Legal Knowledge and Information Systems
S. Villata et al. (Eds.)

doi:10.3233/FAIA200849

Generalizing Culprit Resolution in Legal Debugging with Background Knowledge

Wachara FUNGWACHARAKORN [a] and Ken SATOH [a]
[a] *National Institute of Informatics, Sokendai University, Tokyo, Japan*

Abstract. Since the legal rules cannot be perfect, we have proposed a work called Legal Debugging for handling counterintuitive consequences caused by imperfection of the law. Legal debugging consists of two steps. Firstly, legal debugging interacts with a judge as an oracle that gives the intended interpretation of the law and collaboratively figures out a legal rule called a culprit, which determines as a root cause of counterintuitive consequences. Secondly, the legal debugging determines possible resolutions for a culprit . The way we have proposed to resolve a culprit is to use extra facts that have not been considered in the legal rules to describe the exceptional situation of the case. Nevertheless, the result of the resolution is usually considered as too specific and no generalizations of the resolution are provided. Therefore, in this paper, we introduce a rule generalization step into Legal Debugging. Specifically, we have reorganized Legal Debugging into four steps, namely a culprit detection, an exception invention, a fact-based induction, and a rule-based induction. During these four steps, a new introduced rule is specific at first then becomes more generalized. This new step allows a user to use existing legal concepts from the background knowledge for revising and generalizing legal rules.

Keywords. legal reasoning, legal representation and algorithmic debugging

1. Introduction

Since we cannot codify every essential condition in the law, the law may still lack essential conditions which may later be revealed in a real-life case. This problem is also known in artificial intelligence as a *qualification problem* [1]. When we apply literal interpretation of such law to an exceptional case, it would lead to counterintuitive consequences, which cause absurdity or harm the public interest.

This paper focuses on Legal Debugging [2], which proposes on the detection of a cause of counterintuitive consequence called a *culprit* by asking users systematically, then it attempts to resolve a culprit. However, the previous work of Legal Debugging has encountered a problem that the result of resolution is too specific since it does not cooperate with background knowledge. Therefore, in this paper, we present the cooperation of Legal Debugging and external knowledge. We reorganized Legal Debugging into four steps. The first step is a culprit detection as described in [2]. The second step is an exception invention based on

Closed World Specification [3]. The third step is a fact-based induction based on V-operator described in [4]. The fourth step is a rule-based induction, in which the system cooperates with background knowledge and generalizes the culprit resolution using the same V-operator as in the previous step.

This paper is structured as follows. Sections 2 illustrates an example case used throughout this paper. Section 3 describes the legal formalization used in this paper. Section 4 explains four steps of Legal Debugging, including a step of rule-based induction that is the main proposal of this paper. Section 5 provides the discussion and future works. Finally, section 6 provides the conclusion.

2. Example

We adapted an example case from [5] as follows.

1. The plaintiff made a lease contract for his room between him and the defendant.
2. When the defendant returned home for a while, he let his son use the room.
3. Then, the plaintiff claimed that the contract was ended by his cancellation for the reason that the defendant subleases without permission.

The related piece of law in this case is Japanese Civil Code Article 612, which is stated as follows.

Phrase 1: A Lessee may not assign the lessee's rights or sublease a leased thing without obtaining the approval of the lessor.
Phrase 2: 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.

From the literal interpretation of this article, the cancellation is valid. The third party in this case is the defendant's son and the defendant allowed his son to use the room without obtaining the approval of the lessor (the plaintiff in this case). However, it seems too harsh in this exceptional situation as the court decided as follows.

Phrase 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 [6].

In this court decision, the court introduced the idea of destruction of confidence as an exception of Phrase 2 to prevent the counterintuitive consequence from the literal interpretation of Article 612.

3. Legal Formalization

3.1. Formalizing the law

One primary representation used for formalizing the law is to represent it into a logic program with negation as failure (later referred as a logic program) as in [7]. The logic program is defined as follows.

Definition 1 (Logic Program). *A logic program is a set of rules of the form:*

$$h :- b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n. \tag{1}$$

where h, $b_1, \ldots, b_n$ $(1 \leqslant i \leqslant n)$ are first-order atoms and not presents negation as failure.

