expressions contain variables for parameterizing the acceptability of arguments, allocation facilitates the description not only of a specific set of acceptable arguments, but also of the relationships between such sets. As a result, allocation determines the characteristics of the argumentation framework. In this paper, we also propose a local form of allocation, facilitating discussion of semantics as part of the framework.

Keywords. argumentation framework, labeling, three-valued logic

1. Introduction

Argumentation is a useful technique for handling problems requiring conflict resolution. In the 20 years since Dung's proposal, computational argumentation has been the focus of enthusiastic research across various fields in AI, including defeasible reasoning, dialogue, and legal reasoning [4,11,15,16].

Dung proposed his abstract argumentation framework in 1995, demonstrating its relationship with both nonmonotonic reasoning and logic programming [10]. He formalized an argumentation framework consisting of a pair of a set of arguments and attack relations between arguments. This framework may be represented as a directed graph. Based on this abstraction, analyses and reasoning in argumentation may be reduced to a simple inference procedure, represented on a graph, with no requirement to present the thought contents of arguments.

Regarding argumentation frameworks, the acceptable set is among the most important concepts. Intuitively, an acceptable set constitutes a set of consistent

[1]Corresponding Author: Contract Assistant, Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo, 669-1337 Japan; E-mail:chiguri@acm.org.

arguments, the veracity of which has been agreed or proven. Much theoretical scholarship has focused on the issues inherent in arriving at definitions and calculations of acceptable sets, and comparing such sets according to the different semantics on which they are based [2,9]. Two major approaches are useful for assigning definitions to acceptable sets: an extension-based method and a labeling-based method. Dung offered an extension-based definition that is, an acceptable set is defined as a conflict-free admissible set called an extension [10]. On the other hand, Caminada took a labeling-based approach [8]. The labeling-based approach assigns one of three labels in, out, undec to each argument, depending on the rule, and the set of arguments labeled in are considered to constitute the acceptable set.

Caminada proved a strong correlation between acceptable sets obtained using both approaches. Both extension and labeling assign basic notions to semantics, such as complete, grounded, preferred, and stable. Moreover, labeling gives another viewpoint for arguments in some acceptance sets. For example, if an argument is labeled differently in two complete labelings, it is labeled undec in the grounded labeling.

However, undec does not mean the argument is in some acceptable sets. An argument attacked by only itself is labeled undec in any complete labelings. When the arguments A and B attack each other (and the others do not), they labeled undec in the grounded labeling, and opposite ones (in and out) in (other) complete labelings. To grasp such behaviors, we need more expressive representations for semantics.

In this paper, we propose a unified form expressing all of these semantic behaviors and the calculation of any acceptable set using a three-valued logic. Specifically, we allocate a logical expression to each argument. Three-valued logical expressions allow us to directly express three values in labeling (in, out, and undec), to abstract acceptability of an argument in the framework, and to show relationships with other arguments. Each acceptance set is obtained by assigning one of three values *true*, *false* and *undec* to each atomic term; the value of the logical expression to an argument is then determined. We also propose local allocation, permitting the analysis of one part of the framework independently of the whole.

The key contributions of this paper are as follows: (a) We prove that labeling and allocation using only logical constants coincide (section 3.2). We also prove that allocation permits the framework to keep any complete labelings together (section 4). (b) We demonstrate the relationship between local allocation (i.e., a part of the framework) and global allocation (i.e., the whole of the framework) (section 5.2).

This paper is organized as follows. In section 2, we review the basic notions of argumentation frameworks and labeling. In section 3, we define the three-valued logical expressions used in allocation, and offer the definitions and specifications for allocation. In section 4, we discuss the process of assigning a *general* expression to each argument. In section 5, we propose a local allocation method. Finally, we conclude this paper in section 6.

Figure 1. Two examples of argumentation frameworks. (a) Left: acyclic graph, and (b) right: cyclic graph.

2. Argumentation Framework

We will begin by defining argumentation frameworks, in the terms used in relation to Dung's concepts in [10].

