Detecting and Resolving Counterintuitive Consequences in Law as Legal Debugging

by

Wachara Fungwacharakorn

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI
September 2021

A dissertation submitted to Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies, SOKENDAI,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Advisory Committee

1. Prof. Ken Satoh	National Institute of Informatics, SOKENDAI
2. Asst. Prof. Kanae Tsushima	National Institute of Informatics, SOKENDAI
3. Assoc.Prof. Ryutaro Ichise	National Institute of Informatics, SOKENDAI
4. Prof. Akiko Aizawa	National Institute of Informatics, SOKENDAI
5. Prof. Satoshi Tojo	Japan Advanced Institute of Science and Technology
6. Prof. Katsumi Nitta	Tokyo Institute of Technology

Acknowledgements

First and foremost I am extremely grateful to my supervisor, Prof. Ken Satoh for his continuous support and patience during my research and doctoral study. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. Without his guidance and support, this thesis would not have been possible. I greatly appreciate all his contributions of times, ideas and supervision to make my research experience productive and stimulating.

I would like to thank advisory committee members: Asst. Prof. Kane Tsushima, Assoc. Prof. Ryutaro Ichise, Prof. Akiko Aizawa, Prof. Satoshi Tojo and Prof. Katsumi Nitta, for not only their insightful and constructive comments and encouragement, but also for the fruitful suggestions that inspired me to broaden my research from various perspectives.

Additionally, I would like to thank Asst. Prof. Kanae Tsushima and Prof. Katsumi Nitta for their technical support on my study. I appreciate Dr. Tiago Oliveira and Dr. Jeremie Dauphin for their extensive comments on early drafts of our papers. I would like to thank internship students, visiting professors, lab staffs, university staffs, and anyone who encouraged me and provided thoughtful comments on my research. It is their kind help and support that have made my study and life in Japan a wonderful time.

Mostly importantly, I would like to deeply express my heart-felt gratitude to my family members, who have been a constant source of love, concern, encouragement and continuous support throughout my years of studying, researching, and writing this thesis. None of this would have not been possible without them.

Lastly, I would like to thank SOKENDAI (The Graduate University for Advanced Studies), National Institute of Informatics and Ministry of Education, Culture, Sports, Science and Technology for the educational and financial support during my doctoral studies. I appreciate all the incredible opportunities and memorable experiences they offered so far.

The Graduate University for Advanced Studies, SOKENDAI

Abstract

School of Multidisciplinary Sciences
Department of Informatics

Doctor of Philosophy

Detecting and Resolving Counterintuitive Consequences in Law as Legal Debugging

by Wachara Fungwacharakorn

Since legal judgements are complex but yet essential in our society, it is very challenging for artificial intelligence (AI) researchers to mechanize statutes and legal judgements. AI and Law Researchers have long been interested in representing legal knowledge with computational legal representations in order to enable mechanization. Such legal representations are, for example, normal logic programs or Prolog programs, or the legal knowledge representation called PROLEG adopted from normal logic programs in order to suit the ultimate fact theory in Japanese Civil Code litigation. In countries with a civil law system, such as Japan or Thailand, where statutes are the primary source of reference in court, most legal representations rely on the literal interpretation of statutes. However, in some real-life cases, the literal interpretation of statute does not meet social expectations and produces counterintuitive consequences, leading to absurdity, harming public interests, or endorsing strange behaviors in society. Judges, particularly in high courts, may handle these consequences by taking the exceptional situations in the case that are not addressed by the statute into account and revising the interpretation of the statute by adding new conditions or exceptions to address the exceptional situation.

Recently, there have been many approaches for revising logic programs that represent the interpretation of the statutes in order to resolve legal conflicts. Unfortunately, revisions in order to meet social expectations cannot be done automatically, as opposed to revisions in order to resolve legal conflicts, which can be done automatically in secondary legislation given that we have codified primary and secondary legislation. Furthermore, as early works in AI and Law have suggested, formalizing legal changes for meeting social expectations requires debugging-like mechanism in legal reasoning systems. However, there are no theoretical foundations of debugging in law to our knowledge. Therefore, in this dissertation, we propose *Legal Debugging*, extending from Algorithmic Debugging in software engineering, for judges in civil law systems to detect and resolve counterintuitive

consequences in law. In Legal Debugging, we formalize counterintuitive consequences as the symmetric difference between the literal interpretation of the statute delivered by the computational legal reasoning system, and the interpretation intended by the user i.e. a judge or a legal scholar.

Legal Debugging consists of two main algorithms, namely Culprit Detection Algorithm and Culprit Resolution Algorithm. Culprit Detection Algorithm assists the user to discover more counterintuitive consequences by checking with the user whether related consequences are counterintuitive until the user finds no more counterintuitive consequences related. The last found counterintuitive consequence, called a *culprit*, is determined as a root cause of such counterintuitive consequences. Culprit Resolution Algorithm assists the user to revise the rule-base representing statutes by let the user choose necessary conditions that indicate the exceptional situations in the case. Since statutes are represented by rule-bases but changes in law are initiated by cases, we adopt a *prototypical case with judgement* specified by a set of rules. Then, the result of the culprit resolution algorithm is a revised rule-base such that new prototypical cases with judgement representing exceptional situations of the present case are included. Furthermore, we present in this dissertation one application of Inverse Resolution, which is the well-known inductive programming technique, for generalizing culprit resolution in order to cooperate with a user and background theory for more practical revision of the rule-base.

In this dissertation, we also present our formalization of *semantics-based minimal revision for legal reasoning*, which focuses on minimal revisions on legal interpretations varying among cases, and *dominant-based minimal revision*, a sub-type of semantics-based minimal revision that does not require to calculate a set of all conclusions for each case and unaffected by the fact-domain extension. We use such minimal revisions to warn the user about the possibility of unintentional changes of semantics during the generalization of culprit resolution. We determine additional prototypical cases with judgment beyond ones occur in the minimal revision as non-trivial effects. Hence, legal reasoning systems can check with the user to confirm the intention of such non-trivial effects.

Legal debugging is applicable to any statutory laws in general because most of statutory laws are designated to produce one unique interpretation for each case in litigation, hence they can be represented by a non-recursive and stratified logic program with corresponding prototypical cases with judgement, which is the applicable range of Legal Debugging. Given that the statute contains a large number of rule conditions, we expect that Legal Debugging would help in discovering which condition causes counterintuitive consequences and how to revise logic programs representing the interpretation of the

statutes to resolve the counterintuitive consequences so that it can formalize revisions in order to meet social expectations.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this chapter, we present

- Background of our research
- Motivation of our research
- Contribution of our research
- Outline of the rest of the dissertation

1.1 Background

As artificial intelligence (AI) are applied in different parts of society, law is no exception. Actually, AI and Law is the classic field where the law motivates AI research, and the other way around, the law takes the benefits from AI [1]. The synergy trends of Symbolic AI and AI-and-Law can be shown in Table 1.1, in which each decade roughly indicates the beginning of the trends (for details, see Chapter 2).

TABLE 1.1: Summary of trends of Symbolic AI and AI-and-Law

	Symbolic AI	AI and Law
The 1980s	Non-monotonic Reasoning	Rule-based Legal Reasoning Systems
	Case-based Reasoning	Case-based Legal Reasoning Systems
The 1990s	Argumentation	Hybrid Legal Reasoning Systems
		Judicial Legal Change
The 2000s	Semantics Web	Theory Construction
		Promoting Value
The 2010s	Explainable AI	Precedential Constraints
		Case Model

The mid 1980s is when early *rule-based legal reasoning systems* e.g. [2, 3] emerged as the applications of logic programming. The systems can consider whether a legal consequence (e.g. a particular individual is or is not a British citizen [2]) is valid with respect to statutes and the factual information of the circumstance. The systems also produce reasoning based on the deductive proof obtained by the theorem prover. At that time, symbolic AI researchers were interested in *non-monotonic reasoning*, which deals with incomplete information by allowing the conclusions to be defeated later [4–7]. Since the law often deals with the cases in which the evidence is insufficient, early rule-based legal reasoning systems were also developed to support non-monotonic reasoning. Besides that, in the late 1980s, HYPO [8], a classical *case-based legal reasoning system*, was presented. HYPO produces plausible arguments for the plaintiff's side and the defendant's side from merely precedent cases, and predicts which side would win the case. It becomes one major contribution to establish a new field in symbolic AI called *case-based reasoning* which studies inductive reasoning without requiring any rules, unlike deductive reasoning where rules or axioms are required.

In the early 1990s, AI and Law researchers develop a series of *hybrid legal reasoning systems* between rule-based and case-based e.g. CABARET [9], GREBE [10], and HELIC-II [11]. Hybrid reasoning systems support the reasoning in law when one another approach

is insufficient to decide the case. These systems begin introducing argumentation as a tool to interchange information between rules and cases. In the mid 1990s, theoretical foundations of *argumentation* have been established in symbolic AI, such as Dung's abstract argumentation framework [12] and Walton's argumentation scheme [13]. Such theoretical foundations of augmentations become rapidly interested in AI and Law community e.g. [14, 15]. Meanwhile, AI and Law researchers become recognized for judicial changes in legal concepts, especially in case-based legal reasoning systems, since the strength of the precedent cases may change when the time passes [16, 17]. The judicial legal change occurs once a special judgement (usually called a principle judgement, a guideline judgement, or a case law) adds or modifies the legal concepts that the later cases follow.

The 2000s entered the era of *semantic web* as the rapid growth of representing and visualizing information can be seen especially in web technologies. Therefore, the case-based legal reasoning systems need not only to produce plausible arguments for each side but to provide a set of rules or explanations, called a theory, that can explain the reasons behind the precedent cases. Therefore, a new field in AI and Law called *Theory Construction* becomes established e.g. [18–20]. Meanwhile, the role of legal context in judicial legal change has been investigated in the early 2000s [21] and three roles of legal context have been classified, including teleological relations, temporal relations, and procedural postures. This leads to the field of *Promoting Values* to formalize judicial legal change [22].

The 2010s is when neural networks and deep learning become rapidly mechanized. However, lots of artificial intelligence applications, including AI and law, require not only accurate predictions but also the explainable reasons behind such predictions. Therefore, the current trend of artificial intelligence called *Explainable AI* has emerged. According to Miller [23], explainable AI contains four major features: (1) contrastive (2) selective (3) non-probabilistic (4) social. As we can see, legal reasoning systems have been already included these features for a long time [24]. Meanwhile, in the 2010s, AI and Law researchers produces a series of works related to *precedential constraints* [25, 26] (for an intensive survey, see [27]) and *case models* [28–30]. Such formalizations cope with the coherence of the law to maintain consistencies when judicial legal change occurs.

1.2 Motivation and Research Question

Generally, laws can be divided into two systems. The first system is common law, where precedent cases are the main reference in legal judgements. The second system is civil law, where statutes are the main reference in legal judgements. In the common

law system, it is obvious that the judge can change the law via the judgements that later become the precedent. Even in the civil law system, change is also one enduring characteristic [31] since the social expectation is evolved and exceptional cases occur unexpectedly in real-life. Although judges in the civil law system make their best efforts to use the literal interpretation of the statute due to separation of powers, they sometimes reinterpret the statute when the literal interpretation of the statute has consequences that do not meet the social expectation such that if the consequences were not properly handled, they will cause absurdity, strange behaviors, or risks to public interest. In this dissertation, we refer to such consequences as *counterintuitive consequences*.

TABLE 1.2: Comparison between Legal Debugging and other ILPs

ILP Systems	Main Technique	Negations	Applications in Law
MIS [32]	Refinement Operator	n/a	n/a
CIGOL [33]	Inverse Resolution	n/a	Legal Ontology Refinement [34]
ASPAL [35]	Answer Set Programming	Supported	Automatic Revision in Secondary Legislation [36]
Legal Debugging	Prototypical Cases with Judgement	Supported	Detecting and Resolving Counterintuitive Consequences

AI and Law researchers often deal with similar problems using *Inductive Logic Programming* (ILP) for revising statutes (represented by logic programs) as shown in Table 1.2. For example, Kurematsu and colleagues have applied a concept of Inverse Resolution from an ILP system called CIGOL [33] for refining legal ontology [34]. Li and colleagues have applied an ILP system called ASPAL [35] for automatic revision in secondary legislation when it has some conflicts with primary legislation. Unfortunately, legal changes for meeting social expectation need to handle exceptions in law (represented by negations as failure in logic programs) unlike the legal ontology refinement, and such legal changes cannot be done automatically since we cannot codify social expectation, unlike the revision in secondary legislation. In addition, formalizing legal changes for meeting social expectation rather require debugging-like mechanism in legal reasoning systems as early works in AI and Law [2, 31, 37] have suggested. However, there are no applications of ILP systems for debugging in law to our knowledge although algorithmic software debugging [38] is one earliest application of an ILP system called MIS. Therefore, in this dissertation, we propose *Legal Debugging* as the first formalization of legal changes for meeting social expectation as the detection and resolution of counterintuitive consequences in law. As the term *bug* in the computer field refers to a part of the program producing unexpected results, we define broadly by analogy that a bug in law (which we coined it as a new term *culprit*, to distinguish from a bug in a program) is a

part of the statute producing counterintuitive consequences. We aim Legal Debugging for detecting and resolving such counterintuitive consequences in law by answering three research questions in the following, and hence it brings us one step closer to formalizing legal change for meeting social expectation.

- **RQ1: How to detect a culprit ?** Since we cannot directly codify social expectation, it raises the question of how to detect a culprit.
- **RQ2: How to resolve a culprit ?** Since judges in the civil law system consider legislators' intention when making change, it raises the question of how to resolve a culprit to reflect such intention.
- **RQ3: How to evaluate the resolution ?** Since the culprit resolution should not be too generalized otherwise counterintuitive consequences occur unintentionally, it raises the question of how to evaluate the resolution to prevent such unintentional effects.

1.3 Contribution

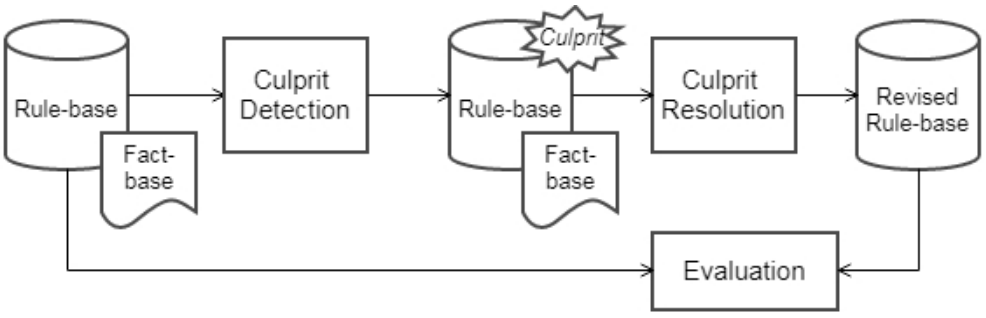

FIGURE 1.1: The Work Flow of Legal Debugging

To address and solve the research questions above, we propose two algorithms involving with a user, namely Culprit Detection Algorithm and Culprit Resolution Algorithm, and one method to evaluate the resolution, as illustrated Figure 1.1. We also present extensions of those algorithms for a first-order rule-base and a PROLEG rule-base. We briefly describe our contributions in the following, where segments of our work that have been published or are going to be published are all cited.

1. **Culprit Detection Algorithm** [39, 40]. Given a rule-base representing the statute and a fact-base representing a case. Culprit Detection Algorithm detects a

culprit from a *counterintuitive consequence*, initially provided by a user. The algorithm iteratively asks the user whether related consequences are counterintuitive until it can no longer find related counterintuitive consequences. Then, the last found counterintuitive consequence is determined as a culprit. This algorithm aims to solve RQ1 under the assumption that a rule-base is non-recursive and stratified. We also present the extensions of the algorithm for a first-order rule-base [40] and for a PROLEG rule-base [39].

2. **Culprit Resolution Algorithm** [40]. Given a rule-base representing the statute, a fact-base representing a case, and a culprit detected from the first algorithm. To solve RQ2 under the assumption that a rule-base is built from *critical sets*, we present a new structure called *prototypical cases with judgement*. The algorithm primarily revises the rule-base. Then, the algorithm asks a user to select facts from the fact-base that are relevant to the head of the new rule. After that, the algorithm reproduces prototypical cases with judgement from the primary revised rule-base and the primary new rules. Finally, the algorithm secondarily revises rule-base to preserve the judgement of prototypical cases as well as to resolve the culprit. We also present the extensions of the algorithm for a first-order rule-base [40], for a PROLEG rule-base [41, 42], and generalization of resolutions by background theory [43].

3. **A method to evaluate the resolution** [44]. To solve RQ3, we present formalization of semantics-based minimal revision for legal reasoning. Based on the formalization, we propose one method to evaluate the revised rule-base by representing effects as the additional prototypical cases with judgement produced from the resolution [45]. We determine the effect not involving the present exceptional case as non-trivial effects and we can check the intention of non-trivial effects with the user.

1.4 Outline

The remainder of the dissertation is organized as follows

Chapter 2: Foundation and Related Work

Chapter 2 provides the extended content of research background and related work. This chapter presents foundations of logic programming, inductive logic programming (ILP), rule-based legal reasoning systems, case-based legal reasoning systems, judicial legal change, and related work to each research question.

Chapter 3: Culprit Detection Algorithm

Chapter 3 corresponds to the research question: RQ1: How to detect a culprit. This chapter presents the formalization of counterintuitive consequences in law and culprits then presents Culprit Detection Algorithm with the proofs of correctness and completeness under the assumption that a rule-base is non-recursive and stratified. Moreover, this chapter also provides the discussion of culprit detection in first-order representation and PROLEG.

Chapter 4: Culprit Resolution Algorithm

Chapter 4 corresponds to the research question: RQ2: How to resolve a culprit. This chapter presents the formalization of a counterintuitive consequence resolution task for references in remaining chapters then presents our culprit resolution algorithm with the proofs of correctness and completeness under the assumption that a rule-base is built from *critical sets*. Moreover, this chapter also provides the discussion of culprit resolution in first-order representation and PROLEG.

Chapter 5: Evaluating Generalization

Chapter 5 corresponds to the research question: RQ3: How to evaluate the resolution. This chapter presents the formalization of semantics-based minimal revision for legal reasoning. Based on the formalization, we present one method to evaluate the revised rule-base by detecting possibly unintentional effect. This chapter also presents generalization of culprit resolution using background theory and evaluation of the generalization using the presented method.

Chapter 6: Conclusion and Future Work

Chapter 6 provides the conclusion of this dissertation and the potential future directions of the research on Legal Debugging.

Chapter 2

Foundations and Related Work

In this chapter, we present

- Foundations of logic programming referred to in this dissertation
- Foundations of inductive logic programming (ILP) and some ILP techniques referred to in this dissertation
- Foundations of rule-based legal reasoning systems and works about rule-based legal reasoning systems referred to in this dissertation
- Foundations of legal case-based reasoning systems and works about case-based legal reasoning systems referred to in this dissertation
- Foundations of judicial legal change, its formalization, and the example of judicial legal change
- Related work of this research

2.1 Logic Programming

In this dissertation, we mainly consider applications of logic programming for representing law. Therefore, in this section, we present some foundations of logic programming referred to in this dissertation as follows.

2.1.1 Normal Logic Programming and Resolution

The concept of using mathematical logic to represent and execute computer programs has been developed as a feature of *lambda calculus* [46] in the 1930s. This concept has been developed into one of the earliest logic programming languages called *LISP* in the 1950s. In the 1960s, applications of logic programming for automated theorem proving have received great attention. New concepts related to automated theorem proving in logic programming have been coined in this period such as resolution [47, 48] and subsumption [49]. Eventually, *Prolog*, one of popular logic programming languages until the present, was developed in the early 1970s, based on the concept of resolution. In this dissertation, we call a program written in Prolog as a *normal logic program* (to distinguish with a legal representation called PROLEG, which will be described later). The normal logic program is formally defined as follows.

Definition 2.1 (Normal Logic Program)**.** A normal logic program is a finite set of rules of the form $h \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n$. where $h, b_1, \ldots, b_n$ are first-order logic atoms and *not* represents a negation as failure, namely *not* b shall be determined true when the attempt to prove b finitely fails.

Let R be a rule of the form, we have h as a head of a rule denoted by $head(R)$, $\{b_1, \ldots, b_m\}$ as a positive body of a rule denoted by $pos(R)$, $\{b_{m+1}, \ldots, b_n\}$ as a negative body of a rule denoted by $neg(R)$, and $\{b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n\}$ as a body of a rule denoted by $body(R)$. Sometimes, we express the rule in the form $h \leftarrow B$. where B is the body of rule. We express $h.$ (called a fact) if the body of the rule is empty. A rule is called a *Horn clause* if the negative body of the rule is empty. For ease of explanation, we assume that all variables in the rule must occur in the positive body of the rule.

Throughout this dissertation, a program means a normal logic program unless stated otherwise. For first-order atoms, we begin variables with uppercase letters and constants with lowercase letters. We present here definitions of substitution, unification, and subsumption as follows.

Definition 2.2 (Substitution)**.** Let A be an atom and $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ be a mapping from variables to terms called a *substitution* with $\{v_1, \ldots, v_n\}$ called the domain

of θ. $A\theta$ is formed by replacing every variable v_i in A by a term t_i $(1 \leq i \leq n)$ [48]. We also extend the substitution to a set of atoms B by $B\theta = \{A\theta | A \in B\}$ and a rule R by $R\theta = head(R)\theta \leftarrow body(R)\theta$.

Definition 2.3 (Unification)**.** Let A_1 and A_2 be atoms. A unifier for A_1 and A_2 is a substitution θ such that $A_1\theta = A_2\theta$. A unifier for A_1 and A_2 is a most general unifier (mgu) if it is a most general substitution which unifies A_1 and A_2 [48].

Definition 2.4 (Subsumption)**.** Let C and D be a set of atoms. We say C subsumes D $(C \preceq D)$ if there is a substitution θ such that $C\theta \subseteq D$ [49].

Although first-order programs representing statutes sometimes have function symbols, but such function symbols are seldom nested i.e. if there is a function symbol named $g(.)$ there would not be $g(g(.)), g(g(g(.))), \ldots$. Therefore, for ease of exposition, we assume the programs in this dissertation have finite constants, finite predicates, and no function symbols so the *Herbrand universe* (the set of all ground terms that can be built) is merely the set of constants, which is also finite. We use the word *ground* in the sense of variable-free. The *Herbrand base* $\mathcal{HB}$ of a program T is the set of all ground atoms whose argument terms are the Herbrand universe. Since the program is presumed to have finite predicates, the Herbrand base is also finite. The ground program of T, denoted by $gr(T)$ is the set of ground rules obtained from T by replacing all variables in each rule with every element of its Herbrand universe.

Now, let introduce resolution. Resolution is a rule of inference used for theorem proving in Prolog. Resolution views a rule as a *clause*, which is formally defined as follows.

Definition 2.5 (Clause)**.** A clause is an expression of the forms $l_1 \vee \cdots \vee l_n$ where $l_1, \ldots, l_n$ are literals (either a positive literal, that is, an atom, or a negative literal, that is, a negation of an atom). We can view a rule $h \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n.$ as a clause $h \vee \neg b_1 \vee \cdots \vee \neg b_m \vee b_{m+1} \vee \cdots \vee b_n$. Hence, we can view a program as a finite set of clauses in the same manner as well.

Definition 2.6 (Resolution)**.** Given two clauses C_1 and C_2 with no common variables. Let L_1 and L_2 be literals within C_1 and C_2 respectively such that there is the mgu θ of $\neg L_1$ and L_2. We denote the *resolvent* of C_1 and C_2 by $C = C_1 \cdot C_2$ where $C = (C_1 \setminus \{L_1\})\theta \cup (C_2 \setminus \{L_2\})\theta$ [48].

2.1.2 Stratification, Stable Model Semantics, and Unfolding/Folding

In the 1980s, researchers in logic programming have become increasingly interested in formalizations of non-monotonic reasoning. Many formalizations of non-monotonic reasoning developed in the period, such as Default Logic [4], Circumscription [5], Non-monotonic Logic [6], and Minimal Belief Revision [7]. Since non-monotonic reasoning allows multiple sets of conclusions, formalizations and uniqueness of semantics have been widely studied. In this section, we present the study of stratified logic program [50, 51] the uniqueness of stable model semantics for stratified logic program [52], and the unfolding/folding transformations of logic programs that preserve stable model semantics [53, 54].

Since a program representing statutes is generally a non-recursive and stratified program, we also hold this presumption in this dissertation. The definition of non-recursive and stratified program, adopted from [50, 51], is defined as follows.

Definition 2.7 (A non-recursive and stratified program)**.** A program T is non-recursive and stratified if there is a partition $T = T_0 \cup T_1 \cup \ldots \cup T_n$ (T_i and T_j disjoint for all $i \neq j$) such that, if a predicate p occurs in a body of rule in T_i then a rule with p in the head is only contained within $T_0 \cup T_1 \cup \ldots T_j$ where $j < i$.

An answer set is a set of ground atoms (including ones with rule predicates and fact predicates) which can be concluded from the logic program T. In this dissertation, we apply the definition of stable model semantics for finding an answer set [52], defined as follows.

Definition 2.8 (Stable Model Semantics)**.** Let T be a normal logic program and M be a set of ground atoms. Let $trim(T)$ be a trimming function defined as follows: $\{head(R) \leftarrow pos(R) | R \in T\}$ and $T^M = trim(\{R | R \in gr(T) \text{ and } neg(R) \cap M = \emptyset\})$. M is a stable model of T if and only if M is the minimum set (in the sense of set inclusion) such that M satisfies every rule $R' \in T^M$, that is $pos(R') \subseteq M$ implies $head(R') \in M$.

It is proved that a stratified program has a unique stable model semantics [52]. We denote a ground atom p is in the stable model of T by $T \vdash p$. We have that the stable model is related to the concept of supporting rule shown as follows.

Definition 2.9 (Supporting Rule)**.** Let R be a rule, p be a ground atom, and M be a set of ground atoms. We say R supports p with respect to M if there is a substitution θ such that $head(R)\theta = p$, $pos(R)\theta \subseteq M$, and $neg(R)\theta \cap M = \emptyset$.