Sometimes, the rule is expressed in the form $h :- B$ where $B = \{b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n\}$. For a rule R in the form (1), we denote the head h of the rule by $head(R)$; the positive body of the rule $\{b_1, \ldots, b_m\}$ by $pos(R)$; the negative body of the rule $\{b_{m+1}, \ldots, b_n\}$ by $neg(R)$; and the whole body of the rule B by $body(R)$. We express h. if the body of the rule is empty.

A first-order atom consists of a predicate and a set of arguments. a predicate begins with a lowercase and an argument is a variable (beginning with an uppercase) or a constant (beginning with a lowercase). A ground atom refers to an atom without any variable. A ground rule refers to a rule which contains only ground atoms.

We follow the previous work [2] to divide a predicate into two types: a rule predicate and a fact predicate. A rule predicate means a predicate that occurs at least once in a head of a rule while a fact predicate means a predicate that never occurs in a head of a rule. An atom with a rule predicate, called a rule atom, shall represent a legal consequence while an atom with a fact predicate, called a fact atom, shall represent a legal fact.

Table 1: An example of a logic program representing Article 612

```
cancellation_due_to_sublease(Lessor,Lessee) :-
        effective_lease_contract(Lessor,Lessee,Property),
        effective_sublease_contract(Lessee,Thirdparty,Property),
        using_leased_thing(Thirdparty,Property),
        manifestation_cancellation(Lessor,Lessee),
        not approval_of_sublease(Lessor,Lessee).

effective_lease_contract(Leaser,Lessee,Property):-
        agreement_of_lease_contract(Leaser,Lessee,Property),
        handover_lease_contract(Leaser,Lessee,Property).

effective_sublease_contract(Leaser,Lessee,Property):-
        agreement_of_sublease_contract(Leaser,Lessee,Property),
        handover_sublease_contract(Leaser,Lessee,Property).

approval_of_sublease(Lessor,Lessee):-
        approval_before_the_day(Lessor,Lessee).
```

Table 1 illustrates an example of logic program representing Article 612. It is adapted from the example described in [5]. From the example, we shall count `cancellation_due_to_sublease`, `effective_lease_contract`, and `effective _sublease_contract` as rule predicates where others are fact predicates.

3.2. Formalizing a case

Computational law researchers have been long interested in representing a legal case or a court decision. One popular way is to represent a legal case with a set of facts and a note that the plaintiff or the defendant won in such case [8,9]. However, we represent a legal case as a set of facts and a set of intentions since we focus on the consideration of legal consequence immediately before the judgement. Our case is formally defined as follows.

Definition 2 (Case). *A case is a tuple (X, V, I) where X is a set of ground fact atoms refer to a fact situation of the case, V is a set of ground rule atoms that shall be valid and I is a set of ground rule atoms that shall be invalid (V and I are disjoint).*

Table 2: A set of legal fact representing the example case

```
agreement_of_lease_contract(plaintiff,defendant,room).
handover_lease_contract(plaintiff,defendant,room).
agreement_of_sublease_contract(defendant,son,room).
handover_sublease_contract(defendant,son,room).
using_leased_thing(son,room).
manifestation_cancellation(plaintiff,defendant).
father(defendant,son).

shall_be_invalid(cancellation_due_to_sublease(plaintiff,defendant)).
```

Table 2 illustrates a representation of example case in Section 2. The case around a ground fact atom in which a fact predicate never occurs in the program before. This ground fact atoms as *extra facts* e.g. `father(defendant,son)` in the example. Since the judge intended that cancellation due to sublease shall be invalid in this case, we note `cancellation_due_to_sublease(plaintiff,defendant)` in the set of ground rule atoms that shall be invalid.

4. Four Steps in Legal Debugging

4.1. Culprit Detection

The first step of the legal debugging is to detect a culprit. As discussed in [2] a *culprit* may be determined as a root cause of counterintuitive consequences. Counterintuitive consequences are defined as differences between literal interpretation of the law and the intended interpretation from the user. Since the intention may not be known in the first place, the system will ask the intention from the user until it finds a legal consequence that falls into two criteria of a culprit.

