Identification of Implicit Legal Requirements with Legal Abstract Knowledge

Seiichiro SAKURAI
Department of Systems Science
Tokyo Institute of Technology
Yokohama, Japan

Hajime YOSHINO
Faculty of Law
Meiji Gakuin University
Tokyo, Japan

Abstract

In order to acquire legal rules from legal texts, legal requirements and legal effects must be identified. However, some of legal requirements are expressed implicitly. Such implicit legal requirements can be found by lawyers when they understand legal texts. In this paper, to mechanize legal knowledge acquisition process, a lawyer's understanding process of legal texts is analyzed. The lawyer's understanding process can be viewed as an abductive reasoning process, since the lawyer can introduce implicit legal requirements which are not appeared in legal texts. This paper models such a reasoning process when lawyers understand legal texts. Based on the analysis of lawyer's understanding process, a knowledge acquisition support system is proposed.

1 Introduction

In order to realize a legal expert system, we must translate legal texts into machine executable forms. If legal rules are translated into rules like Prolog program, a legal knowledge-base can be constructed. Since a legal rule consists of legal requirements and a legal effect, they must be identified correctly. However, since some of legal requirements are described implicitly, machine translation is very difficult. In spite of implicit legal requirements, lawyers can correctly identify legal rules by clarifying implicit legal requirements. Since lawyers seem to use their legal expertise in order to fill the gap between the real meaning of legal texts and the literal meaning of them, such legal expertise should be clarified. This paper analyzes a lawyer's understanding process of legal rules. In the analysis, a kind of legal knowledge seems to guide the lawyer's understanding process. We call such legal knowledge as a legal abstraction, since it is widely applicable knowledge and it is abstracted from specific legal knowledge. Based on the analysis of human understanding process, a knowledge acquisition support system is proposed. Our system acquires new knowledge by using given legal abstractions, which is represented by a Higher Order Horn Clause.

2 Knowledge Representation

2.1 Compound Predicate Formula

Legal knowledge can be represented by a set of *compound predicate formulas(CPF)*. A CPF is a rule, which can be interpreted by a deductive reasoning engine like a Prolog interpreter. A legal rule can be represented by a CPF as follows:

$$\overbrace{r_1, r_2, ..., r_n}^{\text{legal requirements}} \rightarrow \overbrace{e}^{\text{legal effect}}$$

r_i represents a legal requirement and e represents a legal effect. r_i and e are called compound predicate terms. A compound predicate term can be represented as follows:

$$predicate(ID, CaseList)$$

predicate is a predicate name, and ID is an identifier of predicate. The identifier is used as a reference of predicate instance. *CaseList* is a list of pairs and each pair represents case role and filler. Each filler may be a compound predicate term. Case and filler are separated by ":". For example, a concept, "a contract is concluded", is represented as follows:

$$be\_concluded(id1, contract(id2, [agt : \_, obj : \_, ...]))$$

In the above example, "id1" and "id2" are predicate identifiers, and "agt", which stands for *agent* case, and "obj", which stands for *object* case, are case name abbreviations. "_" is an anonymous variable. Note that the predicate "contract" appears as a subterm in the above example, and the predicate contract should be proved

in order to verify whether the concept, "a contract is concluded", holds or not.

2.2 Interpreting CPF

A CPF is a formula for representing a legal rule and it directly corresponds to a legal sentence. Since legal concepts in a CPF are represented as chunks of primitive legal concepts, it seems to be comparatively easy for lawyers to understand a CPF. For example, the following legal rule can be represented by a CPF in Figure 1.

> An offer becomes effective when it reaches the offeree.

In the above legal rule, the legal effect is "an offer becomes effective" and the legal requirement is "the offer reaches the offeree". They are represented by compound predicate terms in Figure 1.

```
become_effective(_,[obj:offer(O,[agt:A,
                                 goa:offeree(B,C)]),
                    tim:D]) ←
    reach(_,[obj:offer(O,[agt:A,goa:offeree(B,C)]),
             tim:D,
             goa:offeree(B,C)]).
```

Figure 1: Example of CPF

By using compound predicate terms, the legal requirements and the legal effect can be distinguished easily. However, since a compound predicate term may contain other compound predicate terms as its subterms, computational interpretation of CPFs is not easy. In order to enable efficient interpretation of CPFs, a fixed legal dictionary, including legal taxonomy, is assumed. By using the legal dictionary, a CPF is converted into a flat CPF automatically like [5]. In the conversion, the case names are eliminated and the case names are represented by argument places. For example, the CPF in Figure 1 is converted as follows:

```
become_effective(_,O,D) ←
    offer(O,A,B), offeree(B,_), reach(_,O,D,B).
```

