Reasoning by a Bipolar Argumentation Framework for PROLEG

Tatsuki Kawasaki, Sosuke Moriguchi, and Kazuko Takahashi

School of Science&Technology, Kwansei Gakuin University,
2-1, Gakuen, Sanda, 669-1337, JAPAN
dxk96093@kwansei.ac.jp, chiguri.s@acm.org, ktaka@kwansei.ac.jp

Abstract. We develop a system allowing lawyers and law school students to analyze court judgments. We describe a transformation from the logic programming language PROLEG to a bipolar argumentation framework (BAF) and the legal reasoning involved. Legal knowledge written in a PROLEG program is transformed into a BAF, in which the structure of argument in a judgment is clear. We describe two types of reasoning by the BAF: the entire structure and causality of arguments, and identification of the evidence required.

Keywords: bipolar argumentation framework, PROLEG, reasoning, semantics

1 Introduction

Recently, information technology and artificial intelligence are now vigorously applied in various fields, including those that have not yet been fully digitized or automated. In the context of legal reasoning, although the use of artificial intelligence has attracted a great deal of attention, higher-level and more practical support exploiting recent technological developments is required. Deriving support for a judgment is important. When seeking to support a judgment, it is essential to develop a system that can be easily used by lawyers who are not computer scientists; also, the system must be highly reliable and must reason accurately and rapidly. Lawyers must be able to access the system in a straightforward manner, and the system must both describe the process leading to judgment and the way in which the law was applied.

In terms of the former consideration, as law is logical, it is reasonable to base the system on such logic and reason from that perspective. Several legal reasoning systems have adopted logic programming such as Prolog as their descriptive languages. However, it is difficult for a lawyer who is not familiar with computer science to directly write Prolog code. A PROLEG system was developed to solve this problem [17]. The system was designed to support inferences based on the Japanese Presupposed Ultimate Facts Theory (termed "Yoken-jijitsu-ron" in Japanese) of the Japanese civil code, and it is currently applied to the Japanese penal code. The theory deals with uncertainties that sometimes arise in court, where a judge must give a decision even if evidence is lacking. The PROLEG

language is an extension of Prolog. Each presupposed ultimate fact is represented using general rules written in the form of *if-then* statements and exceptions. Exceptions of fact apply to all general rules and are used as court defenses. The use of exceptions rather than negative atoms creates a structure equivalent to that of a law, allowing lawyers to intuitively understand the program. An interpreter is implemented in Prolog. A burden-of-proof [14] is attached to each ultimate fact to allow for decision-making even if the fact is not proven to be true. This is achieved using the negation-as-failure inference of Prolog; thus, for a given goal, a general rule is applied and the goal is true unless there is an exception.

In terms of legal process and application, it is appropriate to employ argumentation to describe both the judgment process and how the law was applied [1]. An argumentation system reveals both the causality of arguments (for example, how arguments interacted to create a judgment) and the influence of evidence. Argumentation is powerful when used to resolve conflicts, not only formalizing the structure of the process but also incorporating any uncertainties.

In the time since Dung proposed the abstract Argumentation Framework (AF) [9], many extensions and revisions of the system have been published [15]. AF represents an argumentation by a pair of a set of arguments and a set of attacks between arguments, ignoring the contents of arguments. Several AF semantics have been defined; acceptable arguments are calculated based on these semantics. Visualization tools appropriate for argumentation systems have also been developed (e.g., [16]).

Although PROLEG facilitates the representation of a law, it is difficult to grasp the judgment process or argument causality from the execution trace. On the other hand, although it is possible to create an AF representing the interaction between a plaintiff and a defendant in court, it is difficult to directly write the structure of a law per se, or the part of the law used to create an argument in an AF form. Therefore, we combined the two systems.

We developed a transformation from PROLEG to a bipolar argumentation framework (BAF) [6], an extended AF, and showed its correctness [11]. More specifically, we gave a semantics for the BAF obtained as a result of the transformation, and proved that the answer set of the PROLEG program was the same as the set of acceptable BAF arguments. Here, we describe how the BAF reasons.

Consider the following PROLEG program representing the penal code that defines the "crime of murder."[1] The first clause indicates the general rule and the second clause an exception. The text states that if the object is a human (not a dead body) and there exists both the action of murder and also the intention to murder, then the crime of murder has been committed unless there is a legitimate defense.

```
crime_of_murder <= human, action_of_murder, intention_to_murder.
exception(crime_of_murder, legitimate_defense).
```

[1] Note that the examples shown here are simplified versions of the actual penal code; the conditions per se are simplified and the legal terminology is not precise.