A *false-positive* culprit means a culprit that shall be valid but literally invalid. On the other hand, a *false-negative* culprit means a culprit that shall be invalid but literally valid.

Definition 3 (Intended Interpretation). *An intended interpretation IM is an oracle and possibly infinite set of ground atoms representing an intended interpretation in the user's mind. We denote it by an oracle set since we cannot know the whole intended interpretation but for a case (X, V, I), we know that $X \subset IM$, $V \subset IM$ and I and IM are disjoint.*

Definition 4 (Support). *Let IM be an intended interpretation. We say IM supports a ground rule atom p with respect to a program T if there is a rule in T that can be substituted into a rule in the form (1) such that $\{b_1, \ldots, b_m\} \subset IM$, $\{b_{m+1}, \ldots, b_n\}$ and IM are disjoint, and $p = h$. The substituted rule is called a supporting rule of p w.r.t. IM.*

Definition 5 (Culprit). *Let IM be an intended interpretation. A ground rule atom p is a* culprit *with respect to IM and a program T if*

(i) $p \notin IM$ but IM supports p w.r.t. T (false-positive) or
(ii) $p \in IM$ but IM does not support p w.r.t. T (false-negative).

We follow the previous work [2] to trace down a sequence of counterintuitive consequences and a culprit will be one in the last of the sequence.

Table 3: An example of a culprit detection dialogue

```
Considering
cancellation_due_to_sublease(Lessor,Lessee) :-
        effective_lease_contract(Lessor,Lessee,Property),
        effective_sublease_contract(Lessee,Thirdparty,Property),
        using_leased_thing(Thirdparty,Property),
        manifestation_cancellation(Lessor,Lessee),
        not approval_of_sublease(Lessor,Lessee).

Shall effective_lease_contract(plaintiff,defendant,Property) be valid
    ? yes
Which Property? room.
Shall effective_sublease_contract(defendant,Thirdparty,room) be valid
    ? yes
Which Thirdparty? son.
Shall approval_of_sublease(plaintiff,defendant) be valid? no

Detect a culprit
cancellation_due_to_sublease(plaintiff,defendant).
With a supporting rule(s)
cancellation_due_to_sublease(plaintiff,defendant):-
    effective_lease_contract(plaintiff,defendant,room),
    effective_sublease_contract(defendant,son,room),using_lease_thing
    (son,room),manifestation_cancellation(plaintiff,defendant),not(
    approval_of_sublease(plaintiff,defendant)).
```

Table 3 illustrates an example of a culprit detection dialogue. The system asks a user whether there are some instantiation of rule atoms that shall be valid.

From this dialogue, we realize more members in the intended interpretation. As a result, we know that `cancellation_due_to_sublease(plaintiff, defendant)` is a culprit since it shall be invalid but the intended interpretation supports it.

4.2. Exception Invention

For a false-negative culprit, we may simply resolve by introducing a culprit. On the other hand, for a false-positive culprit, we require to invent a new predicate for an exception to rebut the supporting rule. This section describes how to invent a new predicate when the identified culprit shall be invalid. To this end, we apply Closed World Specification algorithm [3] as in Algorithm 1. It describes how to revise a logic program with negation as failure when we intend a ground atom A to be invalid. The algorithm states that if there is an exception in the supporting rule of A, we should use an instantiation of the exception; otherwise, we should invent a new atom with a new predicate for an exception.

Algorithm 1 An original Closed World Specification (CWS) algorithm

Input a logic program with negation as failure T with the unique stable model M and a ground atom A such that A is valid w.r.t. T but A is intended to be invalid.

```
for all supporting rule $R$ of $A$ w.r.t. $M$ and a substitution $\theta$ do
    if $body(R)$ contains $not\ b$ then
        Let $T' = T \cup \{b\theta\}$
    else
        Let $\{V_1, \ldots, V_n\}$ be the domain of $\theta$
        Let $q$ be a predicate symbol not found in $T$
        Let $b$ be $q(V_1, \ldots, V_n)$
        Let $T' = T \backslash \{R\} \cup \{head(R) :- (body(R) \cup \{not\ b\})\} \cup \{b\theta\}$
return $T'$
```

However, if we apply this algorithm to the example case, `approval_of_sublease(plaintiff,defendant)` is introduced. Such introduction is contradicted to the user intention that the `approval_of_sublease(plaintiff,defendant)` shall be invalid. From this reason, we may solve by forcing the algorithm to merely introduce an exception with a new predicate, as shown in Algorithm 2. Table 4 illustrates the exception invention in the example case. Now the system knows that the example case is an exceptional situation but what is a sufficient condition in the example case that makes the case exceptional would be determined in the next step.