Definition 1. *An argumentation framework comprises a pair of a set of arguments and their attack relations (i.e., binary relation on arguments). We use* $(Arg,\ Att)$ *for denoting an argumentation framework.*

The following is the definition of labeling (with regard to semantics) offered by Caminada [8].

Definition 2. *Labeling in terms of frameworks is a function from arguments to labels, i.e.,* $L : Arg \rightarrow \{\text{in, undec, out}\}$. *Labeling* L *is complete iff the following conditions are satisfied.*

- $L(A) = \text{in}$ *iff* $L(A') = \text{out}$ *for all arguments* A' *such that* $(A', A) \in Att$.
- $L(A) = \text{out}$ *iff there exists an argument* A' *such that* $(A', A) \in Att$ *and* $L(A') = \text{in}$.
- $L(A) = \text{undec}$ *iff there exists an argument* A' *such that* $(A', A) \in Att$ *and* $L(A') = \text{undec}$ *and there are no arguments* A'' *such that* $(A'', A) \in Att$ *and* $L(A'') = \text{in}$.

Complete labeling L *is grounded iff* $\{A|L(A) = \text{in}\}$ *is smallest in those complete labelings. We use* L_g *for grounded labeling. Complete labeling* L *is stable iff* $\{A|L(A) = \text{undec}\}$ *is empty, and* L *is preferred iff* $\{A|L(A) = \text{undec}\}$ *is minimal in those complete labelings.*

The arguments labeled in are considered to be accepted.

Example 1. In the figure 1, we illustrate two argumentation frameworks, (a) and (b). The left framework (a) is an acyclic framework, with only one complete labeling, $L(1) = L(3) = L(5) = \text{in}$ and $L(2) = L(4) = \text{out}$. Since the complete labeling is unique, it also constitutes grounded labeling.

The right framework (b) incorporates a cyclic part between 1 and 2. There are three complete labelings, L_1, L_2 and L_3:

- $L_1(A) = \text{undec}$ for all arguments A. This is a grounded labeling.
- $L_2(1) = L_2(4) = \text{in}$ and $L_2(2) = L_2(3) = \text{out}$.
- $L_3(2) = L_3(4) = \text{in}$ and $L_3(1) = L_3(3) = \text{out}$.

From these labelings, we can see arguments 1 and 2 have always opposite labels, and 3 and 4 are labeled undec only when 1 and 2 are labeled undec. However, each labeling does not imply such observation.

3. Allocation of Three-valued Logical Expression

As an alternative to the labels described in section 2, we apply three-valued logical expressions to the acceptance of arguments. We term the process by which arguments are mapped to logical expressions *allocation*.

3.1. Three-valued Logical Expression

Here, we define the three-valued logical expressions (henceforth, expressions for short) as follows:

$$p \quad ::= \quad T \quad | \quad F \quad | \quad U \quad | \quad x \quad | \quad \neg p \quad | \quad p \wedge p \quad | \quad p \vee p$$

where x is a variable (an element of Var) and T, F, and U are constants denoting true, false, and undecided (the middle value), respectively.

We define the evaluation of the expressions under valuation for the variables $v : Var \to \{1, -1, 0\}$. Note that 1, -1, and 0 denote T, F, and U respectively.

$$[\![T]\!]_v = 1, \quad [\![F]\!]_v = -1, \quad [\![U]\!]_v = 0, \quad [\![x]\!]_v = v(x),$$
$$[\![\neg p]\!]_v = -[\![p]\!]_v, \quad [\![p \wedge q]\!]_v = \min([\![p]\!]_v, [\![q]\!]_v), \quad [\![p \vee q]\!]_v = \max([\![p]\!]_v, [\![q]\!]_v).$$

We also define the equivalence between expressions as $p \equiv q \Leftrightarrow \forall v, [\![p]\!]_v = [\![q]\!]_v$. This equivalence relation is clearly reflexive, symmetric, and transitive.