When evidence is provided, the facts on which that evidence bears are proved, and it is then decided whether the crime_of_murder has been committed or not.

A judge should explain the judgment process to persuade those concerned with the transparency of justice. In such a legal situation, what is required is not only the outcome of judicial reasoning but also an explanation of the reasoning process or the cause-and-effect relationships of arguments used in reasoning. For example, if the crime_of_murder was adjudged to not in fact have been committed; this may be because of a lack of evidence of intention_to_murder, or because a legitimate_defense was available.

Our transformed BAF not only shows the process and structure of judgment, but also suggests a strategy by which a user can achieve a desired goal. If a defendant/plaintiff wishes to argue that a law should or should not be applied, the BAF identifies the evidence that must be presented and any counter-arguments that may arise. For example, when a prosecutor wishes to charge the crime_of_murder, but finds that the lack of intention_to_murder is a complicating factor, s/he will look harder for evidence of intention_to_murder. Here, we discuss such reasoning on our BAF.

This paper is organized as follows. In Section 2, we briefly explain PROLEG. In Section 3, we describe the BAF that we develop and its semantics. In Section 4, we develop the transformation rule from PROLEG to BAF. In Section 5 we describe how the BAF reasons. In Section 6, we compare our method with those of others. Finally, in Section 7, we offer conclusions and describe our planned future work.

2 Legal Description Language: PROLEG

The PROLEG program P is defined as pairs of $\langle \mathcal{R}, \mathcal{E} \rangle$, a finite set of rules, and a finite set of exceptions. Each rule is a Horn clause of the form $H \Leftarrow B_1, \ldots B_n$, where $H, B_1, \ldots, B_n$ are atoms (n may be 0; we term such a rule *a fact rule* or simply *a fact*). Each exception is in the form exception(H, B).

A fact is something given as an evidence in a court case, whereas the other rules and exceptions describe the general case. That is, the former (the facts) are generally given in an instantiated form whereas the other rules and exceptions include variables. In the examples that follow, we use a proposition for simplicity.

For each rule R or exception E, we employ the functions *head* and *body* such that $head(R) = H$ and $body(R) = \{B_1, \ldots, B_n\}$ if $R = H \Leftarrow B_1, \ldots, B_n$; $head(E) = H$ and $body(E) = \{B\}$ if $E =$ exception(H, B). An atom may be defined by more than one rule. This means that there may exist R_1 and R_2 $(R_1 \neq R_2)$ such that $head(R_1) = head(R_2)$.

Example 1. The following is an example of a PROLEG program.

```
p <= q1, q2.
exception(q1, r).
q2 <=.
r <=.
```

The semantics of the PROLEG program P are defined as an answer set (a set of ground atoms). M is the *answer set* of P iff M is the minimum model of the set of Horn clauses, $\{R \in \mathcal{R} \mid \forall E \in \mathcal{E},$ if $head(E) = head(R)$ then $body(E) \not\subseteq M\}$. The expressive power of PROLEG is the same as that of a normal logic program with an answer set [10, 18].

PROLEG allows cyclic definitions. However, here, we deal with an acyclic PROLEG program, because the Japanese civil and penal codes are usually written in an acyclic manner.

3 Bipolar Argumentation Framework

First, we define our argumentation framework.

Definition 1 (argumentation framework). An argumentation framework *is defined as a pair* $\langle AR, AT\rangle$ *where* AR *is the set of arguments and* AT *is a binary relation on* AR*, termed* an attack*. If* $(A, A') \in AT$*, we state that* A attacks A'*.*

A BAF is an extension of an AF in which the two relations of attack and support are defined over a set of arguments [6]. We define a support relation between a power set of arguments and a set of arguments; this differs from the usual BAF format, because the body of a rule generally includes more than one atom in PROLEG.

Definition 2 (bipolar argumentation framework). *A BAF is defined as a triple* $\langle AR, ATT, SUP\rangle$ *where* AR *is a finite set of arguments,* $ATT \subseteq AR \times AR$ *and* $SUP \subseteq (2^{AR} \setminus \{\emptyset\}) \times AR$*. We denote* $\mathrm{att}(B, A)$ *if* $(B, A) \in ATT$*, and* $\mathrm{sup}(\mathbf{A}, A)$ *if* $(\mathbf{A}, A) \in SUP$*.*

Example 2. Figure 1 is a graphical representation of a bipolar argumentation framework $\langle\{a, b, c, d, e\},\ \{(b, a), (e, d)\},\ \{(\{c, d\}, a)\}\rangle$. In the figure, the straight arrow indicates an attack relation and the wavy arrow a support relation.

