On Generality of PROLEG Knowledge Representation

Ken Satoh, Takamune Kogawa, Nao Okada, Kentaro Omori, Shunsuke Omura, and Kazuki Tsuchiya

National Institute of Informatics and Sokendai
ksatoh@nii.ac.jp

Abstract. In this paper, we show a generality of PROLEG knowledge representation in two-fold. One direction is from a view of logic programming that we give a semantics of PROLEG representation and we show that the expressive power of PROLEG is same as a general logic program with answer set semantics. Another direction is that we can use PROLEG not only for the Japanese "theory of presupposed ultimate facts" in Japanese civil law but also for argumentation system and any legal reasoning system where general principles and exceptions are separated.

1 Introduction

The Japanese Presupposed Ultimate Fact Theory[Ito08](called "*Yoken-jijitsu-ron*" in Japanese, "JUF theory" we call in this paper) is used to handle the uncertainty that sometimes occurs in the court because of a lack of enough evidence.

Kitamura [Kitamura93] concisely explained the JUF theory, so we refer to the article here (in the subsection of "Precision Justice in Civil Cases" in [Kitamura93]).

> *Referred to as the "theory of presupposed ultimate facts" (Yoken-jijitsu-ron), it is based essentially on a procedural approach to the burden of persuasion. Civil judges should know which facts must be proved, by which party and what to do in case of failure to prove. Accordingly, advocates of this method propose to distinguish, from among the legal events prescribed as being the cause of a given effect, those facts that the plaintiff or defendant must rely on and establish. They call these facts Yoken jijitsu (presupposed ultimate facts). ... in each text stipulated in the Civil Code, facts of positive presupposition relied on by the plaintiff and facts of negative presupposition raised in the defendant's rebuttal must be distinguished precisely. .. Judges are unhappy with scholarly treaties on civil law which neglect to draw this strict distinction. As a result, the judges give a systematic character to their analysis by adding this new element of interpretation to the traditional method in order to extract a code of "judicial norms" from the Civil Code.*

As this explanation shows, the JUF theory is originated from "burden of persuasion". Independently, we pointed out that "negation as failure" in logic programming is actually related with "burden of persuasion"[Satoh07] and therefore, it is natural that we use logic programming to represent the JUF theory and we showed out how to implement the JUF theory in logic programming[Satoh09]. However, since possible users of computational JUF theory are law school students and lawyers, and they are not familiar with semantics of logic programming, we need more user-oriented knowledge representation for the JUF theory. So, we introduced a new system to reason about the JUF theory by a PROLOG-based meta-interpreter called the *PROLEG* system (PROlog based LEGal reasoning support system)[Satoh12] which has been paid attention by some lawyers [Kawamura12, page 11].

The main investigator of JUF theory, Ito explains JUF theory using *openness* of the conditions[Ito08]. He divides ultimate facts in a condition of a rule into two categories; one category corresponds with facts which normally lead to the conclusion of a rule and the other category corresponds with facts which represent exceptional situation. Ito argues that a fact in the latter category is regarded as "open" so that the truth value is not decided until the counterpart explicitly prove the fact. Therefore, A judge only considers facts in the former category in order to draw a conclusion. Based on this idea, we developed the PROLEG system where we define the JUF theory by a set of general rules and exceptions.

In this paper, we investigate a generality of PROLEG representation of legal knowledge consisting of general rules and exceptions. Since it seems that general rules and exceptions is a kind of principle of normal legal systems and therefore, in this paper, we would like to evaluate how general our representation is.

We abstract the original definition of PROLEG in order to compare its representation power with PROLOG as follows.

- In the original definition, we do not implicitly consider a loop of exceptions since if it were introduced, non existence of conclusion or multiple existence of different conclusions would have occurred whereas we allow a loop in this paper.
- In the original definition, we distinguish from facts and rules where facts are actions performed in the court activity like allegements whereas, in this paper, we just consider usual logical atoms in stead of these court actions.

Then, abstracting the PROLEG representation, we can show that the representation power is theoretical equivalent to PROLOG representation.

