Fixing What Goes Wrong: A Comparison between Legal Debugging and Contestable Argumentation Learning

Wachara Fungwacharakorn[1,*], Adam Dejl[2], Emanuele De Angelis[3], Maurizio Proietti[3], Francesca Toni[2] and Ken Satoh[1]

[1]Center for Juris-Informatics, ROIS-DS, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 100-0003, Japan
[2]Department of Computing, Imperial College London, 4 Queen's Gate, London, SW7 2RH, United Kingdom
[3]IASI-CNR, Via dei Taurini, 19, 00185 Roma, Italy

Abstract

Rule-based reasoning systems inevitably encounter situations where their conclusions conflict with the intended interpretation of domain experts. In legal reasoning, such conflicts arise when the literal application of statutory rules produces counterintuitive outcomes in exceptional cases. In computational argumentation, learned Assumption-Based Argumentation (ABA) frameworks may accept undesirable claims or reject desirable ones. Although legal debugging and contestable ABA learning were developed independently to address these problems, both aim to repair existing rule-based systems while preserving as much validated knowledge as possible. This paper presents a functional comparison between these two approaches, investigates their formal correspondences, and demonstrates the correspondences using an example based on the Japanese Civil Code. The comparison identifies complementary strengths of both approaches and suggests new opportunities for integrating legal debugging with contestable argumentation learning to support human-centered and explainable repair of rule-based legal reasoning systems.

Keywords

Legal reasoning, Assumption-based Argumentation, Debugging

1. Introduction

Rule-based systems for legal and argumentative reasoning inevitably encounter cases where their formal conclusions diverge from what a human decision-maker considers correct. In law, a literal application of statutory rules can produce a counterintuitive outcome in an exceptional case that the legislature never anticipated. In learned argumentation systems, a model trained on examples can produce claims that a stakeholder finds undesirable or that fail to match a desired label. In both settings, simply discarding the system and starting over is unattractive: the existing rule base or learned framework typically still encodes a great deal of correct, validated knowledge, and the goal is instead to find the specific piece responsible for the unwanted conclusion and repair it with minimal disruption to everything else.

This paper compares two independently developed lines of work that address this problem in different sub-fields. The first, **legal debugging** [1], was introduced for PROLEG [2], a Prolog-based system encoding the Japanese Presupposed Ultimate Facts theory (Yoken-jijitsu-ron) for civil-code reasoning [3]. Legal debugging frames the problem as identifying a culprit, which is a specific rule condition responsible for a counterintuitive outcome, through an interactive process with a judge acting as an oracle. A subsequent work [4] shows that, once a culprit is found, it can be resolved by introducing extra facts describing the exceptional situation of the case. Since such resolutions tend to be case-specific, the second subsequent work [5] explores generalizing the resolution from a one-off patch into a properly generalized rule, using existing legal concepts as background knowledge.

SAFA'26: Sixth International Workshop on Systems and Algorithms for Formal Argumentation, September, 2026, Barcelona, Spain
*Corresponding author.
wacharaf@nii.ac.jp (W. Fungwacharakorn); adam.dejl18@imperial.ac.uk (A. Dejl); emanuele.deangelis@iasi.cnr.it (E. De Angelis); maurizio.proietti@iasi.cnr.it (M. Proietti); ft@ic.ac.uk (F. Toni); ksatoh@nii.ac.jp (K. Satoh)
0000-0001-9294-3118 (W. Fungwacharakorn); 0009-0006-0274-4160 (A. Dejl); 0000-0002-7319-8439 (E. De Angelis); 0000-0003-3835-4931 (M. Proietti); 0000-0001-8194-1459 (F. Toni); 0000-0002-9309-4602 (K. Satoh)

The second line of work, **contestable ABA learning**, arises in the context of Assumption-Based Argumentation (ABA) [6]. Building on ABA Learning [7], this line formally defines contestation as a situation in which a learned framework rejects a desirable claim or accepts an undesirable one [8]. It then shows how the learned framework can be repaired by extending it by considering some of its rules defeasible, while preserving as much of the original framework as possible rather than relearning it from scratch.

Although these two lines of work emerged independently, they share several similarities. For example, both operate over non-monotonic, negation-as-failure-style rule representations. Furthermore, both treat an unwanted conclusion as a signal to be localized to a specific piece of the rule base rather than rebuilt the whole rule base from scratch. However, there are also some differences which point to concrete opportunities for developing both lines of work. Legal debugging suggests a way of grounding ABA learning's choice of which rules to be considered defeasible in actual legal doctrine. Conversely, contestable ABA suggests a way of putting PROLEG's culprit-identification process on a more general, example-driven approach.

The contributions of this paper are threefold. First, we provide the first systematic functional comparison between legal debugging and contestable ABA learning. Second, we establish a formal correspondence by translating PROLEG programs into ABA frameworks and relating culprits to contestations. Third, we illustrate how contestable ABA learning can simulate legal debugging using an example from the Japanese Civil Code.

The remainder of the paper is as follows. Section 2 provides background on both lines of work. Section 3 presents a functional comparison between both lines of work. Section 4 investigates the formal correspondence between the two lines of work. Section 5 demonstrates an example based on the Japanese Civil Code Article 612. Section 6 discusses the implication from the comparison. Finally, Section 7 concludes the paper.