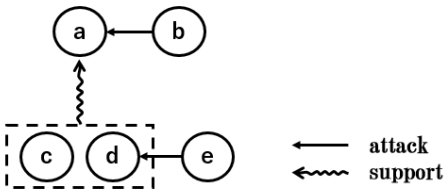


Fig. 1. Example of BAF.

We derived semantics for the BAF based on labeling [5]. Usually, labeling is a function from a set of arguments to $\{in, out, undec\}$, but *undec* is unnecessary here, because the BAF is acyclic. An argument labeled *in* is considered to be an accepted argument.

Definition 3 (labeling). *For* $\langle AR, ATT, SUP\rangle$, labeling $\mathcal{L}$ *is a function from* AR *to* $\{in, out\}$.

Labeling of a set of arguments is denoted as follows: $\mathcal{L}(\mathbf{A}) = in$ if $\mathcal{L}(A) = in$ for all $A \in \mathbf{A}$; $\mathcal{L}(\mathbf{A}) = out$, otherwise.

We use the label *in* to identify arguments that are neither attacked nor supported by any other argument. When an argument is both attacked and supported, the attack is supposed to be stronger than the support. We assign a label *out* to an argument that is attacked by another argument with the label *out*, and simultaneously supported by the set of arguments with the label *out*. Note that an argument lacking support is labeled *out*, even if it is attacked by an argument labeled *out*.

Definition 4 (complete labeling). *For* $\langle AR, ATT, SUP\rangle$, *labeling* $\mathcal{L}$ *is* complete *iff the following conditions are satisfied for any argument* $A \in AR$.

- $\mathcal{L}(A) = in$ *if*
  - $(\forall B \in AR, \neg\mathrm{att}(B, A)) \wedge (\forall \mathbf{A} \subseteq AR, \neg\mathrm{sup}(\mathbf{A}, A))$
    or
  - $(\forall B \in AR, \mathrm{att}(B, A) \Rightarrow \mathcal{L}(B) = out) \wedge (\exists \mathbf{A} \subseteq AR, \mathrm{sup}(\mathbf{A}, A) \wedge \mathcal{L}(\mathbf{A}) = in)$.
- $\mathcal{L}(A) = out$, *otherwise.*

Figure 2 shows the complete labeling of four BAFs.

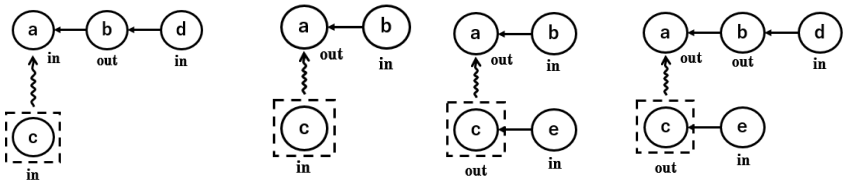


Fig. 2. Examples of BAFs with complete labelings.

Example 3. For a BAF in Figure 1, $\mathcal{L}(b) = \mathcal{L}(c) = \mathcal{L}(e) = in$ and $\mathcal{L}(a) = \mathcal{L}(d) = \mathcal{L}(\{c, d\}) = out$.

The following theorem holds [11].

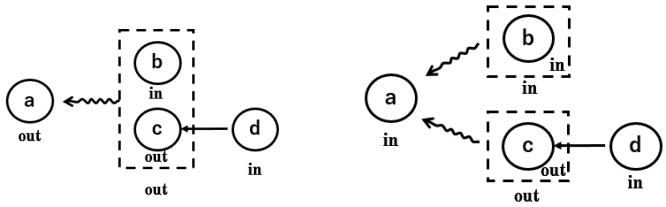
Theorem 1. *For any acyclic BAF, there is exactly one complete labeling.*

Note that we distinguish the case in which an argument is supported by a set of arguments from that in which it is supported by multiple arguments separately.