We also consider the generality from the practical side by showing that examples from argumentation theory and other legal domains besides civil code can be expressed in the PROLEG language. As a side effect we also give a unified view of Pollock's two kinds of counter-arguments[Pollock95].

The structure of this paper is as follows. In Section 2 and Section 3, we give a theoretical generality of PROLEG. In Section 2, we give a formal semantics of PROLEG and in Section 3, we show equivalence of PROLEG and PROLOG representation. In Section 4, we give a practical generality of PROLEG by showing

that PROLEG representation can be applied to other domains such as argumentation theory and other legal system besides civil code. In Section 5 we discuss related work and we summarize our contribution in Section 6.

2 Formalization

We define a PROLEG program as follows. A rule is a horn clause of the form $H \Leftarrow B_1, ..., B_n$[1] where $H, B_1, ..., B_n$ are atoms. We denote the dead of the rule R as $head(R)$. An exception is an expression of the form $exception(H, E)$ where H, E are atoms. A program P is a pair $\langle \mathcal{H}, \mathcal{E} \rangle$ where $\mathcal{H}$ is a set of rules and $\mathcal{E}$ is a set of exceptions.

Let M be a set of atoms. We define *a set of applicable rules w.r.t.* M, $\mathcal{H}^M$, as follows:

$$\{R \in \mathcal{H} | \neg \exists E \in \mathcal{E} \text{ s.t. } exception(head(R), E) \text{ and } E \in M\}.$$

This means that if some exception is found for a conclusion H of rule R, we do not consider such rule R for derivation.

Then, the semantics of P (called an *extension* of P) is given as a set of atoms M s.t. $M = min(\mathcal{H}^M)$ where $min(T)$ is the minimum model of a set of Horn clauses T.

The intuitive explanation of an extension is that we firstly guess a possible candidate of extension and if it is self consistent, that is everything we can derive from the candidate and program whose rules are excluded if the exception is included in the candidate and nothing extra is in the candidate.

Example 1.

- Suppose that plaintiff claims that a contract between him and defendant exists because plaintiff offered defendant to sell his car, and defendant accepted.
- Then, suppose that defendant concedes plaintiff's claims concerning offer and acceptance, but claims an exception, viz. that she was insane when she accepted plaintiff's offer.
- To prove defendant's insanity, the defendant must prove that there is an official-looking document containing a judicial decision declaring her insane.
- In order to claim that this document is not authentic by proving that the correct stamp is not put on the document, the burden of its proof is on the one who claims this, so a switch of burden of proof occurs again.

We introduce propositions to discuss the above problem. "`valid_contract` " means that there is a valid contract, "`offer`" expresses that there is an offer of the contract, "`acceptance`" means that there is an acceptance for the offer, "`insane`" means that she is insane when the contract is made, "`document_insane`"

[1] B_i part can be empty. In this case, we write it as $H \Leftarrow$.

means that the court issues a document which declares that defendant was insane at the time of contract, "`false_document`" means that the document is not authentic, and "`correct_stamp`" that the document has a correct stamp.

Let $P_1 = \{\mathcal{H}_1, \mathcal{E}_1\}$ and $\mathcal{H}_1$ be the following set of rules[2]:

```
valid_contract <= offer,acceptance.
offer <=.
acceptance <=.
insane <= document_insane.
document_insane <=.
false_document <= incorrect_stamp.
incorrect_stamp <=.
```

and $\mathcal{E}_1$ be the following set of exceptions:

```
exception(valid_contract,insane).
exception(insane,false_document).
```

Let M_1 be {`valid_contract`, `false_document`, `offer`, `acceptance`, `document_insane`, `incorrect_stamp`}.
Then, since `false_document` $\in M_1$ and `"exception(insane,false_document)."` exists we cannot include `"insane <= document_insane."` in $\mathcal{H}^{M_1}$. Therefore, $\mathcal{H}^{M_1}$ becomes the following rules

```
valid_contract <= offer,acceptance.
offer <=.
acceptance <=.
document_insane <=.
false_document <= incorrect_stamp.
incorrect_stamp <=.
```

Then, since $M_1 = min(\mathcal{H}^{M_1})$, M_1 is an extension.