Theorem 2.10. *Let p be a ground atom, T be a program, M be the answer set of T. $p \in M$ if and only if there is a rule $R \in T$ that supports p with respect to M.*

Proof. Case 1: Suppose there is a rule $R \in T$ supports p with respect to M but $p \notin M$ then there is a substitution such that $head(R)\theta = p, pos(R)\theta \subseteq M$, and $neg(R)\theta \cap M = \emptyset$. Therefore, $R\theta \in T^M$ but M does not satisfy $R\theta$ because $pos(R)\theta \subseteq M$ but $head(R)\theta \notin M$. So, it leads to contradictions.

Case 2: Suppose there is no rule $R \in T$ that supports p with respect to M but $p \in M$, then, $M \setminus \{p\}$ must satisfy every rule in T^M. It leads to contradiction since M should be the minimum set but it is not. □

In this dissertation, we apply *unfolding/folding* [53, 54], which are transformations of logic programs that preserve the stable model semantics. Below are the definition of unfolding and folding.

Definition 2.11 (Unfolding)**.** Given a program T_1. $T_1, \ldots, T_m$ is an unfolding sequence such that T_{k+1} is a result after applying unfolding to $T_k = \{E_1, \ldots, E_r, C, E_{r+1}, \ldots, E_s\}$ $(1 \leq k < m)$ where C be a rule $H \leftarrow F, A, G.$ where A is a positive literal and F and G are (possibly empty) sequences of literals. Suppose that:

1. $\{D_1, \ldots, D_n\}$ with $n > 0$ is the set of all rules in T_j $(0 \leq j \leq k)$ such that A is unifiable with $head(D_1), \ldots, head(D_n)$ with m.g.u. $\theta_1, \ldots, \theta_n$ respectively, and
2. C_i is the rule $(head(C) \leftarrow (F, bd(D), G.)\theta_i$ where $bd(D)$ is a sequence of literals occurring in the body of D

If we unfold C with respect to A using $D_1, \ldots, D_n$ in T_j, we derive the rules $C_1, \ldots, C_n$ and we get the new program $T_{k+1} = \{E_1, \ldots, E_r, C_1, \ldots, C_n, E_{r+1}, \ldots, E_s\}$ [54]. We say an unfolding sequence $T_1, \ldots, T_m$ is a complete unfolding sequence if all positive body atoms in the rule in T_m cannot be further unfolded.

Definition 2.12 (Folding)**.** Given a program T_1. $T_1, \ldots, T_m$ is an folding sequence such that T_{k+1} is a result after applying folding to $T_k = \{E_1, \ldots, E_r, C_1, \ldots, C_n, E_{r+1}, \ldots, E_s\}$ $(1 \leq k < m)$. Let $\{D_1, \ldots, D_n\} \subseteq T_j (0 \leq j \leq k)$. Suppose that there exists a positive literal A such that, for $i = 1, \ldots, n$:

1. $head(D_i)$ is unifiable with A via m.g.u. θ_i,
2. C_i is the rule $(H \leftarrow F \cup bd(D_i), G)\theta_i$ where F and G are (possibly empty) sequences of literals, and
3. for any rule $D \in P_j \setminus \{D_1, \ldots, D_n\}$, $head(D_i)$ is not unifiable with A.

If we fold $C_1, \ldots, C_n$ using $D_1, \ldots, D_n$ in T_j, we get the new program $T_{k+1} = \{E_1, \ldots, E_r,$ $C, E_{r+1}, \ldots, E_s\}$ where $C = H \leftarrow F, A, G.$ [54].

2.2 Inductive Logic Programming

Inductive logic programming (ILP) is a sub-field of artificial intelligence that studies building a logic program (usually called a *theory* or a *hypothesis*) based on training examples. It is one foundation of modern machine learning techniques such as decision trees. Since the beginning of the development of ILP in the 1970s [55], many ILP techniques have been emerged. In this section, we present some some ILP techniques related to our research as follows.

2.2.1 Algorithmic Debugging and Model Inference System (MIS)

Algorithmic Debugging [38] is the theoretical foundation of software debugging, developed by Ehud Shapiro in the 1980s. In his work, he aims to answer two questions.

1. How do we identify a bug in a program ? (Bug detection)
2. How do we fix a bug, once one is identified ? (Bug correction)

To answer the first question, Shapiro presented Program Diagnosis Algorithm. The algorithm aims for identifying three kinds of bugs. The first kind is an incorrect procedure, which makes a program give a wrong answer. The second kind is an incomplete procedure, which makes a program finitely fail to produce a correct answer. The third kind is a diverging procedure, which makes a program consume resources beyond boundaries (e.g. the depth of stack). Generally speaking, Program Diagnosis Algorithm interacts with the user to systematically trace the differences between the computational results and the user expectation until a bug is detected.

To answer the second question, Shapiro extended Bug-correction Algorithm from his *Model Inference System* (MIS) [32], which is one of the earliest ILP since we can apply MIS for building a whole new logic program from the empty program with merely training examples. MIS is assumed to construct a logic program with only Horn Clauses by developing *refinement operators* and *de-refinement operators*. Below shows the definition of the *refinement operator* in our setting and one example of refinement operator in MIS [38]. The *de-refinement operator*, whose definition is not shown here, is briefly an inverse of the refinement operator.

Definition 2.13. Let L be a set of definite clauses with $p_1, p_2, ..., p_n, p, q \in L$, ρ be a mapping from L to finite subsets of L, and $<_\rho$ be the binary relation over L for which $p <_\rho q$ if and only if there is a finite sequence of clauses $p_1, p_2, ..., p_n$ such that $p_1 = p$, $p_n = q$, $p_{i+1} \in \rho(p_i)$ for $0 \leq i \leq n$. ρ is said to be a *refinement operator* over L if and

only if the relation $<_\rho$ is a well-founded ordering over L and for every interpretation M and atom A, if q supports A with respect to M and $p <_\rho q$ then p supports A with respect to M.

Definition 2.14. Let p be a clause. $q \in \rho(p)$ if and only if one of the following holds

- $p = \square$ (empty clause) and $q = a(X_1, X_2, \ldots, X_n)$, for some n-predicate symbol a, and $X_1, X_2, \ldots, X_n$ are n distinct variables
- q is obtained by unifying two variables in p
- q is obtained by instantiating a variable X in p to a term $t(X_1, X_2, \ldots, X_n)$ where t is an n-function symbol and $X_1, X_2, \ldots, X_n$ are new variables
- q is obtained by adding to p' s body a goal B and every variable in B occurs in $head(p)$

Generally speaking, MIS specializes the input program if it is too generalized (succeeding on a goal known to be false) and generalizes the input program if it is too specialized (failing on a goal known to be true). The framework of MIS can be roughly shown as Algorithm 1. Then, Shapiro extended MIS to Bug-correction Algorithm by using the detected bug as the hint. With the assumption that the programmer tries to code the program with the best effort, Bug-correction Algorithm specializes (via *refinement operators* in MIS) and generalizes (via *derefinement opertors* in MIS) the program to refine the buggy part into the correct one.

Algorithm 1 Model Inference System (MIS)

Input A set of example E, an oracle for an intended interpretation IM and an ordered set of clauses

```
Let T to be an empty program and the set of marked rules be empty
while There is an unknown example do
    Learn one unknown example
    while T is not totally correct with respect to the known example do
        if T succeeds on a goal known to be false then
            Find R that supports an incorrect goal with respect to IM in T
            Remove R from T
            Mark R
        else if T fails on a goal known to be true then
            Find an incomplete goal A
            Search for an unmarked rule R that supports A with respect to IM
            via refinement and de-refinement operators
            Add R to T
return T
```

2.2.2 Closed World Specification (CWS)

Closed World Specification (CWS) [56] is one ILP technqiue for non-monotonic reasoning under Closed World Assumption (the presumption that a statement that is not currently known to be true, is false). Instead of, like MIS, removing the rule found supporting an incorrect goal then searching for a new rule, CWS supports non-monotonic reasoning by introducing the representation of the exceptional situation. In brief, CWS represents the exceptional situation by using the instantiation of one atom in the negative body of the found rule if it exists. Otherwise, CWS introduces a new inventing atom into the negative body of the found rule then uses the instantiation of the atom to represent the exceptional situation. The algorithm for CWS can be illustrated as in Algorithm 2.

Algorithm 2 Closed World Specification (CWS)

Input A non-recursive and stratified normal logic program T and an incorrect ground atom p which $T \vdash p$.

Let M be the stable model of T
for all rules $R \in T$ that supports p with respect to M by a substitution θ **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 θ
    Let q be a predicate symbol not found in T
    Let b be $q(v_1, \ldots, v_n)$
    Let $T' = T \setminus \{R\} \cup \{head(R) : -(body(R) \cup \{not\ b\})\} \cup \{b\theta\}$
return T'

Example 2.1. *(Adopted from [56]) Let* $T = \{flies(X) \leftarrow bird(X).bird(eagle).bird(emu).\}$ *and* $A = flies(imu)$ *is incorrect. Then, we have* $M = \{bird(eagle), bird(emu), flies(eagle), flies(emu)\}$ *as an answer set of* T *and we have* $R = flies(X) \leftarrow bird(X).$ *and* $X = \{X/emu\}$ *such that* M *supports* $flies(emu)$ *with respect to* $R\theta$*. Let* ***flightless*** *be a new predicate. Since* $\{X\}$ *is the domain of* θ*,* $T' = \{flies(X) : -bird(X),\ not\ \textbf{\textit{flightless}}(X).\ bird(eagle).bird(emu).\textbf{\textit{flightless}}(emu).\}$*. Hence, it implies that there are some flightless birds and emu is one of them.*

2.2.3 Inverse Resolution (IR)

Inverse Resolution (IR) [33] is an inductive logic programming technique that generalizes the input program by introducing a general rule based on the existing rules. It involves

inverting the resolution operator, aiming to build parent clauses given the resolvent(s) and possibly one parent clause. IR develops two main operators. The first operator, called *V-operator*, builds a clause C_2 from clauses C_1 and C such that $C_1 \cdot C_2 = C$. The second operator, called *W-operator*, builds clauses C_1, C_2, C_3 from clauses C_4 and C_5 such that $C_1 \cdot C_2 = C_4$ and $C_2 \cdot C_3 = C_5$. Since V-operator is non-deterministic, the original work of Inverse Resolution reduces the indeterminacy by assuming C_1 is a unit clause (a clause with a single literal). In this dissertation, we obtain the clause C_2 by adopting the definition of *folding* (see Definition 2.12) since to fold a rule C using C_1 into the folded rule C_2 is one way of obtain the parent clause in V-operator since C then becomes the resolvent of C_1 and C_2 ($C = C_1 \cdot C_2$). The example can be illustrated as follows.

Example 2.2. *(Adopted from [57])*
Given $T =$

$$flies(X) \leftarrow sparrow(X),\ not\ flightless(X).$$
$$bird(Y) \leftarrow sparrow(Y).$$
$$sparrow(tweety).$$

Let fold $flies(X) \leftarrow sparrow(X),\ not\ flightless(X).$ *using* $bird(Y) \leftarrow sparrow(Y).$ *We get that there exists an atom* $bird(X)$ *that satisfies the conditions in Definition 2.12. Hence, the folded program is* $T' = \{flies(X) \leftarrow bird(X),\ not\ flightless(X).\ bird(Y) \leftarrow sparrow(Y).\ sparrow(tweety).\}$. *It implies the generalization of the statement "sparrows can fly" into the statement "some birds can fly".*

2.3 Rule-based Legal Reasoning Systems

Legal reasoning system are ones of the earliest applications of logic programming. In 1986, the work of representing British Nationality Act as a Prolog program has been presented [2] following by the work of representing Income Tax Act of Canada as a Prolog program in 1987 [3]. By the mid 1900s, AI and Law community has shifted their focuses to the applications of argumentation in law (e.g. [14, 15]). Hence, in the 2000s, many integrations of argumentation models and rule-based reasoning models have been developed such as Argumentation Scheme for Legal Rules [58] and Defeasible Logic [59]. Besides that, foundations in normal logic program has been investigated in the argumentation perspective, such as formalizing a switch of burden of proof by logic programming [60] and translating the Japanese Presupposed Ultimate Fact Theory into logic programming [61]. In this section, we present some studies on rule-based legal reasoning related to our research as follows.

2.3.1 *The British Nationality Act as a Logic Program*

The British Nationality Act as a Logic Program [2] is the first work formalizing statutes into a normal logic program, and the literal interpretation of the statute in the case can be obtained by the semantics of the program i.e. the answer set. Their work is one pointing out non-monotonic reasoning behavior in legal reasoning. They illustrated this behavior by the example of abandoned infant in the British Nationality Act 1981 Section 1, stating

> **(1)** A person born in the United Kingdom after commencement shall be a British citizen if at the time of birth his father or mother is
>
> (a) a British citizen; or
>
> (b) settled in the United Kingdom.
>
> **(2)** A newborn infant who, after commencement, is found abandoned in the United Kingdom shall, *unless the contrary is shown*, be deemed for the purposes of subsection (1)
>
> (a) to have been born in the United Kingdom after commencement; or
>
> (b) to have been born to a parent who at the time of the birth was a British citizen or settled in the United Kingdom.

The phrase *unless the contrary is shown* in subsection (2) is one expression that legislators use for dealing with incomplete information. In this case, the incomplete information is possibly the time of birth or the nationality of the abandoned infant's parent. In their work, they deal with non-monotonic reasoning by using negation as failure (`not[]`) combined with ordinary negation (`not`). Then, subsection (2) can be represented as follows.

```
x acquires British citizenship on date y by section 1.2 if
 x was found as a newborn infant abandoned in the U.K.
 and x was found on date y and
 and y is after or on commencement
 and not[x was not born in the U.K. after or on commencement]
 and not[x was not born to a parent who qualifies
        under section 1.1. at time of birth]
```

2.3.2 *Translating the Japanese Presupposed Ultimate Fact Theory into logic programming*

Translating the Japanese Presupposed Ultimate Fact Theory into logic programming [61] is the work aiming to formalize the Japanese Presupposed Ultimate Fact Theory [62], which is the legal theory for interpreting the Japanese Civil Code. Their theory has been developed by judges in the Japanese Legal Training Institute for handling incomplete information that sometimes occur due to a lack of enough evidence. Hence, they use a normal logic program (with negation as failure) to represent Japanese Civil Code. They show that the normal logic program representing statutes can be built from *critical sets*, which are defined as follows.

Definition 2.15 (Critical Set)**.** A critical set is a set of ultimate facts (an element in the Japanese Presupposed Ultimate Fact Theory) defined as follows [61].

- $\emptyset$ is a critical set that does not lead to the conclusion.
- Let S be a critical set that does not lead to a conclusion and T be a superset of S (that is $S \subset T$) that leads to a conclusion. T is a critical set if the deletion of any element of $T \setminus S$ (the difference set of S and T) from T does not lead to a conclusion. We call $T \setminus S$ an attack set with respect to S.
- Let S be a critical set that leads to a conclusion and T be a superset of S (that is $S \subset T$) that does not lead to a conclusion. T is a critical set if the deletion of any element of $T \setminus S$ (the difference set of S and T) from T leads to a conclusion. We call $T \setminus S$ a defense set with respect to S.

They have illustrated the formalization by the example of the Japanese Civil Code Article 612 and we will adopt this example throughout this dissertation. The Japanese Civil Code Article 612 states 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.

A normal logic program representing the Japanese Civil Code Article 612 can be illustrated as follows[1]. Hereafter, this rule-base is denoted by JRB_0.

[1]This representation is adopted for ease of exposition. We sometimes use :- instead of $\leftarrow$ as in PROLOG convention.

```
cancellation_due_to_sublease :- effective_lease_contract,
    effective_sublease_contract, using_leased_thing,
    manifestation_cancellation, not approval_of_sublease.
effective_lease_contract :-
    agreement_of_lease_contract, handover_to_lessee.
effective_sublease_contract :-
    agreement_of_sublease_contract, handover_to_sublessee.
approval_of_sublease :- approval_before_the_day.
```

This representation illustrates that to prove the contract was ended due to sublease (represented as `cancellation_due_to_sublease`), we must prove four requisites

1. the lease contract was effective (represented as `effective_lease_contract`)
2. the sublease contract was effective (represented as `effective_sublease_contract`)
3. the third party was using the leased thing (represented as `using_leased_thing`)
4. the plaintiff manifested the intention of cancellation of the contract (represented as `manifestation_cancellation`)

And there is one exception, `approval_of_sublease`, which is explicitly stated in the Civil Code. To prove the exception, we must prove that the approval before the cancellation (represented as `approval_before_the_day`).

To prove that the lease contract was effective (`effective_lease_contract`), we must prove two requisites.

1. the lease contract was established (represented as `agreement_of_lease_contract`)
2. the leased thing was handed over to the lessee (represented as `handover_to_lessee`)

To prove that sublease contract was effective (`effective_sublease_contract`), we must prove two requisites.

1. the sublease contract was established (represented as `agreement_of_sublease_contract`)
2. the leased thing was handed over to the sublessee (represented as `handover_to_sublessee`)

With respect to the conclusion `cancellation_due_to_sublease`, the rule-base is built from three critical sets as below.

1. the empty set $\emptyset$ is a critical set that does not lead to the conclusion according to the definition

2. $T_1 = \{$`agreement_of_lease_contract`, `handover_to_lessee`, `agreement_of_sublease_contract`, `handover_to_sublessee`, `using_leased_thing`, `manifestation_cancellation`$\}$ is a critical set that leads to `cancellation_due_to_sublease` and if we delete any element of T_1, the resulting set does not lead to the conclusion. Therefore, T_1 is also an attack set with respect to $\emptyset$.

3. $T_2 = T_1 \cup \{$`approval_before_the_day`$\}$ is a critical set that does not lead to `cancellation_due_to_sublease` since any deletion of an element in $T_2 \setminus T_1 = \{$`approval_before_the_day`$\}$ leads to `cancellation_due_to_sublease` Therefore, $T_2 \setminus T_1$ is a defense set with respect to T_1.

2.3.3 Prolog-based Legal Reasoning Support System (PROLEG)

Prolog-based **Leg**al Reasoning Support System (PROLEG) [63] is a legal reasoning reasoning system based on the Japanese Presupposed Ultimate Fact Theory. This system is extended from the work mentioned in the previous subsection. It invented a new legal representation (Horn clauses and separated exceptions) to express reasoning schemes familiar to lawyers since it turns out lawyers are not familiar with negations as failure. It currently has approximately 2500 rules and exceptions which covers the most part of contract laws in the Japanese Civil Code. A set of rules and exceptions has been compiled by graduates from the top law school in Japan and checked correctness by representing and solving all yes/no questions to use the Japanese Presupposed Ultimate Fact Theory in civil code bar examinations.

PROLEG consists of a rule-base and a fact-base and we will follow this division in this dissertation. We determine a predicate occurring in a head of a rule as a *rule predicate* and a predicate not occurring in a head of a rule as a *fact predicate*. An atom with a rule predicate is called a *rule atom* and an atom with a fact predicate is called a *fact atom*. By this division, we denote a set of all ground fact atoms by $\mathcal{F}$ called a *fact-domain*. We call a program RB a rule-base if fact predicates in RB only occur in the body of a rule. We denote a set of all ground fact atoms that is substitutable to some fact atoms in RB by $f(RB)$. Hence, a PROLEG rule-base and a PROLEG fact-base can be defined as follows.

Definition 2.16 (PROLEG rule-base)**.** Let $\mathcal{F}$ be a fact-domain, S_r be a set of finite Horn clauses, S_e be a set of finite exceptions in the form $exception(h, e)$. where h and e are rule atoms with the meaning of a rebuttal of h by e. We call $RB = \mathcal{S}_r \cup \mathcal{S}_e$ a

PROLEG program and we call RB a PROLEG rule-base if fact predicates in RB only occur in the body of a rule. We denote $rules(RB) = \mathcal{S}_r$ and $exceptions(RB) = \mathcal{S}_e$. We call a set of facts (rules with empty bodies) constructed from a subset of $\mathcal{F}$ a *fact-base*. A fact-base then represents a case.

For example, the Japanese Civil Code Article 612 can be represented as a PROLEG rule-base[2] as follows [63].

```
cancellation_due_to_sublease <= effective_lease_contract,
    effective_sublease_contract, using_leased_thing,
    manifestation_cancellation.
exception(cancellation_due_to_sublease, approval_of_sublease).
effective_lease_contract <=
    agreement_of_lease_contract, handover_to_lessee.
effective_sublease_contract <=
    agreement_of_sublease_contract, handover_to_sublessee.
approval_of_sublease <= approval_before_the_day.
```

Analogous to the definitions in normal logic programs, we also assume that the PROLEG program in this dissertation is non-recursive and stratified. Given a PROLEG program T, the ground PROLEG program of T, denoted by $gr(T)$ is the set of ground rules and ground exceptions obtained from T by replacing all variables in each rule and each exception with every element of its Herbrand universe. An *extension* is the semantics of a PROLEG program calculated in the same manner of the stable model of a normal logic program as follows.

Definition 2.17 (Extension)**.** Let T be a PROLEG program and M be a set of ground atoms. The set of applicable rules of T with respect to M is a set $T^M = \{R \in rules(gr(T))|$ no $exception(head(R), e) \in exceptions(gr(RB))$ such that $e \in M\}$. M is an extension of T if and only if M is the minimum set (in the sense of set inclusion) such that M satisfies every rule $R' \in T^M$, that is $pos(R') \subseteq M$ implies $head(R') \in M$.

For example, let a fact-base FB_0 be

```
{agreement_of_lease. handover_to_leasee. agreement_of_sublease.
handover_to_subleasee. using_leased_thing. manifestation_cancellation.
approval_before_the_day.}
```

[2]We use `<=` in a PROLEG rule-base instead of `:-` RB_0 to distinguish from a rule in a normal logic program

Then, the extension of $FB_0 \cup RB_0$ is as follows, which represents the literal interpretation of the Japanese Civil Code Article 612.

```
{agreement_of_lease, handover_to_leasee, agreement_of_sublease,
handover_to_subleasee, using_leased_thing,manifestation_cancellation,
approval_before_the_day, effective_lease_contract,
effective_sublease_contract, approval_of_sublease}
```

Since $exception(h, e)$ would apply to all rules whose heads are unifiable with h, we would like to discuss here that PROLEG is similar to Defeasible Logic [59] in the sense of using rebutting exceptions. However, the main difference between both representations is that Defeasible Logic allows negations in the head of the rule while PROLEG does not. Actually, the later work [64] shows that the expressive power of PROLEG is the same as normal logic programs since we can translate PROLEG programs into normal logic programs and vice versa. For example, Table 2.1 shows translating from a PROLEG program to a normal logic program.

TABLE 2.1: Translating from a PROLEG program to a normal logic program

$p \Leftarrow b_{11}, \ldots, b_{1n_1}.$ $p \Leftarrow b_{21}, \ldots, b_{2n_2}.$ $\vdots$ $p \Leftarrow b_{k1}, \ldots, b_{kn_k}.$ $exception(p, e_1).$ $\vdots$ $exception(p, e_m).$	$p \leftarrow b_{11}, \ldots, b_{1n_1}, not\ e_1, \ldots, not\ e_m.$ $p \leftarrow b_{21}, \ldots, b_{2n_2}, not\ e_1, \ldots, not\ e_m.$ $\vdots$ $p \leftarrow b_{k1}, \ldots, b_{kn_k}, not\ e_1, \ldots, not\ e_m.$

Table 2.2 shows translating from a normal logic program to a PROLEG program.

TABLE 2.2: Translating from a normal logic program to a PROLEG program

		$exception(c_1, e_{11}).$
		$\vdots$
	$p \Leftarrow c_1.$	$exception(c_1, e_{1m_1}).$
$p \leftarrow b_{11}, \ldots, b_{1n_1}, not\ e_{11}, \ldots, not\ e_{1m_1}.$	$c_1 \Leftarrow b_{11}, \ldots, b_{1n_1}.$	$exception(c_2, e_{21}).$
$p \leftarrow b_{21}, \ldots, b_{2n_2}, not\ e_{21}, \ldots, not\ e_{2m_2}.$	$p \Leftarrow c_2.$	$\vdots$
$\vdots$	$c_2 \Leftarrow b_{21}, \ldots, b_{2n_2}.$	$exception(c_2, e_{2m_2}).$
$p \leftarrow b_{k1}, \ldots, b_{kn_k}, not\ e_{k1}, \ldots, not\ e_{km_k}.$	$\vdots$	$\vdots$
	$p \Leftarrow c_k.$	$exception(c_k, e_{k1}).$
	$c_k \Leftarrow b_{k1}, \ldots, b_{kn_k}.$	$\vdots$
		$exception(c_k, e_{km_k}).$

2.4 Case-based Legal Reasoning Systems

In the 1980s, case-based reasoning systems emerged as alternative reasoning systems that rely on the decision of past cases, and one earliest application of case-based reasoning systems is law. One notable case-based legal reasoning system is HYPO [8], which is later extended into many case-based legal reasoning systems such as CATO [65], Horty Bench-Capon Theory [25], and Rigoni Theory [26] (for a list of extensions of HYPO, see [66]). Case-based legal reasoning systems are developed rapidly especially in common law systems, where the precedent cases are the main reference in the judgement. Generally, case-based legal reasoning systems represent a case as a set of features. Such features are called *dimensions* in HYPO or *factors* in CATO. In the 1990s, hybrid legal reasoning systems between case-based and rule-based have been explored such as CABARET [9], GREBE [10], and HELIC-II [11]. In the 2000s, *Theory Construction*, a sub-field of artificial intelligence that studies building a logic program for explaining reasons behind precedent cases, has been established. Researchers have applied many machine learning techniques, for example, heuristic search [18], association rule [20], and argument-based machine learning [19]. Below presents a case-based reasoning system called an abstract argumentation for case-based reasoning (AA-CBR) and the work of theory construction from a case-base in AA-CBR, which we refer to in this dissertation.

2.4.1 *Abstract Argumentation for Case-based Reasoning (AA-CBR)*

Abstract Argumentation for Case-based Reasoning (AA-CBR) [67] represents the principle of how we make a judgment from precedent cases with a default judgement. We call a subset of a fact-domain a case in AA-CBR. A case is noted with '+' or '−' to imitate which side won in the case. A case-base in AA-CBR is defined as follows.