Example 4. Consider two BAFs baf_1 and baf_2 shown in Figure 3. Formally, baf_1 is described as $\langle\{a, b, c, d\}, \{(d, c)\}, \{(\{b, c\}, a)\}\rangle$ and baf_2 is described as $\langle\{a, b, c, d\}, \{(d, c)\}, \{(\{b\}, a), (\{c\}, a)\}\rangle$.

In baf_1, the argument a has one support that is a set of two arguments, whereas in baf_2, it has two supports, both of which are singletons.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the complete labeling of baf_1 and baf_2, respectively. In baf_1, $\mathcal{L}_1(b) = \mathcal{L}_1(d) = in$ and $\mathcal{L}_1(c) = out$ hold. It follows that $\mathcal{L}_1(\{b, c\}) = out$ holds. Therefore, $\mathcal{L}_1(a) = out$. On the other hand, in baf_2, $\mathcal{L}_2(b) = \mathcal{L}_2(d) = in$ and $\mathcal{L}_2(c) = out$ hold by the similar reasoning. However, $\mathcal{L}_2(\{b\}) = in$ and $\mathcal{L}_2(\{c\}) = out$. Therefore, $\mathcal{L}_1(a) = in$.



(a) baf_1: supported by a set (b) baf_2: supported independently

Fig. 3. Two types of support.

.

4 Transformation

4.1 Transformation rule

We here show a transformation from a PROLEG program to a BAF. The atoms, rules, and exceptions of the PROLEG program are transformed into arguments, supports, and attacks, respectively.

We add two types of arguments to the BAF that do not feature as explicit atoms in PROLEG. One is an argument reflecting the *absence* of any rules of inference in PROLEG. In PROLEG, an atom H that does not appear in the header of any rule or exception is not in the answer set. On the other hand, arguments that are neither attacked nor supported are labeled *in*. To fill this gap, we add the argument $\mathrm{ab}(H)$ that attacks H. We term this argument *an absence argument*. We also add arguments showing the *existence* of fact rules. For a fact rule (i.e., a rule in the form $H \Leftarrow$), there are no arguments that support H in BAF; any support is a binary relation. Therefore, we add an argument $\mathrm{ex}(H)$ that supports H. We term this argument an *existence argument*.

Definition 5 (transformation rule).
Transformation from a PROLEG program $\langle \mathcal{R}, \mathcal{E} \rangle$ *to a BAF* $\langle AR, ATT, SUP \rangle$ *is defined as follows.*

- $Atom = \bigcup_{R \in \mathcal{R}} (\{head(R)\} \cup body(R)) \cup \bigcup_{E \in \mathcal{E}} (\{head(E)\} \cup body(E))$
- $Rule = \{(body(R), head(R)) \mid R \in \mathcal{R} \wedge body(R) \neq \emptyset\}$
- $Exc = \{(B, H) \mid \text{exception}(H, B) \in \mathcal{E}\}$
- $Existence = \{H \mid H \Leftarrow\in \mathcal{R}\}$
- $ExistenceSupport = \{(\{\text{ex}(H)\}, H) \mid H \in Existence\}$
- $Absence = Atom \backslash (\{head(R) \mid R \in \mathcal{R}\} \cup \{head(E) \mid E \in \mathcal{E}\})$
- $AbsenceAttack = \{(\text{ab}(B), B) \mid B \in Absence\}$
- $AR = Atom \cup \{\text{ex}(H) \mid H \in Existence\} \cup \{\text{ab}(B) \mid B \in Absence\}$
- $ATT = Exc \cup AbsenceAttack$
- $SUP = Rule \cup ExistenceSupport$

The following theorem indicates that the semantics is preserved during transformation [11].

Theorem 2. *For PROLEG program* P*, let* M *be an answer set of* P*. Assume that* $\mathcal{L}$ *is the complete labeling of the BAF transformed from* P*. Then, for each atom* H *in* P*,* $H \in M$ *iff* $\mathcal{L}(H) = in$*.*

Example 5. The program in Example 1 is transformed into the following BAF:
$\langle\ \{p, q_1, q_2, r, ex(q_2), ex(r)\},\ \{(r, q_1)\},$
$\{(\{q_1, q_2\}, p), (\{ex(q_2)\}, q_2), (\{ex(r)\}, r)\}\quad \rangle.$

Complete labeling of the BAF is performed in the following manner. Arguments q_1 and q_2 together support the argument p. The existence arguments $ex(q_2)$ and $ex(r)$ are added to support q_2 and r, respectively. Figure 4 shows a graphical representation of the BAF, with the complete labeling. As $\mathcal{L}(q_1) = out$ and $\mathcal{L}(q_2) = in$, the label of the set of arguments $\mathcal{L}(\{q_1, q_2\}) = out$. Also, as p is supported by $\{q_1, q_2\}$, $\mathcal{L}(p) = out$. When we ignore the existence and absence arguments introduced during transformation, the set of arguments labeled *in* is $\{q_2, r\}$, which coincides with the answer set of the program of Example 1.