Table 4: Exception invention in the example case

```
Inventing an exception using a closed world specification...
please specify a new exception name: new_exception.

The culprit is revised into

cancellation_due_to_sublease(Lessor,Lessee) :-
        effective_lease_contract(Lessor,Lessee,Property),
        effective_sublease_contract(Lessee,Thirdparty,Property),
        using_leased_thing(Thirdparty,Property),
        manifestation_cancellation(Lessor,Lessee),
        not approval_of_sublease(Lessor,Lessee),
        not new_exception(Lessor,Lessee,Property,Thirdparty).
```

Algorithm 2 An adapted Closed World Specification (CWS) algorithm

Input a logic program with negation as failure T with the unique stable model M and a ground atom A such that A is valid w.r.t. T but A is intended to be invalid.

```
for all supporting rule $R$ of $A$ w.r.t. $M$ and a substitution $\theta$ do
    Let $\{V_1, \ldots, V_n\}$ be the domain of $\theta$
    Let $q$ be a predicate symbol not found in $T$
    Let $b$ be $q(V_1, \ldots, V_n)$
    Let $T' = T \backslash \{R\} \cup \{head(R) : -(body(R) \cup \{not\ b\})\} \cup \{b\theta\}$
return $T'$
```

4.3. Fact-based Induction

In this step, we obtain the sufficient condition of why the present case is exceptional by asking from a user. Since we require to form a rule for describing the exceptional situation, the system would apply Inverse Resolution [4], to induce a new rule from known rules. Inverse Resolution is widespread applied for inductive programming, including for refining legal concepts in legal ontology [10]. However, there are some concerns about Inverse Resolution in Logic Program with Negation as Failure [11]. The first concern is that the result of Inverse Resolution is not generally consistent with the input program under the stable model semantics. We can only guarantee for some types of input programs e.g. input programs that are locally stratified and the dependencies of the input program are preserved in the result program. Since logic programs in legal representation are usually locally stratified, we have no problem with the first issue. For the second issue, one practical way is to take some extra facts into a body of a new rule to guarantee that we do not destroy dependencies of the input program. This corresponds to the practice in the law that the extra facts should be identified to distinguish the present exceptional case with the precedent. Another concern is that all variables in a body of a new rule should occur in a head of a new rule. It limits a new rule so that it is not too generalized.

Table 5: An example of fact-based induction dialogue

```
Generating a primary exception rule using Inverse Resolution
Listing possibly relevant facts...
1: agreement_of_lease_contract(plaintiff,defendant,room)
2: handover_lease_contract(plaintiff,defendant,room)
3: agreement_of_sublease_contract(defendant,son,room)
4: handover_sublease_contract(defendant,son,room)
5: using_leased_thing(son,room)
6: manifestation_cancellation(plaintiff,defendant).
7: father(defendant,son)
please specify relevant facts by a list of incremental indices (e.g.
    [1,3,5])
|: [7].

A new exception rule
new_exception(Lessor,Lessee,Property,Thirdparty):-father(Lessee,
    Thirdparty).
```

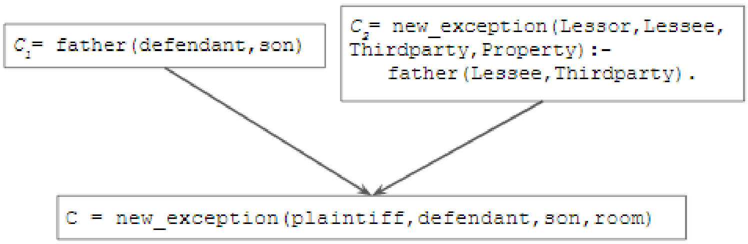
C1= father(defendant,son)
C2= new_exception(Lessor,Lessee, Thirdparty,Property):- father(Lessee,Thirdparty).
C = new_exception(plaintiff,defendant,son,room)

Figure 1. Illustration of applying V-operator to induce a new rule

A user would give the sufficient condition of the exceptional situation as the relevant facts and the system may check whether the set of relevant facts meets above criteria as a body of a new rule (e.g. the set must contain at least one extra fact). If the set passes the criteria, the system would apply the V-operator in Inverse Resolution to induce a new rule from a pair of ground atoms.