Lemma 1. *The following specifications are satisfied.*

1. *For any expression p, $\neg T \equiv F$, $T \wedge p \equiv p \wedge T \equiv p$, $T \vee p \equiv p \vee T \equiv T$, $\neg F \equiv T$, $F \wedge p \equiv p \wedge F \equiv F$, $F \vee p \equiv p \vee F \equiv p$, $\neg U \equiv U$.*
2. *Let the valuation v_0 as $v_0(x) = 0$ for every variable x. If p is not equivalent with either T or F, $[\![p]\!]_{v_0} = 0$.*
3. *Any expression p is either equivalent with T or F, or there is an equivalent expression without any occurrence of T or F.*
4. *Let valuation v_1 satisfy $v_1(x) \neq 0$ for each variable x. If U does not occur in p, $[\![p]\!]_{v_1} \neq 0$.*
5. *For any expression p, if U does not occur in p, $p \not\equiv U$.*

They are easily proven by computation or induction on expressions.

3.2. Allocation

As noted above, we apply the allocation to the process of mapping each argument to an expression, and each mapping instance is termed an *allocator*. Completeness of allocation is defined in a manner similar to that for labeling.

Definition 3. *An allocator E is complete iff the following statements are satisfied.*

- *If argument A is not attacked, then $E(A) = T$.*
- *If argument A is attacked by other arguments, then $E(A) \equiv \bigwedge_{(A',A)\in Att} \neg E(A')$.*

Note that the former may be treated as a special instance of the latter.

For example, (a) in figure 1 has an complete allocator E such that $E(1) = E(3) = E(5) = T$ and $E(2) = E(4) = F$. There are, of course, an infinite number of allocators that $E(3) \equiv T$ but $E(3) \neq T$. For the purposes of this paper, however, equivalent allocators are irrelevant.

The following theorem demonstrates that allocation constitutes a generalization of labeling.

Theorem 1. *For any complete labeling L, the allocator E such that $E(A) = T$ iff $L(A) =$ in, $E(A) = F$ iff $L(A) =$ out, and $E(A) = U$ iff $L(A) =$ undec is complete.*

An allocator mapping only logical constants (T, F or U) is termed a *constant* allocator. The inverse of the above theorem is also valid.

Theorem 2. *For any constant allocator E, labeling L such that $L(A) =$ in if $E(A) = T$, $L(A) =$ out if $E(A) = F$, and $L(A) =$ undec if $E(A) = U$ is a complete labeling.*

Example 2. (b) in figure 1 has three constant allocators corresponding to complete labelings. Simultaneously, it also has a complete allocator E such that $E(1) = a$, $E(2) = \neg a$, $E(3) = a \wedge \neg a$ and $E(4) = a \vee \neg a$. With the valuation v, these expressions are evaluated to $v(a)$, $-v(a)$, $\min(v(a), -v(a))$ and $\max(v(a), -v(a))$.

- When $v(a) = 1$, they are 1, -1, -1, 1, respectively. This result corresponds to L_2, as described in example 1.
- When $v(a) = -1$, they are -1, 1, -1, 1, respectively. This result corresponds to L_3.
- When $v(a) = 0$, they are all 0. This result corresponds to L_1.

The inherent intension of the above observation is that E abstracts these labelings. Note that a complete allocator may allocate logical expressions with variables to arguments *only if* the arguments are in cycles of attack relations (see theorem 5).

We demonstrate the relation between E and constant allocators corresponding to the labelings. We apply substitution to an expression and a valuation, respectively, replacing the variables with constants.

Definition 4. *$p[C/x]$ for logical expression p, variable x and logical constant C are defined as follows.*

- *If p is a logical constant, $p[C/x] = p$.*
- *$x[C/x] = C$ and $y[C/x] = y$ if $y \neq x$.*
- *$(\neg p)[C/x] = \neg(p[C/x])$, $(p \wedge q)[C/x] = p[C/x] \wedge q[C/x]$, $(p \vee q)[C/x] = p[C/x] \vee q[C/x]$.*

We also use similar notation $v[C/x]$ for valuation v, variable x and logical constant C to denote $v[C/x](x) = [\![C]\!]_v$ and $v[C/x](y) = v(y)$ if $x \neq y$.