Example 2. Let $P_2 = \{\mathcal{H}_2, \mathcal{E}_2\}$ and $\mathcal{H}_2$ be the following set of rules:

```
P <=.
Q <=.
```

and $\mathcal{E}_2$ be the following set of exceptions:

```
exception(P,Q).
exception(Q,P).
```

Let $M_2 = \{$`P`$\}$. Then, since $\mathcal{H}^{M_2} = \{$`P <=.`$\}$ and $M_2 = min(\mathcal{H}^{M_2})$, M_2 is an extension. Let $M_3 = \{$`Q`$\}$. Then, since $\mathcal{H}^{M_3} = \{$`Q <=.`$\}$ and $M_3 = min(\mathcal{H}^{M_3})$, M_3 is also an extension.

Example 3. Let $P_3 = \{\mathcal{H}_3, \mathcal{E}_3\}$ and $\mathcal{H}_3$ be the following set of rules:

```
P <=.
```

[2] This example is used in [Prakken01].

and $\mathcal{E}_3$ be the following set of exceptions:

```
exception(P,P).
```

Let $M_4 = \{\texttt{P}\}$. Then, since $\mathcal{H}^{M_4} = \{\}$ and $M_4 \neq min(\mathcal{H}_4^M)$, M_4 is not an extension. Let $M_5 = \{\}$. Then, since $\mathcal{H}^{M_5} = \{\texttt{P <=.}\}$ and $M_5 \neq min(\mathcal{H}_5^M)$, M_5 is neither an extension. Since there is no other possibility of extension of P_2, there is no extension for P_2.

3 Relation between PROLEG program and a general logic program

3.1 Translation from PROLEG to PROLOG

We firstly review the translation from PROLEG to PROLOG [Satoh09]. Let $P = \langle \mathcal{H}, \mathcal{E} \rangle$ be the following PROLEG program[3] where $\mathcal{H}$ is as follows:

$C \Leftarrow B_{11}, ..., B_{1n_1}.$
$C \Leftarrow B_{21}, ..., B_{2n_2}.$
$\vdots$
$C \Leftarrow B_{k1}, ..., B_{kn_k}.$

and $\mathcal{E}$ is as follows:

$exception(C, E_1).$
$\vdots$
$exception(C, E_m).$

Then, a translation of a PROLEG program P, $tr_e(P)$ is defined as the following PROLOG program:

$C : -B_{11}, ..., B_{1n_1},$ `not` $E_1, ...,$ `not` $E_m.$
$C : -B_{21}, ..., B_{2n_2},$ `not` $E_1, ...,$ `not` $E_m.$
$\vdots$
$C : -B_{k1}, ..., B_{kn_k},$ `not` $E_1, ...,$ `not` $E_m.$

Theorem 1. *Let P be a PROLEG program and $tr_e(P)$ be a PROLOG program obtained by the above translation. Then, there is an extension of P, M if and only if there is an answer set M of $tr_e(P)$.*

Example 4. Consider Example 1. $tr_e(P_1)$ is as follows:

```
valid_contract: −offer, acceptance, not insane.
offer.
acceptance.
insane: −document_insane, not false_document.
document_insane.
```

[3] We only consider rules and exceptions which are related with the conclusion C for simplicity. If there are multiple conclusions then we just do the same translation for each rules and exceptions whose conclusions are the same.

`false_document`: $-$`incorrect_stamp`.
`incorrect_stamp`.

Then the stable model of this PROLOG program is {`valid_contract`, `false_document`, `offer`, `acceptance`, `document_insane`, `incorrect_stamp`} which is equivalent to M_1 in Example 1.

3.2 Translation from PROLOG to PROLEG

We now give a translation from a PROLOG program to a PROLEG program. The difficulty lies in that the above translation from PROLEG to PROLOG, rules with the same head in PROLOG will have the same negative literals in the body, whereas a usual PROLOG program does not have such property. We solve this difficulty by introducing a new predicate for each rule.

Let P be the following PROLOG program:

C: $-B_{11}, ..., B_{1n_1}$, `not` $E_{11}, ...,$ `not` E_{1m_1}.
C: $-B_{21}, ..., B_{2n_2}$, `not` $E_{21}, ...,$ `not` E_{2m_1}.
$\vdots$
C: $-B_{k1}, ..., B_{kn_k}$, `not` $E_{k1}, ...,$ `not` E_{km_k}.