2. Background

This section provides background on two lines of work. The first line is about when legal reasoning, particularly focusing on PROLEG and legal debugging. The second line is about argumentative reasoning, particularly focusing on ABA and contestable learning. The details are as follows.

2.1. PROLEG and Legal Debugging

PROLEG (PROlog based LEGal reasoning support system) implements the Japanese Presupposed Ultimate Fact theory (Yoken-jijitsu-ron), a framework developed over decades by judges at the Japanese Legal Training Institute for reasoning about civil-code disputes. PROLEG formalizes the civil-code reasoning as follows.

Definition 1. *A PROLEG program is a tuple* $\langle \mathcal{L}, \mathcal{F}, \mathcal{R}, \mathcal{E} \rangle$ *where*

- $\mathcal{L}$ *is a first-order atomic language;*
- $\mathcal{F}$ *is a set of predicate symbols, called fact predicates;*
- $\mathcal{R}$ *is a set of inference rules, written as* $h <= b_1, \ldots, b_n$*, where* $h \in \mathcal{L}$ *is the head of a rule and* $b_1, \ldots, b_n \in \mathcal{L}$ *is the body of a rule, such that*
  1. *if a body of a rule is empty (i.e.,* $n = 0$*), then* h *must use a fact predicate (such a rule is called a* fact*), or*
  2. *if a body of a rule is not empty (i.e.,* $n > 0$*), then* h *must not use a fact predicate;*
- $\mathcal{E} \subseteq \mathcal{L} \times \mathcal{L}$ *is a set of exceptions, written as* $exception(h_1, h_2)$ *where* $h_1, h_2 \in \mathcal{L}$ *must not use a fact predicate; it represents that a conclusion* h_1 *is defeated if we can prove a conclusion* h_2*. As in legal reasoning generally, this paper assumes that exceptions do not contain cycles.*

The reasoning approach used in PROLEG corresponds to logic programming under negation as failure. However, PROLEG explicitly separates exceptions from rules because this representation is more intuitive to judges. An individual case is then represented as a set of ground facts. PROLEG provides legal reasoning based on stable model semantics [9] (for more details, see [10]). To keep the model unique, PROLEG programs are generally assumed to be equivalent to stratified logic programs (i.e., their exception dependency graph contains no cycle). A ground atom in the stable model of a PROLEG program P is then considered as a valid conclusion of P.

Legal debugging was introduced to address the problem that a literal interpretation of the rules can produce a counterintuitive outcome in an exceptional case. It is inspired by software debugging, where developers identify a bug from a failing test case and revise the program to correct it. Legal debugging operates on rule-based legal reasoning systems like PROLEG by introducing a two-step process. First, legal debugging interacts with a judge, who acts as a user supplying the intended interpretation of the law, to collaboratively identify a culprit, defined as follows.

Definition 2. *Let $P = \langle \mathcal{L}, \mathcal{F}, \mathcal{R}, \mathcal{E} \rangle$ be a PROLEG program and $M_i \subseteq \mathcal{L}$ be a set of ground atoms as an* intended interpretation. *For any ground atom $q \in \mathcal{L}$,*

1. *q is an* incorrect culprit *with respect to P and M_i if $q \notin M_i$ but there is a rule $r \in \mathcal{R}$ that supports q with respect to M_i (i.e., there exists a ground instance of r with head q, all atoms in the body of this ground rule belong to M_i, and there is no ground instance of an exception $exception(q, e) \in \mathcal{E}$ such that $e \in M_i$), and*

2. *q is an* incomplete culprit *with respect to P and M_i if $q \in M_i$ but there is no rule in $\mathcal{R}$ supporting q with respect to M_i.*

Second, it determines a resolution for that culprit. Once a culprit has been identified, it resolves it by introducing extra facts, which describe the exceptional situation of the case but not previously considered in the legal rules. Then, it lets a user revise or generalize legal rules using concepts already recognized in the legal domain.

2.2. ABA and Contestable Learning

Assumption-based Argumentation (ABA) is one of the pioneering computational argumentation frameworks, addressing non-monotonic reasoning using assumptions. The framework is defined as follows.

Definition 3. *ABA is a tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^{-} \rangle$, where*

- *$\langle \mathcal{L}, \mathcal{R} \rangle$ is a deductive system, where L is a language and R is a set of inference rules of the form $s_0 \leftarrow s_1, ..., s_m$ ($m \geq 0$, $s_i \in L$, for $0 \leq i \leq m$);*
- *$\mathcal{A} \subseteq \mathcal{L}$ is a (non-empty) set of assumptions;*
- *${}^{-}$ is a total mapping from $\mathcal{A}$ into $\mathcal{L}$, where $\bar{a}$ is the contrary of a, for any $a \in \mathcal{A}$.*

In original ABA, $\mathcal{L}$ can be any language, but in this paper, we consider ABA frameworks where $\mathcal{L}$ is a first-order language, similar to Definition 1. An ABA framework is said to be *flat* if assumptions are not heads of rules. The semantics of flat ABA frameworks is given by stable extensions, defined as follows [6, 8].