Definition 2.18 (Case-base in AA-CBR)**.** Let $\mathcal{F}$ be a fact-domain. A case is a subset of $\mathcal{F}$. A case with judgement on $\mathcal{F}$ is a pair $cj = (C, J)$ where C is a case and $J \in \{\text{'+'}, \text{'−'}\}$. We denote the complement of J by $\bar{J}$, namely $\bar{J} = \text{'+'}$ if $J = \text{'−'}$; and $\bar{J} = \text{'−'}$ if $J = \text{'+'}$. A *case-base*, denoted with CB, is a set of cases with judgements, namely a subset of $P(\mathcal{F}) \times \{\text{'+'}, \text{'−'}\}$, where $P(\mathcal{F})$ is the powerset of $\mathcal{F}$. A case base is assumed to be consistent, namely there is no pair $(X, \text{'+'}), (X, \text{'−'}) \in CB$.

Let illustrate a case-base with an example adapted from [68].

Example 2.3. *Suppose a case-base regards to claim compensation from a delivery company. A fact domain contains considering fact predicates as follows.*

1. The items were delivered on time (represented as `delivered_on_time`)

2. The items were damaged (represented as `items_were_damaged`)
3. The damaged items are repairable (represented as `damage_is_repairable`)
4. The buyer had fixed an additional period for repairing (represented as `buyer_fixed_repair_time`)
5. The items were repaired in the above additional period (represented as `items_repaired_in_time`)

Let '+' represent that the buyer won and '−' represent that the delivery company won. The case-base CB_1 consists of three past cases with judgement as follows.

1. $cj_1 = (\{\texttt{delivered\_on\_time}\}, \text{'}-\text{'})$ represents a case where items were delivered on time and the delivery company won this case
2. $cj_2 = (\{\texttt{delivered\_on\_time, items\_were\_damaged}\}, \text{'}+\text{'})$ represents a case where items were delivered on time, but the items were damaged, and the buyer won this case
3. $cj_3 = (\{\texttt{delivered\_on\_time}, \texttt{items\_were\_damaged}, \texttt{damage\_is\_repairable}, \texttt{buyer\_fixed\_repair\_time}, \texttt{items\_repaired\_in\_time}\}, \text{'}-\text{'})$
   represents a case where items were delivered on time but the items were damaged. However, the damaged items are repairable. The buyer fixed an additional period for repairing and the items were repaired in the period. The delivery company won this case.

AA-CBR requires two steps to provide reasoning. The first step is addressing a default judgement, a judgement that should be assigned to an empty case. For example, it is obvious that the buyer cannot claim compensation without any special circumstance so the default judgement shall be '−' hence the case-base must include $cj_0 = (\emptyset, \text{'}-\text{'})$. The second step is building an abstract argumentation framework [12]. An abstract argumentation framework is a pair $(AR, attacks)$, where AR is a set of arguments and $attacks$ is a subset of $AR \times AR$. AA-CBR refers to a case-base as a set of arguments CB and says $cj \in CB$ attacks $cj' \in CB$ if and only if cj is a trumping case to cj' with respect to CB defined as follows.

Definition 2.19 (Trumping case)**.** Let CB be a case-base and $(C, J) \in CB$. A case with judgement $(N, \bar{J})$ is a trumping case (not always unique) to (C, J) with respect to CB when $C \subsetneq N$, and no other $(N', \bar{J}) \in CB$ such that $C \subsetneq N' \subsetneq N$.

Definition 2.20 (Abstract argumentation in AA-CBR)**.** Let CB be a case-base with a default case with judgement $(\emptyset, J_0)$. $AF = (CB, attacks)$ is an argumentation framework

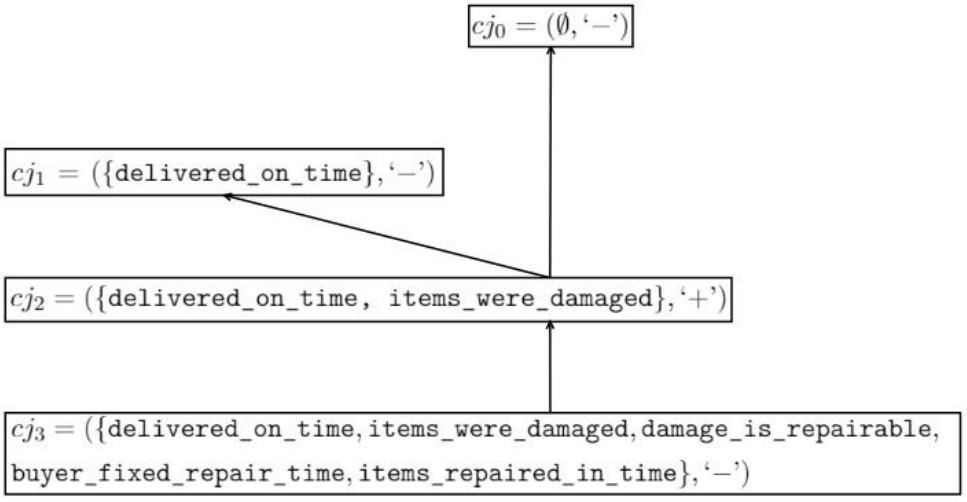

FIGURE 2.1: Corresponding Argumentation Framework in AA-CBR

corresponding to CB and *attacks* is a set of $(X, \bar{J}) \rightsquigarrow (Y, J)$ for all pairs $(X, \bar{J}), (Y, J) \in CB$ such that $(X, \bar{J})$ is a trumping case to (Y, J) with respect to CB.

Definition 2.21 (Predicted judgement in AA-CBR)**.** To predict a judgement for a case N, let $CB_N = \{(C, J) \in CB | C \subseteq N\}$. A predicted judgement of N is J_0 if $(\emptyset, J_0)$ is in the grounded extension of the corresponding argumentation framework of CB_N. Otherwise, a predicted judgement of N is $\bar{J_0}$ [67].

The grounded extension G is a subset of CB_N constructed by $G = \bigcup_{i \geq 0} G_i$, where G_0 is the set of un-attacked cases with judgement in CB_N (that is, for all $(Y, J_y) \in G_0$, there is no $(X, \bar{J_y}) \rightsquigarrow (Y, J_y) \in attacks$), and for all $i \geq 0$, G_{i+1} is the set of cases with judgement in CB_N that G_i defends (that is $(Y, J_y) \in G_{i+1}$ if for all $(X, \bar{J_y}) \rightsquigarrow (Y, J_y) \in attacks$, there is $(Z, J_y) \in G_i$ such that $(Z, J_y) \rightsquigarrow (X, \bar{J_y}) \in attacks$) [12]. We say CB entails (N, J) (denoted by $CB \models (N, J)$) when a predicted judgement of N is J with respect to CB.

Figure 2.1 shows the argumentation framework corresponding to the case-base in Example 2.3. Suppose we would like to predict a judgment for a new case $N = \{$`delivered_on_time`, `items_were_damaged`, `damage_is_repairable`, `buyer_fixed_repair_time`, `items_repaired_in_time`$\}$, representing a case where items were delivered on time but the items were damaged, the damaged items are repairable, the buyer fixed an additional period for repairing, but the items were not repaired in the period. We get that $CB_N = \{cj_0, cj_1, cj_2\}$ and the grounded extension of the argumentation framework of the corresponding argumentation framework of CB_N is $\{cj_2\}$. Since $cj_0 = (\emptyset, \text{`}-\text{'})$ is excluded from the grounded extension, a predicted judgement of a new case is '+', which means the buyer should win this case and be able to claim the compensation.

We can see that not all but only those cases with judgement related to the default case with judgement $(\emptyset, J_0)$ are involved in the prediction of judgement. For example, cj_1 in Figure 2.1 can be omitted. This corresponds to the minimization of case-base representation using critical cases (adopted from *relevant attack* in [68]) defined as follows.

Definition 2.22 (Critical case-base)**.** Let CB be a case-base. A case with judgement $(C, J) \in CB$ is critical if and only if (C, J) is a default case with judgement $(C = \emptyset)$ or $(C, J) \rightsquigarrow (B, \bar{J}) \in \mathit{attacks}$ where $(B, \bar{J})$ is critical. A critical case-base means a case-base in which all cases with judgement are critical.

2.4.2 Theory Construction from a Case-base in AA-CBR

As constructing a theory, a rule-base that has the same expressive power to predict a judgement, becomes interested in case-based reasoning field, Athakravi et al. [68] present the translation of a case-base in AA-CBR into a theory by using answer set programming. The theory is called a *judgement theory*, defined in our setting as follows.

Definition 2.23 (Judgement theory)**.** Let $\mathcal{F}$ be a fact-domain, RB be a rule-base, CB be a case-base and p_0 is a ground rule atom called an ultimate goal. RB is a *theory* of CB with respect to p_0 when given a case $C \subseteq \mathcal{F}$ and a fact-base FB such that $f(FB) = C$, $CB \vDash (C, \text{`}+\text{'})$ if and only if $FB \cup RB \vdash p_0$.

We denote the preliminary theory construction from [68] by a function THEORY-CONS. For ease of exposition, we assume that the given case-base is critical and the default judgement is '$-$', reflecting the principle that no one can claim anything if there is no special situation. It is proved in their work that THEORY-CONS is one way, but not the only way, of constructing theory of the given case-base.

Definition 2.24 (Theory-cons)**.** Given a critical case-base CB with a default judgement '$-$', an ultimate goal p_0, and $(CB, \mathit{attacks})$ as a corresponding argumentation framework. THEORY-CONS(CB, p_0) is defined in our setting by

$$\text{THEORY-CONS}(CB, p_0) = \{p_0 : -X \cup ab_X | (X, \text{`}+\text{'}) \rightsquigarrow (\emptyset, \text{`}-\text{'}) \in \mathit{attacks}\} \cup$$
$$\{p_{X,Y} : -X \setminus Y \cup ab_X | (X, J) \rightsquigarrow (Y, \bar{J}) \in \mathit{attacks}\}$$
where $X \setminus Y$ is the set difference of X and Y,
$ab_X = \{\mathit{not}\ p_{Z,X} | (Z, J) \rightsquigarrow (X, \bar{J}) \in \mathit{attacks}\}$,
and $p_{X,Y}$ be a rectified proposition of cases X and Y and distinct to p_0.

Let CB be a critical case-base $\{cj_0, cj_2, cj_3\}$ with respect to Example 2.3, `compensation` be an ultimate goal and `exception` is a rectified proposition with respect to cj_3 attacks

cj_2 in Figure 2.1. We can construct a theory by calling THEORY-CONS(CB, `compensation`) as follows.

```
compensation:- delivered_on_time, items_were_damaged, not exception.
exception:- damaged_is_repairable, buyer_fixed_repair_time,
    items_repaired_in_time.
```

The constructed rule-base has the same expressive power to predict a judgement as the given case-base. For example, Let N be the new case mentioned in the previous section, the theory explains that the buyer could claim compensation since the buyer could prove that the items were delivered on time, the items were damaged, and the delivery company could not claim the exception since the company fails to prove that the items were repaired in time. This explanation also corresponds with the predicted judgement from AA-CBR that the buyer should be able to claim the compensation in this case.

2.5 Formalizing Judicial Legal Change

Change is one enduring characteristic of the law. Levi [69] has discussed that legal reasoning involves with *moving classification system*, where legal concepts are the classifiers. Levi states that "*the kind of reasoning involved in the legal process is one in which the classification changes as the classification is made* [69]". Ashley [70] has explained more about legal reasoning as *moving classification system* that when judges decide the case, judges may clarify the meaning of legal concepts but the clarification often has some level of obscure. To distinguish a case, judges may introduce an exception to the rule by introducing a new legal concept, the rule is revised and the process is continued. This process of legal rule revision is similar to a *qualification problem* in artificial intelligence [71], in which necessary conditions are hard to be qualified in the first place and such necessary conditions are often discovered when an exceptional case goes to high court and the literal interpretation of the law in that case leads to counterintuitive consequences.

AI and Law researchers have been long interested in legal change [72] especially in case-based legal reasoning systems since such legal change can be detected through a temporal order of precedent cases. For example, Rissland and Friedman [16] investigate changes of legal concepts through time by analyzing a temporal order of cases. Berman and Hafner [17] consider precedent cases that may strengthen or weaken as time has passed. The role of legal context in legal change has been explored in [21] and classified

into three aspects, which are teleological relations, temporal relations, and procedural postures. As AI and Law researchers become interested in *Theory Construction*, a subfield that studies building explainable theories behind precedent cases, recent studies focus on revisions of such theories. For example, Henderson and Bench-Capon [73] have encouraged contrastive revision of a theory by not only supporting accepted consequences but also attacking the objected ones. Rotolo and Roversi [74] have classified theory revision criteria and minimal change is one of such criteria. This criterion comes from the standard revision theory, which intuitively states that we should adjust theories as little as possible. To minimally revise rule-based theories, they present two strategies:

1. keep the set of rules as close as possible to the original one (this strategy is independent of the facts of the case)
2. minimize the changes of the set of conclusions obtained from the theory (the strategy is dependent of the facts of the case since different facts give different set of conclusions)

Explaining legal change can also be seen in recent extensions of HYPO [27, 66]. For example, Horty Bench-Capon theory [25], one extension of HYPO, states that a new legal judgement could be made as long as it preserves the *precedential constraint*. This guides a judge to introduce new factors in a new case to preserve the precedential constraint if the new case is exceptional. Horty Bench-Capon theory has been extended into Rigoni theory [26] for supporting hybrid legal reasoning systems between rule-based and case-based. Moreover, recently presented Verheij's case models [28–30] have become new explanations of hybrid legal reasoning systems by using the preference of cases.

In this section, we present one example of legal change in Japanese Civil Code Article 612, which are presented in [63, 75]. The example of legal change in the Japanese Civil Code Article 612 is based on the Japanese Supreme Court case with the phrase as follows.

> [The Japanese Civil Code Article 612] 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 the lessor proves the lessee's destruction of confidence [76].

In this court decision, the judge introduced a factual concept of non-destruction of confidence (represented as `non_destruction_of_confidence`) as an exception of Phrase 2 to prevent the counterintuitive consequence from the literal interpretation of the Japanese

Civil Code Article 612. Hence, the case that goes to the Supreme Court can be represented as the following set of fact-base $JFB_1 =$

```
{agreement_of_lease_contract. handover_to_lessee.
agreement_of_sublease_contract. handover_to_sublessee.
using_leased_thing. manifestation_cancellation.
non_destruction_of_confidence.}
```

2.6 Related Work

In this section, we organize the related works according to three research questions as described in Chapter 1 and compare our whole system with related inductive logic programming systems (ILPs) as follows.

2.6.1 RQ1: How to detect a culprit ?

Representing and detecting *errors* in law have been formalized in several aspects. We list here three examples of them as follows.

1. **Conflicts**: Generally speaking, conflicts in law is a situation where two rules disagree on the same consequence i.e. the first rule permits some action but the second rule does not allow to do so. The conflict can be detected and resolved easily if one rule involved in the conflicts is higher than the other. In that case, we can have an automatic revision of the lower rule to detect and resolve conflicts e.g. [36]. It becomes harder when two rules causing conflicts are in the same level or incomparable i.e. such rules are in the same statute or each rule applies in different areas. Although there are some semi-formal frameworks [77, 78] introduced to represent and detect this conflict, this problem requires a legal expert to be involved in conflict resolutions.

2. **Legal Gap**: A legal gap is, roughly speaking, a situation where some legal action is blocked since the regulation is poorly drafted. Actually, legal gaps may be considered as a kind of conflict. Legal gap can be divided roughly into two types [78]. The first type is called a genuine legal gap, for example, there exist two rules stating "event E must be done before event A" and "event E must be done after event A". This can be determined as a conflict since we could say one rule obligates event E to be done before event A but the second rule does not allow to do so. The second type is called an axiological legal gap, for example, there exist

two rules stating "event E must be done before event A" and "event E must be done after event B". Such rules cause no conflicts unless A is done before or at the same time of B. Although there is a semi-formal framework called AFLEG [78] tackling these legal gaps, it still loads on a user heavily to resolve them.

3. **Loophole**: A loophole is a board word, meaning a situation where some action is unanticipated (i.e. not following the purpose of law) but, according to the literal interpretation of the law, it is valid [79]. For example, in *Adler v. George (1964)*, one has obstructed a military guard in the execution of his duty inside a military establishment. This exploits the prohibition of obstructing a military guard in the *vicinity* of a military establishment, in which *vicinity* literally means *outside or in the proximity*. Judges may resolve this loophole by broadening or modifying the legal concept. In *Adler v. George (1964)*, judges have stated that such an interpretation would lead to an absurd result, and reinterpret the word *vicinity* in this rule to cover inside a military establishment. Such interpretative argumentation in law can be represented by Interpretative Argumentation Scheme [80], where we can vary interpretations of law with respect to the settings or the legal context of law.

As we can see, resolving such errors usually involves a legal expert. Hence, debugging is one of earliest requirement in legal formalisms. The earliest work [2] of formalizing statutes into normal logic program has suggested that *debugging* the rules is one interesting application of formalizing statutes. Later works [31, 37] also suggest in the same manner that debugging is one scheme required for legal reasoning systems to cope with legal changes. However, to our knowledge, our work is the first work that theoretically formalizes debugging in law. There are few closely related research fields such as the field of legal ontology refinement [34] but it represents legal knowledge in Horn clauses, unlike our work that considers the setting in a normal logic program.

Debugging is also applied for navigating users in other fields besides software [81]. In Intelligent Tutoring Systems (ITS), Algorithmic Debugging is applied for detecting student misconceptions as bugs [82]. However, the program in this work is presumed to be correct, in contrast to general program debugging that assumes the users as the oracle query with correct interpretations. In Hardware Design and Verification, Algorithmic Debugging for detecting faulty components. It is applicable by viewing digital circuits as auxiliary functions [83] and logic programs [84].

2.6.2 RQ2: How to resolve a culprit ?

As we have mentioned, early techniques in ILP (e.g. MIS) focus on monotonic reasoning especially a logic program with only Horn clauses. One reason is that there are many open issues in ILP for non-monotonic reasoning [85]. For example, one issue is that what can be treated as *known* and *unknown* in a logic program with negation as failure. De Raedt and Bruynooghe [86] have once discussed that a logic program with negation as failure acts as if everything is known so it is hard to identify things that should be determined as *unknown*. This issue is one main difference between legal change in common law systems and civil law systems. In common law systems, judges decide based on precedent cases (or most of precedent cases that are not obsolete and consistent) so precedent cases make the constraint that the present case must follow, otherwise judges must introduce new facts to distinguish the present case from the precedents (see [25, 27]). In ILP terms, precedent cases become examples that the induced program must follow. In civil law systems, in contrast, judgements are independent from the precedent cases. However, arbitrary revision of law is obviously not preferable. Therefore, in this research, we solve the problem by *prototypical cases with judgement* built from the rule-base and we propose that such prototypical cases with judgement should be determined as *known* examples, as if precedent cases in common law systems. In the other words, the revision should preserve the judgements of such prototypical cases. This proposal is aligned with the proposal that statutes are constructed from *critical sets* [61] and hence the revision should preserve them.

2.6.3 RQ3: How to evaluate the resolution ?

In monotonic reasoning, generalization operations hold the implication order, that is if a monotonic theory T derives p then a generalization of T also derives p. This implication order does not hold in non-monotonic reasoning since an exception, e.g. the negative body of the rule, is allowed in non-monotonic reasoning. So, the order of generalization operations in non-monotonic reasoning is still an open issue [85]. Actually, the order of generalization operations in non-monotonic reasoning usually depends on semantics and this affects the design of debuggers. For example, a debugger for an answer set programming must treat multiple situations concurrently because the program allows multiple answer sets [87]. A debugger for Datalog has to deal with a non-stratified program differently because the semantics of some non-stratified programs in Datalog is the empty set, unlike such programs in Prolog that get stuck in a loop [88]. Besides semantics, some studies on debuggers have focused on specific types of bugs and unexpected results such as inconsistency [89–91] and absence of answer sets [92, 93]. However, our research is

based on the assumption that legal rules are usually stratified and non-recursive. Hence, these specific types of bugs can be omitted because such a stratified program has been proved to have a unique answer set [52]. However, in our study, we consider a problem in legal reasoning that the semantics of a rule-base is dependent of the facts of the case. Although this problem has been addressed in the works of hybrid legal reasoning systems such as Rigoni Theory [26] and Verheij case models [28–30], effects of legal change on revision of rules have not been fully investigated.

2.6.4 Comparison with Related ILPs

Given a logic program B representing background knowledge (called *background theory*), a set of positive examples E^+, and a set of negative examples E^-. ILP can be generally formalized as a task to find a hypothesized logic program H such that $B \wedge H \models E^+; B \wedge H \not\models E^-$. From this generalization, we can compare ILP techniques on the following dimensions [94].

TABLE 2.3: Comparison of Legal Debugging with related ILPs

ILPs	Background Theory	Bias and Additional Inputs	Search Method
MIS [32]	Definite	-	Top-down
PROGOL [95]	Normal	Modes M	Top-down Bottom-up
ASPAL [96]	Normal	Modes M Penalty γ	ASP
Our System	Normal	Modes M Examples E^+_{pcj}, E^-_{pcj} (M,E^+_{pcj},E^-_{pcj} are all implicit)	Top-down Bottom-up

- **Background Theory**: There are various type of background theory supported by ILP systems. In Table 2.3, we show that *MIS* is one example that supports definite background theories where complex data structures such as lists are mainly focused negations as failure are not mainly focused. Later ILP systems such as *PROGOL*, *ASPAL*, and our system tend to support background theories in normal logic program form, where negations as failure are allowed.

- **Bias and Additional Inputs**: To restrict the *hypothesis space*, some language bias can be introduced as additional inputs. One common language bias is a *mode declaration* introduced in *PROGOL*. In mode declaration, we restrict which literals that can occur in the head of a rule and which literals that can occur in the body of a rule. Later ILP systems tend to use mode declaration such as *ASPAL* as well as our system but we use it implicitly through the division of rule predicates and fact predicates. *ASPAL* also introduced a function called penalty as another

additional input, using for ordering the hypothesis while our system propose new concepts of reproducing implicit examples from the rule-base using *prototypical cases with judgement* (E^{+}_{pcj},E^{-}_{pcj}).

- **Search Methods**: Basically, there are two methods used for searching the *hypothesis space*, which is the set of all possible hypotheses that can be built in the language. The two methods are the top-down method and the bottom-up method. Both methods are based on the generality relation orders between two clauses i.e. the refinement operator in *MIS*. The top-down method starts searching from a general hypothesis and specializes it. *CWS* is also another example of the top-down method since we start from one general hypothesis and we introduce new conditions to specialize the hypothesis. In contrast, the bottom-up method starts searching from a specific hypothesis and generalizes it. *IR* is one example of the bottom-up method since we start from one specific hypothesis and we generalize the hypothesis using V-operators or W-operators. PROGOL and our system also introduced *IR* for bottom-up search where our system developed top-down search by extending *CWS*.

2.7 Summary

- Logic Programming has been established in the twentieth century. A normal logic program is defined as a finite set of rules. Early applications of logic programming are, for example, automated theorem proving and representing statutes.
- A normal logic program representing statutes tends to be a non-recursive and stratified program since such a program has a unique stable model. This also reflects a constraint that judges need one unique judgement from legal rules.
- We divide predicates into two types, namely *rule predicates* and *fact predicates*. A *rule-base* is defined as a program in which fact predicates only occurring in the body of a rule. A *fact-base* is a set of ground facts, in which only fact predicates occur.
- The literal interpretation of the statute in the case can be obtained by the stable model semantics of the whole program (the union of the rule-base and the fact-base). A consequence (represented by a ground atom with a rule predicate) is in the stable model if and only if there is the supporting rule in the program.
- We have presented three Inductive Logic Programming techniques as follows
  - *Model Inference System (MIS)* revises the program by specializing the input program if it is too generalized and generalizing the input program if it is too

specialized. *MIS* is also applied for inductive program synthesis, in which a logic program is built from the empty program with merely training examples

  - *Closed World Specification (CWS)* specializes the input program by introducing an exception when the input program succeeds on a goal known to be false.
  - *Inverse Resolution (IR)* involves inverting the resolution operator to generalize the input program by introducing a general rule from the existing rules.

- We have presented *Abstract Argumentation for Case-based Reasoning (AA-CBR)*, a case-base reasoning framework based on the principle of how we make a judgment from precedent cases with a default judgement. Then, we have presented one theory construction, which constructs a normal logic program for explaining reasons behind precedent cases, from a case-base in AA-CBR.

- Legal rule revision in the judicial process goes as follows. When judges apply interpretation of statutes in a particular case and it leads to counterintuitive consequences, judges may introduce an exception to the rule by introducing a new legal concept, the rule is revised and the process is continued.

- This research, as the first theoretical formalization of debugging in law, tries to tackle two issues in Inductive Logic Programming for non-monotonic reasoning, especially for legal reasoning, as follows.

  - Given a normal logic program representing statutes, which *cases* we should fix their decision, especially in rule-based legal reasoning systems with no precedent cases are available. To solve this issues, we propose to build *prototypical cases with judgement* from the rule-base and we determine them as *known examples* as if they are precedent cases.
  - How to deal with generalization operations in non-monotonic reasoning, which usually depend on semantics. In this research, we consider especially on legal reasoning. We focus on the effects of legal change on revision of rule under the consideration of semantics which depends on the facts of the cases.