Then, a translation of a PROLOG program P, $tr_o(P)$ is defined as the following PROLEG program $\langle \mathcal{H}_o, \mathcal{E}_o \rangle$ where $\mathcal{H}_o$ is as follows:

$C \Leftarrow C_1$.
$C_1 \Leftarrow B_{11}, ..., B_{1n_1}$.
$C \Leftarrow C_2$.
$C_2 \Leftarrow B_{21}, ..., B_{2n_2}$.
$\vdots$
$C \Leftarrow C_k$.
$C_k \Leftarrow B_{k1}, ..., B_{kn_k}$.

and $\mathcal{E}_e$ is as follows:

$exception(C_1, E_{11})$.
$\vdots$
$exception(C_1, E_{1m_1})$.
$exception(C_2, E_{21})$.
$\vdots$
$exception(C_2, E_{2m_2})$.
$\vdots$
$exception(C_k, E_{k1})$.
$\vdots$
$exception(C_k, E_{km_k})$.

Theorem 2. *Let P be a PROLOG program and $tr_o(P)$ be a PROLEG program obtained by the above translation. Then, there is an answer set M of P if and*

only if there is an extension of $tr_o(P)$ such that a set deleting newly introduced predicates from the extension equals to M.

Example 5. Consider the PROLOG program $tr_e(P_1)$ in Example 4. Then, $tr_o(tr_e(P_1)) = \langle \mathcal{H}_{oe}, \mathcal{E}_{oe} \rangle$ becomes (by introducing some new predicates) where $\mathcal{H}_{oe}$ is as follows:

```
valid_contract <= c1.
c1 <= offer,acceptance.
offer <=.
acceptance <=.
insane <= c2.
c2 <= document_insane.
document_insane <=.
false_document <= incorrect_stamp.
incorrect_stamp <=.
```

and $\mathcal{E}_{oe}$ be the following set of exceptions:

```
exception(c1,insane).
exception(c2,false_document).
```

Then, an extension of $tr_o(tr_e(P_1))$ becomes {`valid_contract`, `false_document`, `offer`, `acceptance`, `document_insane`, `incorrect_stamp`, `c1`}. By removing `c1` from the extension we have the stable model of $tr_e(P_1)$. Note that we do not infer `insane` by the rule `"insane <= c2."` since `false_document` is derived and `"exception(c2,false_document)."` exists.

4 Practical Generality of PROLEG Representation to Other Domains

In the previous sections, we talk about theoretical generality of PROLEG whereas in this section, we give a practical generality of PROLEG by showing that PROLEG representation can be applied to other domains such as argumentation theory and other legal domains.

4.1 Application to Pollock's Argumentation Theory

Walton neatly explains Pollock's distinction of counter-arguments as follows [Walton09]:

> Pollock's distinction between two kinds of counter-arguments called rebutting defeaters and undercutting defeaters (often referred to as rebutters versus undercutters) is drawn as follows. A rebutting defeater gives a reason for denying a claim by arguing that the claim is a false previously held belief[Pollock95, page 40]. An undercutting defeater attacks the inferential link between the claim and the reason supporting it by weakening or removing the reason that supported the claim. The way Pollock uses these terms, a rebutter gives a reason to show the conclusion is false, whereas an undercutter merely raises doubt whether the inference supporting the conclusion holds. It does not show that the conclusion is false.

Pollock uses the following example[Pollock95, page 41] to explain undercutting defeater.

> For instance, suppose x looks red to me, but I know that x is illuminated by red lights and red lights can make objects look red when they are not. Knowing this defeats the prima facie reason, but it is not a reason for thinking that x is not red. After all, red objects look red in red light too. This is an undercutting defeater.

At a glance, it seems that the denial of the conclusion and the denial of applicability of the rule are different since the denial of applicability does not directly mean the denial of the conclusion. However, if we consider the *closed world assumption* or 2-valued semantics, the denial of the conclusion and the denial of applicability of the rule can be converted each other as the following discussion shows.