Definition 4. *For any flat ABA $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^{-} \rangle$,*

- *An argument for (the claim) $s \in \mathcal{L}$ supported by $A \subseteq \mathcal{A}$ and $R \subseteq \mathcal{R}$ (denoted $A \vdash_R s$, or simply $A \vdash s$, when R is immaterial) is a finite tree with nodes labelled by sentences in $\mathcal{L}$ or by true, the root labelled by s, leaves either true or from A, and non-leaves s' with, as children, the elements of the body of some rule in R with head s' (and all rules in R are used in the tree).*
- *Argument $A_1 \vdash_{R_1} s_1$ attacks argument $A_2 \vdash_{R_2} s_2$ iff $s_1 = \bar{a}$ for some $a \in A_2$.*

- *Let $Args$ be the set of all arguments and $Att = \{(\beta, \gamma) \in Args \times Args | \, \beta$ attacks $\gamma\}$, for "arguments" and "attacks" defined as above. Then, $\Delta \subseteq Args$ is a* stable extension *iff (1) $\nexists \, \beta, \gamma \in \Delta$ such that $(\beta, \gamma) \in Att$ (i.e., Δ is conflict-free); and (2) $\forall \, \gamma \in Args \setminus \Delta, \; \exists \, \beta \in \Delta$ such that $(\beta, \gamma) \in Att$ (i.e., Δ "attacks" all arguments outside Δ).*
- *We write $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle \models_\Delta s$ to denote that Δ is a stable extension of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$ and $s \in \mathcal{L}$ is the claim of an argument in Δ. We also say that s is a* (credulous) consequence *of $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$.*
- *We say $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$ is* satisfiable *if it admits at least one stable extension.*

ABA Learning is a symbolic learning approach that generates ABA frameworks from positive and negative examples, given some background knowledge, defined as follows. By $pred(E)$ we denote the set of predicate symbols occurring in E, where E is an atom, a rule, a set thereof, or an ABA framework.

Definition 5. *An ABA learning problem is a structure $(F, \langle E^+, E^- \rangle, \mathcal{T})$ where*

1. *$F = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$ is a satisfiable ABA framework as background knowledge;*
2. *$E^+ \subseteq \mathcal{L}$ is a set of positive examples;*
3. *$E^- \subseteq \mathcal{L}$ is a set of negative examples, with $E^+ \cup E^- \subseteq \mathcal{L}$ and $E^+ \cap E^- = \emptyset$;*
4. *$\mathcal{T}$ is a set of learnable predicates, with $\mathcal{T} \cap pred(A) = \emptyset$ and $pred(E^+ \cup E^-) \subseteq \mathcal{T}$.*

A solution to $(F, \langle E^+, E^- \rangle, \mathcal{T})$ is an ABA framework $F' = \langle \mathcal{L}, \mathcal{R}', \mathcal{A}', ^{-'} \rangle$ such that 1. $\mathcal{R} \subseteq \mathcal{R}'$, 2. for each head h of the new rule in $\mathcal{R}' \setminus \mathcal{R}$, $pred(h) \cap pred(F) \subseteq \mathcal{T}$, 3. $\mathcal{A} \subseteq \mathcal{A}'$, 4. $\overline{\alpha}' = \overline{\alpha}$ for all $\alpha \in \mathcal{A}$, and 5. F' is satisfiable and admits a stable extension Δ such that:

1. *for all $e \in E^+$, $F' \models_\Delta e$, i.e., all positive examples are covered in Δ; and*
2. *for all $e \in E^-$, $F' \not\models_\Delta e$, i.e., no negative example is covered in Δ.*

Contestable ABA Learning builds on this by asking what should happen when a learned ABA framework gets something wrong: specifically, when it rejects a claim that should be accepted, or accepts one that should be rejected. This situation is formally defined as contestations.

Definition 6. *Let F' be a solution to an ABA learning problem $(F, \langle E^+, E^- \rangle, \mathcal{T})$, and $c \notin E^+ \cup E^-$ be a claim in $\mathcal{L}$ whose predicate belongs to $\mathcal{T}$. Then:*

- *F' is* contested by want of *c iff there is no stable extension Δ of F' such that (1) $E^+ \cup \{c\}$ are covered in Δ; and (2) E^- are not covered in Δ.*
- *F' is* contested by want of not *c iff there is no stable extension Δ of F' such that (1) E^+ are covered in Δ; and (2) $E^- \cup \{c\}$ are not covered in Δ.*

It was shown that ABA learning can be adapted to resolve a contestation, by using an answer set programming (ASP) based algorithm. Rather than building a framework from scratch, the algorithm can extend the existing learned framework by considering some rules defeasible (i.e., by deriving for the contraries of the assumptions in the bodies of the rules; see [8, 11] for details) so that the previously rejected claim becomes accepted (or the previously accepted claim becomes rejected) while as much of the original framework as possible is preserved.

3. Functional Comparison

This section provides functional comparison between legal debugging and contestable ABA learning, as summarized in Table 1.

Table 1
Comparison between legal debugging and contestable ABA learning

Aspect	Legal Debugging	Contestable ABA Learning
Repair Trigger	Triggered by a specific case	Triggered by a contestation
Fault Localization	Identifies a single faulty conclusion at a time	Identifies multiple rules to be considered defeasible simultaneously
Repair Generalization	Starts with a case-specific repair then generalizes it	Generalizes during the repair process
Human Involvement	Guides the repair process	Initiates the contestation

3.1. Repair Trigger

Legal debugging is triggered by a specific case that produces a counterintuitive outcome under a literal interpretation of the legal rules. This reflects the role of judges, whose primary task is to decide the case before them rather than revise the law in the abstract. In contrast, contestable ABA learning is triggered whenever a claim that should be accepted is rejected by the learned ABA framework, or vice versa. Consequently, the repair trigger in contestable ABA learning is more global, whereas the repair trigger in legal debugging is inherently local.