Chapter 3

Culprit Detection Algorithm

In this chapter, we present

- Formalization of counterintuitive consequences in law and culprits
- Culprit Detection Algorithm
- Extensions of Culprit Detection in First-order Rule-base and PROLEG Rule-base

3.1 Overview

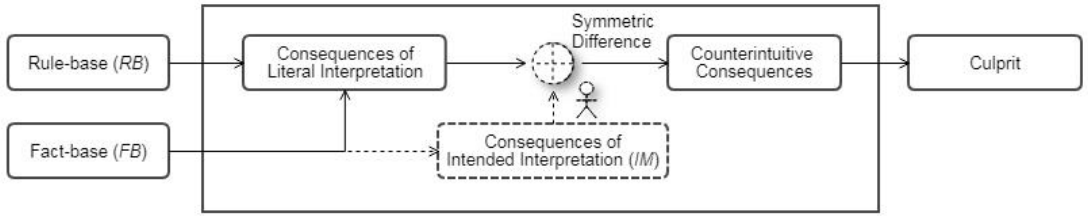

FIGURE 3.1: Culprit Detection Workflow
(Dash-lines represent the data or the process that a user involves)

Figure 3.1 shows the work flow of Culprit Detection. Given a rule-base RB representing the statute and a fact-base FB representing a case, a legal reasoning system derives consequences of the literal interpretation of the statute in the case. A user is assumed to be a legal expert who can answer correctly about whether the derivation is counterintuitive according to the commonly held opinion of reasonable people (*Communis opinio* [97]). In formal words, we say a user is an oracle query [98] of an unknown set of *intended interpretation* and the counterintuitive consequence is the symmetric difference of the literal interpretation and the intended interpretation. Culprit Detection Algorithm iterates to ask the user whether related consequences are counterintuitive until it can no longer find any counterintuitive consequences related. We call the last counterintuitive consequence found a *culprit*. Intuitively, we determine a culprit as a root cause of counterintuitive consequences or a legal bug, which is a legal consequence that has no logical alignment with the intended interpretation.

3.2 Counterintuitive Consequences and Culprits

When the literal interpretation is counterintuitive, it means that the user has another interpretation in mind, even if it is unknown in the first place. Given the fact domain $\mathcal{F}$, we assume an *intended interpretation* denoted by an unknown set of ground atoms IM such that a set of all fact atoms in IM is equal to a set of all fact atoms in FB ($\mathcal{F} \cap IM = f(FB)$ where $f(T)$ is a set of all ground fact atoms that can be substitutable to some fact atoms in a program T, see Chapter 2 for details). IM represents the intended interpretation of the statute in the considered case. We describe a user as a membership query [98] that can answer correctly about whether a ground atom A is a member of IM. We define a counterintuitive consequence as a member of the symmetric difference (denoted by $\oplus$) between the literal and the intended interpretation.

Definition 3.1 (Counterintuitive consequence)**.** Given a fact-base FB representing a case, a rule-base RB representing statutes, and a set of ground atoms IM such that $\mathcal{F} \cap IM = f(FB)$ representing the *intended interpretation*. Let $T = FB \cup RB$ and M be the answer set of T. A ground atom p is a *counterintuitive consequence* with respect to IM and RB if $p \in M \oplus IM$, that is $p \in M \setminus IM$ or $p \in IM \setminus M$.

To let a user specify which condition is problematic, we extend Legal Debugging from Algorithmic Debugging [38] for legal reasoning. Firstly, we define a culprit as follows.

Definition 3.2 (Culprit)**.** A ground rule atom p is a *culprit* with respect to an intended interpretation IM and a rule-base RB if

- $p \notin IM$ but there is a rule $R \in gr(RB)$ that supports p with respect to IM (called an incorrect culprit) or
- $p \in IM$ but there is no rule $R \in gr(RB)$ that supports p with respect to IM (called an incomplete culprit)

By the definition of culprit (Definition 3.2), we get the following proposition.

Proposition 3.3. *Let IM be an intended interpretation and RB be a rule-base. There is no counterintuitive consequence with respect to IM and RB if and only if there is no culprit with respect to IM and RB.*

Proof. If there is no counterintuitive consequence with respect to IM and RB, then IM is the answer set of RB. Hence, $p \in IM$ if and only if there is a rule $R \in T$ that supports p with respect to IM according to Theorem 2.10. Hence, there is no culprit with respect to IM and RB (This applies vice versa). □

We show two examples of counterintuitive consequences and culprits as follows.

Example 3.1. *Suppose a fact-domain $\mathcal{F} = \{a, b, c\}$, a fact-base $FB_1 = \{a.\ c.\}$, a rule-base $RB_1 = \{p \leftarrow q.\ q \leftarrow a.\ q \leftarrow b.\}$, and $IM_1 = \{a, c, r\}$. Let $T_1 = FB_1 \cup RB_1$ and the answer set of T_1 is $M = \{a, c, p, q\}$. Hence p, q, r are counterintuitive consequences and*

- *p is not a culprit since $p \notin IM_1$ and there is no rule supporting p with respect to IM_1.*
- *q is an incorrect culprit since $q \notin IM_1$ but $q \leftarrow a$ supports q with respect to IM_1.*
- *r is an incomplete culprit since $r \in IM_1$ but there is no rule supporting r with respect to IM_1.*

Example 3.2. *Suppose a fact-domain* $\mathcal{F} = \{a, b, c\}$*, a fact-base* $FB_2 = \{a.\ c.\}$*, a rule-base* $RB_2 = \{p \leftarrow q.\ q \leftarrow a, not\ r.\ q \leftarrow b.\ r \leftarrow c.\}$*, and* $IM_2 = \{a, c, q\}$*. Let* $T_2 = FB_2 \cup RB_2$ *and the answer set of* T_2 *is* $M = \{a, c, r\}$*. Hence* q, r *are counterintuitive consequences and*

- p *is an incorrect culprit since* $p \notin IM_2$ *but* $p \leftarrow q$ *supports* p *with respect to* IM_2*.*
- q *is not a culprit since* $q \in IM_2$ *and* $q \leftarrow a, not\ r$ *supports* q *with respect to* IM_2*.*
- r *is an incorrect culprit since* $r \notin IM_2$ *but* $r \leftarrow c$ *supports* r *with respect to* IM_2*.*

From both examples, we get that counterintuitive consequences and culprits have some relations. Although a counterintuitive consequence may not be a culprit (e.g. p in Example 3.1 and q in Example 3.2) and a culprit may not be a counterintuitive consequence (e.g. p in Example 3.2), a non-culprit counterintuitive consequence always links to another counterintuitive consequence (e.g. p links to q in Example 3.1 and q links to r in Example 3.2). Thus, we formalize this into the following theorem.

Theorem 3.4 (Counterintuitive Consequence Propagation)**.** *Let* IM *be an intended interpretation and* RB *be a rule-base. If* p *is a non-culprit counterintuitive consequence with respect to* IM *and* RB *then there is a rule* $R \in gr(RB)$ *such that* $p \in head(R)$ *and a counterintuitive consequence occurs in* $body(R)$*.*

Proof. Let FB be some fact-base, $T = FB \cup RB$, and M be the answer set of T. We divide the proof into two cases:

Case 1: If $p \in IM$ and p is not a culprit. Then, there is a ground rule $R \in gr(RB)$ such that $pos(R) \subseteq IM$, $neg(R) \cap IM = \emptyset$, and $head(R) = p$. Since p is a counterintuitive consequence, $p \notin M$ so R does not support p with respect to M. Thus, $pos(R) \not\subseteq M$ or $neg(R) \cap M \neq \emptyset$. Hence, there is either $q \in pos(R)$ such that $q \in IM \setminus M$ or $q \in neg(R)$ such that $q \in M \setminus IM$.

Case 2: If $p \notin IM$ and p is a counterintuitive consequence. Then, $p \in M$. Therefore, there is a ground rule $R \in gr(RB)$ such that $pos(R) \subseteq M$, $neg(R) \cap M = \emptyset$, and $head(R) = p$. Since p is not a culprit, R does not support p with respect to IM. Thus, $pos(R) \not\subseteq IM$ or $neg(R) \cap IM \neq \emptyset$. Hence, there is either $q \in pos(R)$ such that $q \in M \setminus IM$ or $q \in neg(R)$ such that $q \in IM \setminus M$. □

3.3 Culprit Detection Algorithm

We design Culprit Detection Algorithm as in Algorithm 3. The algorithm begins with one counterintuitive consequence and attempts to find another counterintuitive consequence according to Theorem 3.4. The algorithm is recursively called until it can no longer find related counterintuitive consequences. The algorithm always succeeds with a culprit as long as the rule-base is finite, non-recursive, and stratified. This algorithm aims to detect only one culprit at a time. If there are two culprits or more, the user has to repeat the algorithm for a new culprit after resolving the old one.

Algorithm 3 Culprit Detection Algorithm

Given a fact-base FB, a rule-base RB, a counterintuitive consequence atom p, a user as a membership query of the intended interpretation IM

procedure CULPRIT-DETECTION(p)
  Let M be the answer set of $FB \cup RB$
  Find $R \in gr(RB)$ that supports p with respect to IM
  if $p \in IM$ and there is no such R **then** return p;
  else if $p \in IM$ **then**
    Find $q \in pos(R)$ such that $q \notin M$
    if there is such q **then** return CULPRIT-DETECTION(q)
    Find $q \in neg(R)$ such that $q \in M$
    if there is such q **then** return CULPRIT-DETECTION(q)
  if $p \notin IM$ and there is such R **then** return p;
  else if $p \notin IM$ **then**
    Find $R' \in gr(RB)$ that supports p with respect to M
    Find $q \in pos(R')$ such that $q \notin IM$
    if there is such q **then** return CULPRIT-DETECTION(q)
    Find $q \in neg(R)$ such that $q \in IM$
    if there is such q **then** return CULPRIT-DETECTION(q)

Theorem 3.5 (Correctness of Culprit Detection Algorithm)**.** *Given a fact-base FB, a finite non-recursive and stratified rule-base RB, a counterintuitive consequence p_0, and an intended interpretation IM such that $\mathcal{F} \cap IM = f(FB)$. Algorithm 3 always returns a culprit with respect to IM and $FB \cup RB$.*

Proof. From a counterintuitive consequence p_0, we can find another related counterintuitive consequence $p_1 \neq p_0$ according to Theorem 3.4. Then, we may find related counterintuitive consequences iteratively as a sequence of $p_0, p_1, p_2, \ldots, p_n$, $p_i \neq p_j$ for every $i \neq j$ because the rule-base is non-recursive and stratified. The sequence is finite because the ground rule-base is finite. Since no other counterintuitive consequences occurs in the body of the rule whose head is p_n (because it is at the end of the sequence), from the contraposition of Theorem 3.4, p_n is a culprit. □

Theorem 3.6 (Completeness of Culprit Detection Algorithm)**.** *Given a fact-base FB, a finite non-recursive and stratified rule-base RB, a counterintuitive consequence p_0, and an intended interpretation IM such that $\mathcal{F} \cap IM = f(FB)$. If there is a culprit p_c with respect to IM and $FB \cup RB$, we can detect p_c in finite steps by using Algorithm 3 and incremental resolving the detected culprits (assuming a user is an equivalence query [98] that correctly answers 'yes' when the answer set of $FB \cup RB$ is equal to IM, or provides one counterintuitive consequence when it is not).*

Proof. Let M be the answer set of $FB \cup RB$ and $p_1, \ldots, p_n$ be an incremental sequence of culprits that have been resolved. With an equivalence query, the algorithm always gets a counterintuitive consequence when $M \neq IM$. Since $p_1, \ldots, p_n$ have been incrementally resolved, Algorithm 3 returns a new culprit p_{n+1}. Since the ground rule-base is finite, the sequence reaches to p_c in finite steps. □

Example 3.3. *Let us illustrate Culprit Detection Algorithm using the rule-base representing the Japanese Civil Code Article 612 as follows.*

```
1  cancellation_due_to_sublease :- effective_lease_contract,
2          effective_sublease_contract, using_leased_thing,
3          manifestation_cancellation, not approval_of_sublease.
4  effective_lease_contract :-
5          agreement_of_lease_contract,
6          handover_to_lessee.
7  effective_sublease_contract :-
8          agreement_of_sublease_contract,
9          handover_to_sublessee.
10 approval_of_sublease :- approval_before_the_day.
```

Figure 3.2 shows the flowchart of Culprit Detection Algorithm in the example case. Since the user (representing the judges) has considered that `cancellation_due_to_sublease` is not intended and hence it is a counterintuitive consequence. The algorithm considers whether there is another counterintuitive consequence related to `cancellation_due_to_sublease` by tracing the supporting rule of `cancellation_due_to_sublease` (lines 1-3). The algorithm then asks the user whether `effective_lease_contract` is counterintuitive. If it is counterintuitive, it becomes an incorrect culprit because there is a rule supporting it but it is not intended (situation 1). If it is not, the algorithm then asks the user about `effective_sublease_contract` in the same manner. If `effective_sublease_contract` is counterintuitive, it becomes an incorrect culprit because there is a rule supporting it but it is not intended (situation 2). If it is not, then the algorithm asks the user whether `approval_of_sublease` is counterintuitive.

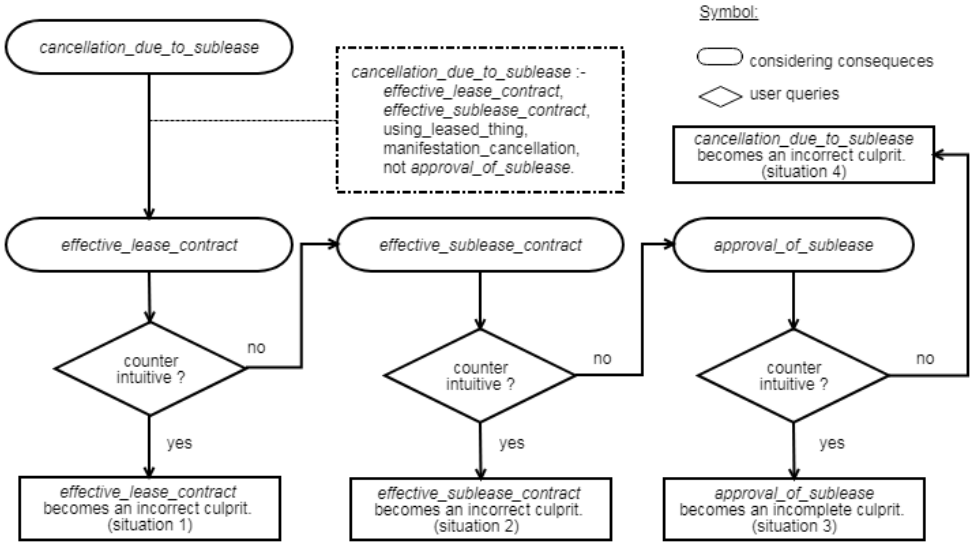

FIGURE 3.2: Flowchart of Culprit Detection Algorithm

If it is counterintuitive, it became an incomplete culprit because there is no rule supporting it but it is intended (situation 3). If it is not counterintuitive, there would be no counterintuitive consequences related to `cancellation_due_to_sublease`, hence `cancellation_due_to_sublease` becomes an incorrect culprit itself (situation 4). Intention of consequences in each situation can be summarized in Table 3.1 and below the table shows an example dialogue of Culprit Detection Algorithm with respect to situation 4.

TABLE 3.1: Summary of intended interpretation and culprit in each situation

Situation	Intention of consequences	Detected Culprit
1	`cancellation_due_to_sublease` $\notin IM$ `effective_lease_contract` $\notin IM$	`effective_lease_contract` (incorrect culprit)
2	`cancellation_due_to_sublease` $\notin IM$ `effective_lease_contract` $\in IM$ `effective_sublease_contract` $\notin IM$	`effective_sublease_contract` (incorrect culprit)
3	`cancellation_due_to_sublease` $\notin IM$ `effective_lease_contract` $\in IM$ `effective_sublease_contract` $\in IM$ `approval_of_sublease` $\in IM$	`approval_of_sublease` (incomplete culprit)
4	`cancellation_due_to_sublease` $\notin IM$ `effective_lease_contract` $\in IM$ `effective_sublease_contract` $\in IM$ `approval_of_sublease` $\notin IM$	`cancellation_due_to_sublease` (incorrect culprit)

```
Considering cancellation_due_to_sublease:- effective_lease_contract,
        effective_sublease_contract, using_leased_thing,
        manifestation_cancellation, not approval_of_sublease.
effective_lease_contract is valid with respect to the literal interpretation
Counterintuitive (yes/no) ? |: no.
effective_sublease_contract is valid with respect to the literal interpretation
Counterintuitive (yes/no) ? |: no.
using_leased_thing occurs in the case
manifestation_cancellation occurs in the case
approval_of_sublease is invalid with respect to the literal interpretation
Counterintuitive (yes/no) ? |: no.

Detect an incorrect culprit: cancellation_due_to_sublease.
```

3.4 Extensions of Culprit Detection Algorithm

3.4.1 First-Order Rule-base

Although Algorithm 3 is applicable with first-order rule-base, there is a question to consider for designing user interactions. That is how should the user input an intended interpretation in first-order rule-base. In this dissertation, we let a user list all intended instances for non-ground counterintuitive consequences.

Example 3.4. *Let us illustrate culprit detection in a first-order rule-base by representing Japanese Civil Code Article 612 as the following first-order rule-base. We begin variables with uppercase letters and constants with lowercase letters.*

```
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(Lessor,Lessee,Property):-
        agreement_of_lease_contract(Lessor,Lessee,Property),
        handover_to_lessee(Lessor,Lessee,Property).
effective_sublease_contract(Subleaser,Sublessee,Property):-
        agreement_of_sublease_contract(Subleaser,Sublessee,Property),
        handover_to_sublessee(Subleaser,Sublessee,Property).
approval_of_sublease(Lessor,Lessee) :-
        approval_before_the_day(Lessor,Lessee).
```

The exceptional case can be represented by this following first-order fact-base.

```
{agreement_of_lease_contract(plaintiff,defendant,room).
handover_to_lessee(plaintiff,defendant,room).
agreement_of_sublease_contract(defendant,son,room).
handover_to_sublessee(defendant,son,room).
using_leased_thing(son,room).
manifestation_cancellation(plaintiff,defendant).
father(defendant,son).
minor(son).}
```

Given `cancellation_due_to_sublease(plaintiff,defendant)` as an initial counterintuitive consequence. The example dialogue of culprit detection for first-order rule-bases is shown below. In this example, we show a culprit detection in situation 2 (see Figure 3.2) where `effective_sublease_contract(defendant,son,room)` becomes an incorrect culprit.

```
1  Considering cancellation_due_to_sublease(Lessor,Lessee) :-
2          effective_lease_contract(Lessor,Lessee,Property),
3          effective_sublease_contract(Lessee,Thirdparty,Property),
4          using_leased_thing(Thirdparty,Property),
5          manifestation_cancellation(Lessor,Lessee),
6          not approval_of_sublease(Lessor,Lessee).
7
8  effective_lease_contract(plaintiff,defendant, room) is valid
9  with respect to the literal interpretation
10 Counterintuitive (yes/no) ? |: no.
11 effective_sublease_contract(defendant,son,room) is valid
12 with respect to the literal interpretation,
13 Counterintuitive (yes/no) ? |: yes.
14 Is there a valid instance for
15 effective_sublease_contract(defendant,ThirdParty,room)
16 Which ThirdParty ? (answer no. if there is no valid instance) |: no.
17
18 Detect an incorrect culprit:
19     effective_sublease_contract(defendant,son,room).
```

To trace a culprit, the algorithm goes to the first rule, whose head is unifiable with `cancellation_due_to_sublease(plaintiff,defendant)`. It gets a substitution $\theta = \{\texttt{Lessor/plaintiff}, \texttt{Lessee/defendant}\}$. The algorithm asks the user to check whether `effective_contract(plaintiff,defendant,room)` is counterintuitive. Suppose the

user says it is not counterintuitive, it will be marked into the intended interpretation and we add a mapping `Property/room` into θ. Then, the algorithm would ask the user to check whether `effective_sublease_contract(defendant,Thirdparty,room)` is counterintuitive. Suppose the user says it is counterintuitive, In this time, the algorithm will ask for an instance of `Thirdparty`. Since the user identifies that there is no valid instance for `Thirdparty`. Hence, there is no valid instance for `effective_sublease_contract(defendant,Thirdparty,room)` and `effective_contract(defendant,son,room)` is an incorrect culprit since it is not intended but there is a rule (in lines 12-14) supporting it.

3.4.2 PROLEG Rule-base

Since the expressive power of PROLEG is the same as normal logic programs, we require to extend the definition of supporting rules to cover separated exceptions in PROLEG in the following. Then, we design Culprit Detection Algorithm for PROLEG rule-base to check separated exceptions in PROLEG as in Algorithm 4.

Definition 3.7 (Supporting rule in PROLEG)**.** Let M be a set of ground atoms, p be a ground atom, R be a ground rule, and T be a PROLEG program. We say R supports p with respect to M if there is a substitution θ such that $head(R)\theta = p$, $pos(R)\theta \subseteq M$, and there is no $exception(p, e) \in gr(T)$ such that $e \in M$.

Algorithm 4 Culprit Detection Algorithm for PROLEG rule-base

Given a fact-base FB, a PROLEG rule-base RB, a counterintuitive consequence atom p, a user as a membership query of the intended interpretation IM

procedure CULPRIT-DETECTION(p)
  Let M be the extension of $FB \cup RB$
  Find $R \in gr(RB)$ that supports p with respect to IM
  if $p \in IM$ and there is no such R **then** return p;
  else if $p \in IM$ **then**
    Find $q \in pos(R)$ such that $q \notin M$
    if there is such q **then** return CULPRIT-DETECTION(q)
    Find q **such that** $exception(p, q) \in gr(RB)$ **and** $q \in M$
    if there is such q **then** return CULPRIT-DETECTION(q)
  if $p \notin IM$ and there is such R **then** return p;
  else if $p \notin IM$ **then**
    Find $R' \in gr(RB)$ that supports p with respect to M
    Find $q \in pos(R')$ such that $q \notin IM$
    if there is such q **then** return CULPRIT-DETECTION(q)
    Find q **such that** $exception(p, q) \in gr(RB)$ **and** $q \in IM$
    if there is such q **then** return CULPRIT-DETECTION(q)

3.5 Summary

- A *counterintuitive consequence* is defined as a consequence in the symmetric difference between the literal interpretation and the intended interpretation, that is a consequence in the intended interpretation but not in the literal interpretation or a consequence in the literal interpretation but not in the intended interpretation.
- A *culprit* is defined as a consequence that is not intended but there is a rule supporting it with respect to the intended interpretation (we call this an *incorrect culprit*) or a consequence that is intended but there is no rule supporting it with respect to the intended interpretation (we call this an *incomplete culprit*).
- We present Culprit Detection Algorithm (Algorithm 3), which detects a culprit from a counterintuitive consequence by iteratively asking with the user whether related consequences are counterintuitive until the user cannot find further counterintuitive consequences related. Then, we get that the last found counterintuitive consequence is a culprit.
- To extend Culprit Detection Algorithm for first-order rule-base, we require to consider how to check the intended interpretation in first-order rule-base with the user. In this dissertation, we preliminary let a user list all intended instances for non-ground counterintuitive consequences.
- To extend Culprit Detection Algorithm for a PROLEG rule-base, we require to extend the definition of supporting rules for handling separated exceptions. Then, we present an extended Culprit Detection Algorithm for a PROLEG rule-base (Algorithm 4).

Chapter 4

Culprit Resolution Algorithm

In this chapter, we present

- Formalization of Counterintuitive Consequence Resolution Task (CCR Task)
- Culprit Resolution Algorithm
- Prototypical Case with Judgement
- Extensions of Culprit Resolution in First-order Rule-base and PROLEG Rule-base
- Generalizing Culprit Resolution using background theory

4.1 Overview

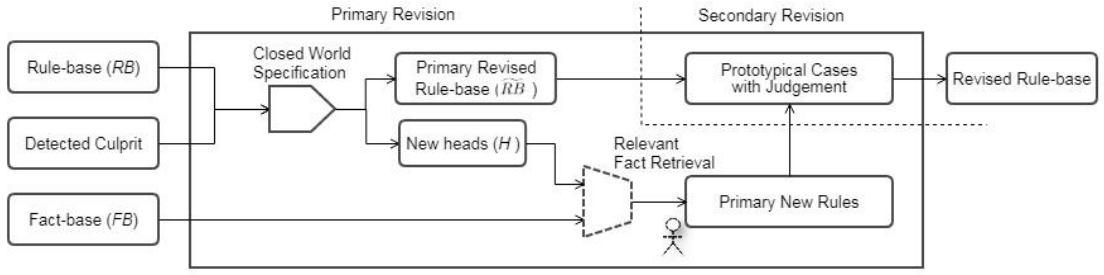

FIGURE 4.1: Culprit Resolution Work Flow
(Dash-lines represent the data or the process that a user involves)

Figure 4.1 shows the work flow of Culprit Resolution. Given a rule-base RB representing the statute, a fact-base FB representing a case, and a detected culprit, we aimed to develop Culprit Resolution Algorithm to resolve a culprit. However, since judges in civil law system consider legislator's intention when making change, we need to consider legislator's intention from the original rule-base in a form of cases so we present a new structure called *prototypical cases with judgement*, reproducible from the rule-base. Then, Culprit Resolution Algorithm works on the following steps.

1. The algorithm primarily revises the rule-based based on Closed World Specification, and asks a user to select facts from the fact-base that are relevant to the head of the new rule

2. The algorithm reproduces *prototypical cases with judgement* from the primary revised rule-base and the primary new rules, and secondarily revises rule-base to preserve the judgement of prototypical cases as well as to resolve the culprit.

4.2 Counterintuitive Consequence Resolution (CCR) Task

When judges apply interpretation of statutes in a particular case and it leads to counterintuitive consequences, the judges may revise interpretation of statutes. We call such a case an exceptional case. In such scenarios, judges would introduce a new factual concept in the exceptional case. Then, judges use the introduced factual concept to revise the statute so that the counterintuitive consequence is resolved. To formalize judicial legal change, we firstly define agreement and disagreement as follows.

Definition 4.1 (Agreement and Disagreement)**.** Let FB_1, FB_2 be fact-bases, RB_1, RB_2 be rule-bases, and p be a ground atom. We say $FB_1 \cup RB_1$ agrees with $FB_2 \cup RB_2$ on

p if the following condition is satisfied. Otherwise, we say $FB_1 \cup RB_1$ disagrees with $FB_2 \cup RB_2$ on p.

- $FB_1 \cup RB_1 \vdash p$ and $FB_2 \cup RB_2 \vdash p$ or
- $FB_1 \cup RB_1 \nvdash p$ and $FB_2 \cup RB_2 \nvdash p$

Formalization of the task goes as follows. Firstly, we have a ground atom p representing a counterintuitive consequence and FB_e representing an exceptional case, which contains at least one fact atom not occurring in an original rule-base RB ($f(FB_e) \not\subseteq f(RB)$ where $f(T)$ is a set of all ground fact atoms that can be substitutable to some fact atoms in a program T, see Chapter 2 for details). Then, we revise an original rule-base RB to a new rule-base RB'. RB' is a correct revision of RB' with respect to FB_e and p if it can resolve the counterintuitive consequence p. The task is formally described as follows.