As shown in the above example, the filler of the obj of the become_effective is replaced with the identifier. The second argument of the become_effective represents the obj's filler and the third argument represents the tim's filler. "Tim" is the abbreviation of time. Similarly, other predicates are converted. In a flat CPF, only identifiers are allowed as its fillers. Since a flat CPF is a function free Horn Clause, it can be executed directly by a Prolog interpreter. If the predicate is not referred from other predicate, its identifier is omitted, too. Therefore, the above flat CPF can be represented as follows:

```
become_effective(O,D) ←
    offer(O,A,B), offeree(B,_), reach(O,D,B).
```

In the real application, we should provide other rules, which bridge the CPFs and the given facts. For example, since the given facts may not designate the offeree, the offeree's definition rule will be needed. Such definition rules are assumed in the legal dictionary.

3 Human Understanding Process

A lawyer's understanding process is analyzed by using United Nations Convention on Contracts for the International Sale of Goods. Since the convention is applied to contracts of sale of goods, Article 23, which is applied to "conclusion of contract," is concerned. Article 23 is shown in Figure 2.

> Article 23
>
> A contract is concluded at the moment when an acceptance of an offer becomes effective in accordance with the provisions of this Convention.

Figure 2: United Nations Convention on Contracts for the International Sale of Goods

Since a legal rule consists of legal requirements and a legal effect, the requirements and the effect must be identified. In Figure 2, even a novice or a beginner of law can identify that the legal effect is "a contract is concluded". Since most of articles describe such legal effects explicitly, the identification of the legal effect is an easy task. However, it is not so easy to identify legal requirements of articles because of implicitly described legal requirements. Of course, the identification of explicit legal requirements is easy. Article 23 is initially understood as a rule shown in Figure 3. In Figure 3, unused cases are omitted for simplicity. Indeed, since some relations between primitive concepts are not sufficiently specified, the ideal rule should specify such relations.

A novice would understand Article 23 as such a rule, since the rule seems to represent the literal meaning of the article. Once such a rule is obtained, even a novice tries to its validity. The rule can be verified by assuming an ideal situation in which a contract is concluded. In such a situation, the acceptance of the offer must

```
be_concluded(_,[agt:[A,B],
               obj:contract(_,[agt:[A,B]]),
               tim:D]) ←
    become_effective(_,
        [obj:acceptance(_,
              [agt:B,
               obj:offer(_, [agt:A,goa:B]),
               goa:A]),
         tim:D).
```

Figure 3: Initial Understanding of Article 23

become effective. Since the following Article 18-2 describes when the acceptance becomes effective, the rule of Article 18-2 will be used for the verification of the CPF in Figure3.

> Article 18
> 1. A statement made by or other conduct of the offeree indicating assent to an offer is an acceptance.
> 2. An acceptance of an offer becomes effective at the moment the indication of assent reaches the offeror.

The CPF of Article 18-2 is shown in Figure 4.

```
become_effective(_,
        [obj:
         acceptance(IdA,
           [agt:A,
            obj:offer(C,[agt:offeror(B,[obj:C]),
                         goa:A]),
            goa:offeror(B,[obj:C])])]) ←
  reach(_,
   [obj:indication(IdA,
        [agt:A,
         obj:assent(_,
              [agt:A,
               obj:offer(C,[agt:offeror(B,[obj:C]),
                            goa:A]),
               goa:offeror(B,[obj:C])]),
         goa:offeror(B,[obj:C])]),
    goa:offeror(B,[obj:C])]).
```

Figure 4: CPF of Article 18-2

If the legal requirement of Article 18-2, "the indication of assent reaches the offeror", is assumed in the ideal situation, "a contract is concluded" can be proved by using two legal rules, Article 23 and Article 18-2. In this way, rules translated from legal texts are verified. Because of such successful explanation, the novice's understanding process may halt. Are the above rules really correct ones? In spite of such successful explanation, most lawyers can provide its counter-example. For example, by means of "withdrawal of offer," even if the indication of assent reaches, the contract cannot be concluded. Therefore, lawyers' understanding process don't halt when only such explanation is made successfully. The lawyers' massive legal expertise seems to enable them to investigate further.