3.2. Fault Localization

Legal debugging localizes the fault to a single conclusion, called a *culprit*, identified through a structured interaction with the judge. This process produces a precise and clear navigation for the repair. In contrast, contestable ABA learning does not seek a single point of failure. Instead, it identifies which existing rules should be considered defeasible so that after extending the framework, the contested claim receives the intended outcome.

This distinction has implications for scalability and explainability. The single-culprit approach of legal debugging is easier to explain to judges, but it may be less suitable when multiple faults contribute to the counterintuitive outcome. Conversely, the ASP-based algorithm used in contestable ABA learning can accommodate more complex repair scenarios, although it provides less transparency regarding how the fault is identified.

3.3. Repair Generalization

Legal debugging initially resolves the culprit by introducing extra facts that describe the exceptional circumstances of the case at hand, resulting in a concrete but case-specific repair. It then generalizes this repair by replacing the case-specific description with broader legal concepts derived from background knowledge, thereby producing a rule that can apply to future cases. By contrast, contestable ABA learning incorporates generalization directly into the repair process. The learned ABA framework is extended while selected rules are considered defeasible in a single step.

Thus, although both approaches seek repairs that extend beyond the triggering case, they differ in when generalization occurs. Legal debugging first constructs a case-specific repair and subsequently generalizes it. Meanwhile, the first step of the ABA Learning algorithm is *Rote Learning*, which can be seen as finding specific (i.e., ground) exceptions, then the algorithm performs generalization as an integral part of the repair process.

3.4. Human Involvement

Legal debugging is highly dependent on human judges throughout the repair process. Judges determine the intended interpretation of the law, identify the exceptional circumstances of the case, and select the legal concepts used to generalize the repair. In contrast, contestable ABA learning treats contestation as a right that can be exercised by a human or another computational agent, while assuming that the relevant background knowledge is already encoded in the learned framework.

Consequently, the human role in legal debugging is primarily to guide and refine the repair process, whereas the human role in contestable ABA learning is primarily to initiate and justify the contestation.

4. Formal Correspondence

This section investigates the formal correspondence between two kinds of rule-based representations (PROLEG and ABA) and between two kinds of fault (culprits in legal debugging and contestations in ABA learning). First, we define the formal construction for an ABA framework corresponding to a PROLEG program as follows.

Definition 7. *Given a PROLEG program $P = \langle \mathcal{L}, \mathcal{F}, \mathcal{R}, \mathcal{E} \rangle$, the corresponding ABA framework $\langle \mathcal{L}, \mathcal{R}', \mathcal{A}, ^- \rangle$ is constructed as follows:*

- *For every rule $r \in \mathcal{R}$ of the form $h <= b_1, \ldots, b_n$, there is a rule $h \leftarrow a, b_1, \ldots, b_n$ in $\mathcal{R}'$ where $a \in \mathcal{A}$ is a unique assumption atom associated with r that uses a new predicate symbol not found in $\mathcal{R}$ and contains all variables occurring in h [1].*
- *For every $exception(h_1, h_2) \in \mathcal{E}$ and every pair of rules $r_i, r_j \in \mathcal{R}$ such that (1) the head of r_i is h_i; (2) the head of r_j is h_j; and (3) there are two substitution θ and δ such that $h_i = h_1\theta$ and $h_j\delta = h_2\theta\delta$, there is $\overline{a_i} \leftarrow h_j\delta$ in $\mathcal{R}'$ where a_i is an assumption atom associated with r_i.*
- *$\mathcal{R}'$, $\mathcal{A}$ and $^-$ contain no rules, assumptions, or contraries other than those specified above.*

The key idea is to reinterpret every PROLEG rule as defeasible by associating each PROLEG rule with an assumption representing the absence of an applicable exception. Following the structure of PROLEG, a corresponding ABA framework is flat and well-founded (i.e., no cyclic attacks) and hence it has a unique stable extension. The following theorem shows that there is a semantic equivalence between ABA and the original PROLEG program.

Theorem 1. *Let $P = \langle \mathcal{L}, \mathcal{F}, \mathcal{R}, \mathcal{E} \rangle$ be a PROLEG program and $F = \langle \mathcal{L}, \mathcal{R}', \mathcal{A}, ^- \rangle$ be its corresponding ABA framework. For any ground atom $q \in \mathcal{L}$, q is a valid conclusion of P if and only if q is a consequence of F.*

Proof. **Forward direction:** Assume q is a valid conclusion of P. By PROLEG semantics, there exists a successful derivation tree for q using a set of rules $R_q \subseteq \mathcal{R}$ such that no applicable exception in $\mathcal{E}$ successfully defeats it. For every rule $r_i \in R_q$, the corresponding ABA rule in $\mathcal{R}'$ requires an assumption $a_i \in \mathcal{A}$. Let $A_q \subseteq \mathcal{A}$ be the set of all such assumptions. Because the PROLEG derivation of q is valid, there is no exception $exception(h_1, h_2)$ that is both applicable and undefeated. Consequently, in F, there is no derivable contrary $\overline{a_i}$ for any $a_i \in A_q$. If an attacking assumption set A_{attack} derives a contrary $\overline{a_i}$ (representing a PROLEG exception), the PROLEG semantics guarantee a counter-exception exists (an exception to the exception). By our construction, this maps to a set of rules in $\mathcal{R}'$ that derives the contrary of an assumption in A_{attack}. Therefore, A_q defends itself against all attacks, making it an admissible set that can be extended to a stable extension with q.