5 Reasoning by the BAF

We describe the two types of reasoning performed by the BAF transformed from the PROLEG program:

1. A portrayal of the entire structure of judgment and the causality of arguments.
2. Identification of the required evidence.

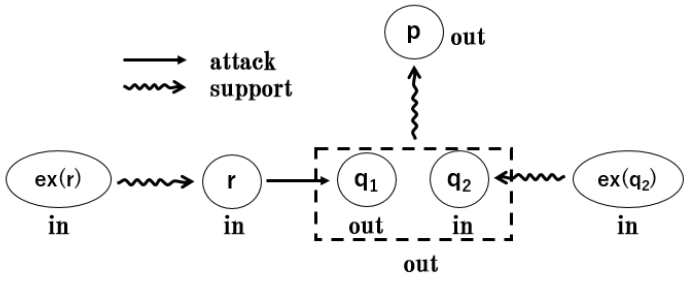


Fig. 4. BAF for the program in Example 1.

5.1 PROLEG program

Example 6. Consider the following PROLEG program. The first set of rules and exceptions states that if the object is a human (not a dead body) and there exists both the action of murder and also the intention to murder, then the crime of murder has been committed unless there was a legitimate defense. The second set of rules and exceptions states that if the accused is attacked and takes emergency, necessary, and appropriate action to defend himself/herself, then this is a legitimate defense, unless there was no intention to harm the deceased. The remainder of the program deals with the facts in evidence.

```
% rules regarding crime_of_murder
crime_of_murder <= human, act_of_murder, intention_to_murder.
exception(crime_of_murder, legitimate_defense).

legitimate_defense <=
  infringement, emergency, necessity, appropriateness,
  defense_intention.
exception(legitimate_defense, aggressive_intention_to_harm).

% facts
human <=.
act_of_murder <=.
intention_to_murder <=.
infringement <=.
emergency <=.
necessity <=.
appropriateness <=.
defense_intention <=.
```

5.2 The entire structure of judgment and causality of the arguments

In this case, the entire PROLEG program is transformed into a BAF using the rules shown in Section 4.

For each atom, a general rule defining both the atom and the exceptions is transformed into the BAF using a transformation rule. If there exists a fact, then a corresponding existence argument supporting the fact is added. If an atom does not appear in the header of any rule or exception, then a corresponding absence argument attacking the atom is added to the transformed BAF. We show a graphical representation of the transformed BAF in Figure 5.

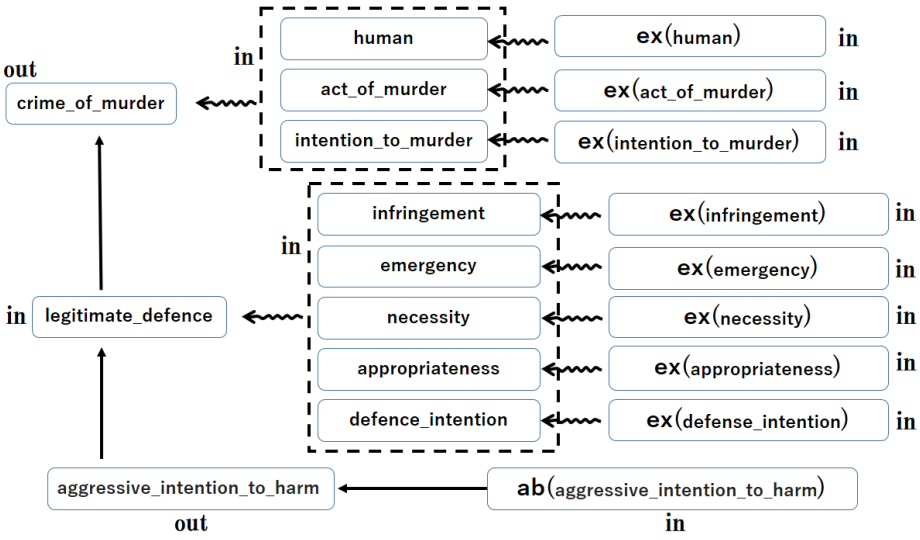


Fig. 5. Graphical representation of a transformed BAF for a murder case.