Lemma 2. $[\![p[C/x]]\!]_v = [\![p]\!]_{v[C/x]}$

Lemma 3. *Let x be a variable and C be a logical constant. If $p_1 \equiv p_2$, then $p_1[C/x] \equiv p_2[C/x]$.*

For any complete allocator, a variable may be replaced with one of the constants.

Theorem 3. *For the complete allocator E, we write $E^x_C(A) = E(A)[C/x]$ where C is a logical constant. Consequently, for any complete allocator E, E^x_C is also complete.*

Hereafter, we term a set of variables occurring in expressions allocated by allocator E, i.e. $\{x | x$ occurs in $E(A)$ for some $A \in Arg\}$, the *allocation variables* of E. Theorem 3 generates a complete allocator from another complete allocator that has more allocation variables. With theorem 2, a complete allocator gives some complete labelings. When a complete allocator E' is equivalent to another allocator E, substituting some variables with constants, E' is said to be instantiated from E.

Theorem 4. *For any complete allocator E and valuation v, a constant allocator E_v is defined as $E_v(A) = T$ if $[\![E(A)]\!]_v = 1$, $E_v(A) = F$ if $[\![E(A)]\!]_v = -1$ and $E_v(A) = U$ if $[\![E(A)]\!]_v = 0$. Then E_v is instantiated from E.*

There are some arguments to which any complete allocator allocates expressions equivalent to logical constants. For example, the argumentation framework $(\{1,2,3\}, \{(1,2),(2,3),(3,1)\})$ has a unique complete allocator E, such that $E(A) \equiv U$ for $A = 1, 2, 3$. The following theorem shows other examples.

Theorem 5. *Let L_g be a grounded labeling of a framework and E be a complete allocator. If $L_g(A) =$ in, then $E(A) \equiv T$. Also, if $L_g(A) =$ out, then $E(A) \equiv F$.*

4. Construction of Allocator

Instantiation raises the following question regarding inverse direction: can two allocators be instantiated from a complete allocator? The following theorem provides a positive answer to this question.

Theorem 6. *Let E_1 and E_2 be complete allocators and a be a fresh variable. The allocator E, such that $E(A) = a \wedge E_1(A) \vee \neg a \wedge E_2(A) \vee a \wedge \neg a$ if $L_g(A) =$ undec and $E(A) = E_1(A)$ otherwise, is complete.*

Proof. For valuation v,

- Let $v(a) = 1$. If argument A satisfies $L_g(A) =$ undec, then $[\![E(A)]\!]_v = [\![E_1(A)]\!]_v$. Otherwise, $E(A) = E_1(A)$, so $[\![E(A)]\!]_v = [\![E_1(A)]\!]_v$. Since E_1 is a complete allocator, $E_1(A) \equiv \bigwedge_{(A',A)\in Att} \neg E_1(A')$. So $[\![E(A)]\!]_v = [\![E_1(A)]\!]_v = [\![\bigwedge_{(A',A)\in Att} \neg E_1(A')]\!]_v = [\![\bigwedge_{(A',A)\in Att} \neg E(A')]\!]_v$.

- Let $v(a) = -1$. If argument A satisfies $L_g(A) = \text{undec}$, then $[\![E(A)]\!]_v = [\![E_2(A)]\!]_v$. And otherwise $E(A) = E_1(A)$, and from theorem 5 $E_1(A) = E_2(A)$, so $[\![E(A)]\!]_v = [\![E_2(A)]\!]_v$. Applying the same process, $[\![E(A)]\!]_v = [\![\bigwedge_{(A',A)\in Att} \neg E(A')]\!]_v$.
- Let $v(a) = 0$. If argument A satisfies $L_g(A) = \text{undec}$, then $[\![E(A)]\!]_v = 0$. Otherwise $E(A) = E_1(A)$, and from theorem 5 $[\![E_1(A)]\!]_v = 1$ if $L_g(A) = \text{in}$ and $[\![E_1(A)]\!]_= - 1$ if $L_g(A) = \text{out}$. This corresponds to L_g; according to theorem 1 it is complete.