Definition 6 (Resolution). *Let C_1 and C_2 be two rules with no common variables. Let p be an atom within $pos(C_2)$ such that p is unifiable with $head(C_1)$ by the most general unifier (mgu) of θ. We denote the* resolvent *of C_1 and C_2 by $C = C_1 \cdot C_2$ where $C = head(C_2)\theta : -(body(C_2)\backslash\{p\})\theta \cup body(C_1)\theta$.*

Definition 7 (V-operator). *Given two rules C_1 and C, We call C_2 an induced rule by the V-operator from C_1 and C if $C_1 \cdot C_2$ is substitutable to C .*

Table 5 illustrates an example of fact-based induction dialogue. The systems ask the user to select a set of relevant facts. In the example, a user selects that fact that the defendant is a father of the third party, represented by `father(defendant,son)`, is the reason why this case is exceptional. Since the fact is an extra fact, it passes the criteria. Let C be `new_exception(plaintiff,defendant,room,son)`, a ground exception from the

Table 6: An example of fact-based induction dialogue

```
Would you like to generalize the rule more by using the background
    theory (y./n.) |: y.

Found more general rule
new_exception(Lessor,Lessee,Property,Thirdparty):-
    relatives(Lessee,Thirdparty).

Would you like to generalize the rule more by using the background
    theory (y./n.)
|: y.
Found no more general rule
```

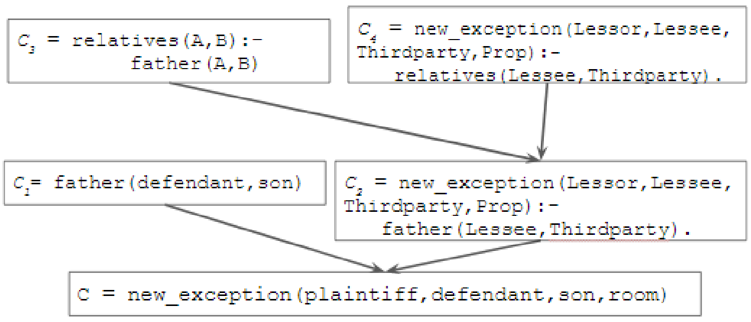
C_3 = relatives(A,B):- father(A,B)
C_4 = new_exception(Lessor,Lessee, Thirdparty,Prop):- relatives(Lessee,Thirdparty).
C_1= father(defendant,son)
C_2 = new_exception(Lessor,Lessee, Thirdparty,Prop):- father(Lessee,Thirdparty).
C = new_exception(plaintiff,defendant,son,room)

Figure 2. Illustration of applying V-operator to induce a new rule

exception invention step by the adaption of Closed World Specification Algorithm; and let C_1 be `father(defendant,son)`, the reason given by the user, the system induce a new rule by the V-operator as in Fig. 1. An induced rule is more generalized than the ground exception from the previous step since an induced rule by the V-operator does not specifically apply to the example case. From the example, the system knows that the sufficient condition to make a case exceptional is when the lessee is the father of the sublessee.

4.4. Rule-based Induction

Beyond the primary induced rule, in this newly introduced step, the system may apply Inverse Resolution further with background knowledge. For ease of exposition, we assume that the background knowledge is convertible to a logic program called a background theory. This background theory is assumed to contain general knowledge rules as well as legal knowledge rules. For example, the background theory may contain a rule "A father is a kind of relative", which is represented as `relative(X,Y) :- father(X,Y)`.

Table 6 illustrates an example of rule-based induction dialogue. If a user would like to generalize a rule induced in the previous step, the system would find a rule in a theory such that it can induce more general rule using the V-operator. From the dialogue, the system found a rule C_3 `relative(X,Y) :- father(X,Y)`. The

V-operator induces a new rule C_4 from C_2 (from the previous step) and C_3 as in Figure 2. The result rule C_4 implies that a new exception may be executed if the lessee is a relative of the sublessee.