This BAF was obtained from an entire PROLEG program including facts, and shows the structure of the entire argumentation from which we can grasp the cause-and-effect relationships of the arguments.

Using this BAF, the argumentation process is explained as follows. As the label of the absence argument ab(aggressive_intention_to_harm) is *in*, that of the argument aggressive_intention_to_harm is *out* (there was no intention to harm). Therefore, the label of the argument legitimate_defense is *in* (it is a legitimate defense). The argument crime_of_murder has one attacking argument, the label of which is *in*, and one supporting set of arguments, the label of which is *in*. Hence, the label of the argument crime_of_murder is *out* (the crime of murder was not committed).

The BAF is updated as the judgment proceeds. Counter-arguments and evidences may be incrementally added as the corresponding nodes. Then, the

node labels can be changed. For example, if there is an exception to a legitimate_defense argument, and this is proven, a new argument is added; legitimate_defense is attacked by this argument and its label is changed to *out*. As another example, if evidence of aggressive_intention_to_harm is given, then its absence argument is replaced by an existence argument, and attack by the absence argument is replaced by support from the existence argument. As a result, the label of the node aggressive_intention_to_harm is changed to *in*. It follows that the label of legitimate_defense is changed to *out*, and that of the crime_of_murder to *in*.

5.3 Identification of required evidence

The BAF also identifies the evidence required to apply the law or prevent its application.

We transform a PROLEG program except for the fact part, and determine the existence arguments required to apply or not apply the law. Unlike the first type of reasoning, all available rules and exceptions are assumed to be represented, and no rules or exceptions are added.

From the definition of complete labeling, $\mathcal{L}(A) = in$ holds iff the labels of all arguments that attack A are *out* and there exists an argument that supports A, of which the label is *in*, or A is neither attacked nor supported.

Assume that a defendant wants to apply a law or that a plaintiff wants to prevent its application. Then they seek to label the corresponding argument *in* and *out*, respectively. The BAF determines the evidence required for attainment of either goal. This is achieved by repeatedly applying the following process:

Let A be an argument.

- Make $\mathcal{L}(A) = in$.
  Both of the following conditions should be satisfied.
  - (attack condition) Make $\mathcal{L}(B) = out$ for each B such that $\mathrm{att}(B, A)$. If there does not exist such an argument B, then the condition is satisfied.
  - (support condition) Make $\mathcal{L}(\mathbf{A}) = in$ for some $\mathbf{A}$ such that $\mathrm{sup}(\mathbf{A}, A)$, that is, for each $A' \in \mathbf{A}, \mathcal{L}(A') = in$. If there does not exist such $\mathbf{A}$, then an existence argument $ex(A)$ and a support $\mathrm{sup}(\{ex(A)\}, A)$ should be added.
- Make $\mathcal{L}(A) = out$.
  Either of the following conditions should be satisfied.
  - (attack condition) Make $\mathcal{L}(B) = in$ for some B such that $\mathrm{att}(B, A)$. If there does not exist such an argument B, then this condition is not satisfied.
  - (support condition) Make $\mathcal{L}(\mathbf{A}) = out$ for each $\mathbf{A}$ such that $\mathrm{sup}(\mathbf{A}, A)$, that is, for some $A' \in \mathbf{A}, \mathcal{L}(A') = out$. If there does not exist such $\mathbf{A}$, then this condition is not satisfied.

As a result, a set of existence arguments, that is, a set of evidences that should be provided, is found; this allows either party to attain his/her goal no matter what evidence his/her opponent offers.

Example 7. Figure 6 shows a PROLEG program excluding the fact part of Example 6. For convenience, each node is named $a, b, \ldots, k$, respectively.