My understanding of rebutting is that no matter how many rules are applicable for a given conclusion, the rebutting just ignores these applicable rules and makes the conclusion false. This is very similar to PROLEG system; that is, if $exception(H, E)$ is in a PROLEG program and no matter how many rules can be used to derive this conclusion H, H will not be derived if E is true and by the closed world assumption H becomes false.

On the other hand, we could represent an undercutting defeater in PROLOG by H: $-B_1, ...B_n$, `not` E meaning that a rule H: $-B_1, ...B_n$ is not applied if E happens. It does not interfere applicability of another rule for H, H: $-B'_1, ...B'_m$ [4].

Then, the according to the equivalent translation between PROLOG and PROLEG programs in Section 3, rebutting and undercutting could be interchangeable as follows:

From rebutting to undercutting:

$H \Leftarrow B_{11}, ...B_{1n_1}$
$\vdots$
$H \Leftarrow B_{m1}, ...B_{mn_m}$
$exception(H, E)$

can be translated into the following rule sets:

H: $-B_{11}, ...B_{1n_1}$, `not` E
$\vdots$
H: $-B_{m1}, ...B_{mn_m}$, `not` E

From undercutting to rebutting:

H: $-B_1, ...B_n$, `not` E

can be translated into the following rule sets:

$H \Leftarrow C_1$.

[4] Consider that we add a following rule to Pollock example: "x is found to be a mature apple by chemical experiment, x is red". If the condition of this rule is true, we could conclude that x is red even if the Pollock's original rule is not applicable.

$C_1 \Leftarrow B_1, ...B_n.$
$exception(C_1, E).$

This interchangeability is coming from two-valued semantics of PROLEG and PROLOG. So, at least in two-valued semantics of logic programming, distinction of rebutting and undercutting can be converted each other.

4.2 Application to Penal Code

A general rule of criminal law is that if the prosecutor proves that the external elements (*actus reus*) and the internal elements (*mens rea*) of the crime then the actor of the crime is prosecuted. However, there are some exceptions such as insanity, self-defense. However, for self-defense, imminent and unlawful infringement must exist but for a situation such that the defender have an intention of attack using this chance, then imminent condition will be denied[5].

So, we could represent these general rules and exceptions in PROLEG. For example, for the murder we could formalize a general rule as:

```
guilty(X,Y,murder)  <=
  intention_of_killing(X,Y), killing_act(X,Y),died(Y).
insane(X)  <=  psychiatric_test_insanity(X).
self_defense(X,Y) <=
  imminent_infringement(Y,X), unlawful_infringement(Y,X),
  defense_action_appropriate(X,Y).
```

and exceptions as:

```
exception(guilty(X,Y,murder),insane(X)).
exception(guilty(X,Y,murder),self_defense(X,Y)).
exception(imminent_infringement(Y,X),intention_of_attack(X,Y)).
```

4.3 Application to Code of Criminal Procedure

In Japanese Code of Criminal Procedure, Phrase 1 of Article 197 states that "with regard to investigation, such examination as is necessary to achieve its objective may be conducted; provided, however, that compulsory dispositions shall not be applied unless special provisions have been established in this Code". So, a general rule is that compulsory dispositions are prohibited, but in the exceptional situation where special provisions are made, the compulsory dispositions are allowed. Another general rule says that if dispositions are not compulsory (or in other words depositions are with consent), they are allowed, but in the exceptional situation where the dispositions are the above necessary level, they are prohibited.

We can write these general rules as:

[5] Supreme Court Case:1977.7.21,31-4 Keisyu. 747.

```
prohibited(Police,X,Disposition)  <=
  compulsory(Disposition).
allowed(Police,X,Disposition)  <=
  with_consent(X,Disposition).
```

and exceptions as:

```
exception(prohibited(Police,X,Disposition),
  provisions(Disposition)).
exception(allowed(Police,X,Disposition),
  unnecessary(Police,X,Disposition)).
```

4.4 Application to Constitution Law

In the Constitution of Japan, Article 21 says that freedom of assembly and association as well as speech, press and all other forms of expression are guaranteed. A general rule for constitutionality of constraining expression is that its purpose must be compelling and its constraining method must be narrowly tailored. However, if the constraint is content-neutral regulation, then the standard of judicial review becomes softer as an exception.