Backward direction: Assume q is a credulous consequence of F and let Δ be a stable extension of F (which is unique, as F is well-founded). This means there is a set of assumptions $A_\Delta \subseteq \mathcal{A}$ that supports all elements of Δ, which includes q. By construction, every rule in $\mathcal{R}'$ used to derive q corresponds uniquely to a rule in $\mathcal{R}$. This forms a rule chain for q in P. Because Δ is a stable extension, no contrary $\overline{a}$ for $a \in A_\Delta$ is justified unless it is successfully counter-attacked. In P, the derivation of any contrary $\overline{a}$ corresponds strictly to the conditions of an $exception(h_1, h_2) \in \mathcal{E}$. Because Δ defends itself, any such exception in P is either unprovable or defeated by a higher-level exception. Thus, the rule chain for q succeeds in P. □

[1] This reflects PROLEG's exceptions, which are defined using head atoms. We assume that PROLEG rules are written such that the head contains all variables occurring in the rule, as it typically names every entity relevant to its body (e.g., all parties to a legal relationship).

Having established semantic equivalence, we next show that the translated ABA framework can be regarded as a learned ABA framework, allowing contestable ABA learning to be applied.

Theorem 2. *Let $P = \langle \mathcal{L}, \mathcal{F}, \mathcal{R}, \mathcal{E} \rangle$ be a PROLEG program and $F' = \langle \mathcal{L}, \mathcal{R}', \mathcal{A}, ^- \rangle$ be its corresponding ABA framework. There exists an ABA learning problem $(F, \langle E^+, E^- \rangle, \mathcal{T})$ to which F' is a solution.*

Proof. (Sketch) The theorem follows the construction of PROLEG program from critical sets (for more details, see [12]), which consist of defense sets (reducible to E^+) and attack sets (reducible to E^-). Since greedy ABA Learning [11] has been shown to correspond with abstract argumentation for case-based reasoning (AA-CBR) [13], F' can be learned from empty background knowledge F, with E^+, E^-, and $\mathcal{T}$ extracted from the critical sets, as they are structurally similar to AA-CBR. □

The following theorem shows the formal correspondence between culprits in legal debugging and contestations in ABA learning.

Theorem 3. *Let $P = \langle \mathcal{L}, \mathcal{F}, \mathcal{R}, \mathcal{E} \rangle$ be a PROLEG program, M_i be an intended interpretation, and $F' = \langle \mathcal{L}, \mathcal{R}', \mathcal{A}, ^- \rangle$ be its corresponding ABA framework. There exists an ABA learning problem $(F, \langle E^+, E^- \rangle, \mathcal{T})$ to which F' is a solution and for any ground atom $q \in \mathcal{L} \setminus (E^+ \cup E^-)$, with respect to P and M_i:*

- *if q is an incorrect culprit, F' is contested by want of not q.*
- *if q is an incomplete culprit, F' is contested by want of q.*

Proof. (Sketch) The main difference between culprits and contestations is that culprits are defined based on an intended interpretation while contestations are defined based on examples. In addition to the examples extracted from the critical sets discussed in the previous proof, a legal debugger interacts with the user, who supplies the intended interpretation, following the culprit detection algorithm [1]. This interaction provides additional examples that bridge the gap between culprits and contestations. □

5. Working Example

This section demonstrates a working example based on the Japanese Civil Code Article 612, which states as follows:

> A lessee may not assign the lessee's rights or sublease a leased thing without obtaining the approval of the lessor.
>
> If the lessee allows any third party to make use of or take profits from a leased thing in violation of the provisions of the preceding paragraph, the lessor may cancel the contract.

According to the JUF theory [3], the lease contract shall be ended due to sublease by Article 612 if all of these conditions are satisfied: (1) the lease contract was effective; (2) the sublease contract was effective; (3) the third party was using the leased thing; and (4) the lessor manifested the cancellation of the lease contract. From the original text of the civil code, there is one exception to the rule; that is, the sublease shall not be approved by the lessor before the cancellation. An example of PROLEG program representing Article 612 is as follows.

```
cancellation_due_to_sublease(Lessor, Lessee, Thirdparty, Property) <=
        effective_contract(Lessor, Lessee, Property),
        effective_contract(Lessee, Thirdparty, Property),
        using_leased_thing(Thirdparty, Property),
        manifestation_cancellation(Lessor, Lessee, Property).
effective_contract(Lessor, Lessee, Property)<=
        agreement_of_contract(Lessor, Lessee, Property),
        handover(Lessor, Lessee, Property).
approval_of_sublease(Lessor, Lessee, Thirdparty, Property) <=
```

```
          approval_before_the_day(Lessor, Lessee, Thirdparty, Property).
   exception(cancellation_due_to_sublease(Lessor, Lessee, Thirdparty, Property),
          approval_of_sublease(Lessor, Lessee, Thirdparty, Property)).
```