Definition 4.2 (Counterintuitive Consequence Resolution (CCR) Task)**.** A counterintuitive consequence resolution (CCR) task is a tuple $\langle RB, FB_e, p\rangle$ where RB is a rule-base representing statutes, FB_e is a fact-base representing an exceptional case ($f(FB_e) \not\subseteq f(RB)$), and p is a considered counterintuitive consequence. A rule-base RB' is a *resolution* for the task $\langle RB, FB_e, p\rangle$ if $FB_e \cup RB'$ disagrees with $FB_e \cup RB$ on p (hence, it implies that the considered counterintuitive consequence is resolved).

Example 4.1. *Let a fact-domain $\mathcal{F} = \{a, b, c, d\}$, and a rule-base $RB_1 = \{p \leftarrow q.\ q \leftarrow a.\ q \leftarrow b.\}$. Suppose p is a counterintuitive consequence from applying RB_1 in an exception case represented by $FB_1 = \{a.\ b.\ c.\ d.\}$. Then, $RB_2 = \{p \leftarrow q.\ q \leftarrow a, not\ s.\ q \leftarrow b, not\ t.\ s \leftarrow b, c.\ t \leftarrow d.\}$ and $RB_3 = \{p \leftarrow q, not\ r.\ q \leftarrow a.\ q \leftarrow b.\ r \leftarrow c.\}$ are both resolutions to the CCR task $\langle RB_1, FB_1, p\rangle$ since $FB_1 \cup RB_1$ disagrees with $FB_1 \cup RB_2$ and $FB_1 \cup RB_3$ on p which means p is resolved in both revisions.*

4.3 Culprit Resolution Algorithm

In this section, we describe Culprit Resolution Algorithm, which can be divided into primary revision and secondary revision described as follows.

4.3.1 Primary Revision

We can see that a resolution is not always unique, as illustrated in Example 4.1. One reason is that there are possibly many rules to put exceptions (*not r* in Example 4.1).

Now, we introduce the first part of the primary revision to specify which rule should an exception be put in. For an incorrect culprit, we put new exceptions in all supporting rules of the culprit in the same manner of Closed World Specification [56] then the new exceptions become the heads of the new rules. For an incomplete culprit, we just skip the step of introducing new exceptions since the incomplete culprit itself becomes the head for the new rule. Following this instruction as illustrated in Algorithm 5, we can reduce a CCR task to a task finding which facts are relevant to each new head.

Algorithm 5 Closed World Specification in Primary Revision

Given A CCR task $\langle RB, FB_e, p\rangle$ For ease of explanation, we assume that all variables in the rule must occur in the positive body of the rule.

Let $\widetilde{RB} = RB$ and $H = \emptyset$
for all culprits p_c detected from p using Algorithm 3 **do**
  if p_c is an incorrect culprit **then** ▷ Applying Closed World Specification
    for all rules R that support p_c with respect to the answer set of $FB_e \cup RB$ **do**
      for all substitutions θ that make R supports p_c as such **do**
        Let $V_1, \ldots, V_n$ be all variables in R
        Let q be a new rule predicate
        Let p_e be $q(V_1, \ldots, V_n)$
        Add $p_e\theta$ to H
        Add *not* p_e to the body of R in $\widetilde{RB}$
  else ▷ when p_c is an incomplete culprit
    Add p_c to H

After we have a primary revised rule-base ($\widetilde{RB}$) and a set of new heads (H), the second part of the primary revision asks the user which facts are relevant to each new head. We restrict that each new rule must contain an extra fact, a new fact that never depends on the considered rule predicate. Hence, we define the *culprit resolution* to CCR task as follows.

Definition 4.3 (Culprit Resolution to CCR task)**.** Given a CCR task $\langle RB, FB_e, p\rangle$, $\widetilde{RB}, H$ obtained by Algorithm 5 where $\widetilde{RB}$ is a primary revised rule-base and $H = \{h_1, \ldots, h_n\}$ is a set of new heads. If we have a set of Horn clauses $RB_H = \{h_1 \leftarrow B_1., \ \ldots, \ h_n \leftarrow B_n.\}$ such that there is a substitution θ such that $B_i\theta \subseteq f(FB_e)$ (known as Plotkin's subsumption [49]) for all $1 \leq i \leq n$, We call a rule-base $RB' = \widetilde{RB} \cup RB_H$ a *culprit resolution* to the CCR task $\langle RB, FB_e, p\rangle$. A culprit resolution is *restricted* if each B_i has a fact predicate that never depends on p.

Example 4.2. *Continuing from Example 4.1, Suppose q is an incorrect culprit and we put not s and not t in supporting rules of q, then $\widetilde{RB}_1 = \{p \leftarrow q.\ q \leftarrow a, not\ s.\ q \leftarrow b, not\ t.\}$ and $H = \{s, t\}$. We get that $RB_2 = \{p \leftarrow q.\ q \leftarrow a, not\ s.\ q \leftarrow b, not\ t.\ s \leftarrow b, c.\ t \leftarrow d.\}$ in Example 4.1 is a restricted culprit resolution to the CCR task $\langle RB_1, FB_1, p\rangle$ since it includes new Horn clauses $s \leftarrow b, c.$ and $t \leftarrow d.$ that satisfy the conditions in Definition 4.3 (as illustrated in Figure 4.2).*

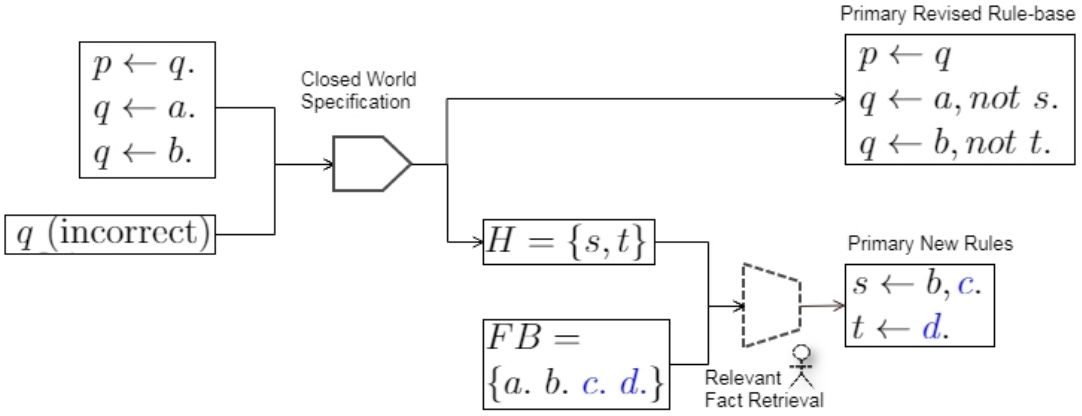

FIGURE 4.2: Example of Primary Revision

4.3.2 Prototypical Case with Judgement (PCJs)

In the previous step, we reduce a CCR task to a task of finding which facts are relevant to each new head. In this step, we consider that finding which facts are relevant to each new head is equivalent to finding which sub-cases make the present case exceptional.

Firstly, we assume that a rule-base representing statutes is built from a collection of *critical sets* [61]. In this dissertation, we represent a collection of critical sets by a critical case-base in AA-CBR [67] (see Chapter 2 for details). Given a fact-domain $\mathcal{F}$, let $CB_1, \ldots, CB_n$ be some effective enumeration of a class of critical case-bases. Given a rule-base construction THEORY-CONS from a case-base to a rule-base described in Chapter 2 and an ultimate goal p_0. We have an enumeration $RB_1 = \text{THEORY-CONS}(CB_1, p_0)$, $\ldots, RB_m = \text{THEORY-CONS}(CB_m, p_0)$. Let an original rule-base RB_i be a member of the enumeration $(1 \leq i \leq m)$ such that $RB_i = \text{THEORY-CONS}(CB_i, p_0)$, we would like to design Culprit Resolution Algorithm so that given a fact-base FB_e representing an exceptional case with $CB_i \models (f(FB_e), J_e)$ and representative sub-cases with judgement $(RC_1, \bar{J}_e), \ldots, (RC_n, \bar{J}_e)$ ($RC_k \subseteq f(FB_e)$ and $RC_k \not\subseteq f(RB)$ for every $1 \leq k \leq n$) given by the user, the output of Culprit Resolution Algorithm would be $RB_j = \text{THEORY-CONS}(CB_j, p_0)$ $(1 \leq j \leq m)$ where $CB_j = CB_i \cup \{(RC_1, \bar{J}_e), \ldots, (RC_n, \bar{J}_e)\}$.

Firstly, we adopt *prototypical cases with judgement* (PCJs) as cases that associate sub rule-bases. Intuitively, PCJs reflect cases in the legislators' intention when they drafted the statutes. PCJs are designated as an inverse of THEORY-CONS so that we can extract PCJs from a rule-base RB by using PCJ-DAG defined as follows.

Definition 4.4 (Prototypical case with judgement directed acyclic graph (PCJ-DAG))**.** Let RB be a non-recursive and stratified rule-base. A *prototypical case with judgement directed acyclic graph* (PCJ-DAG) G of RB is a directed acyclic graph which,

- nodes are tuples $\langle p, U, C, J \rangle$ where
  - p is a special ground rule atom $\top$ or any other ground rule atom in RB
  - U is a special rule with $\top$ in its head if $p = \top$. Otherwise, $U \in RB_m$ where $RB_1, \ldots, RB_m$ is a complete unfolding sequence and $head(U) = p$ (if $pos(U)$ contains free variables, substitute them with Skolem constants so that U is grounded)
  - C is a set of ground fact atoms
  - J is either '+' or '−'
- for every node $\langle p, U, C, J \rangle$ in G, the children of the node are all nodes in the form $\langle q, U_q, C_q, J_q \rangle$ such that $q \in neg(U)$, $C_q = C \cup pos(U_q)$, and $J_q = \bar{J}$

Definition 4.5 (Prototypical Case with Judgement)**.** Given a non-recursive and stratified rule-base RB and a ground rule-atom p_0, the set of *prototypical cases with judgement* (PCJs) from RB with respect to p_0 is the set of all pairs (C, J) in all nodes $\langle p, U, C, J \rangle$ of a PCJ-DAG in which the root node is $\langle \top, \top \leftarrow not\ p_0., \emptyset, \text{'}-\text{'} \rangle$.

Example 4.3. *Continuing from the previous example, let* $RB_2 = \{p \leftarrow q.\ q \leftarrow a, not\ s.\ q \leftarrow b, not\ t.\ s \leftarrow b, c.\ t \leftarrow d.\}$ *and* p *is a considered ground rule atom. The set of prototypical cases with judgement from* RB_2 *with respect to* p*, built from a PCJ-DAG illustrated in Figure 4.3, is the set* $\{(\emptyset, \text{'}-\text{'}), (\{a\}, \text{'}+\text{'}), (\{b\}, \text{'}+\text{'}), (\{a, b, c\}, \text{'}-\text{'}), (\{b, d\}, \text{'}-\text{'})\}$.

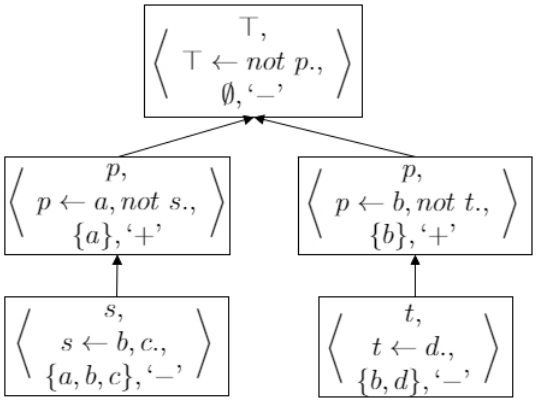

FIGURE 4.3: Example of PCJ-DAG

4.3.3 Secondary Revision

Since primary new rules sometimes fail to reflect some attacks in prototypical cases with judgement, Secondary revisions fixes the primary revision to reflect attacks in the prototypical cases with judgement correctly as in Algorithm 6. Proofs of correctness and completeness are shown immediately after the algorithm.

Algorithm 6 Secondary Revision in Culprit Resolution Algorithm

Given A CCR task $\langle RB, FB_e, p\rangle$ where RB is a propositional rule-base[1], a primary revised rule-base $\widetilde{RB}$ and a set of primary new rules RB_H

Let $RB' = \widetilde{RB} \cup RB_H$
Let G be a PCJ-DAG of $\widetilde{RB}$ in which the root node is $\langle \top, \top \leftarrow not\ p_0., \emptyset, \text{`}-\text{'}\rangle$
Let PCJ be the set of PCJs from $\widetilde{RB}$ with respect to p
Let PCJ' be the set of PCJs from RB' with respect to p
Let $J_e =$ '+' if $FB_e \cup RB \vdash p$ otherwise $J_e =$ '$-$'
for all $(RC, \bar{J}_e) \in PCJ' \setminus PCJ$ **do**
  for all $(RC, \bar{J}_e) \rightsquigarrow (C, J_e)$ in the argumentation framework corresponding to PCJ' **do**
    Let $\langle p_n, U, C, J_e\rangle$ be a node in G that (C, J_e) is extracted from
    for all rules R that support p_n with respect to the answer set of $RC \cup RB$ **do**
      ▷ If R supports p_n then it misses the corresponding exceptions to the attack
      Let h be a new rule proposition (a rule predicate with zero arguments)
      Add *not* h to the body of R
      Add a new rule $h \leftarrow RC \setminus C.$ to RB'

Theorem 4.6 (Correctness of Culprit Resolution Algorithm)**.** *Given a CCR task* $\langle RB, FB_e, p\rangle$ *and representative sub-cases with judgement* $RepCJ = (RC_1, \bar{J}_e)$, $\ldots, (RC_n, \bar{J}_e)$ *where* $RC_k \subseteq f(FB_e)$ *and* $RC_k \not\subseteq f(RB)$ *for all* $1 \leq k \leq n$ *and* J_e *is* '+' *if* $FB_e \cup RB \vdash p$ *and* '$-$' *otherwise. Algorithm 6 always returns a restricted culprit resolution for the task* $\langle RB, FB_e, p\rangle$*.*

Proof. Since $RepCJ = (RC_1, \bar{J}_e), \ldots, (RC_n, \bar{J}_e)$ where $RC_k \subseteq f(FB_e)$ for all $1 \leq k \leq n$ and for every new Horn clause $h \leftarrow B.$, there is $(RC, \bar{J}_e) \in RepCJ$ such that $B \subseteq RC$ hence $B \subseteq f(FB_e)$. Therefore, Algorithm 6 always returns a culprit resolution to the task $\langle RB, FB_e, p\rangle$ by the Closed World Specification Algorithm. □

Theorem 4.7 (Completeness of Culprit Resolution Algorithm)**.** *Given a CCR task* $\langle RB, FB_e, p\rangle$ *such that there is a critical case-base* CB *such that* THEORY-CONS $(CB, p) = RB$*, and* $CB \models (f(FB_e), J_e)$*. If there is a culprit resolution* RB' *of the task* $\langle RB, FB_e, p\rangle$ *such that there is a critical case-base* $CB' = CB \cup RepCJ$ *which* THEORY-CONS(CB', p) *where* $RepCJ = \{(RC_1, \bar{J}_e), \ldots, (RC_n, \bar{J}_e)\}$ *and* $RC_k \subseteq f(FB_e)$ *and* $RC_k \not\subseteq f(RB)$ *for every* $1 \leq k \leq n$*, we can get* RB' *in finite steps by using Algorithm 6.*

Proof. Let a set of heads of new rules $H = \{h_1, \ldots, h_m\}$. Since a user require to select sub fact-base $RF_{m,1} \ldots RF_{m,n} \subseteq FB_e$ relevant to h_i for each $1 \leq i \leq m$ and FB_e is finite, we can have a finite enumeration of $RF_{1,1} \ldots RF_{m,n}$ and a finite enumeration of a

class of representative sub-cases with judgement. For every representative sub-cases with judgement, new rules are introduced corresponding to the definition of THEORY-CONS. Hence, we can get RB' in finite steps. □

Example 4.4. *Continuing from the previous example, let* $RB_2 = \{p \leftarrow q.\ q \leftarrow a, not\ s.\ q \leftarrow b, not\ t.\ s \leftarrow b, c.\ t \leftarrow d.\}$. *We get that there is* $(\{a, b, c\}, \text{`}-\text{'}) \rightsquigarrow (\{b\}, \text{`}+\text{'})$ *that has no rule in* RB_2 *corresponding to. Therefore, we add not* t_2 *and* $t_2 \leftarrow a, c.$ *as illustrated in Figure 4.4. The result from the secondary revision is* $RB_4 = \{p \leftarrow q.\ q \leftarrow a, not\ s.\ q \leftarrow b, not\ t, not\ t_2.\ s \leftarrow b, c.\ t \leftarrow d.\ t_2 \leftarrow a, c.\}$

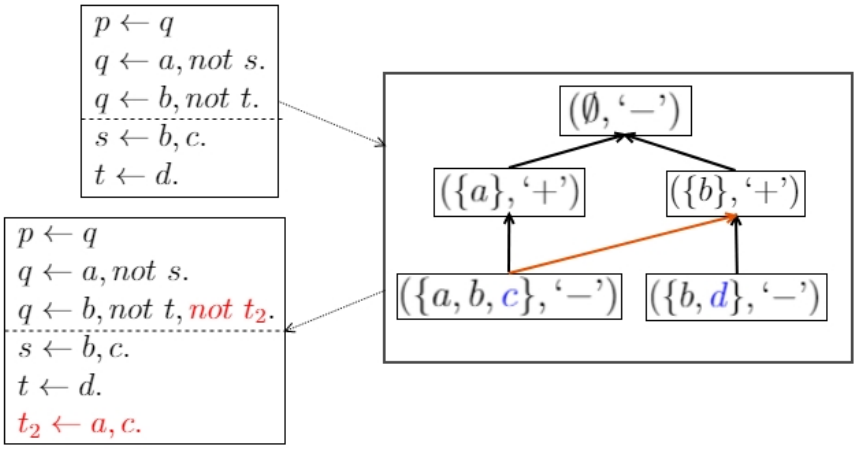

FIGURE 4.4: Example of Secondary Revision

Let us illustrate Culprit Resolution Algorithm using the example case related to the Japanese Civil Code Article 612 (continuing from Example 3.3)

Example 4.5. *The example dialogue of culprit resolution is shown as follows.*

```
13 Detect an incorrect culprit: cancellation_due_to_sublease.
14
15 Introduce a new exception to a support rule of a culprit
16 cancellation_due_to_sublease:- effective_lease_contract,
17         effective_sublease_contract, using_leased_thing,
18         manifestation_cancellation, not approval_of_sublease,
19         not new_exception.
20
21 A prototypical case with judgement associated with new_exception:
22 ({agreement_of_lease_contract,handover_to_lessee,
23 agreement_of_sublease_contract,handover_to_sublessee,
24 using_leased_thing,manifestation_cancellation},'+')
25
```

```
26 Listing possibly relevant facts...
27 1: non_destruction_of_confidence
28 2: sublessee_is_a_minor
29 ...
30
31 Please specify facts relevant to new_exception
32 by a list of indices (e.g. [1,3,5]).
33 If you would like to input new facts, please input newfact([fact1,fact2,...]).
34 |: [1].
35
36 New representative case(s) with judgement:
37 ({agreement_of_lease_contract,handover_to_lessee,
38 agreement_of_sublease_contract,handover_to_sublessee,
39 using_leased_thing,manifestation_cancellation,
40 non_destruction_of_confidence},'-')
41
42 Introduce a new rule: new_exception:- non_destruction_of_confidence.
```

In the primary revision, since the detected culprit (`cancellation_due_to_sublease`) is an incorrect culprit, we introduce a new exception (represented as `new_exception`) for a supporting rule of the culprit with respect to the answer set and `new_exception` becomes a head of a new rule. Then, the algorithm asks the user which facts are relevant to the new head. Since the user specify `non_destruction_of_confidence` as the fact relevant to `new_exception`, primary revision produces a new representative case with a judgement '−' as shown in lines 36-39, which refers to an intention that cancellation due to sublease shall not be valid in cases covering all facts in the representative case. Since there is no fixation needed in the primary revision, the algorithm just introduces a new rule `new_exception:- non_destruction_of_confidence.` as specified.

4.4 Extensions of Culprit Resolution Algorithm

4.4.1 First-order Rule-base

It turns out that Algorithm 6 is not fully applicable with first-order rule-base since there are questions about levels of abstraction (or *de-refinement* in [38]) and the definitions of case-bases in first-order rule-base. Therefore, we extend AA-CBR to support a first-order case-base by introducing an abstraction to deal with a first-order case-base as follows.

Definition 4.8 (Abstraction)**.** Let B be an atom, a set of atoms, a tuple of atoms, or a rule. An abstraction of B, denoted by $abs(B)$, is obtained by constructing an injective

mapping $\{c_1/v_1, \ldots, c_n/v_n\}$ for all constants in B and replacing every constant c_i in B by a new distinct variable v_i $(1 \leq i \leq n)$. For instance, $abs(p(a) \leftarrow q(a), r(b).) = p(X) \leftarrow q(X), r(Y)$.

Definition 4.9 (Extended Subsumption)**.** Let C and D be a set of atoms. As we say C subsumes D ($C \preceq D$) if there is a substitution θ such that $C\theta \subseteq D$, we denote such $C\theta$ by $[C]_D$. We say C is substitutable to D ($C \sqsupseteq D$) if there is a substitution θ such that $C\theta = D$ (Please note that sometimes $C \sqsupseteq D$ but $D \not\sqsupseteq C$) . C non-substitutably subsumes D ($C \prec D$) if $C \preceq D$ but $C \not\sqsupseteq D$.

Let CB be a finite set of first-order cases with judgement called a first-order case-base. We assume that a first-order case-base is consistent, that is no pair $(X, J_x), (Y, J_y) \in CB$ such that $abs(X) \sqsupseteq Y$ but $J_x \neq J_y$. We say a case with judgement $(N, \bar{J})$ is a trumping case with judgement to (C, J) w.r.t. CB when $abs(C) \prec N$, and no other $(N', \bar{J}) \in CB$ such that $abs(C) \prec N'$ and $abs(N') \prec N$.

We also extend how AA-CBR predicts a judgement for a case N (called an analogous prediction) by constructing an analogous case-base of CB w.r.t. N, denoted by $CB_N = \{([abs(C)]_N, J) | (C, J) \in CB \text{ and } abs(C) \preceq N\}$. An analogously predicted judgement of N is J_0 if $(\emptyset, J_0)$ is in the ground extension of an argumentation framework corresponding to CB_N. Otherwise, an analogously predicted judgement of N is $\bar{J_0}$.

A theory construction for a first-order case-base is revised as follows.

Definition 4.10 (Theory-cons in first-order)**.** Given a ground rule atom g_0, a first-order critical case-base CB with a default judgement '$-$', and $(CB, attacks)$ as its corresponding argumentation framework, THEORY-CONS-FO(CB, g_0) is defined by

- Given two atoms A_1, A_2 and three sets of atoms X, Y, Z, let
  - $pos_t(X, Y) = X \setminus [abs(Y)]_X$.
  - $head_t(X, Y)$ be a ground atom with a new reified rule predicate $p_{X,Y}$ and arguments from all arguments occurring in $pos_t(X, Y)$.
  - $var_t(A_1, Y, Z) = A_2$ means A_2 is a variant of A_1 in the same manner of Y is a variant in Z, in other words, there is a substitution θ such that $abs(\langle A_1, Y\rangle)\theta = (\langle A_2, [abs(Y)]_Z\rangle)$ ($\langle\cdot\rangle$ in this sentence means a tuple).
  - $neg_t(X, Y) = \{not\ var_t(head_t(Z, X), pos_t(X, Y), Z) | (X, J) \rightsquigarrow (Y, \bar{J}) \in attack$ and $(Z, \bar{J}) \rightsquigarrow (X, J) \in attack\}$.
- Then, THEORY-CONS-FO$(CB, g_0) =$
  $\{abs(g_0 : -pos_t(X, \emptyset) \cup neg_t(X, \emptyset)) | (X, \text{`+'}) \rightsquigarrow (\emptyset, \text{`}-\text{'}) \in attacks\} \cup$
  $\{abs(head(X, Y) : -pos_t(X, Y) \cup neg_t(X, Y)) | (X, J) \rightsquigarrow (Y, \bar{J}) \in attacks \wedge Y \neq \emptyset\}$.

Theorem 4.11. *Let CB be a first-order case-base and g_0 be a ground rule atom. T = THEORY-CONS-FO(CB, g_0) is an analogous theory of CB w.r.t. g_0, namely an analogously predicted judgement of a case N is '+' if and only if $FB \cup T \vdash g$ for some $abs(g_0) \sqsupseteq g$ and $f(FB) = N$.*

Proof. Let CB_N be an analogous case-base of CB with respect to N and $T_N = \{R \in T | pos(R) \subseteq N\}$. We get that there is a ground rule atom g such that $abs(g_0) \sqsupseteq g$ and T_N = THEORY-CONS(CB_N, g). Hence, THEORY-CONS-FO(CB, g_0) is an analogous theory of CB w.r.t. g_0. □

Therefore, we extend Culprit Resolution Algorithm for resolving culprits in a first-order rule-base as Algorithm 7.