So, we can write this general rule as:

```
prohibitted(X,Y,Constraint)  <=
  constrain(X,Y,Constraint).
allowed(X,Y,Constraint) <=
  purpose_of_constraint_compelling(X,Y,Constraint,Purpose),
  method_of_constraint_narrowly_tailored(X,Y,Constraint,Method).
allowed(X,Y,Constraint) <=
  content_neutral_constraint(Constraint),
  purpose_of_constraint_imporant(X,Y,Constraint,Purpose),
  method_of_constraint_least_restrictive(X,Y,Constraint,Method).
```

and exception can be expressed as follows:

```
exception(prohibitted(X,Y,ConstrainedExpression),
          allowed(X,Y,ConstrainedExpression)).
```

5 Related Work

The idea of representing legal knowledge representation by general rules with exceptions is not new. For example, Gordon[Gordon88] explains how general rules with exceptions is useful for legal knowledge representation and proposes Ob-log-2 system[Gordon87].

Gordon[Gordon93] proposes another system which uses technique of rule naming and meta-predicate of applicability of rules. If certain conditions are fulfilled, then applicability of a rule is prevented.

McCarty and Cohen[McCarty94] represent explicit exceptions using intuitionistic logic programming and show that nonmonotonic reasoning examples can be easily encoded in their framework.

Prakken and Sartor[Prakken97] propose a method of handling exceptions by using priority representation over rules in an extended logic program.

Verheij[Verheij03] introduces notions of dialectical negation and conditional justification which can express rebutters and undercutters.

Governatori et al[Governatori04] proposes defeasible logic and show that the logic can handle exceptions.

Gordon et al[Gordon07] propose a method of handling arguments in a sophisticated manner so that a way of rebutting/undercutting/undermining arguments can be captured in a unified framework.

Although we need to compare with these works theoretically in the future, what we can say at the moment is that the main purpose of our research is quite different from the above works. The direction would be almost opposite. Most of the works are trying to enhance representation power of the system to distinguish subtle difference in various legal reasoning schemes, we pursue minimum sufficient functionality of legal reasoning related with the JUF theory and show that our simple mechanism is sufficient to handle the JUF theory and also show that it can be applied to other legal domain.

The most closely related work is by Kowalski and Sadri [Kowalski91]. Kowalski and Sadri propose a language which consists of rules and exceptions. In their language, rules with explicit negative literals in the head are used to express exceptions for rules with its positive literal. A conclusion of the positive literal is prevented if a rule of the conclusion with its counterpart of the negative literal is applicable. They use a syntax of extended logic program where they can use not only negation as failure but also explicit negation and define new semantics which can handle exceptions by giving priority of derivation to rules with explicit negative literals in the head. They show a translation of their formalism into PROLOG, but they do not show a translation of PROLOG into their formalism so it is not clear that the expressive power is the same as PROLOG.

On the other hand, we do not allow negation as failure in the body of the rule nor use explicit exception for rules. This simplifies a semantics and it is easy to show that PROLOG and PROLEG has the same expressive power.

There is another similar work to [Kowalski91] which uses explicit negative literals in the head in Datalog settings [Halfeld98]. The semantics of [Halfeld98] and [Kowalski91] are different, At least they only consider Datalog without function symbols whereas we consider a general logic program. Like [Kowalski91], whether representation power of their framework is same as or above the power of PROLOG is not known.

6 Conclusion

In this paper, we discuss a generality of PROLEG representation. Our claims are as follows:

– We show that the representation power of PROLEG is same as PROLOG.
– The representation of "general rules and exceptions" in PROLEG is very simple but powerful enough to formalize legal reasoning in not only for Japanese Civil Code, but also in other legal domain.

As a future research, we would like to pursue practical support tools based on PROLEG representation not only for the JUF theory but also for other legal domains.

Acknowledgments: I am very grateful to anonymous referees for pointing out various related works. This work is partially supported by Grant-in-Aid for Scientific Research(B),23300062.