Therefore E is complete. □

From this abstraction, we arrive at the notion of general allocators.

Definition 5. *A complete allocator, such that any complete labelings are obtained as its instantiated allocators, is called a* general allocator.

We have already encountered an example of a general allocator as E, described in example 2.

Since the number of complete labelings within a finite framework is finite, we get a general allocator through the repeated application of the generalization process detailed above.

Theorem 7. *There is a general allocator for each finite framework.*

As single framework has several general allocators, and discerning which (or how many) constant allocators are instantiated from a general allocator is challenging. However, grounded labeling is obtained by a specific valuation.

Lemma 4. *When E is a general allocator, E_{v_0}, where $v_0(x) = 0$ for any variable x is a constant allocator corresponding to a grounded labeling.*

Stable labelings are also obtained by specific valuations *with conditions*.

Lemma 5. *When E is a general allocator that $E(A)$ does not contain U for any argument A, E_{v_1}, where $v_1(x) \neq 0$ for any variable x, is a constant allocator corresponding to a stable labeling.*

The means by which preferred labelings are obtained is unclear, but it is evident that they are obtained by some valuations v such that $v(x) \neq 0$ for any variable x.

Unfortunately, the general allocator produced by theorem 6 is frequently larger than desired. As the proof demonstrates, E_1, E_2 and the allocator corresponding to L_g are instantiated from the composed allocator described in theorem 6. This means that each composition instantiates to another complete labeling for each introduced variable. For example, the argumentation framework shown in figure 2 has nine complete labelings. The method requires seven variables to construct a general allocator (each process has one abstract labeling and one grounded labeling). However, since the left part ({ 1, 2 }) and right part ({ 3, 4 }) only affect the central argument 5, without affecting one another, the a general allocator E is constructed using only two variables, as $E(1) = a$, $E(2) = \neg a$, $E(3) = b$, $E(4) = \neg b$ and $E(5) = a \wedge b$.

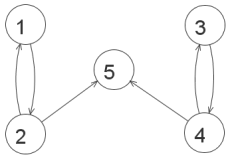

Figure 2. A framework with two cyclic parts.

Prior to the application of the method, reducing the framework down to its strongly connected components may decrease the number of allocation variables. However, it is unclear how allocators for such components are constructed and whether, if indeed possible, such allocators are minimal.

Another problem of the process is that its complexity depends on the enumeration of complete labelings, which is not processed in polynomial time (unless $P \neq NP$) [14]. To implement allocation, an algorithm for constructing general allocators without enumeration of complete labelings is required.

5. Local Allocation

As the size of a given framework is so big, we want to split into some blocks. Since both extensions and labelings are global entities, there are no functionalities for abstraction outside of the blocks. Additionally, merging blocks is required, but each block may exhibit its own preference, for example, grounded, preferred, stable, etc. However, it is difficult to assemble the blocks according to such preferences.

Allocation offers a solution to these problems. Variables abstract expressions allocated to arguments. Furthermore, they facilitate assembly of the blocks by substitution.

In this section, we propose allocations for the blocks, not for the global framework. The term "allocators" here applies only to specific blocks, not the global framework, and use term *local allocators*.

5.1. Variable Argument

Let the block be a subset of Arg. First, the arguments are selected *attacking arguments in the block*. These arguments may externally affect the block's allocation in the block from outside. These arguments are termed *variable arguments*.

To abstract the acceptance of variable arguments, a local allocator provides each with a unique variable. Here, we offer formal definitions for blocks, variable arguments and local allocators.