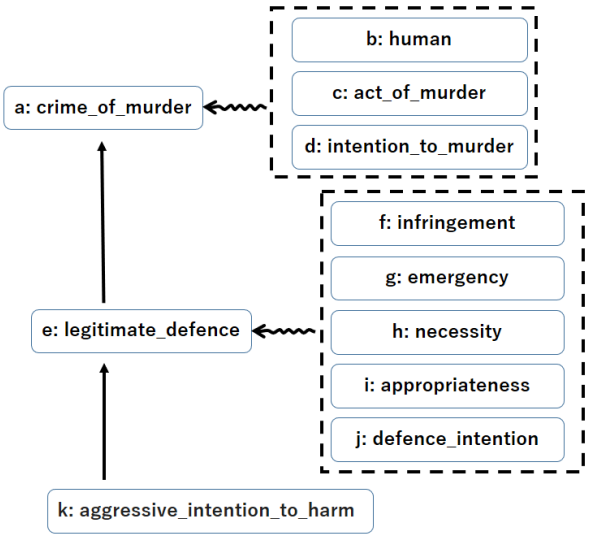


Fig. 6. Reasoning about required evidences.

– In this BAF, consider the conditions required to make $\mathcal{L}(a) = in$.
  - By attack condition for a, $\mathcal{L}(e) = out$ should be satisfied. To achieve this, attack condition for e or support condition for e should be satisfied. By attack condition for e, $\mathcal{L}(k) = in$ should be satisfied, and since k has no support, $ex(k)$ is required. By support condition for e, at least one of $\mathcal{L}(f) = out$, $\mathcal{L}(g) = out$, $\mathcal{L}(h) = out$, $\mathcal{L}(i) = out$ or $\mathcal{L}(j) = out$ holds. However, this is impossible since f, g, h, i and j are neither attacked nor supported.
  - By support condition for a, $\mathcal{L}(b) = \mathcal{L}(c) = \mathcal{L}(d) = in$ should be satisfied. Therefore, $ex(b), ex(c)$ and $ex(d)$ are required.

  As a result, the defendant should provide the four evidences $ex(k), ex(b), ex(c)$ and $ex(d)$ to apply the law.
– On the other hand, consider the conditions required to make $\mathcal{L}(a) = out$.
  - By attack condition for a, $\mathcal{L}(e) = in$ should be satisfied. To achieve this, $\mathcal{L}(k) = out$ should be satisfied, but this is impossible since k is neither attacked nor supported.
  - By support condition for a, either $\mathcal{L}(b)$, $\mathcal{L}(c)$ or $\mathcal{L}(d)$ should be out, but this is impossible since b, c and d are neither attacked nor supported.

  Therefore, the plaintiff never prevents application of the law.

In this example, only one set of existence arguments is found to make $\mathcal{L}(a) = in$, and no argument is found $\mathcal{L}(a) = out$. However, in general, we may find multiple sets in both cases. For example, assume that a defendant wishes to make $\mathcal{L}(a) = in$ in the BAF shown in Figure 7. The support required to make $\mathcal{L}(a) = in$ is one of $ex(b)$ or $ex(c)$. The support required to make $\mathcal{L}(f) = in$ is one of $ex(g)$ or $ex(h)$. Thus, we find four sets of required evidences.

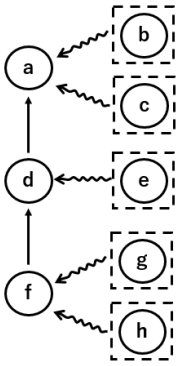


Fig. 7. Multiple sets of evidences are obtained.

6 Related Works

We compare our BAF semantics to the semantics derived by others.

Several works on BAF semantics have been undertaken. In almost all of them, the BAFs are given in advance or obtained by a translation from artificial logic programs. Such programs principally discuss argumentation structures that are seldom seen in actual judgments. We sought to apply real-world legal reasoning. A significant issue during transformation is to create BAF semantics preserving legal reasoning; no previous BAF semantics met this criterion.

Cayrol et al. investigated BAF semantics, defining several types of indirect attacks by combining attacks with supports. They also defined several types of extension [6]. Next, the concept of "coalition" (a set of arguments) was introduced and used to define a meta-AF [7, 8]. The idea was to reduce a BAF to an AF by deleting the support relations between arguments in the same coalition. An argument in BAF is accepted if it is included in an accepted coalition of the meta-AF. Boella et al. pointed out that this approach does not allow use of the Dung semantics, and revised the semantics by introducing different meta-arguments and meta-supports [2]. However, if we adopt these semantics, the semantics of PROLEG and BAF do not coincide [11]. It follows that we cannot

combine arguments to form a single support without considering their original relationships in PROLEG.

Noueioua et al. proposed a BAF that considered a support relation to be a "necessity" relation [12]. In this approach, each atom corresponds to each argument, similar to our approach. They proved the correspondence between a normal logic program and their BAF. The main drawback of the method is that it does not discriminate support by a set of arguments from support given by separate multiple arguments. They do not reflect the case in which a set of body goals support its head goal in a logic program.