References

[Gordon87] Gordon, T. F., "Oblog-2: A Hybrid Knowledge Representation System for Defeasible Reasoning", Proceedings of the First International Conference on Artificial Intelligence and Law, pp.231-239 (1987).

[Gordon88] Gordon, T. F., "The Importance of Nonmonotonicity for Legal Reasoning", Expert Systems in Law; Impacts on Legal Theory and Computer Law, (H. Fiedler, F. Haft and R. Traunmu"ller eds.) pp.111-126 (1988).

[Gordon93] Gordon, T. F., "The Pleadings Game; Formalizing Procedural Justice", Proceedings of the Fourth International Conference on Artificial Intelligence and Law, pp.10–19(1993).

[Governatori04] Governatori, G., Maher, M. J., Antoniou, G., and Billington, D., "Argumentation Semantics for Defeasible Logic", Journal of Logic and Computation, Vol.14, pp.675–702 (2004).

[Gordon07] Gordon, T. F., Prakken, H., and Walton, D., "The Carneades Model of Argument and Burden of Proof", Artificial Intelligence, vol.171, No.10-11 pp.875–896 (2007).

[Halfeld98] Halfeld Ferrari Alves,M., Laurent D., Spyratos N., "Update Rules in Datalog Programs", Journal of Logic and Computation, Vol 8, No 6, pp.745-775 (1998).

[Ito08] Ito, S., "Lecture Series on Ultimate Facts", Shojihomu (2008) (in Japanese).

[McCarty94] McCarty, L. T., and Cohen, W. V., "The Case for Explicit Exceptions", Methods of Logic in Computer Science, Vol.1, pp.19–50 (1994).

[Kawamura12] Kawamura, H.,, "Fundamental Theory of Civil Litigation: Structural Analysis of Legal Judgment 1 (the second volume), What is The Japanese Presupposed Ultimate Fact Theory?", Horitsu Jiho, Vol. 2146, pp.9-24 (2012) (in Japanese).

[Kitamura93] Kitamura, I., "The Judiciary in Contemporary Society: Japan", Case Western Reserve Journal of International Law, 00087254, Sprint93, Vol. 25, Issue 2 (1993).

[Kowalski91] Kowalski, R. A., Sadri, F., "Logic Programs with Exceptions", New Generation Computing, Vol.9(3/4), pp.387-400 (1991).

[Pollock95] Pollock, J., "Cognitive Carpentry", Cambridge, Mass: MIT Press (1995).

[Prakken97] Prakken, H., and Sartor, G., "Argument-based Extended Logic Programming with Defeasible Priorities", Journal of Applied Non-classical Logics, Vol. 7, pp.25–75 (1997).

[Prakken01] Prakken, H., Modelling Defeasibility in Law: Logic or Procedure?, *Fundam. Inform.*, 48(2-3), pp. 253-271 (2001).

[Satoh07] Satoh, K., Tojo, S., Suzuki, Y., "Formalizing a Switch of Burden of Proof by Logic Programming", Proc. of the 1st International Workshop on Juris-Informatics (JURISIN 2007), pp. 76 – 85 (2007). (`http://research.nii.ac.jp/~ksatoh/papers/jurisin2007.pdf`)

[Satoh09] Satoh, K., Kubota, M., Nishigai, Y., Takano, C., "Translating the Japanese Presupposed Ultimate Fact Theory into Logic Programming", Proc. of JURIX 2009, pp.162-171 (2009).

[Satoh12] Satoh, K., Asai, K., Kogawa, T., Kubota, M., Nakamura, M., Nishigai, Y., Shirakawa, K., Takano, C., "PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology", New Frontiers in Artificial Intelligence: JSAI-isAI 2010 Workshops, Revised Selected Papers, LNAI 6797, pp. 153-164 (2012).

[Verheij03] Verheij, B., "DefLog: on the Logical Interpretation of Prima Facie Justified Assumptions", Journal of Logic and Computation, Vol. 13, 319–346 (2003).

[Walton09] Walton, D., "Objections, Rebuttals and Refutations", in: J. Ritola (Ed.), Argument Cultures: Proceedings of OSSA 09, pp. 1-10 (2009).