When lawyers understand legal texts, they also confirm whether the legal effect can be derived from the identified legal effects or not. Then they judge the validity of the derivation by using their legal expertise. The verification process can be seen as a backward reasoning process of a legal effect. Figure 5 shows such an explanation.

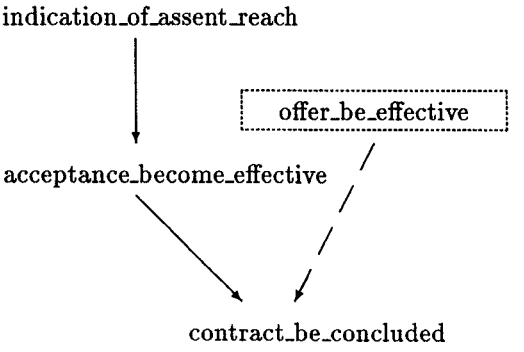


Figure 5: Explanation

In Figure 5, each subgoal, compound predicate term, is represented by a proposition. The first top goal is the "contract_be_concluded". In the explanation, although "offer_be_effective" does not appear in the legal texts, it is regarded as a legal requirement by a lawyer. The understanding process can be seen as an abductive reasoning process since the explanation is constructed by adding implicit legal requirements. In the search of such legal requirements, the candidates of the legal requirements seem to be weakly constrained by lawyers' abstract legal knowledge. In Figure 5, a general principle about contract, "a contract is a match of offer and acceptance" weakly constrains the candidates.

After the verification, a new legal requirement, "offer is effective", is added into the body of the legal rule of Article 23. For example, the following rule can be made.

```
contract_be_concluded ←
    acceptance_become_effective,          (1)
    offer_be_effective
```

If the legal rule whose effect is "the acceptance becomes effective" is modified so that its requirement part includes "the offer is effective", the same result can be derived. For example, the following rule can be made.

$$\begin{array}{l}\text{acceptance\_become\_effective} \leftarrow \\ \quad \text{indication\_of\_assent\_reach}, \\ \quad \text{offer\_be\_effective}\end{array} \qquad (2)$$

The former legal rule (1) is preferred since the former one matches the lawyer's abstract legal knowledge. Such a knowledge is called a legal abstraction, since it represents abstract knowledge which is abstracted from a specific legal knowledge. An example of a legal abstraction is shown in Figure 6. In Figure 6, the legal norm sentence is a super concept of the contract, and the indication of intention is a super concept of both the acceptance and the offer. Since the offer is prior to the acceptance, the former legal rule (1) matches the legal abstraction in Figure 6. If there exists another legal abstraction, the latter legal rule (2) may be preferred.

> If two indications of intention forms a legal norm sentence, the legal norm sentence is concluded when the prior indication of intention is effective and the posterior indication of intention becomes effective.

Figure 6: Example of Legal Abstraction

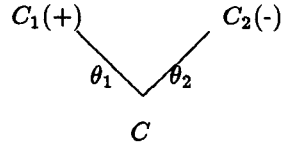
In this way, the lawyer's understanding process seems to be guided by legal abstractions. Therefore, legal knowledge acquisition from legal texts can be seen as abduction constrained by legal abstractions.

4 Abductive Reasoning and Abstract Legal Knowledge

Legal rule acquisition process can be viewed as an extraction process of a legal effect and an abductive explanation process of legal requirements. As an abductive explanation engine, an inverse resolution [3, 4, 5] method is used. The inverse resolution is an inverse operation of resolution. Figure 7 shows typical operations of the inverse resolution. The basic operation is performed by V operator. In the resolution, C is derived from C_1 and C_2. In the inverse resolution, C_1 is derived from C and C_2 or C_2 is derived from C and C_1.

Rouveirol[4] has proposed "Saturation Operator", which is an inverse resolution operator. The algorithm of Saturation [4] is shown in Figure 8.