ABA has established a Prolog-style representation by introducing two wrapping predicates (1) assumption/1, which indicates an atom as an assumption, and (2) contrary/2, which indicates the contrary relation. An ABA framework translated from representing the interpretation of the Japanese Civil Code Article 612 is as follows.

```
cancellation_due_to_sublease(Lessor, Lessee, Thirdparty, Property):-
   no_exception_to_cancellation(Lessor, Lessee, Thirdparty, Property),
   effective_contract(Lessor, Lessee, Property),
   effective_contract(Lessee, Thirdparty, Property),
   using_leased_thing(Thirdparty ,Property),
   manifestation_cancellation(Lessor, Lessee, Property).
effective_contract(Lessor, Lessee, Property):-
   no_exception_to_effective_contract(Lessor, Lessee, Property),
   agreement_of_contract(Lessor , Lessee,Property),
   handover(Lessor, Lessee, Property).
approval_of_sublease(Lessor, Lessee, Thirdparty, Property) :-
   no_exception_to_approval(Lessor, Lessee, Thirdparty, Property),
   approval_before_the_day(Lessor, Lessee, Thirdparty, Property).
exception_to_cancellation(Lessor, Lessee, Thirdparty, Property) :-
   approval_of_sublease(Lessor, Lessee, Thirdparty, Property).

assumption(
   no_exception_to_cancellation(Lessor, Lessee, Thirdparty, Property)).
assumption(
   no_exception_to_effective_contract(Lessor, Lessee, Property)).
assumption(
   no_exception_to_approval(Lessor, Lessee, Thirdparty, Property)).

contrary(
   no_exception_to_cancellation(Lessor, Lessee, Thirdparty, Property),
      exception_to_cancellation(Lessor, Lessee, Thirdparty, Property)):-
      assumption(no_exception_to_cancellation(Lessor, Lessee, Thirdparty, Property)
         ).
contrary(
   no_exception_to_effective_contract(Lessor, Lessee, Property),
      exception_to_effective_contract(Lessor, Lessee, Property)) :-
      assumption(no_exception_to_effective_contract(Lessor, Lessee, Property)).
contrary(
   no_exception_to_approval(Lessor, Lessee, Thirdparty, Property),
      exception_to_approval(Lessor, Lessee, Thirdparty, Property)):-
      assumption(no_exception_to_approval(Lessor, Lessee, Thirdparty, Property)).
```

The following example, adapted from actual cases, causes such counterintuitive outcomes when restricting to the literal interpretation of the Japanese Civil Code Article 612.

> The plaintiff made a lease contract for his room between him and the defendant. When the defendant went away for a while, he let his son use the room without obtaining the approval of the plaintiff. The plaintiff found out so he claimed the contract was cancelled by manifesting the cancellation.

According to the literal interpretation of Japanese Civil Code Article 612, the cancellation of contract is valid because the plaintiff (lessor) made a lease contract for his room (the leased thing) with the defendant (lessee) and the defendant allowed his son (third party) to make use of the room without obtaining the approval of the lessor so the plaintiff can cancel the contract by manifesting the cancellation. However,

the cancellation appears unduly harsh if we consider that the third party in this case is merely the defendant's son and the room is left to use for a while. The court may therefore decide, as the real case did, that Article 612 paragraph 2 is not applicable in exceptional situations where the sublease does not harm the confidence between a lessee and a lessor, and therefore the lessor cannot cancel the contract unless they prove the lessee's destructing of confidence.

To simulate legal debugging with contestable ABA learning, we initialize the ABA representing Article 612 and the following facts representing the example case. The facts are represented as variable-instantiation rules.

```
agreement_of_contract(Lessor, Lessee, Property):-
   Lessor = plaintiff, Lessee = defendant, Property = room.
handover(Lessor, Lessee, Property):-
   Lessor = plaintiff, Lessee = defendant, Property = room.
agreement_of_contract(Lessor, Lessee, Property):-
   Lessor = defendant, Lessee = son, Property = room.
handover(Lessor, Lessee, Property):-
   Lessor = defendant, Lessee = son, Property = room.
using_leased_thing(Thirdparty, Property):-
      Thirdparty = son, Property = room.
manifestation_cancellation(Lessor, Lessee, Property):-
      Lessor = plaintiff, Lessee = defendant, Property = room.
parent(Parent, Children):-
      Parent = defendant, Children = son.
```

Following the example case, the ABA framework is contested by want of not

```
cancellation_due_to_sublease(plaintiff, defendant, son, room),
```

which is also an incorrect culprit if the following non-fact conditions are in the intended interpretation (other conditions are directly derivable from facts and hence in the intended interpretation).

```
effective_contract(plaintiff, defendant, room)
effective_contract(defendant, son, room)
```

These two conditions are included as positive examples in the ABA learning. With a greedy folding mode [11], ABA learning extends the framework by introducing a learned exception rule:

```
exception_to_cancellation(Lessor, Lessee, Thirdparty, Property) :-
      agreement_of_contract(Lessor, Lessee, Property),
      agreement_of_contract(Lessee,Thirdparty,Property),
      handover(Lessor, Lessee, Property),
      handover(Lessee, Thirdparty, Property),
      using_leased_thing(Thirdparty, Property),
      manifestation_cancellation(Lessor, Lessee, Property),
      parent(Lessee, Thirdparty).
```

The learned rule generalizes all relevant facts into conditions for the exception to cancellation. It can be further simplified by retaining only the additional facts that are not already present in the rules used to derive the attacked argument, as follows.

```
exception_to_cancellation(Lessor, Lessee, Thirdparty, Property) :-
      parent(Lessee, Thirdparty).
```

This simplified rule indicates that the new exception is derived from the parental relationship between the lessee and the third party in the example case. This relationship explains the exceptional circumstances of the example case and justifies the introduction of the new exception.