Definition 6. *Let* $\mathcal{B} \subseteq \mathcal{P}(Arg)$. *If* $\forall B_1, B_2 \in \mathcal{B}$, $B_1 \neq B_2 \rightarrow B_1 \cap B_2 = \emptyset$ *and* $\forall A \in Arg, \exists B \in \mathcal{B}, A \in B$, *then* $\mathcal{B}$ *is called the* splitter *of the framework, and each element of the splitter is called a block. For the block* B, *we define the function* VA *as* $\mathrm{VA}(B) = \{A' | A \in B \wedge A' \notin B \wedge (A', A) \in Att\}$. *An argument in* $\mathrm{VA}(B)$ *is called a variable argument. A local allocator* E_l *for block* B *is a function* $B \cup \mathrm{VA}(B) \rightarrow p$ *where* p *is an expression.*

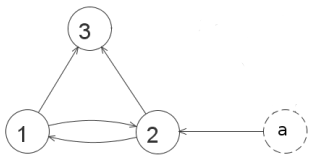

Figure 3. Example of block. $\{a\}$ is a variable argument of block $\{1, 2, 3\}$.

When we discuss the local allocation and allocation defined in section 3.2 simultaneously, we apply the term *global* allocation to the latter. The completeness of local allocators is defined similarly to that of global allocators.

Definition 7. *A local allocator E_l for block B is* complete *iff the following conditions are satisfied.*

- *For any variable argument A, $E_l(A) = a$ for some variable a and $E_l(A) \neq E_l(A')$ for any other variable argument A'.*
- *For any argument A in block B, $E_l(A) \equiv \bigwedge_{(A',A)\in Att} \neg E_l(A')$.*

The difference between the completeness of local allocators and that of global allocators is the allocation of variable arguments to variables. As such, the complete local allocator for all arguments (i.e., no variable arguments) is a complete (global) allocator.

Example 3. Figure 3 is an example of block. It has three arguments $\{1, 2, 3\}$ and its variable argument is $\{a\}$. We can assign expressions to this block, like $E_l(a) = a$, $E_l(1) = a \vee b$, $E_l(2) = \neg a \wedge \neg b$, and $E_l(3) = (a \vee b) \wedge \neg a \wedge \neg b$.

This allocator shows that the acceptability of 3 depends on not only external argument a, but also internal argument 1 (or 2). This means that this block requires a preference for 1 (or a negativity for 2) in the whole framework including this block.

5.2. Local and Global Completeness

Here, we construct a complete global allocator from local allocators. Generally speaking, the construction of local allocators is challenging.

For example, we assume a splitter for figure 4 as $\{\ \{1, 2, 3\},\ \{4, 5, 6\}\ \}$. Each block has a local allocator, as described in example 3. To distinguish the variables in each allocator, we use a_l and b_l for $\{1, 2, 3\}$, and a_r and b_r for $\{4, 5, 6\}$. To construct a global allocator from these, we need to remove a_l and a_r. However, it is difficult to remove them since a_l relates to $(a_r \vee b_r) \wedge \neg a_r \wedge \neg b_r$ and a_r relates $(a_l \vee b_l) \wedge \neg a_l \wedge \neg b_l$.

To avoid this, we restrict the acyclic splitters.

Definition 8. *We define attacks on a splitter as B_1 attacking B_2 iff $B_1 \neq B_2$ and there exist $A \in B_1$ and $A' \in B_2$, such that $(A, A') \in Att$. A splitter is acyclic iff the graph, whose nodes are blocks and arcs are attacks on the splitter, is acyclic.*

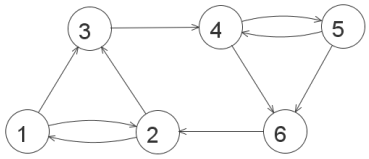

Figure 4. Cyclic attack between blocks $\{1, 2, 3\}$ and $\{4, 5, 6\}$.

Lemma 6. *Assume an acyclic splitter $\mathcal{B}$ and two blocks $B_1, B_2 \in \mathcal{B}$ such that B_1 is not attacked by any other blocks and B_2 is not attacked by other than B_1. Then, $\mathcal{B}' = \mathcal{B} \backslash \{B_1, B_2\} \cup \{B_1 \cup B_2\}$ is also acyclic.*

Next, we compose local allocators for such blocks.