Algorithm 7 Culprit Resolution Algorithm for First-order Rule-bases

Given A CCR task $\langle RB, FB_e, p\rangle$ where RB is a (first-order) rule-base, a primary revised rule-base $\widetilde{RB}$ and a set of primary new rules RB_H (Note that RB_H are all grounded)

Let $RB'_H = \{abs(R) | R \in RB_H\}$
Let $RB' = \widetilde{RB} \cup RB'_H$
Let G be a PCJ-DAG of $\widetilde{RB}$ in which the root node is $\langle \top, \top \leftarrow not\ p_0., \emptyset, \text{'}-\text{'}\rangle$
Let PCJ be the set of PCJs from $\widetilde{RB}$ with respect to p
Let PCJ' be the set of PCJs from RB' with respect to p
Let J_e = '+' if $FB_e \cup RB \vdash p$ otherwise J_e = '−'
for all $(RC, \bar{J}_e) \in PCJ' \setminus PCJ$ **do**
  for all $(RC, \bar{J}_e) \rightsquigarrow (C, J_e)$ in the argumentation framework corresponding to PCJ' **do**
    Let $\langle p_n, U, C, J_e\rangle$ be a node in G that (C, J_e) is extracted from
    for all rules R that support p_n with respect to the answer set of $RC \cup RB$ **do**
      ▷ If R supports p_n then it misses the corresponding exceptions to the attack
      for all substitutions θ that make R supports p_n as such **do**
        Let $V_1, \ldots, V_n$ be all variables in R
        Let q be a new rule predicate
        Let p_e be $q(V_1, \ldots, V_n)$
        Add $p_e\theta$ to H
        Add *not* p_e to the body of R in RB'
        Add a new rule $abs(p_e\theta \leftarrow pos_t(RC, C))$. to RB'

Example 4.6. *Continuing from Example 3.4, the example dialogue of culprit resolution is shown as follows.*

```
12 Detect an incorrect culprit: effective_sublease_contract(defendant,son,room).
13
14 Introduce a new exception to a support rule of a culprit
15 effective_sublease_contract(Subleaser,Sublessee,Property):-
16         agreement_of_sublease_contract(Subleaser,Sublessee,Property),
17         handover_to_sublessee(Subleaser,Sublessee,Property),
```

```
18          not new_exception(Subleaser,Sublessee,Property).
19
20  A prototypical case with judgement associated with new_exception
21  (analogous to the fact-base):
22  ({agreement_of_lease_contract(plaintiff,defendant,room),
23      handover_to_lessee(plaintiff,defendant,room),
24      agreement_of_sublease_contract(defendant,son,room),
25      handover_to_sublessee(defendant,son,room),
26      using_leased_thing(son,room),
27      manifestation_cancellation(plaintiff,defendant)},'+')
28
29  Listing possibly relevant facts...
30  1: parent(defendant,son)
31  2: minor(son)
32  ...
33
34  Please specify facts relevant to new_exception(plaintiff,defendant,son,room)
35  by a list of indices (e.g. [1,3,5]).
36  If you would like to input new facts, please input newfact([fact1,fact2,...]).
37  |: newfact([non_destruction_of_confidence(plaintiff,defendant)]).
38
39  A representative sub-case with judgement:
40  ({agreement_of_lease_contract(plaintiff,defendant,room),
41      handover_to_lessee(plaintiff,defendant,room),
42      agreement_of_sublease_contract(defendant,son,room),
43      handover_to_sublessee(defendant,son,room),
44      using_leased_thing(son,room),
45      manifestation_cancellation(plaintiff,defendant),
46      non_destruction_of_confidence(plaintiff,defendant)},'-')
47
48  Introduce a new rule:
49  new_exception(Subleaser,Sublessee,Property):-
50      non_destruction_of_confidence(Subleaser,Sublessee).
```

Since the detected culprit is an incorrect culprit, we introduce a new exception with a new predicate all variables in the supporting rule as argument (`new_exception(Subleaser, Sublessee,Property)`). Then, we also simulate prototypical cases with judgement that associates with the new exception. The prototypical case is simulated analogously to the fact-base. After that, we show facts that are not covered in the prototypical case and let the user select facts or input new facts that are relevant to the exception. In this example dialogue, we assume the user give a new fact `non_destruction_of_confidence(plaintiff,defendant)`. The algorithm implies a new ground rule `new_exception(defendant,son,room):- non_destruction_of_confidence(plaintiff,defendant)`.

According to Closed World Specification, the head of the new rule is instantiated from `new_exception(Subleaser,Sublessee,Property)`θ where θ is the m.g.u. used for instantiating the supporting rule hence $\theta = \{$`Subleaser/plaintiff`, `Sublessee/son`, `Property/room`$\}$ is applied here (see Section 3.4.1 for the details). Therefore, the algorithm introduces `new_exception(Subleaser,Sublessee,Property):- non_destruction_of_confidence(Subleaser,Sublessee).`, which is an abstraction of the new rule previously mentioned (the algorithm may use different variables for the abstract of the rule but we use such variables for the sake of readability).

4.4.2 PROLEG Rule-base

Since PROLEG separates exceptions from rules, a PROLEG exception would rebut all rules whose heads are unifiable with the rebuttal consequence. Therefore, when we apply Closed World Specification, we may translate the introduction of an undercutting exception e in the rule of the form $h \leftarrow b_1, \ldots, b_m, not\ e.$ into the following Horn clauses and the PROLEG exception $\{h \leftarrow c.\ c \leftarrow b_1, \ldots, b_m.\ exception(c, e).\}$ where c is a new atom with a new rule predicate and with the same arguments as h [64]. However, if there is only one Horn clause whose head is unifiable with the rebuttal atom, then we can introduce $exception(h, e)$ without necessity to translate the rule.

4.5 Generalizing Culprit Resolution

To make the resolution not too specific to the present case, we generalize the resolutions in the relevant fact retrieval. Actually, we generalize the resolution implicitly when the user selects which facts relevant to each new head since the new rule needs not to be applied specifically in the case larger or equal to the present case. We call this *implicit generalization*. However, the body of the new rules after implicit generalization would contain only fact atoms. In this section, we present the *explicit generalization* with background theory. Background theory is a separated rule-base representing some general background knowledge as well as legal knowledge. Explicit generalization would introduce rule atoms into the body of the new rules.

We consider the explicit generalization by Inverse Resolution [33] . However, there are some concerns about Inverse Resolution in normal logic programs [85]. The first concern is that the result of Inverse Resolution is not generally consistent with the original program under the stable model semantics. We can only guarantee when a program is locally stratified and the dependencies in the program are preserved. Since a rule-base representing statutes is locally stratified, we have no problem with the first concern. For

the second concern, one practical way is to preserve at least one predicate that never depends on the considered rule predicate, to guarantee that we preserve dependencies in the rule-base. This corresponds to the practice in the law to identify extra facts that distinguishes the present exceptional case with precedent cases.

Inverse Resolution consists of two main operators, namely V-operator and W-operator. In this section, we will discuss the application of each operator respectively for resolving culprits.

4.5.1 Generalizing Culprit Resolution Using V-Operator

Given a resolvent clause (usually a primary new rule) and one parent clause (usually a rule in background theory), V-operator builds any other parent clause that can produce the resolvent clause via resolution with the given parent clause. In this dissertation, we induce the parent clause in V-operator by using folding.

Example 4.7. *Continuing from the previous example (Example 4.6), suppose a user select* `parent(plaintiff,defendant)` *as a fact relevant to the new exception and we have a rule in the background theory stating "A parent is a kind of relatives" represented as* `relatives(X,Y) :- parent(X,Y).`. *The example dialogue regarding this generalization is shown as follows (Lines 1-28 are same as those in the example dialogue of Example 4.6).*

```
29 Listing possibly relevant facts...
30 1: parent(defendant,son)
31 2: minor(son)
32 ...
33
34 Please specify facts relevant to new_exception(plaintiff,defendant,son,room)
35 by a list of indices (e.g. [1,3,5]).
36 If you would like to input new facts, please input newfact([fact1,fact2,...]).
37 |: [1].
38
39 Found general rule(s):
40 1: new_exception(Subleaser,Sublessee,Property):-
41     parent(Subleaser,Sublessee).
42 2: new_exception(Subleaser,Sublessee,Property):-
43     relatives(Subleaser,Sublessee).
44
45 Which one would you like to introduce ? |: 2.
```

From the result of the culprit resolution algorithm, we get the rule $R =$ `new_exception(Subleaser,Sublessee,Property):-parent(Subleaser,Sublessee).` We can generalize this rule by folding it with a rule $R_1 =$ `relatives(X,Y) :- parent(X,Y).` from the background theory although there is a rule `relatives(X,Y) :- mother(X,Y).` whose head is unifiable. From the folding, we get a new rule R_2 from R and R_1 as illustrated in Figure 4.5. The induced rule R_2 implies that a new exception should be derived if the subleaser is the relatives of the sublessee.

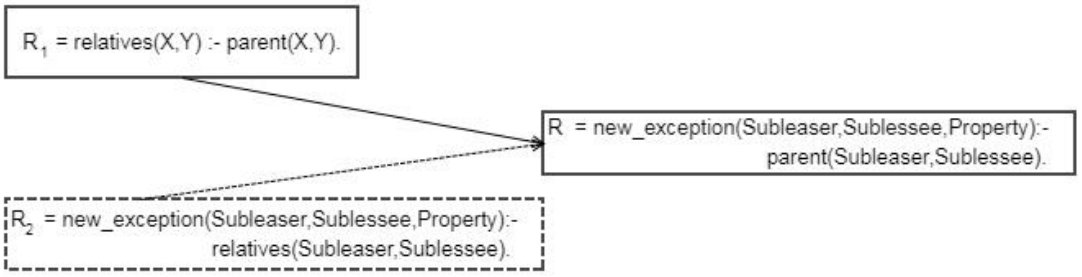

FIGURE 4.5: Generalizing culprit resolution with V-operator

4.5.2 Generalizing Culprit Resolution Using W-Operator

Given two resolvent clauses (usually one old rule and one new rule), W-operator builds three parent clauses that can produce the resolvent clauses via resolution. W-operator is useful for grouping similar concepts into a new concept. To induce three parent clauses in W-operator, we first construct a new atom then use such atoms for two-times folding since W-operator can be determined as two V-operators combined back-to-back.

Example 4.8. *Continuing from the previous example, suppose we have a new exceptional case that the new exception should be derived if the lessee is a colleague of the sublessee. The new exceptional case can be represented as follows.*

```
agreement_of_lease_contract(plaintiff,defendant,room).
handover_to_lessee(plaintiff,defendant,room).
agreement_of_sublease_contract(defendant,colleague,room).
handover_to_sublessee(defendant,colleague,room).
using_leased_thing(colleague,room).
manifestation_cancellation(plaintiff,defendant).
colleague(defendant,colleague).
```

With W-operator, the concept of *relatives* in the previous example and the concept of *colleague* in this example may be grouped into one new concept. Let's say the new

concept is *acquaintance*. Then, the rule of the new exception is generalized to cover cases such that the subleaser is an acquaintance of the sublessee. The example dialogue regarding this generalization is shown as follows.

```
1  Considering cancellation_due_to_sublease(plaintiff,defendant):-
2      effective_contract(plaintiff,defendant,Property),
3      effective_contract(defendant,Thirdparty,Property),
4      using_leased_thing(Thirdparty,Property),
5      manifestation_cancellation(plaintiff,defendant)),
6      not approval_of_sublease(plaintiff,defendant),
7      not new_exception(plaintiff,defendant,Thirdparty,Property).
8
9  effective_contract(plaintiff,defendant,room) is valid
10 with respect to the literal interpretation,
11 Counterintuitive (yes/no) ? |: no.
12 effective_sublease_contract(defendant,colleague,room) is valid
13 with respect to the literal interpretation,
14 Counterintuitive (yes/no) ? |: yes.
15 Is there a valid instance for
16 effective_sublease_contract(defendant,ThirdParty,room)
17 Which ThirdParty ? (answer no. if there is no valid instance) |: no.
18
19 Considering
20 effective_sublease_contract(defendant,colleague,room):-
21         agreement_of_sublease_contract(defendant,colleague,room),
22         handover_to_sublessee(defendant,colleague,room),
23         not new_exception(defendant,colleague,room).
24
25 new_exception(defendant,colleague,room) is not valid
26 with respect to the literal interpretation,
27 Counterintuitive (yes/no) ? |: yes.
28
29 Detect an incomplete culprit:  new_exception(defendant,colleague,room).
30
31 Listing possibly relevant facts...
32 1: colleague(defendant,colleague)
33 ...
34
35 Please specify facts relevant to new_exception(defendant,colleague,room)
36 by a list of indices (e.g. [1,3,5]).
37 If you would like to input new facts,
38 please input newfact([fact1,fact2,...]).
39 |: [1].
```

```
40
41 Introduce a new rule:
42 new_exception(Subleaser,Sublessee,Property):-
43     colleague(Subleaser,Sublessee).
44
45 Found similar rule(s):
46 new_exception(Subleaser,Sublessee,Property):-
47     relatives(Subleaser,Sublessee).
48
49 Would you like to group these rules ? (yes/no) |: yes.
50 Please specify the new concept name |: acquaintance.
51
52 Remove rule(s)
53 new_exception(Subleaser,Sublessee,Property):-
54     relatives(Subleaser,Sublessee).
55 Add rule(s)
56 new_exception(Subleaser,Sublessee,Property):-
57     acquaintance(Subleaser,Sublessee).
58 acquaintance(Subleaser,Sublessee):-relatives(Subleaser,Sublessee).
59 acquaintance(Subleaser,Sublessee):-colleague(Subleaser,Sublessee).
```

In this example, the culprit detection algorithm goes into the rule with `effective_sublease_contract` in its head since now it has `new_exception(defendant,colleague,room)` in its body so the algorithm checks whether the invalidity of new exception is counterintuitive. As the user in this example intends the new exception should be derived, `new_exception(defendant,colleague,room)` becomes an incomplete culprit in this example. By asking relevant facts and consulting the culprit resolution algorithm, we get the rule `new_exception(Subleaser,Sublessee,Property):-colleague(Subleaser,Sublessee).` we can generalize by folding it with the new aim atom with the predicate `acquaintance`, and two rules that say two people are acquaintances if they are relatives or they are colleagues as illustrated in Figure 4.6. The new rules from W-operator generalize the concepts that the new exception should be derived when the subleaser is the acquaintance of the sublessee although there are two kinds of acquaintance known in the system so far.

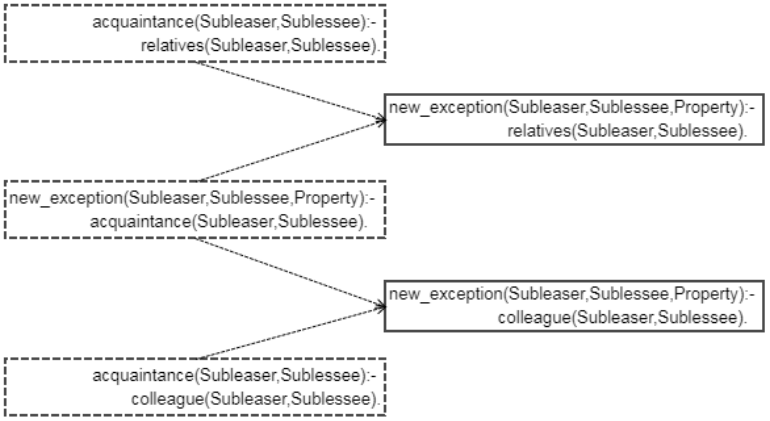

FIGURE 4.6: Generalizing culprit resolution with W-operator

4.6 Summary

- When judges apply interpretation of statutes in a particular case and it leads to counterintuitive consequences, judges would introduce an exception to the rule by introducing a new legal concept. Because of that, we formalize an exceptional case as a case which contains at least one fact with an extra fact predicate not occurring in the current rule-base
- We present Culprit Resolution Algorithm divide into two parts
  - the primary revision extends Closed World Specification (Algorithm 5) then asks the user which facts are relevant to each new head.
  - the second revision builds *prototypical cases with judgement (PCJs)* from the primary revised rule-base and the primary new rules, then fix the revision so that it reflects all attacks in the prototypical cases with judgement correctly (Algorithm 6) hence judgements of prototypical cases, which reflect legislator's intention, are all preserved.
- We consider the formalization of first-order case-base and analogous theory and extend Culprit Resolution Algorithm for resolving a culprit in a first-order rule-base (Algorithm 7).
- To extend Culprit Resolution Algorithm for a PROLEG rule-base, we translate the introduction of undercutting exceptions into rebutting exceptions when resolving a culprit
- We consider the application of Inverse Resolution for generalizing the resolution with background theory by V-operator and grouping similar concepts by W-operator

Chapter 5

Evaluating Resolution

In this chapter, we present

- Formalization of Semantics-based Minimal Revision
- Evaluation using Semantics-based Minimal Revision

5.1 Overview

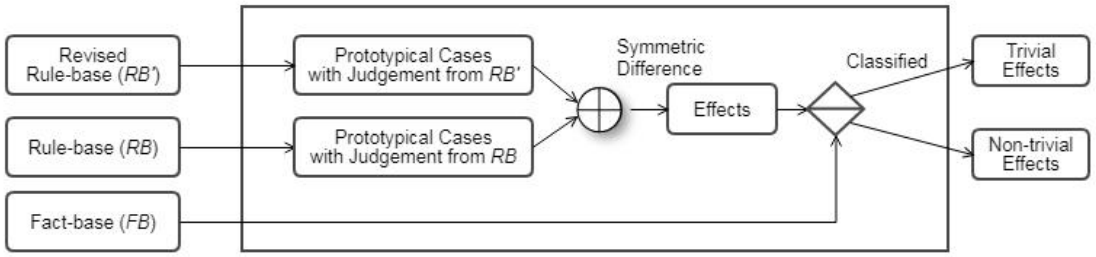

FIGURE 5.1: Evaluation Workflow

Figure 5.1 shows the workflow of the method to evaluate the resolution. Given the revised rule-base RB' from the culprit resolution, the original rule-base RB before the resolution, and the fact-base FB representing the present case, we aimed to prevent the resolution to cause counterintuitive consequences unintentionally. Therefore, we firstly examine minimal revision since it is one common constraint used when making revision in AI and Law (e.g. [36, 75]). Generally, researchers often used *syntax-based minimal revision* (e.g. [36, 99]), which tries to revise a rule-base as less as possible. However, we argue that minimal revision should be considered based on semantics, which depends on facts of the case, since legal reasoning is a hybrid between reasoning by rules and reasoning by cases [10, 100]. Although such semantics-based revision has been considered in hybrid legal reasoning systems between rule-based and case-based [26, 30], but semantics-based revision has not been investigated in pure rule-based legal reasoning systems. Therefore, we report our investigation of *semantics-based minimal revision* in the beginning of this chapter.

From our investigation, we achieve that effects of the semantics changes can be captured by the symmetric difference of two sets of prototypical cases with judgement (defined in Chapter 4), one from the revised rule-base and another from the original rule-base. We classify such effects into *trivial effects* and *non-trivial effects* by comparing them with the fact-base representing the present case. A trivial effect is an effect with a case that directly involves the present case but a non-trivial effect is an effect with a case that does not directly involve the present case. By this classification, we can check non-trivial effects with the user to confirm the user intention on such effects.

5.2 Semantics-based Minimal Revision

Minimal revision is one common theme in studies of revision in law since it reflects a principle that judges usually limit themselves from legislative issues. Since legal reasoning is a hybrid between reasoning by rules and reasoning by cases, a revision should consider semantics which depend on the facts of the case. Therefore, we define the semantics of a rule-base as follows.

Definition 5.1 (Semantics of a rule-base)**.** Let $\mathcal{F}$ be a fact-domain and RB be a rule-base, The *semantics* of RB is a set of pairs defined as follows: $\{\langle FB, A\rangle | FB$ is a fact-base constructed from a subset of $\mathcal{F}$ and A is the answer set of $FB \cup RB\}$. We denote this set as $sem(RB)$.

Example 5.1. *Let a fact-domain, $\mathcal{F}$ be $\{a, b, c\}$, and a rule-base $RB_1 = \{p \leftarrow q.\ q \leftarrow a.\ q \leftarrow b.\}$. Then $sem(RB_1) =$*

$$\{\langle\emptyset,\emptyset\rangle, \langle\{a\},\{a,p,q\}\rangle, \langle\{b\},\{b,p,q\}\rangle, \langle\{c\},\{c\}\rangle, \langle\{a,b\},\{a,b,p,q\}\rangle,$$
$$\langle\{a,c\},\{a,c,p,q\}\rangle, \langle\{b,c\},\{b,c,p,q\}\rangle, \langle\{a,b,c\},\{a,b,c,p,q\}\rangle\}$$

Then, we define a difference of semantics as follows.

Definition 5.2 (Difference of Semantics of a Rule-base)**.** Let $\mathcal{F}$ be a fact-domain and RB_1, RB_2 be two rule-bases. A difference of semantics between RB_1 and RB_2 denoted by $DIFF(RB_1, RB_2)$ is a set defined as follows: $\{\langle FB, D\rangle | FB$ is a fact-base constructed from a subset of $\mathcal{F}$ and D is a symmetric difference between the answer sets of $FB \cup RB_1$ and $FB \cup RB_2\}$.

Example 5.2. *Continuing from the previous example, let $RB_2 = \{p \leftarrow q.\ q \leftarrow a, not\ r.\ q \leftarrow b.\ r \leftarrow c.\}$. We get that, $sem(RB_2) =$*

$$\{\langle\emptyset,\emptyset\rangle, \langle\{a\},\{a,p,q\}\rangle, \langle\{b\},\{b,p,q\}\rangle, \langle\{c\},\{c,r\}\rangle, \langle\{a,b\},\{a,b,p,q\}\rangle,$$
$$\langle\{a,c\},\{a,c,r\}\rangle, \langle\{b,c\},\{b,c,p,q,r\}\rangle, \langle\{a,b,c\},\{a,b,c,p,q,r\}\rangle\}$$

and $DIFF(RB_1, RB_2) =$

$$\{\langle\emptyset,\emptyset\rangle, \langle\{a\},\emptyset\rangle, \langle\{b\},\emptyset\rangle, \langle\{c\},\{r\}\rangle, \langle\{a,b\},\emptyset\rangle,$$
$$\langle\{a,c\},\{p,q,r\}\rangle, \langle\{b,c\},\{r\}\rangle, \langle\{a,b,c\},\{r\}\rangle\}$$

Let $RB_3 = \{p \leftarrow q, not\ r.\ q \leftarrow a.\ q \leftarrow b.\ r \leftarrow c.\}$. We get that, $sem(RB_3) =$

$$\{\langle\emptyset,\emptyset\rangle, \langle\{a\},\{a,p,q\}\rangle, \langle\{b\},\{b,p,q\}\rangle, \langle\{c\},\{c,r\}\rangle, \langle\{a,b\},\{a,b,p,q\}\rangle,$$
$$\langle\{a,c\},\{a,c,q,r\}\rangle, \langle\{b,c\},\{b,c,q,r\}\rangle, \langle\{a,b,c\},\{a,b,c,q,r\}\rangle\}$$

and $DIFF(RB_1, RB_3) =$

$$\{\langle\emptyset,\emptyset\rangle, \langle\{a\},\emptyset\rangle, \langle\{b\},\emptyset\rangle, \langle\{c\},\{r\}\rangle, \langle\{a,b\},\emptyset\rangle,$$
$$\langle\{a,c\},\{p,r\}\rangle, \langle\{b,c\},\{p,r\}\rangle, \langle\{a,b,c\},\{p,r\}\rangle\}$$

A difference of semantics reflects changes on a consequence (both adding and removing) of two rule-bases. Now, we can define a minimal revision of this framework as follows.

Definition 5.3 (Semantics-based Minimal Revision)**.** Let RB_1, RB_2, RB_3 be three rule-bases. We say that RB_2 has a *smaller change than* RB_3 *from* RB_1 denoted as $DIFF(RB_1, RB_2) \leq DIFF(RB_1, RB_3)$ if $\langle FB, A2\rangle \in DIFF(RB_1, RB_2)$ and $\langle FB, A3\rangle \in DIFF(RB_1, RB_3)$, then $A2 \subseteq A3$. We define $DIFF(RB_1, RB_2) < DIFF(RB_1, RB_3)$ if $DIFF(RB_1, RB_2) \leq DIFF(RB_1, RB_3)$ but $DIFF(RB_1, RB_3) \not\leq DIFF(RB_1, RB_2)$. We call RB_2 a *semantics-based minimal revision* of RB_1 if RB_2 is a revision of RB_1 and there is no revision RB' of RB_1 such that $DIFF(RB_1, RB') < DIFF(RB_1, RB_2)$.

However, we consider Definition 5.3 is too strong, since it is hard for comparing between two revisions from different schemes. From the previous example, $DIFF(RB_1, RB_2)$ and $DIFF(RB_1, RB_3)$ are incomparable since

$\langle\{a,c\},\{p,q,r\}\rangle \in DIFF(RB_1, RB_2)$ and $\langle\{a,c\},\{p,r\}\rangle \in DIFF(RB_1, RB_3)$
hence $DIFF(RB_1, RB_2) \not\leq DIFF(RB_1, RB_3)$

and

$\langle\{a,b,c\},\{p,r\}\rangle \in DIFF(RB_1, RB_3)$ and $\langle\{a,b,c\},\{r\}\rangle \in DIFF(RB_1, RB_2)$
hence $DIFF(RB_1, RB_3) \not\leq DIFF(RB_1, RB_2)$.

Therefore, we relax Definition 5.3 by considering partial semantics-based minimal revision as follows.

Definition 5.4 (Partial Semantics-based Minimal Revision)**.** Let RB_1, RB_2, RB_3 be three rule-bases and S be a set of propositions. We say that RB_2 has a *smaller change than* RB_3 *from* RB_1 with respect to S denoted as $DIFF(RB_1, RB_2) \leq_S DIFF(RB_1, RB_3)$ if $\langle FB, A2\rangle \in DIFF(RB_1, RB_2)$ and $\langle FB, A3\rangle \in DIFF(RB_1, RB_3)$, then $A2 \cap S \subseteq A3 \cap S$. We define $DIFF(RB_1, RB_2) <_S DIFF(RB_1, RB_3)$ if $DIFF(RB_1, RB_2) \leq_S DIFF(RB_1, RB_3)$ but $DIFF(RB_1, RB_3) \not\leq_S DIFF(RB_1, RB_2)$. We call RB_2 a *partial semantics-based minimal revision* of RB_1 with respect to S if RB_2 is a revision of RB_1 and there is no revision RB' of RB_1 such that $DIFF(RB_1, RB') <_S DIFF(RB_1, RB_2)$.

Example 5.3. *From the previous example, if* $S = \{p\}$ *or* $S = \{p, r\}$*, we get that*

$DIFF(RB_1, RB_2) \leq_S DIFF(RB_1, RB_3)$ *but*
$DIFF(RB_1, RB_3) \not\leq_S DIFF(RB_1, RB_2)$
hence $DIFF(RB_1, RB_2) <_S DIFF(RB_1, RB_3)$.