Oren and Norman developed an evidential argumentation by introducing a special argument, corresponding to an environment, into a BAF [13]. This concept is similar to the existence argument of our method. The difference is that they introduced a single argument from which both attack and support relations arise. In contrast, we add an existence or absence argument for each fact. The structure of their evidential argumentation is more compact than ours, but the evidences bearing upon individual facts are unclear.

Unlike the works cited above, Brewka et al. developed an abstract dialectical framework (ADF) as a generalization of the Dung AF [3, 4]. In the ADF, each node is associated with an acceptance condition depending on the parent nodes, and each link exhibits an individual strength. A bipolar ADF is a subclass of ADF in which a link is either attacked or supported depending on the polarity of its strength. A BAF transformed from PROLEG may be considered to be an instantiation of an ADF. It would be interesting to explore whether an ADF semantics could be simply applied to a BAF transformed from PROLEG.

7 Conclusion

We have described the transformation from a PROLEG description to a BAF, and the legal reasoning using the BAF. We gave semantics to the BAF preserving the features of a PROLEG program. The BAF reflects the structure of the judgment process and causality among arguments. We have developed reasoning on the BAF, that is difficult to emulate or understand using a PROLEG program or trace of its execution. Our system enables lawyers and law school students to analyze judgments.

In future, we will improve reasoning by the BAF and create a graphical interface.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP17H06103.

References

1. Bench-Capon, T., Prakken H. and Sartor, G.: Argumentation in legal reasoning. *Argumentation in Artificial Intelligence*, 363-382 (2009).

2. Boella, G., Gabbay, D. M., Torre, L. van der and Villata, S.: Support in abstract argumentation In *Proc. of COMMA2010*, 40-51 (2010).
3. Brewka, G. and Woltran, S.: Abstract dialectical frameworks. In *Proc. of KR2010*,102-111 (2010).
4. Brewka, G., Ellmauthaler, S., Strass, H., Wallner, J.P. and Woltran, S.: Abstract dialectical frameworks revisited. In *Proc. of IJCAI2013*, 803-809 (2013).
5. Caminada, M.: On the issue of reinstatement in argumentation. In *Proc. of JELIA2006*, 111-123 (2006).
6. Cayrol, C. and Lagasquie-Schiex, M.: On the acceptability of arguments in bipolar argumentation frameworks. In *Proc. of ECSQARU2005*, 378-389 (2005).
7. Cayrol, C. and Lagasquie-Schiex, M.: Coalitions of arguments: A tool for handling bipolar argumentation frameworks. In *International Journal of Intelligent Systems*, Volume 25, 83-109 (2010).
8. Cayrol, C. and Lagasquie-Schiex, M.: Bipolarity in argumentation graphs: Towards a better understanding. *International Journal of Approximate Reasoning*, Volume 54, 876-899 (2013).
9. Dung, P. M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, Volume 77, 321-357 (1995).
10. Gelfond, M. and Lifschitz, V.: The stable model semantics for logic programming. In *Proc. of ICLP*, 1070-1080 (1988).
11. Kawasaki, T., Moriguchi, S. and Takahashi, K.: Transformation from PROLEG to a bipolar argumentation framework. In *Proc. of SAFA2018*, 36-47 (2018).
12. Nouioua, F. and Risch, V.: Argumentation framework with necessities. In *Proc. of SUM2011*, 163-176 (2011).
13. Oren, N. and Norman, T. J.: Semantics for evidence-based argumentation. In *Proc. of COMMA2008*, 276-284 (2008).
14. Prakken, H., Reed, C. and Walton, D.: Dialogues about the burden of proof. In *Proc. of ICAIL2005*, 115-124 (2005).
15. Rahwan, I. and Simari, G.(eds.): *Argumentation in Artificial Intelligence*, Springer (2009).
16. Reed, C. and Rowe, G.: Araucaria: Software for argument analysis, diagramming and representation. In *International Journal of AI Tools*, Volume 13, 961-980 (2004).
17. Satoh, K. et al.: PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology. In *JSAI-isAI 2010: New Frontiers in Artificial Intelligence*, 153-164 (2010).
18. Satoh, K. et al.: On generality of PROLEG knowledge representation. In *Proc. of JURISIN2012*, 115-128 (2012).