6. Discussion and Future Work

In this paper, we explored the formal correspondence between PROLEG and ABA. Several previous studies have investigated the correspondence between PROLEG and other formal representations,

including Prolog programs [10] and Bipolar Argumentation Frameworks [14]. Moreover, PROLEG has its own argumentative extension, ArgPROLEG [15]. Our translation differs from these approaches in that it focuses on assumptions, which constitute the key component for mapping PROLEG into ABA. Assumptions naturally capture default reasoning in legal domains, where certain conditions are presumed to hold unless their contraries are established [16]. Although our translation introduces additional predicates to represent assumptions explicitly, rather than modelling exceptions directly, these assumptions can also be interpreted as *enthymemes*—implicit premises that are presumptively accepted but become defeasible when contrary evidence is presented [17]. Identifying and reconstructing such enthymemes in legal reasoning remains an open challenge.

PROLEG programs are generally assumed to be stratified, guaranteeing a unique stable-model semantics. This assumption simplifies legal debugging because only a single model needs to be considered. From a computational perspective, however, both non-stratified PROLEG and ABA can admit multiple semantics. Although multiple stable models are uncommon in legal reasoning, they arise naturally in other normative domains, such as ethical reasoning, where different stable models may represent alternative acceptable outcomes [18]. Extending legal debugging to support rule-based systems with multiple stable models is therefore an interesting direction for future research.

We also investigated the correspondence between *culprits* in legal debugging and *contestations* in ABA learning. This correspondence suggests that legal debugging can be viewed as a process of identifying informative learning examples from both legal rules and users. In particular, legal rules can be used to derive prototypical examples that expose potential deficiencies in the rule base, complementing user-provided examples. This observation aligns with the extraction of prototypical cases in the culprit resolution algorithm [4].

Another challenge concerns the identification of additional facts that characterize exceptional cases. Since newly introduced exception rules often rely on facts absent from the original rule base, these facts must first be discovered before they can be formalized. Previous work has employed large language models (LLMs) to extract fact formulas from case descriptions given a predefined set of predicates [19]. However, legal debugging often requires the discovery of entirely new predicates rather than merely instantiating existing ones. Recent work has also demonstrated the use of LLMs to identify *factors*, which capture legally relevant factual patterns in case-based reasoning [20]. In parallel, argumentative LLMs [21] have shown promise in producing explainable legal reasoning for legal entailment tasks [22]. These developments suggest that LLMs could be employed to discover additional facts and predicates for legal debugging, while argumentation provides a symbolic mechanism for validating and explaining the resulting revisions within a neuro-symbolic pipeline.

A further challenge is the generalization of exception rules. In practice, case law rarely introduces exceptions solely in terms of case-specific facts. Instead, courts typically formulate more abstract legal concepts — for example, leveraging the concept of destruction of confidence rather than the specific fact of a parental relationship in the example case — to ensure that the resulting interpretation applies to future cases. Recent work on argumentation in medical reasoning addresses a similar issue through the notion of *global contestability*, which evaluates the broader impact of learned revisions on future cases [23]. Adapting this notion to legal debugging may provide a principled approach to learning more general and broadly applicable legal interpretations.

7. Conclusion

This paper has compared legal debugging and contestable ABA learning as two complementary approaches for repairing rule-based reasoning systems. Although they originated in different research communities, we showed that both pursue the common objective of correcting undesirable conclusions while preserving as much of the existing knowledge base as possible. Our functional comparison highlighted important similarities and differences in terms of repair triggers, fault localization, repair generalization, and human involvement.

Beyond this conceptual comparison, we established a formal correspondence between the two

approaches. We presented a translation from PROLEG programs to ABA frameworks, proved the semantic equivalence of the resulting representations, showed that the translated ABA framework can be viewed as a solution to an ABA learning problem, and demonstrated that culprits in legal debugging correspond to contestations in ABA learning. These results provide a common theoretical foundation for studying repair mechanisms across legal reasoning and computational argumentation.

The correspondence also reveals opportunities for future research. Legal debugging may benefit from the automated and example-driven repair techniques developed for contestable ABA learning, while ABA learning may exploit the structured interaction with judges and domain knowledge developed in legal debugging to produce more explainable repairs. More broadly, the proposed correspondence opens the door to developing unified repair frameworks for rule-based reasoning systems that combine formal guarantees with human-centered explanations. Future work includes extending the correspondence to non-stratified programs and multiple semantics, incorporating richer legal principles into the repair process, and investigating global contestation mechanisms that can revise rule bases beyond individual cases.

Acknowledgments

This research was supported by JST as part of Adopting Sustainable Partnerships for Innovative Research Ecosystem (ASPIRE), Grant Number JPMJAP25B2, and by EPSRC as part of the NeSyDebates project. Fungwacharakorn and Satoh were also supported by the “R&D Hub Aimed at Ensuring Transparency and Reliability of Generative AI Models" project of the MEXT, by JSPS KAKENHI Grant Numbers, 25H00522 and 25H01112, and ROIS NII Open Collaborative Research 2026 Grant Number 262S07-24790. Toni was funded by the ERC (ADIX, grant no.101020934). De Angelis and Proietti were supported by MUR PRIN 2022 Project DOMAIN funded by the EU–NextGenEU, M4.C2.1.1, CUP B53D23013220006. De Angelis and Proietti are members of INdAM-GNCS and acknowledge the support of PNRR MUR project PE0000013-FAIR.