Theorem 8. *For the two blocks B_1 and B_2, we may assume that $\mathrm{VA}(B_1) \cap B_2 = \emptyset$. If E_1 and E_2 are complete local allocators for B_1 and B_2 respectively, the following allocator E_{12} is a complete local allocator for $B_1 \cup B_2$.*

$$E_{12}(A) = \begin{cases} E_1(A) & (A \in B_1 \cup \mathrm{VA}(B_1)) \\ E_2(A)[A' \in \mathrm{VA}(B_2) \cap B_1, E_1(A')/E_2(A')] & (A \in B_2 \cup \mathrm{VA}(B_2)) \end{cases}$$

where $p[A' \in \mathrm{VA}(B_2) \cap B_1, E_1(A')/E_2(A')] = p[E_1(A_1)/a_1] \ldots [E_1(A_n)/a_n]$ for all $A_i \in \mathrm{VA}(B_2) \cap B_1$, $E_2(A_i) = a_i$.

Proof. For argument $A \in B_1$, it is evident that the conditions for completeness hold, since its attackers are also in B_1. For argument $A \in B_2$, $E_{12}(A) = E_2(A)[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')] \equiv (\bigwedge_{(A',A) \in Att} \neg E_2(A'))[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')] \equiv \bigwedge_{(A',A) \in Att} \neg(E_2(A')[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')])$. If $(A', A) \in Att$ then $A' \in B_2$ or $A' \in \mathrm{VA}(B_2) \cap B_1$ or $A' \in \mathrm{VA}(B_2)$ but $A' \notin B_1$. The first case, it is clear that $E_2(A')[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')] = E_{12}(A')$. The second case, $E_2(A') = a$ for some variable a and $E_2(A')[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')]) = a[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')]) = E_1(A') = E_{12}(A')$. The last case is unaffected by the substitution, so $E_2(A')[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')]) = a[A' \in \mathrm{VA}(B_2), E_1(A')/E_2(A')]) = a = E_2(A') = E_{12}(A')$. Therefore $E_{12}(A) \equiv \bigwedge_{(A',A) \in Att} \neg E_{12}(A')$. □

Finally, we can prove the following theorem using lemma 6 and theorem 8.

Theorem 9. *For a finite framework, if a splitter is acyclic, then the assembly of complete local allocators, as described in theorem 8, construct a complete global allocator.*

6. Conclusion

In this paper, we have proposed a new semantics for argumentation frameworks characterized by three-valued logical expressions. Allocation offers flexibility for the description of specific semantics (as constant allocators) and whole semantics (as general allocators). We have also proposed a process of local allocation,

focusing on the semantics of part of a given framework. We proved that a global allocator can be constructed from an acyclic splitter and local allocators.

There are several researches to propose novel kinds of semantics of arguments. Abstract dialectical frameworks (ADFs)[6,7] are generalization of argumentation frameworks. In ADFs, acceptance of an argument gives as logical expressions with acceptances of other arguments relating with the argument[2]. Also, there is an equational approach to describe acceptances for arguments [12]. In the equational approach, each argument has its own numerical function, describing relationships with other arguments. The results (acceptability) of the equations are a numeric value of $[0, 1]$ (and interpretable as acceptance like labels).

These approaches give flexibility of relationships between arguments. Although our method allows only attacks for such relationships, by the definition of the completeness (Def. 3), it enables us to discuss acceptability between arguments, or an argument and the whole framework through the allocation variables, which is effective to compare semantics.

There are some approaches to calculate several kinds of semantics using logical formulas. Besnard et al. proposed a methodology to encode argumentation frameworks and set operations to logical formulas [5]. Arieli and Caminada proposed an approach based on signed theories and quantified boolean formulas to calculate several kinds of labelings [1]. These approaches give a formula for each argumentation framework to represent acceptance of arguments. Our approach gives a formula for each argument in given framework to represent how the acceptability of the argument behaves in the framework.