Since a revision is only an advisory, the user can reject the generalization, accept the generalization, or request the system to generalize a rule further. The system may use other cases with intention to determine whether the generalized rule is acceptable.

Another operation that has not been mentioned in the example is W-operator [4]. W-operator is simply a combination of two V-operators back-to-back. It may be used for grouping similar concepts into the new concept. For example, suppose we know that a new exception should be valid not only for a case such that the lessee is a relative of the sublessee but also for a case such that the lessee is a working colleague of the sublessee. With W-operator, these two concepts may be grouped into a new concept, that covers a case such that the lessee is an acquaintance of the sublessee.

5. Discussion and Future Works

This paper is in line with a previous study [12] suggesting the benefit of background knowledge in computational law. Since we assume the legal rules and cases are formalized using first-order predicates, Legal Debugging has not yet supported open-texture concepts, which shows that a qualification problem still exists in our formalization. Another limitation of the proposed method is that a case which causes counterintuitive consequences is presumed to contain an extra fact describing the exceptional situation of the case. Since the V-operator used in the proposed method supports only one extra fact to induce each rule, we think that potential future works are extending the V-operator to support multiple extra facts, obtaining practical extra facts, or combining the facts already existed in legal rules with extra facts.

6. Conclusion

This paper describes the reorganization of Legal Debugging into four steps, namely a culprit detection, an exception invention, a fact-based induction, and a rule-based induction. These steps generalizes the resolution of a culprit by using Closed World Specification and Inverse Resolution. The rule-based induction, which is firstly introduced in this paper, can obtain more general rules for resolving a culprit by cooperating with background knowledge in a form of background theory. With such cooperation, the resolution can obtain more general normative facts to resolve a culprit in a more practical way. In future, we would like to investigate the acquisition of extra facts, the compliance of multiple extra facts, and the combination of extra facts and facts that already existed in legal rules.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Numbers, JP17H06103 and JP19H05470. We would like to thank Tiago Oliveira and anonymous reviewers for their extensive comments.

References

[1] M. Thielscher, The qualification problem: A solution to the problem of anomalous models, *Artificial Intelligence* **131**(1–2) (2001), 1–37.

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

[3] M. Bain and S. Muggleton, Non-monotonic learning, *Inductive logic programming* **38** (1992), 145153.

[4] S. Muggleton and W. Buntine, Machine invention of first-order predicates by inverting resolution, in: *Machine Learning Proceedings 1988*, Elsevier, 1988, pp. 339–352.

[5] K. Satoh, M. Kubota, Y. Nishigai and 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, Amsterdam, The Netherlands, The Netherlands, 2009, pp. 162–171. ISBN ISBN 978-1-60750-082-7.

[6] 1994 (O) 693, Tokyo High Court No. 9 at 2431, Minshu Vol. 50, 1996.

[7] M.J. Sergot, F. Sadri, R.A. Kowalski, F. Kriwaczek, P. Hammond and H.T. Cory, The British Nationality Act as a logic program, *Communications of the ACM* **29**(5) (1986), 370–386.

[8] V. Aleven, Teaching case-based argumentation through a model and examples, PhD thesis, University of Pittsburgh, 1997.

[9] E.L. Rissland and K.D. Ashley, A case-based system for trade secrets law, in: *Proceedings of the 1st international conference on Artificial intelligence and law*, 1987, pp. 60–66.

[10] M. Kurematsu, M. Tada and T. Yamaguchi, A legal ontology refinement environment using a general ontology, in: *Proceedings of Workshop on Basic Ontology Issues in Knowledge Sharing, International Joint Conference on Artificial Intelligence*, Vol. 95, 1995.

[11] C. Sakama, Some properties of inverse resolution in normal logic programs, in: *International Conference on Inductive Logic Programming*, Springer, 1999, pp. 279–290.

[12] V. Aleven, Using background knowledge in case-based legal reasoning: a computational model and an intelligent learning environment, *Artificial Intelligence* **150**(1–2) (2003), 183–237.