- V Operator



- W Operator

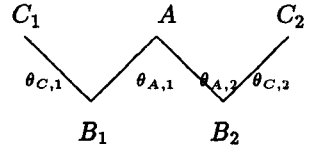


Figure 7: Inverse Resolution as Abduction[3]

Given the clause $C_e : T_e \leftarrow L_{C_e}$ and a domain theory T, proceed to steps 1,2 and 3:

1. Skolemization of C_e into $C_e\theta_s$ (one keeps track of θ_s for skolemization). For each literal $L_{C_{ei}}$ in the body of C_e, a clause C_{sk_i} is created, with $C_{sk_i} : L_{C_{ei}}\theta_s \leftarrow$.
2. Deductive phase: we apply all possible resolutions between the set of clauses $\{C_{sk_i}\}$ and the clauses of T.
3. If atoms have been deduced after step 2, they are transformed into unit clauses and added to $\{C_{sk_i}\}$. The process then iterates on step 2. If no literals have been deduced after step 2, the $\{C_{sk_i}\}$ are deskolemized and added to L_{C_e} to form the body of the saturated clause, C_s.

Figure 8: Algorithm of saturation[4]

Inverse resolution can construct rules which bridge a gap between a newly observed fact and known facts. However, the search space of the inverse resolution is very huge. For example, the offer_be_effective in Figure 5 is not described in the UNCCIS. Therefore, the offer_be_effective should be constructed as a new predicate. While W Operator is an operator which has an ability to invent a new predicate, the control of application of W Operator is very difficult. Furthermore, as shown in the previous section, the invention of new predicate should be controlled by legal abstractions. A legal abstraction is represented by Higher Order Horn Clause[1]. The legal abstraction in Figure 6 can be represented by Higher Order Horn Clause. Figure 9 shows an example of a legal abstraction represented by a Higher Order Horn Clause.

```
be_concluded(_,[agt:_,obj:Xid,tim:C]) ←
    X(Xid,[obj:_]),
    become_effective(_,[obj:Yid,tim:C]),
    Y(Yid,[obj:_]),
    be_effective(_,[obj:Zid,tim:C]),
    Z(Zid,[obj:_]).
Contraints:
    X is legal norm sentence.
    Y is an indication of intention.
    Z is an indication of intention.
    Z is prior to Y.
```

Figure 9: Legal Abstraction Represented by a Higher Order Horn Clause

In Figure 9, X, Y and Z are predicate variables, and the constraints can be considered as predicate types[1]. Since the constraints are used in Higher Order Unification, only the same type predicates are allowed to bind the predicate variables. For the efficiency, all predicates are instantiated by predicate constants. Under the restriction, a legal abstraction can be seen as a First Order Horn Clause. If the legal concept hierarchy is represented by a set of First Order Horn Clauses, the inheritance is realized as a deductive inference. In order to distinguish the legal deductive inference with the inheritance, a legal abstraction is represented by a Higher Order Horn Clause.

A legal abstraction can be converted into a flat legal abstraction. For example, the legal abstraction in Figure 9 can be converted as follows:

```
be_concluded(Xid) ←
    X(Xid,_,_),
    become_effective(Yid),
    Y(Yid,_,_,_),
    be_effective(Zid),
    Z(Zid,_,_,_).
```

(3)

The above example, each predicate arity is determined by using the legal dictionary. Based on the above legal abstraction (3), a new rule can be obtained as an instance. An instance of a legal abstraction can be obtained by substituting the predicate variables with the same type predicate constants. By assuming such legal abstraction, the legal effect can be proved also by a theorem prover like λprolog[1].

5 Knowledge Acquisition Support System

KASS is a legal knowledge acquisition support system based on the analysis of human understanding process. Figure 10 shows the overview of KASS.

KASS has two components for legal knowledge acquisition. One is a bug detection module and the other is an abductive reasoning engine based on the inverse resolution. The bug detection module is based on Shapiro's algorithmic debugger[6]. A set of facts and rules are given to the module. Then the module interacts the user to identify bug rules. Figure 11 shows an erroneous explanation when the offer is not effective. If all leaves of the explanation in Figure 11 are confirmed by the user, the following two rules may be considered as bug rules.

```
be_concluded(i4) ←
    contract(i4,a,b), offer(i5,a,b,c),                    (4)
    acceptance(i6,b,a,i5), become_effective(i6)
become_effective(i6) ←
    offer(i5,a,b,c), acceptance(i6,b,a,i5),               (5)
    reach(i6,a)
```

If multiple rules are considered as bug rules, legal abstractions are used for selection of a bug rule. If the legal abstraction (3) is given to the KASS, the legal rule (4) is selected as a bug rule. If another legal abstraction is given, the legal rule (5) may be selected.