Local allocation enables us to split a framework into blocks. We discussed only acyclic splitters, but there are several researches to decompose a framework into some parts. Baroni et al. discussed decomposability of a framework in several semantics [3]. Variable arguments are similar to input arguments in I/O-gadgets, proposed in [13]. Both works are not limited to acyclic. However, since they are based on labeling (or extension) approaches, we should discuss their results are applicable in our approach.

Our work will focus on two key areas in the immediate future.

- The completeness of an allocator is unambiguous: checking for completeness is obviously more complex than it is for labeling. Since checking that the equivalence of two-valued logical expressions is co-NP complete, the completeness of the allocation should be at least co-NP hard. For the purposes of our study, it is less important to check completeness than it is to construct *general* allocators. We will soon begin developing a method for constructing general (and local) allocators.
- The minimum number of allocation variables is unclear: for example, if the framework has two or three complete labelings, its general allocator can be constructed using one variable according to the method described in section 4 (by selecting not-grounded complete labelings as E_1 and E_2). A variable can be instantiated with T, F or U, so a framework with four complete labelings requires at least two allocation variables to provide the general allocator. In fact, we can construct it from two allocation variables

[2]Conditions in ADFs are defined as functions, but logical expressions are enough expressive.

using the same method. However, where a framework has five complete labelings, it is unclear whether the general allocator can be constructed from two variables.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP17H06103.

References

[1] Arieli, O. and Caminada, M. A QBF-based formalization of abstract argumentation semantics. *Journal of Applied Logic*, 11(2):229–252, 2013.
[2] Baroni, P., Caminada, M. and Giacomin,G. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.
[3] Baroni, P., Boella, G., Cerutti, F., Giacomin, M., Torre, L. and Villata, S. On the Input/Output behavior of argumentation frameworks. *Artificial Intelligence*, 217:144–197, 2014.
[4] Bench-Capon, T. and Dunne, P. Argumentation in artificial intelligence. *Artificial Intelligence*, 171:619–641, 2007.
[5] Besnard, P., Doutre, S. and Herzig, A.: Encoding Argument Graphs in Logic. In *Proc. of IPMU2014*, (2):345-354, 2014.
[6] Brewka, G. and Woltran, S.: Abstract dialectical frameworks. In *Proc. of KR2010*, pp.102–111, 2010.
[7] Brewka, G., Ellmauthaler, S., Strass, H., Wallner, J. P., and Woltran, S.: Abstract dialectical frameworks revisited. In *Proc. of IJCAI2013*, pp.803–809, 2013.
[8] Caminada, M.: On the Issue of Reinstatement in Argumentation. In *Proc. of JELIA 2006*, pp.111–123, 2006.
[9] Dunne, P., Spanring, C., Linsbichler, T. and Woltran, S.: Investigating the relationship between argumentation semantics via signatures In *Proc. of IJCAI2016*, pp.1051–1057, 2016.
[10] 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.
[11] Garc'ia. A and Simari, G.: Defeasible logic programming: An argumentative approach. *Theory and Logic Programming*, 4(1):95–138, 2004.
[12] Gabby, D. M.: Equational approach to argumentation networks. *Argument and Computation*, 3(2-3):87–142, 2012.
[13] Giacomin, M., Linsbichler, T. and Woltran, S.: On the Functional Completeness of Argumentation Semantics. In *Proc. of KR2016*, pp.43–52, 2016.
[14] Kröll, M., Pichler, R. and Woltran, S.: On the Complexity of Enumerating the Extensions of Abstract Argumentation Frameworks In *Proc. of IJCAI2017*, pp.1145–1152, 2017.
[15] Prakken, H.: An overview of formal models of argumentation and their application in philosophy. *Studies in logic*, 4(1):65–86, 2011.
[16] Rahwan,I. and Simari,G.(eds.).: *Argumentation in Artificial Intelligence*. Springer, 2009.