Declaration on Generative AI

During the preparation of this work, the first author used Claude for the draft and ChatGPT for the grammar and spelling check. After using these tools and services, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

[1] W. Fungwacharakorn, K. Satoh, Legal debugging in propositional legal representation, in: JSAI International Symposium on Artificial Intelligence, Springer, 2018, pp. 146–159.

[2] K. Satoh, K. Asai, T. Kogawa, M. Kubota, M. Nakamura, Y. Nishigai, K. Shirakawa, C. Takano, PROLEG: an implementation of the presupposed ultimate fact theory of japanese civil code by prolog technology, in: JSAI international symposium on artificial intelligence, Springer, 2010, pp. 153–164.

[3] S. Ito, Basis of Ultimate Facts, Yuhikaku, 2001.

[4] W. Fungwacharakorn, K. Tsushima, K. Satoh, Resolving counterintuitive consequences in law using legal debugging, Artificial Intelligence and Law 29 (2021) 541–557.

[5] W. Fungwacharakorn, K. Satoh, Generalizing culprit resolution in legal debugging with background knowledge, in: Legal Knowledge and Information Systems, volume 334 of *Frontiers in Artificial Intelligence and Applications*, 2020, pp. 52–62.

[6] P. M. Dung, R. A. Kowalski, F. Toni, Assumption-based argumentation, in: Argumentation in artificial intelligence, Springer, 2009, pp. 199–218.

[7] M. Proietti, F. Toni, Learning assumption-based argumentation frameworks, in: International Conference on Inductive Logic Programming, Springer, 2022, pp. 100–116.

[8] E. De Angelis, M. Proietti, F. Toni, Learning to contest argumentative claims, in: International Joint Conference on Rules and Reasoning, Springer, 2025, pp. 237–255.
[9] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, Bowen, Kenneth (Eds.), Proceedings of the Fifth International Logic Programming Conference and Symposium, ICLP'88, MIT Press, Cambridge, MA, USA, 1988, pp. 1070–1080.
[10] K. Satoh, T. Kogawa, N. Okada, K. Omori, S. Omura, K. Tsuchiya, On generality of proleg knowledge representation, in: Proceedings of the 6th International Workshop on Juris-informatics (JURISIN 2012), 2012, pp. 115–128.
[11] E. De Angelis, M. Proietti, F. Toni, Greedy ABA learning for case-based reasoning, in: Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, 2025, pp. 556–564.
[12] K. Satoh, M. Kubota, Y. Nishigai, C. Takano, Translating the japanese presupposed ultimate fact theory into logic programming, in: Proceedings of the 2009 Conference on Legal Knowledge and Information Systems: JURIX 2009: The Twenty-Second Annual Conference, IOS Press, NLD, 2009, p. 162–171.
[13] K. Čyras, K. Satoh, F. Toni, Abstract argumentation for case-based reasoning, in: Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning, 2016, pp. 549–552.
[14] T. Kawasaki, S. Moriguchi, K. Takahashi, Transformation from proleg to a bipolar argumentation framework, in: Proceedings of the Second International Workshop on Systems and Algorithms for Formal Argumentation (SAFA 2018), 2018, pp. 36–47.
[15] Z. Shams, M. De Vos, K. Satoh, ArgPROLEG: A normative framework for the JUF theory, in: JSAI International Symposium on Artificial Intelligence, Springer, 2013, pp. 183–198.
[16] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, H. T. Cory, The british nationality act as a logic program, Communications of the ACM 29 (1986) 370–386.
[17] A. Popescu, J. P. Wallner, Completing structured arguments in assumption-based argumentation, in: European Conference on Logics in Artificial Intelligence, Springer, 2025, pp. 95–111.
[18] F. Berreby, G. Bourgne, J.-G. Ganascia, Modelling moral reasoning and ethical responsibility with logic programming, in: Logic for programming, artificial intelligence, and reasoning, Springer, 2015, pp. 532–548.
[19] M. M. Zin, H. T. Nguyen, K. Satoh, S. Sugawara, F. Nishino, Improving translation of case descriptions into logical fact formulas using LegalCaseNER, in: Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law, 2023, pp. 462–466.
[20] M. Gray, J. Savelka, W. Oliver, K. Ashley, Using LLMs to discover legal factors, in: Legal Knowledge and Information Systems, IOS Press, 2024, pp. 60–71.
[21] G. Freedman, A. Dejl, D. Gorur, X. Yin, A. Rago, F. Toni, Argumentative large language models for explainable and contestable claim verification, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 14930–14939.
[22] C. Lindsay, F. Toni, T. D. Nguyen, T. M. Nguyen, N. A. T. Pham, Q. H. Chu, H. N. Nguyen, L.-M. Nguyen, Argumentative llms for legal information entailment, in: Legal Knowledge and Information Systems, IOS Press, 2025, pp. 121–132.
[23] A. Dejl, M. Williams, F. Toni, Argumentation for explainable and globally contestable decision support with LLMs, in: Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning, KR '26, 2026.