The bug rule is given to the abductive reasoning engine and it is eliminated from the legal knowledge base. The reasoning engine then finds an appropriate legal abstraction. If the legal abstraction (3) is selected, the following instance can be generated.

```
be_concluded(Xid) ←
    contract(Xid,_,_),
    become_effective(Yid),
    acceptance(Yid,_,_,_),                                (6)
```

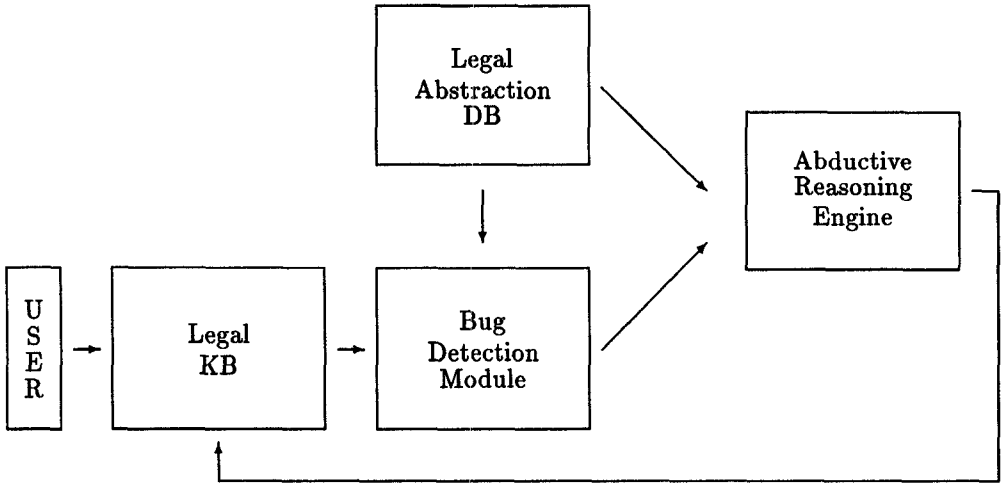


Figure 10: KASS

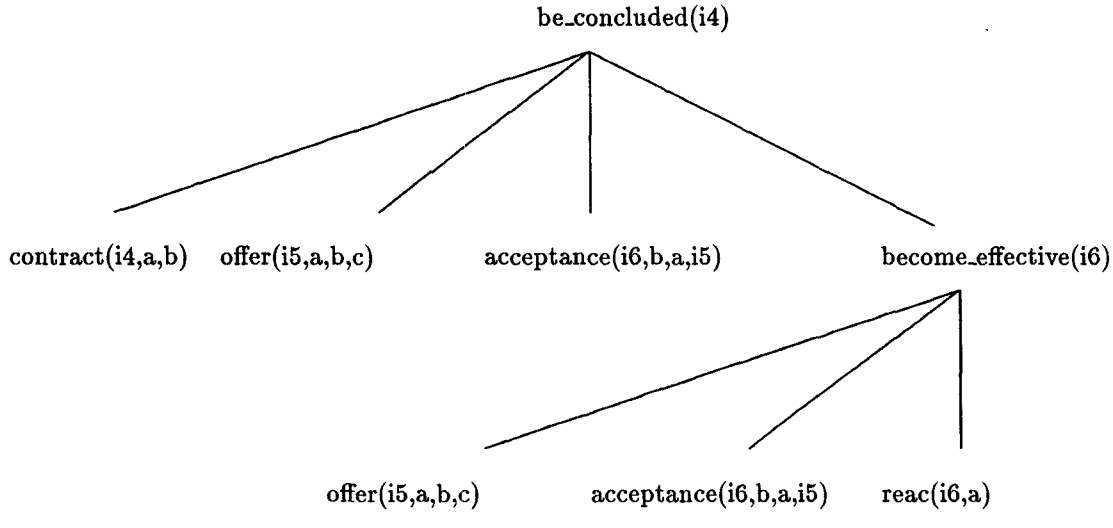


Figure 11: Erroneous Explanation

be_effective(Zid),
offer(Zid,_,_,_).

By referring the original rule, the above rule can be refined as follows:

be_concluded(A) ←
contract(A,B,C),
become_effective(D),
acceptance(D,C,B,E), (7)
be_effective(E),
offer(E,B,C,_).

In the above legal rule (7), a new predicate be_effective(offer(A,B,C,D)) is introduced. This new predicate will be a new subgoal of the abductive reasoning engine and other new rules may be found. If such auxiliary legal rules are found successfully, new legal rules , which satisfy legal abstractions, can be learned.