However, if $S = \{q\}$*, we get that*

$DIFF(RB_1, RB_3) \leq_S DIFF(RB_1, RB_2)$ *but*
$DIFF(RB_1, RB_2) \not\leq_S DIFF(RB_1, RB_3)$
hence $DIFF(RB_1, RB_3) <_S DIFF(RB_1, RB_2)$.

We define a partial difference of semantics as follows.

Definition 5.5 (Partial Difference of Semantics)**.** Let $\mathcal{F}$ be a fact-domain, RB_1, RB_2 be two rule-bases, and S be a set of propositions. A partial difference of semantics between RB_1 and RB_2 with respect to S denoted by $DIFF_S(RB_1, RB_2)$ is a set defined as follows: $\{\langle FB, D \cap S\rangle | \langle FB, D\rangle \in DIFF(RB_1, RB_2)\}$.

By this way, given any three rule-bases RB_1, RB_2, RB_3,

$DIFF(RB_1, RB_2) \leq_S DIFF(RB_1, RB_3) \equiv$
$DIFF_S(RB_1, RB_2) \leq DIFF_S(RB_1, RB_3)$.

Example 5.4. *Continuing from the previous example,* $DIFF_{\{p\}}(RB_1, RB_2) =$

$$\{\langle\emptyset, \emptyset\rangle, \langle\{a\}, \emptyset\rangle, \langle\{b\}, \emptyset\rangle, \langle\{c\}, \emptyset\rangle,$$
$$\langle\{a, b\}, \emptyset\rangle, \langle\{a, c\}, \{p\}\rangle, \langle\{b, c\}, \emptyset\rangle, \langle\{a, b, c\}, \emptyset\rangle\}$$

5.3 Evaluating Resolution

5.3.1 Semantics-based Minimal Culprit Resolution

In this section, we consider evaluation of the culprit resolution in terms of semantics-based minimal revision, or as we call *semantics-based minimal culprit resolution* in shorts. Since we show in Chapter 4 that finding a culprit resolution to a CCR task is equivalent to adding representative sub-cases, which explain why the present case is exceptional, into prototypical cases with judgement. To alternatively analyze semantics of a rule-base by an equivalent case-base, we define the semantics of a case-base as follows.

Definition 5.6 (Semantics of a case-base)**.** Let $\mathcal{F}$ be a fact-domain and CB be a case-base. The *semantics* of CB is a set of pairs defined as follows: $\{(C, J)|C \subseteq \mathcal{F}$ and $CB \models (C, J)\}$. We denote this set as $cbsem(CB)$.

Example 5.5. *Let a fact-domain* $\mathcal{F} = \{a, b, c\}$*, and a case-base* $CB_1 = \{(\emptyset, \text{`}-\text{'}), (\{a\}, \text{`}+\text{'}), (\{b\}, \text{`}+\text{'})\}$*. Then* $cbsem(CB_1) = \{(\emptyset, \text{`}-\text{'}), (\{a\}, \text{`}+\text{'}), (\{b\}, \text{`}+\text{'}), (\{c\}, \text{`}-\text{'}), (\{a, b\}, \text{`}+\text{'}), (\{a, c\}, \text{`}+\text{'}), (\{b, c\}, \text{`}+\text{'}), (\{a, b, c\}, \text{`}+\text{'})\}$.

Then, we define the difference of semantics of a case-base as follows.

Definition 5.7 (Difference of Semantics of a Case-base)**.** Let $\mathcal{F}$ be a fact-domain, CB_1, CB_2 be two case-bases. The difference of semantics between CB_1 and CB_2 denoted by $CBDIFF(CB_1, CB_2)$ is a set defined as follows: $\{C|(C, J) \in cbsem(CB_1)$ and $(C, \bar{J}) \in cbsem(CB_2)\}$.

Example 5.6. *Continuing from the previous example, let* $CB_2 = \{(\emptyset, \text{`}-\text{'}), (\{a\}, \text{`}+\text{'}), (\{b\}, \text{`}+\text{'}), (\{a, c\}, \text{`}-\text{'})\}$*. Then,* $cbsem(CB_2) = \{(\emptyset, \text{`}-\text{'}), (\{a\}, \text{`}+\text{'}), (\{b\}, \text{`}+\text{'}), (\{c\}, \text{`}-\text{'}), (\{a, b\}, \text{`}+\text{'}), (\{a, c\}, \text{`}-\text{'}), (\{b, c\}, \text{`}+\text{'}), (\{a, b, c\}, \text{`}+\text{'})\}$ *and* $CBDIFF(CB_1, CB_2) = \{\{a, c\}\}$.

The difference of semantics of a case-base is closely related to the partial difference of semantics of rule-base as the following proposition.

Proposition 5.8. *Let* $\mathcal{F}$ *be a fact domain,* CB_1, CB_2 *be two critical case-base, and* p *be an ultimate goal.* $\langle FB, \{p\}\rangle \in DIFF_{\{p\}}(\textsc{theory-cons}(CB_1, p), \textsc{theory-cons}(CB_2, p))$ *if and only if* $f(FB) \in CBDIFF(CB_1, CB_2)$ *(where* THEORY-CONS *is a theory construction function defined in Chapter 2).*

Example 5.7. *Continuing from the previous example,* $\textsc{theory-cons}(CB_1, p) = \{p \leftarrow a.\ p \leftarrow b.\}$ *and* $\textsc{theory-cons}(CB_2, p) = \{p \leftarrow a, not\ r.\ p \leftarrow b.\ r \leftarrow c.\}$*. Then, a partial difference of semantics between both rule-bases with respect to* p *has only* $\langle\{a, c\}, \{p\}\rangle$ *that contains* $\{p\}$ *(see Example 5.4).*

Moreover, we can see from the previous example that the new cases with judgement added to the case-base will occur in the difference of semantics of a case-base, as described in the following proposition.

Proposition 5.9. *Let* $\mathcal{F}$ *be a fact domain,* CB *be a critical case-bases, and* $RepCJ$ *be a set of cases with judgement. If* $(C, \bar{J}) \in RepCJ$ *and* $CB \models (C, J)$ *and every* $(X, J_x) \in CB$*,* $C \not\subseteq X$*, then* $C \in CBDIFF(CB, CB \cup RepCJ)$.

Then, we define a semantics-based minimal culprit resolution as follows.

Definition 5.10 (Semantics-based Minimal Culprit Resolution)**.** Given a CCR task $\langle RB_1, FB_e, p\rangle$ such that there is a critical case-base CB which $RB_1 = \text{THEORY-CONS}(CB, p)$ and $CB \models (f(FB_e), J_e)$. We call a culprit resolution RB_2 of the task $\langle RB_1, FB_e, p\rangle$ a *semantics-based minimal culprit resolution* if there is no culprit resolution RB' of the task $\langle RB, FB_e, p\rangle$ such that $DIFF(RB_1, RB') <_{\{p\}} DIFF(RB_1, RB_2)$.

The simplest way to get a semantics-based minimal culprit resolution is to use all atoms in the exceptional case as a representative case. This can be formalized into the following lemma.

Lemma 5.11. *Given a CCR task $\langle RB, FB_e, p\rangle$ such that there is a critical case-base CB which $RB = \text{THEORY-CONS}(CB, p)$ and $CB \models (f(FB_e), J_e)$, we get that $RB_2 = \text{THEORY-CONS}(CB \cup \{(f(FB_e), \bar{J}_e)\}, p)$ is a semantics-based minimal culprit resolution of $\langle RB, FB_e, p\rangle$.*

Proof. Let $RepCJ$ be a set of cases with judgement such that $RepCJ \neq \{(f(FB_e), \bar{J}_e)\}$ and $RB' = \text{THEORY-CONS}(CB \cup RepCJ, p)$ is a culprit resolution of $\langle RB, FB_e, p\rangle$, then $CBDIFF(CB, CB \cup RepCJ) \not\subset CBDIFF(CB, CB \cup \{(f(FB_e), \bar{J}_e)\})$ hence there is no culprit resolution RB' such that $DIFF(RB_1, RB') <_{\{p\}} DIFF(RB_1, RB_2)$. □

5.3.2 Dominant Rule-base

In this section, we would show that the minimal culprit resolution also minimally affects the dominant rule-bases of each possible fact-base. Roughly speaking, the dominant rule-base is a trimmed version of the specific rule-base, which is preliminary defined in [101] as the subset of the rule-base that is specific to the given fact-base for deriving the given consequence. Both rule-bases are formally defined in the following.

Definition 5.12 (Specific Rule-base and Dominant Rule-base)**.** Let RB be a rule-base, FB be a fact-base, and p be a proposition. We say a rule-base $SR \subseteq RB$ is *specific* to FB with respect to RB and p if SR is a minimal set of rules (in the sense of set inclusion) such that

- $FB \cup SR$ agrees with $FB \cup RB$ on p, and
- no rule-base SR' such that $SR \subsetneq SR' \subsetneq RB$ and $FB \cup SR'$ disagrees with $FB \cup SR$ on p.

We call $DR = trim(SR)$ a *dominant* rule-base of FB with respect to RB and p (*trim* is defined in Definition 2.8).

TABLE 5.1: Differences of dominant rule-bases between RB_1 and RB_2

Possible fact-base FB	Dominant rule-base(s) of FB (before revision)	Dominant rule-base(s) of FB (after revision)
$\emptyset$	$\emptyset$	$\emptyset$
$\{a.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$
$\{b.\}$	$\{p \leftarrow q.\ q \leftarrow b.\}$	$\{p \leftarrow q.\ q \leftarrow b.\}$
$\{c.\}$	$\emptyset$	$\emptyset$
$\{a.\ b.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$ and $\{p \leftarrow q.\ q \leftarrow b.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$ and $\{p \leftarrow q.\ q \leftarrow b.\}$
$\{a.\ c.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$	$\{\mathbf{r} \leftarrow \mathbf{c}.\}$
$\{b.\ c.\}$	$\{p \leftarrow q.\ q \leftarrow b.\}$	$\{p \leftarrow q.\ q \leftarrow b.\}$
$\{a.\ b.\ c.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$ and $\{p \leftarrow q.\ q \leftarrow b.\}$	$\{\mathbf{p} \leftarrow \mathbf{q}.\ \mathbf{q} \leftarrow \mathbf{b}.\}$

Example 5.8. *From the examples in previous sections, let a fact-domain* $\mathcal{F} = \{a, b, c\}$ $RB_1 = \{p \leftarrow q.\ q \leftarrow a.\ q \leftarrow b.\}$ *and the fact-base representing the exceptional case is* $FB_1 = \{a.\ c.\}$. *If a culprit is* q *and we get a primary revision* $\{p \leftarrow q.\ q \leftarrow a, not\ r.\ q \leftarrow b.\}$. *If we add all fact atoms in* FB_1 *to a body of a new rule, we get a revision* $RB_2 = \{p \leftarrow q.\ q \leftarrow a, not\ r.\ q \leftarrow b.\ r \leftarrow c.\}$. *We get that the revision affects the dominant rule-bases of* $\{a.\ c.\}$ *and* $\{a.\ b.\ c.\}$, *as illustrated in Table 5.1.*

Example 5.9. *(Continuing from the previous example) In contrast, if a culprit is* p *and we get a primary revision* $\{p \leftarrow q, not\ r.\ q \leftarrow a.\ q \leftarrow b.\}$. *If we resolve a culprit using* $RB_3 = \{p \leftarrow q, not\ r.\ q \leftarrow a.\ q \leftarrow b.\ r \leftarrow c.\}$. *We get that the revision affects the dominant rule-bases of* $\{a.\ c.\}$, $\{b.\ c.\}$, *and* $\{a.\ b.\ c.\}$, *as illustrated in Table 5.2. This corresponds to the difference of semantics of a case-base between the prototypical cases with judgement of* RB_1 *and* RB_3, *which is a set of* $\{\{a.\ c\}, \{b.\ c.\}, \{a.\ b.\ c.\}\}$. *Hence,* RB_3 *is not a minimal culprit resolution from* RB_1 *since there exists* $\{b.\ c.\}$ *that is not superset or equal to the present case (*FB_1*). The minimal culprit resolution when a culprit is* p *is actually* $RB' = \{p \leftarrow q, not\ r.\ q \leftarrow a.\ q \leftarrow b.\ r \leftarrow a, c.\}$.

5.3.3 Evaluation Method

From the investigation above, we achieve that effects of the semantics changes can be captured by the additional prototypical cases with judgement (PCJs), produced after the resolution. We classify such effects into *trivial effects* and *non-trivial effects* as the following definition.

Definition 5.13 (Effects)**.** Given a rule-bases RB representing the statute, a fact-base FB representing the case, a considered ground rule atom p, and a revised rule-base RB'

TABLE 5.2: Differences of dominant rule-bases between RB_1 and RB_3

Possible fact-base FB	Dominant rule-base(s) of FB (before revision)	Dominant rule-base(s) of FB (after revision)
$\emptyset$	$\emptyset$	$\emptyset$
$\{a.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$
$\{b.\}$	$\{p \leftarrow q.\ q \leftarrow b.\}$	$\{p \leftarrow q.\ q \leftarrow b.\}$
$\{c.\}$	$\emptyset$	$\emptyset$
$\{a.\ b.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$ and $\{p \leftarrow q.\ q \leftarrow b.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$ and $\{p \leftarrow q.\ q \leftarrow b.\}$
$\{a.\ c.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$	$\{\mathbf{r} \leftarrow \mathbf{c}.\}$
$\{b.\ c.\}$	$\{p \leftarrow q.\ q \leftarrow b.\}$	$\{\mathbf{r} \leftarrow \mathbf{c}.\}$
$\{a.\ b.\ c.\}$	$\{p \leftarrow q.\ q \leftarrow a.\}$ and $\{p \leftarrow q.\ q \leftarrow b.\}$	$\{\mathbf{r} \leftarrow \mathbf{c}.\}$

after the culprit resolution. An *effect* of a revision from RB to RB' with respect to p is a prototypical case with judgement (PCJ) in the symmetric difference between the set of PCJs from RB' with respect to p and the set of PCJs from RB with respect to p. We call an effect (C, J) a *trivial effect* with respect to FB if $C \subseteq f(FB)$ or $f(FB) \subseteq C$, otherwise we call a *non-trivial effect* with respect to FB.

Intuitively, trivial effects mean additional PCJs that directly involve the present case which non-trivial effects do not. By this classification, we may determine non-trivial effects as ones that possibly make unintentional semantics changes and we can ask the user to confirm the user intention on such effects.

Example 5.10. *Let* $RB_1 = \{p \leftarrow q.\ q \leftarrow a.\ q \leftarrow b.\}$ *and* $RB_3 = \{p \leftarrow q, not\ r.\ q \leftarrow a.\ q \leftarrow b.\ r \leftarrow c.\}$ *and* $FB = \{a.\ c.\}$. *From Example 5.9, we can view* RB_3 *as a resolution to an incorrect culprit* p *as Figure 5.2. We get that there are two effects of a revision from* RB_1 *to* RB_3 *with respect to* p, *which are* $pcj_1 = (\{a, c\}, \text{`}-\text{'})$ *and* $pcj_2 = (\{b, c\}, \text{`}-\text{'})$. *Consequently,* pcj_1 *is a trivial effect with respect to* FB *since* $f(FB) = \{a, c\}$. *On the other hand,* pcj_2 *is a non-trivial effect with respect to* FB *since* $f(FB) \not\subseteq \{b, c\}$ *and* $\{b, c\} \not\subseteq f(FB)$.

Example 5.11. *Let* $RB_1 = \{p \leftarrow q.\ q \leftarrow a.\ q \leftarrow b.\}$ *and* $RB_4 = \{p \leftarrow q, not\ r.\ q \leftarrow a.\ q \leftarrow b.\ r \leftarrow a, c.\}$ *and* $FB = \{a.\ c.\}$. *From Example 5.9, we get that* RB_4 *is actually the semantics-based minimal resolution to an incorrect culprit* p *as illustrated in Figure 5.3. There are two effects of a revision from* RB_1 *to* RB_4 *with respect to* p, *which are* $pcj_1 = (\{a, c\}, \text{`}-\text{'})$ *and* $pcj_2 = (\{a, b, c\}, \text{`}-\text{'})$. *Consequently, both of them are trivial effects with respect to* FB *since* $f(FB) = \{a, c\}$ *and* $f(FB) \subseteq \{a, b, c\}$.

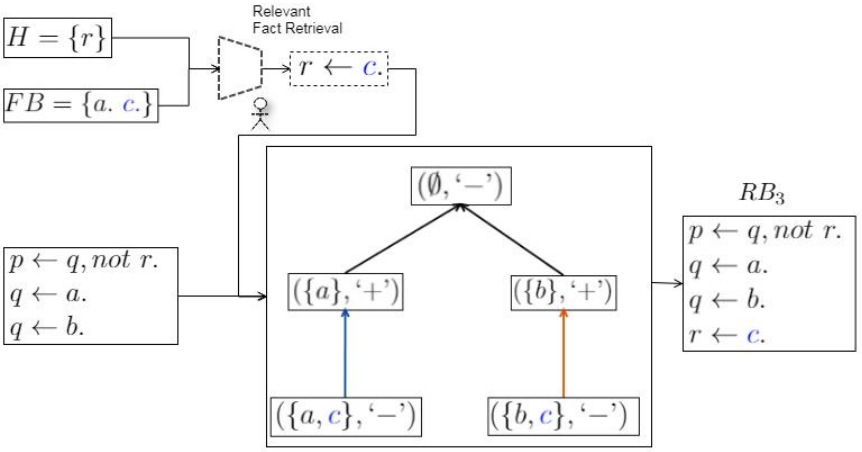

FIGURE 5.2: General Culprit Resolution before Evaluation

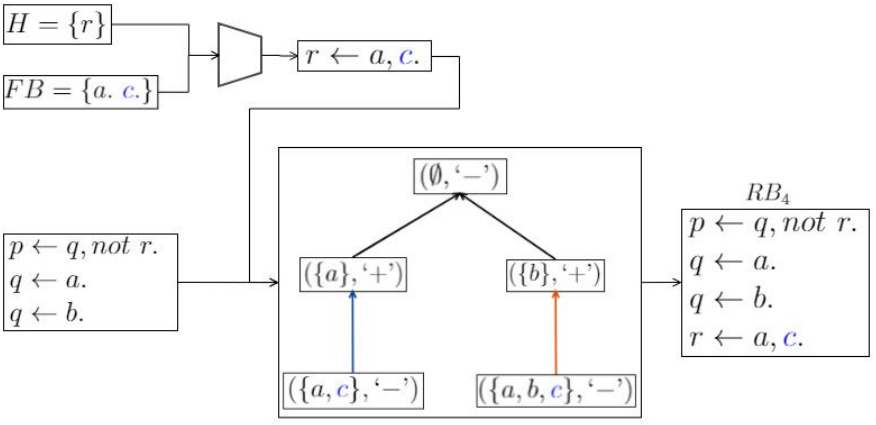

FIGURE 5.3: Semantics-based Minimal Culprit Resolution before Evaluation

5.3.4 Evaluating Implicit Generalization

In Chapter 4, we have discussed that actually we generalize the resolution implicitly when the user selects which facts relevant to each new head since the new rule needs not to be applied specifically in the case larger or equal to the present case. This selection of relevant facts is quite similar to the syntax-based minimal revision if we have a criterion to select as few relevant facts as possible. For example, let consider the syntax-based minimal revision based on Theory Distance Metric [99], which is one common minimal revision used for describing minimal revision in legislation (e.g. [36, 96]). The definition is formally described in our context as follows.

Definition 5.14 (Syntax-based Minimal Revision)**.** Let RB and RB' be rule-bases. A revision transformation r is such that $r(RB) = RB'$, and RB' is obtained from RB by program edit operations as follows: deleting a rule, creating a rule with an empty body, adding a condition to a rule in RB or deleting a condition from a rule in RB. RB' is a

revision of RB with distance $c(RB, RB') = n$ if and only if $RB' = r^n(RB)$ and there is no $m < n$ such that $RB' = r^n(RB)$ [99].

We get that, actually, it is very easy to obtain the syntax-based minimal culprit resolution to a CCR task since, without finding relevant facts, the primary revised rule-base is also a minimal revision since it requires no additional program edit operation. However, if we require a restricted culprit resolution, we can just introduce one extra fact to each new head. This requires only one program edit operation per each new rule so the revision is definitely the syntax-based minimal revision.

Generally, we get that the syntax-based minimal revision is not a semantics-based minimal revision especially when an original rule-base contains multiple rules with unifiable heads. For example, a rule-base representing Japanese Civil Code Article 612 but adding a fictitious rule with `condition_2` can be described in the following.

```
1  cancellation_due_to_sublease :-
2          effective_lease_contract,
3          effective_sublease_contract, using_leased_thing,
4          manifestation_cancellation, not approval_of_sublease.
5  effective_lease_contract :-
6          agreement_of_lease_contract,
7          handover_to_lessee.
8  effective_sublease_contract :-
9          agreement_of_sublease_contract,
10         handover_to_sublessee.
11 effective_sublease_contract :- condition_2.
12 approval_of_sublease :- approval_before_the_day.
```

The present case is represented as follows.

$$\left\{\begin{array}{l}\texttt{agreement\_of\_lease\_contract},\\ \texttt{handover\_to\_leasee},\\ \texttt{agreement\_of\_sublease\_contract},\\ \texttt{handover\_to\_subleasee},\\ \texttt{using\_leased\_thing},\\ \texttt{manifestation\_cancellation},\\ \texttt{non\_destruction\_of\_confidence}\end{array}\right\}$$

Suppose `cancellation_due_to_sublease` is an incorrect culprit. A semantics-based minimal revision for resolving the culprit is as follows.

```
cancellation_due_to_sublease :-
effective_lease_contract,
        effective_sublease_contract, using_leased_thing,
        manifestation_cancellation, not approval_of_sublease,
        not new_exception.
effective_lease_contract :-
        agreement_of_lease_contract,
        handover_to_lessee.
effective_sublease_contract :-
        agreement_of_sublease_contract,
        handover_to_sublessee.
effective_sublease_contract :- condition_2.
approval_of_sublease :- approval_before_the_day.
new_exception :-
        agreement_of_sublease_contract,
        handover_to_sublessee,
        non_destruction_of_confidence.
```

On the other hand, the syntax-based minimal revision for resolving the culprit is as follows.

```
cancellation_due_to_sublease :-
effective_lease_contract,
        effective_sublease_contract, using_leased_thing,
        manifestation_cancellation, not approval_of_sublease,
        not new_exception.
effective_lease_contract :-
        agreement_of_lease_contract,
        handover_to_lessee.
effective_sublease_contract :-
        agreement_of_sublease_contract,
        handover_to_sublessee.
effective_sublease_contract :- condition_2.
approval_of_sublease :- approval_before_the_day.
new_exception :-
        non_destruction_of_confidence.
```

Compared with the semantics-based minimal revision, the syntax-based minimal revision produces a non-trivial effect in the following.

$$\left(\left\{\begin{array}{l}\texttt{agreement\_of\_lease\_contract},\\ \texttt{handover\_to\_leasee},\\ \texttt{condition\_2},\\ \texttt{using\_leased\_thing},\\ \texttt{manifestation\_cancellation},\\ \texttt{non\_destruction\_of\_confidence}\end{array}\right\},\text{`\_'}\right)$$

We consider that such a non-trivial effect occurs due to possibly unintentional change of semantics caused by the syntax-based minimal revision. Hence, we check with the user whether such changes are actually intended. If they are not intended, we may suggest the semantics-based minimal revision as shown above.

5.3.5 Evaluating Explicit Generalization

In Chapter 4, we have presented the *explicit generalization* with background theory by Inverse Resolution, which consists of V-operator and W-operator. In this dissertation, we induce the parent clause in V-operator by using folding. In the original folding [54], the clause cannot be folded if there are any other clauses whose head is unifiable with the aim goal but are not used in the original folding (see Chapter 2). However, in our induction, we use folding with relaxation of this condition. Hence, to prevent unintentional effects, we can consider the difference of prototypical cases with judgement, as discussed in previous sections, to check with the user whether the generalization is intended.

Example 5.12. *Continuing from Example 4.7, suppose we have two rules in the background theory.*

- *"A parent is a kind of relatives" represented as* `relatives(X,Y) :- parent(X,Y).`
- *"A cousin is a kind of relatives" represented as* `relatives(X,Y) :- cousin(X,Y).`

After such a generalization as in Example 4.7, we may detect that the generalization produces non-trivial effect as the continuous dialogue shown below.

```
39 Found general rule(s):
40 1: new_exception(Sublesser,Sublessee,Property):-
41     parent(Sublesser,Sublessee).
42 2: new_exception(Sublesser,Sublessee,Property):-
43     relatives(Sublesser,Sublessee).
44
45 Which one would you like to introduce ? |: 2.
```

```
46
47 Non-trivial Effect(s) detected:
48 ({agreement_of_lease_contract(plaintiff,defendant,room),
49     handover_to_lessee(plaintiff,defendant,room),
50     agreement_of_sublease_contract(defendant,son,room),
51     handover_to_sublessee(defendant,son,room),
52     using_leased_thing(son,room),
53     manifestation_cancellation(plaintiff,defendant),
54     cousin(plaintiff,son)},'-')
55
56 Would you like to confirm the introduction of the rule (yes/no)?
57 new_exception(Sublesser,Sublessee,Property):-
58     relatives(Sublesser,Sublessee).
59 |: no.
60
61 Found general rule(s):
62 1: new_exception(Sublesser,Sublessee,Property):-
63     parent(Sublesser,Sublessee).
64
65 Which one would you like to introduce instead ? |: 1.
```

Intuitively, the non-trivial effect raises the question of whether the new exception should be executed when the subleaser is the cousin of the sublessee. The user may determine the generalization is too generalized and refuse the generalization, or create a new sub-level concept (e.g. `family`) using W-operator for refining the generalization if some non-trivial effects are intended but some are not.

5.4 Summary

- We consider *the semantics of a rule-base* as a set of all possible pairs of a fact-base and its corresponding answer set.
- We consider *the semantics-based minimal revision* as a revision that we cannot find any other revision with smaller change. In the same manner, we consider *the semantics-based minimal culprit resolution* if there is no culprit resolution with smaller change.
- We consider *the semantics of a case-base* as a set of all possible cases with judgement that the case-base entails. We show that the difference of semantics of a case-base corresponds to the partial difference of semantics of a rule-base.