6 Learning Examples

Figure 12 is the initial knowledge base provided to the KASS. Although Article 18 is described by legal sentences, Article 18 is represented by a single legal rule. The temporal information is omitted for simplicity, in the real application, the theory for temporal reasoning should be needed.

UNCCIS

*Article*15 : become_effective(A) ←
offer(A,_,B,_), reach(A,B)
*Article*17 : be_terminated(A) ←
offer(A,B,_,_), rejection(C,A),
reach(C,B)
*Article*18 : become_effective(A) ←
acceptance(A,B,C,D),
offer(D,C,B,_), reach(A,C)
*Article*23 : be_concluded(A) ←
contract(A,B,C), offer(D,B,C,_),
acceptance(E,C,B,D),
become_effective(E)

auxiliary rules
not_be_terminated(X) ← not(be_terminated(X)).

Figure 12: Initial Knowledge

The following query is given to KASS. The right hand side of ⇐ is assumptions and the left hand side is the goal which should be proved.

be_conclude(i1) ⇐
contract(i1,a,b), offer(i2,a,b,c), reach(i2,b), (8)
acceptance(i3,b,a,i2), reach(i6,a)

The above example can be paraphrased as:

If the offer reaches the offerree and the acceptance reaches the offeror, is a contract concluded?

While the above example is only used for the verification of legal knowledge-base, if the following goal is proved, then the legal knowledge-base includes some bug rules.

be_conclude(i4) ⇐
contract(i4,a,b), offer(i5,a,b,c), reach(i5,b), (9)
acceptance(i6,b,a,i5),
rejection(i7,i5), reach(i7,a), reach(i6,a)

In the first step, the proof of be_concluded(i4) contains the application of Article 18 and Article 23, they may be erroneous rules. By using the legal abstraction (3), Article 23 is identified as an erroneous rule, and the following rule (10) is obtained as an instance of the legal abstraction.

be_concluded(A) ←
contract(A,B,C), offer(D,B,C,E), (10)
acceptance(F,C,B,D), become_effective(F),
be_effective(D).

The above rule can be converted into the CPF as shown in Figure 13 when the user wants to see the learned legal rule.

However, the verification process needs another assumption, be_effective. Then, KASS's abductive reasoning engine is invoked recursively, the following legal rule (11) can be obtained if appropriate legal abstractions are given.

be_effective(X) ←
offer(X,_,_,_), become_effective(X), (11)
not_be_terminated(X).

In this way, erroneous rules are removed and the correct rules can be obtained from legal abstractions.

```
be_concluded(_,[agt:[A,B],
               obj:contract(_,[agt:[A,B]]),
               tim:C]) :-
  become_effective(_,
        [obj:
         acceptance(_,
             [agt:B,
              obj:offer(D,[agt:A,goa:B]),
              goa:A]),
        tim:C]),
  be_effective(_,
        [obj:offer(D,[agt:A,goa:B]),
         tim:C]).
```

Figure 13: Learning Example

7 Conclusions

This paper describes a knowledge acquisition from legal texts based on the analysis of human knowledge acquisition process. If background knowledge is provided, we can approach the automatic knowledge acquisition from legal texts. By replacing the legal abstraction database, a variety of legal knowledge can be obtained.

Since the acquired knowledge does not contain dynamically interpretation knowledge, in order to implement a legal expert system, other inference engine such as [7, 2] is required.

References

[1] D. Miller, et al.. Uniform proofs as a foundation for logic programming. In *Annals of Pure and Applied Logic*, **51**, pp. 125-157, 1991.

[2] M. Haraguchi. A form of analogy as an abductive inference. In *ALT91*, pp. 266–274, 1991.

[3] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Workshop on Machine Learning*, pp. 339–352, 1988.

[4] C. Rouveirol. Semantic model for induction of first order theories. In *IJCAI91*. Morgan Kaufmann, 1991.

[5] C. Rouveirol and J. F. Puget. Beyond inversion of resolution. In *Workshop on Machine Learning*, pp. 122–130, 1990.

[6] E. Shapiro. *Algorithmic program debugging*. MIT Press, 1983.

[7] H. Yoshino et al. Foundation of Systematization of Legal Analogy (In Japanese). Proc. of the 5th Annual Conference of JSAI, pp. 219–222, 1991.