- We show that the minimal culprit resolution also minimally affects the dominant rule-base of each possible fact-base.
- We show that the syntax-based minimal revision is not always a semantics-based minimal revision especially when a rule-base contains multiple rules with unifiable heads.
- We consider new prototypical cases that are not subsets, supersets, or equal to the representative case as probably unintentional changes of semantics. Hence, by this consideration, we can detect possibly unintentional changes of semantics during the generalization of culprit resolution by checking with the user whether the generalization is intended.

Chapter 6

Conclusion and Future Work

In this chapter, we present

- Conclusion of this dissertation
- Potential future work of our research

6.1 Conclusion

In this dissertation, we explore *Legal Debugging* in rule-based legal reasoning systems for formalizing judicial legal change when literal interpretation of statutes has consequences that do not meet the social expectation, as we call *counterintuitive consequences*. By analogy, we define a *culprit*, which is newly coined in this research, to describe the root cause of counterintuitive consequences. Legal Debugging detects and resolves counterintuitive consequences in law by answering these three research questions.

- RQ1: How to detect a culprit ?
- RQ2: How to resolve a culprit ?
- RQ3: How to evaluate the resolution ?

To answer these questions, we present two algorithms, namely Culprit Detection Algorithm and Culprit Resolution Algorithm, and one method to evaluate generalization of exceptions in the revised rule-based. Those are designed as follows.

- **Culprit Detection Algorithm**, which extends from Algorithmic Debugging [38], aims to assist the user to discover more counterintuitive consequences by checking with the user whether related consequences are counterintuitive until no related counterintuitive consequences can be found. Then, we get that the last found counterintuitive consequence is a culprit (Theorem 3.4). It is proved that if there is a counterintuitive consequence, there is definitely a culprit and if there is a culprit, there is definitely one counterintuitive consequence that can trace to such a culprit. We show that the algorithm can be extended for first-order representation and PROLEG, as in PROLEG, the algorithm can detect a culprit if we generalize the definition of the supporting rule.
- **Culprit Resolution Algorithm**, which extends from Closed World Specification [56], aims to assist the user to revise the rule-base representing statutes by let the user select which facts relevant to each new head. In this algorithm, we describe a class of rule-bases that built from critical sets [63, 68] and we show that the algorithm can definitely resolve culprits (and hence resolve counterintuitive consequences) and the algorithm can give the designated rule-base in an effective enumeration of the class in finite time. We also provide our formalization of first-order case-base and analogous theory to show that the algorithm can be extended for culprit resolution in first-order representation and we apply the

translation of undercutting exceptions into rebutting exceptions for culprit resolution in PROLEG. In addition, we present the application of Inverse Resolution [33] for generalizing culprit resolution. In this way, we can produce more general rules for resolving a culprit by cooperating with background theory. With such cooperation, the resolution can obtain more general normative facts to resolve a culprit more practically.

- **A method to evaluate resolutions**, which we develop based on the semantics-based minimal revision for legal reasoning, which we newly formalized in this research. By this formalization, we achieve a semantics-based minimal culprit resolution. We show that the minimal culprit resolution also minimally affects the dominant rule-base of each possible fact-base, and the semantics-based minimal revision is different from the syntax-based minimal revision in general. From these findings, we evaluate the resolution by detecting possibly unintentional changes of semantics due to the generalization of culprit resolution.

Since most of statutory laws are designated to produce one unique interpretation for each case in litigation, they can be represented by a non-recursive and stratified logic program with corresponding prototypical cases with judgement and hence our legal debugging is applicable to any statutory laws in general.

6.2 Future Work

In this section, we report potential future directions of the research on Legal Debugging as follows.

1. **Potential future investigations in Legal Debugging**: Since this dissertation has investigated Legal Debugging with some presumptions, we report potential future investigations in Legal Debugging as follows.

    - **Extending Legal Debugging to support rule-bases that allow conflict judgements**: In this dissertation, we assume that a rule-base representing statutes is non-recursive, stratified, and theoretically constructed from prototypical cases with judgement (e.g. [61, 68]) so that the result program has a unique answer set. This reflects that judges in civil law systems tend to have one unique judgement from the same statutes in the similar case. This is in contrast with representations in common law systems that tend to allow conflict judgements even if the cases are similar. To deal with conflict judgements, multiple answer sets programming is sometimes introduced in legal

representation for common law systems such as Defeasible Logic [59]. Hence, assuming programs has a unique answer set is one limitation of our work and studies of debugging answer set programming [87, 93, 102] are interesting for expanding Legal Debugging to support rule-bases that allow conflict judgements.

- **Extending evaluation for any other types of revisions**: since we have investigated evaluation using semantics-based minimal revision only for the culprit resolution, it is interesting to investigate semantics-based minimal revision for evaluating any other types of revisions in general.

2. **Potential applications of Legal Debugging**: Besides detecting and resolving counterintuitive consequences in law as suggested in this dissertation, we suggest potential applications of Legal Debugging as follows.

   - **Legislation**: As one application of inductive programming is to synthesize a new program from an empty program, Legal Debugging might prove to be useful for drafting new statutes from scratch.
   - **Conflict Resolution**: Instead of considering the interpretation intended by legal experts, we might simulate counterintuitive consequences as the difference of the literal interpretation in one legal system and another interpretation in other legal systems (probably the superior legal systems such as federal law and state law in the United States, or the comparison of international law between two countries).
   - **Contract Debugging**: Instead of applying Legal Debugging with rule-bases representing statutes, we may apply it with representations of contracts in the same manner of detecting conflicts in legislation. Hence, we may apply Legal Debugging for detecting and resolving parts of contracts that may introduce unexpected results.

Appendix A

English Court Example Cases

In Appendix A, we show some example cases involving with statutory interpretation from English Courts. English courts have developed three rules of statutory interpretation to deal with counterintuitive consequences, which are the plain meaning rule, the golden rule, and the mischief rule. Generally speaking, the plain meaning rule suggests judges to keep literal interpretation although it causes counterintuitive consequences. The golden rule suggests judges to generalize or specify meaning of the statute to prevent counterintuitive consequences. The mischief rule suggest judges to focus on the purpose of the statute that tries to resolve some mischief in the society. In this Appendix, we apply Legal Debugging for formalizing legal changes in the example cases, where the judges applied the golden rule and the mischief rule, to illustrate the applicable range of legal debugging as follows.

A.1 *R v. Allen (1872)*

This example case [103] is a well-known case in which the judge applied the golden rule by generalizing the meaning of the wording in the statute. In this case, the defendant was charged under the Offences Against the Person Act 1861 Section 57, which states

> **Section 57** Whosoever being married shall marry any other person during the lifetime of the former husband or wife is guilty of an offence [104].

The Offences Against the Person Act 1861 Section 57 can be represented as the following rule-base.

```
guilty_of_an_offence(Person):-
        married(Person,Spouse),
        different_person(Spouse,Paramour),
        second_married(Person,Spouse,Paramour).
married(Person,Spouse):-
    legitimate_married(Person,Spouse,Certificate1).
second_married(Person,Spouse,Paramour):-
    legitimate_married(Person,Paramour,Certificate2),
    during_the_lifetime_of(Certificate2,Spouse)
```

However, the defendant in this case has not legitimate married but go through the ceremony of marriage with any other person. When the case went to the court, the judge stated that the word marry in this section could not mean "legitimate marry" since "legitimate marry" is regulated to have only one at a time. Hence, the judge reinterpreted 'marry' in this section as "go through the ceremony of marriage." We assume that the case can be represented by the following fact-base.

```
{legitimate_married(defendant,person_a).
different_person(person_a,person_b),
go_through_the_ceremony_of_marriage(defendant,person_b,the_ceremony).
during_the_lifetime_of(the_ceremony,person_a).}
```

With this rule-base and fact-base, `guilty_of_an_offence(defendant)` is not valid. This is consistent with the literal interpretation of the law, in which we consider the meaning of marry with another person as "legitimate marry" . The example dialogue of the culprit detection and the culprit resolution algorithm in this case is shown below.

```
Considering guilty_of_an_offence(defendant):-
        married(defendant,Spouse),
        different_person(Spouse,Paramour),
        second_married(defendant,Paramour,Spouse).

married(defendant,person_a) is valid w.r.t. the literal interpretation,
Counterintuitive (yes/no) ? |: no.
different_person(person_a,person_b) occurs in the case
```

```
second_married(defendant,person_a,person_b) is not valid
w.r.t. the literal interpretation,
Counterintuitive (yes/no) ? |: yes.

Detect an incomplete culprit:
    second_married(defendant,person_a,person_b)

A prototypical case with judgement associated with
second_married(defendant,person_a,person_b):
({legitimate_married(defendant,person_a),
different_person(person_a,person_b)},'-')

Listing possibly relevant facts...
1: go_through_the_ceremony_of_marriage(defendant,person_b,the_ceremony).
2: during_the_lifetime_of(the_ceremony,person_a).

Please specify facts relevant to
    second_married(defendant,person_a,person_b)
by a list of indices (e.g. [1,3,5]).
If you would like to input new facts,
please input newfact([fact1,fact2,...]).
|: [1,2].

A representative sub-case with judgement:
({legitimate_married(defendant,person_a).
    different_person(person_a,person_b),
    go_through_the_ceremony_of_marriage(defendant,person_b,the_ceremony).
    during_the_lifetime_of(the_ceremony,person_a).},'+')

Introduce a new rule:
second_married(Person,Spouse,Paramour):-
    go_through_the_ceremony_of_marriage(Person,Paramour,Ceremony),
    during_the_lifetime_of(Ceremony,Spouse).
```

Since the user (representing the judge) has considered that `guilty_of_an_offence(defendant)` is intended in this case and hence it is a counterintuitive consequence. Culprit Detection Algorithm would consider whether there is another counterintuitive consequence related to `guilty_of_an_offence(defendant)` by going to the first rule (lines 1-4), whose head is unifiable with the atom by a substitution $\theta = \{\texttt{Person/defendant}\}$.

The algorithm asks the user whether `married(defendant,person_a)` is counterintuitive. Since it is not counterintuitive, the algorithm adds a mapping `Spouse/person_a` into θ. When the algorithm traces to `different_person(person_a,Paramour)`, it gets `different_person(person_a,person_b)` from the fact-base hence it adds a mapping `Paramour/person_b` in θ. Then, the algorithm asks whether `second_married(defendant,person_a,person_b)` is counterintuitive. Since the user determines the consequence as counterintuitive, the consequence becomes an incomplete culprit since it is intended but the intended interpretation does not support it. Since the culprit is an incomplete culprit, the algorithm asks the user to select facts or input new facts that are relevant to the culprit. To reflect that the judge generalized the meaning of 'marry any other person' in the statute to cover "go through the ceremony of marriage" in this case, the user select all the left facts as relevant. Thus, the algorithm introduces the new rule (lines 39-41 in the dialogue) to cover the change introduced by the judge.

A.2 *Adler v. George (1964)*

This example case is adopted from the English Court Case. It is one of the well-known cases in which the judge applied the golden rule by modifying the meaning of the wording in the statute. In this case, the defendant was charged under the Official Secrets Act 1920 Section 3, which states

> **Section 3** No person in the vicinity of any prohibited place shall obstruct, knowingly mislead or otherwise interfere with or impede, the chief officer or a superintendent or other officer of police, or any member of His Majesty's forces engaged on guard, sentry, patrol, or other similar duty in relation to the prohibited place, and, if any person acts in contravention of, or fails to comply with, this provision, he shall be guilty of a misdemeanour.

The Official Secrets Act 1920 Section 3 can be represented as the following rule-base.

```
1 guilty_of_a_misdemeanour(Person):-
2     prohibited(Place), officer(Officer,Place),
3     interfere(Person,Officer), in_the_vicinity_of(Person,Place).
4
5 officer(Officer,Place) :- chief_officer(Officer,Place).
6 officer(Officer,Place) :- superintendent(Officer,Place).
7 officer(Officer,Place) :- police(Officer,Place).
8 officer(Officer,Place) :- royal_guard(Officer,Place).
```

```
officer(Officer,Place) :- sentry(Officer,Place).
officer(Officer,Place) :- patrol(Officer,Place).
officer(Officer,Place) :- other_officer(Officer,Place).

interfere(Person,Officer) :- obstruct(Person,Officer).
interfere(Person,Officer) :- knowingly_mislead(Person,Officer).
interfere(Person,Officer) :- impede(Person,Officer).
interfere(Person,Officer) :- otherwise_interfere(Person,Officer).
```

However, the defendant in this case obstructed a military guard in the execution of his duty inside a military establishment. From the literal interpretation of the law, the defendant shall be guilty only if the obstruction takes place 'in the vicinity of' a military establishment, which literally means "outside or in the proximity". When the case went to the court, the judge stated that such an interpretation would lead to an absurd result, and reinterpreted 'in the vicinity of' to cover a person already on the premises. We assume that the case can be represented by the following fact-base.

```
{prohibited(military_est).  royal_guard(guard,military_est).
obstruct(defendant,guard).  inside(defendant,military_est).}
```

With this rule-base and fact-base, `guilty_of_a_misdemeanour(defendant)` is not valid since we cannot prove `in_the_vicinity_of(defendant, military_est)`. Given the atom as an initial counterintuitive consequence, the example dialogue of the culprit detection and the culprit resolution algorithm in this case is shown below.

```
Considering guilty_of_a_misdemeanour(defendant):-
    prohibited(Place), officer(Officer,Place),
    interfere(defendant,Officer), in_the_vicinity_of(defendant,Place).

prohibited(military_est) occurs in the case
officer(guard,military_est) is valid
w.r.t. the literal interpretation,
interfere(defendant,guard) is valid
w.r.t. the literal interpretation,
in_the_vicinity_of(defendant,military_est) does not occur in the case
Detect an incomplete culprit:
    guilty_of_a_misdemeanour(defendant)

```

```
A prototypical case with judgement associated with
guilty_of_a_misdemeanour(defendant): ({},'-')

Listing possibly relevant facts...
1: prohibited(military_est).
2: obstruct(defendant,guard).
3: royal_guard(guard).
4: inside(defendant,military_est).

Please specify facts relevant to guilty_of_a_misdemeanour(defendant)
by a list of indices (e.g. [1,3,5]).
If you would like to input new facts,
please input newfact([fact1,fact2,...]).
|: [1,2,3,4].

A representative sub-case with judgement:
({prohibited(military_est), royal_guard(guard,military_est),
obstruct(defendant,guard), inside(defendant,military_est)},'+')

Introduce a new rule:
guilty_of_a_misdemeanour(defendant):-
    prohibited(Place), royal_guard(Officer,Place),
    obstruct(Person,Officer), inside(Person,Place).
Found general rule(s):
1: guilty_of_a_misdemeanour(Person):-
    prohibited(Place), royal_guard(Officer,Place),
    obstruct(Person,Officer), inside(Person,Place).
2: guilty_of_a_misdemeanour(Person):-
    prohibited(Place), officer(Officer,Place),
    obstruct(Person,Officer), inside(Person,Place).
3: guilty_of_a_misdemeanour(Person):-
    prohibited(Place), royal_guard(Officer,Place),
    interfere(Person,Officer), inside(Person,Place).
4: guilty_of_a_misdemeanour(Person):-
    prohibited(Place), officer(Officer,Place),
    interfere(Person,Officer), inside(Person,Place).

Which one would you like to introduce ? |: 4.
```

Since the user (representing the judge) has considered that `guilty_of_a_misdemeanour(defendant)` is intended in this case and hence it is a counterintuitive consequence. Culprit Detection Algorithm would consider whether there is another counterintuitive consequence related to `guilty_of_a_misdemeanour(defendant)` by going to the first rule (lines 1-3), whose head is unifiable with the atom by a substitution $\theta = \{$`Person/defendant`$\}$. Since there is only the fact `prohibited(military_est)` that is in the fact-base and unifiable with the condition `prohibited(Place)`, the algorithm add the mapping `Place/military_est`. The algorithm asks the user whether `officer(guard,military_est)` is counterintuitive. Since it is not counterintuitive, the algorithm adds a mapping `Officer/guard` into θ. Then, the algorithm asks whether `interfere(defendant,guard)` is counterintuitive. Since it is not counterintuitive, there are no counterintuitive consequences related to `guilty_of_a_misdemeanour(defendant)`. Hence, it becomes an incomplete culprit itself since it is intended but the intended interpretation does not support it. Since the culprit is an incomplete culprit, the algorithm asks the user to select facts or input new facts that are relevant to the culprit. To reflect that the judge modified the meaning of 'in the vicinity of' in the statute to cover "inside" in this case, the user select all the left facts as relevant. Thus, the algorithm introduces the new rule (lines 31-33 in the dialogue) to cover the change introduced by the judge. We can obtain general rules (lines 35-53 in the dialogue) by V-operator.

Bibliography

[1] Edwina L Rissland, Kevin D Ashley, and Ronald Prescott Loui. Ai and law: A fruitful synergy. *Artificial Intelligence*, 150(1-2):1–15, 2003.

[2] Marek J. Sergot, Fariba Sadri, Robert A. Kowalski, Frank Kriwaczek, Peter Hammond, and H Terese Cory. The British Nationality Act as a logic program. *Communications of the ACM*, 29(5):370–386, April 1986.

[3] David M Sherman. A prolog model of the income tax act of Canada. In *Proceedings of the 1st international conference on Artificial intelligence and law*, pages 127–136, New York, NY, USA, 1987.

[4] Raymond Reiter. A logic for default reasoning. *Artificial intelligence*, 13(1-2): 81–132, 1980.

[5] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial intelligence*, 13(1-2):27–39, 1980.

[6] Drew McDermott and Jon Doyle. Non-monotonic logic I. *Artificial intelligence*, 13(1-2):41–72, 1980.

[7] Ken Satoh. *Nonmonotonic reasoning by minimal belief revision.* Japan, 1988.

[8] Edwina L Rissland and Kevin D Ashley. A case-based system for trade secrets law. In *Proceedings of the 1st international conference on Artificial intelligence and law*, pages 60–66, New York, NY, USA, 1987.

[9] Edwina L Rissland and David B Skalak. CABARET: rule interpretation in a hybrid architecture. *International journal of man-machine studies*, 34(6):839–887, 1991.

[10] L Karl Branting. Building explanations from rules and structured cases. *International journal of man-machine studies*, 34(6):797–837, 1991.

[11] Katsumi Nitta, Yoshihisa Ohtake, Shigeru Maeda, Masayuki Ono, Hiroshi Ohsaki, and Kiyokazu Sakane. Helic-ii: legal reasoning system on the parallel inference machine. *New generation computing*, 11(3-4):423–448, 1993.

[12] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[13] Douglas N Walton. *Argumentation schemes for presumptive reasoning*. 1996.

[14] Ronald Prescott Loui and Jeff Norman. Rationales and argument moves. *Artificial Intelligence and Law*, 3(3):159–189, 1995.

[15] Henry Prakken and Giovanni Sartor. Modelling reasoning with precedents in a formal dialogue game. In *Judicial applications of artificial intelligence*, pages 127–183. 1998.

[16] Edwina L Rissland and M Timur Friedman. Detecting change in legal concepts. In *Proceedings of the 5th international conference on Artificial intelligence and law*, pages 127–136, New York, NY, USA, 1995.

[17] Donald H. Berman and Carole D. Hafner. Understanding precedents in a temporal context of evolving legal doctrine. In *Proceedings of the 5th International Conference on Artificial Intelligence and Law*, ICAIL '95, page 42–51, New York, NY, USA, 1995.

[18] Alison Chorley and Trevor Bench-Capon. AGATHA: Using heuristic search to automate the construction of case law theories. *Artificial Intelligence and Law*, 13 (1):9–51, 2005.

[19] Martin Možina, Jure Žabkar, Trevor Bench-Capon, and Ivan Bratko. Argument based machine learning applied to law. *Artificial Intelligence and Law*, 13(1): 53–73, 2005.

[20] Maya Wardeh, Trevor Bench-Capon, and Frans Coenen. Padua: a protocol for argumentation dialogue using association rules. *Artificial Intelligence and Law*, 17 (3):183–215, 2009.

[21] Carole D Hafner and Donald H Berman. The role of context in case-based legal reasoning: teleological, temporal, and procedural. *Artificial Intelligence and Law*, 10(1):19–64, 2002.

[22] Trevor J. M. Bench-Capon. Value-based argumentation frameworks. In *Proceedings of Non Monotonic Reasoning*, pages 443–454, 2002.

[23] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[24] Katie Atkinson, Trevor Bench-Capon, and Danushka Bollegala. Explanation in AI and law: Past, present and future. *Artificial Intelligence*, 289, 2020.

[25] John F Horty and Trevor JM Bench-Capon. A factor-based definition of precedential constraint. *Artificial intelligence and Law*, 20(2):181–214, 2012.

[26] Adam Rigoni. An improved factor based approach to precedential constraint. *Artificial Intelligence and Law*, 23(2):133–160, 2015.

[27] Henry Prakken. A formal analysis of some factor-and precedent-based accounts of precedential constraint. *Artificial Intelligence and Law*, pages 1–27, 2021.

[28] Bart Verheij. Correct grounded reasoning with presumptive arguments. In *European Conference on Logics in Artificial Intelligence*, pages 481–496, Cham, 2016.

[29] Bart Verheij. Formalizing value-guided argumentation for ethical systems design. *Artificial Intelligence and Law*, 24(4):387–407, 2016.

[30] Bart Verheij. Formalizing arguments, rules and cases. In *Proceedings of the 16th Edition of the International Conference on Articial Intelligence and Law*, ICAIL '17, page 199–208, New York, NY, USA, 2017.

[31] Paul Bratley, Jacques Frémont, Ejan Mackaay, and Daniel Poulin. Coping with change. In *Proceedings of the 3rd International Conference on Artificial Intelligence and Law*, ICAIL '91, page 69–76, New York, NY, USA, 1991.

[32] Ehud Y Shapiro. The model inference system. In *IJCAI*, page 1064, 1981.

[33] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Machine Learning Proceedings 1988*, pages 339–352. 1988.

[34] Masaki Kurematsu, Masayoshi Tada, and Takahira 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*, volume 95, 1995.

[35] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In *Inductive Logic Programming*, pages 91–97, Berlin, Heidelberg, 2012. ISBN 978-3-642-31951-8.

[36] Tingting Li, Tina Balke, Marina De Vos, Julian Padget, and Ken Satoh. A model-based approach to the automatic revision of secondary legislation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law*, ICAIL '13, pages 202–206, New York, NY, USA, 2013.

[37] Paul Bratley, Daniel Poulin, and Jacques Savoy. The effect of change on legal applications. In *Database and Expert Systems Applications*, pages 436–441, Vienna, 1991.

[38] Ehud Y. Shapiro. *Algorithmic Program DeBugging*. Cambridge, MA, USA, 1983.

[39] Wachara Fungwacharakorn and Ken Satoh. Legal debugging in propositional legal representation. In *JSAI International Symposium on Artificial Intelligence*, pages 146–159, Cham, 2018.

[40] Wachara Fungwacharakorn, Kanae Tsushima, and Ken Satoh. Resolving counterintuitive consequences in law using legal debugging. *Artificial Intelligence and Law*, pages 1–17, 2021.

[41] Wachara Fungwacharakorn and Ken Satoh. Culprit resolution for legal debugging in first-order PROLEG. In *Proceedings of the Thirteenth International Workshop on Juris-informatics (JURISIN 2019)*, volume 2019, November 2019.

[42] Wachara Fungwacharakorn, Kanae Tsushima, and Ken Satoh. On the legal revision in PROLEG program. In *34th Proceedings of the Annual Conference of JSAI*, volume 2020, pages 3G5ES104–3G5ES104, Japan, 2020.

[43] Wachara Fungwacharakorn and Ken Satoh. Generalizing culprit resolution in legal debugging with background knowledge. In *Legal Knowledge and Information Systems - JURIX 2020: The Thirty-third Annual Conference, Brno, Czech Republic, December 9-11, 2020*, volume 334 of *Frontiers in Artificial Intelligence and Applications*, pages 52–62, 2020.

[44] Wachara Fungwacharakorn, Kanae Tsushima, and Ken Satoh. On semantics-based minimal revision for legal reasoning. In *Proceedings of the 18th international conference on Artificial intelligence and law - ICAIL '21*, NY, USA, 2021.

[45] Wachara Fungwacharakorn, Kanae Tsushima, and Ken Satoh. Effects of legal revision in PROLEG. In *35th Proceedings of the Annual Conference of JSAI*, volume 2020, Japan, 2021.

[46] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[47] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[48] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

[49] Gordon D Plotkin. A note on inductive generalization. *Machine intelligence*, 5(1): 153–163, 1970.

[50] Kenneth Kunen. Signed data dependencies in logic programs. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1987.

[51] Krzysztof R Apt, Howard A Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of deductive databases and logic programming*, pages 89–148. Burlington, MA, USA, 1988.

[52] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, volume 88, pages 1070–1080, Cambridge, MA, USA, 1988.

[53] Hisao Tamaki and Taisuke Sato. Unfold/fold transformation of logic programs. *Proceedings of the Second International Conference on Logic Programming*, pages 127–138, 1984.

[54] Alberto Pettorossi and Maurizio Proietti. Transformation of logic programs: Foundations and techniques. *The Journal of Logic Programming*, 19:261–320, 1994.

[55] Gordon Plotkin. *Automatic methods of inductive inference*. 1972.

[56] Michael Bain and Stephen Muggleton. Non-monotonic learning. *Inductive logic programming*, 38:145–153, 1992.

[57] Chiaki Sakama. Nonmonotomic inductive logic programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 62–80. Springer, 2001.

[58] Bart Verheij. Dialectical argumentation with argumentation schemes: An approach to legal logic. *Artificial intelligence and Law*, 11(2):167–195, 2003.

[59] Guido Governatori, Michael J Maher, Grigoris Antoniou, and David Billington. Argumentation semantics for defeasible logic. *Journal of Logic and Computation*, 14(5):675–702, 2004.

[60] Ken Satoh, Satoshi Tojo, and Yoshitaka Suzuki. Formalizing a switch of burden of proof by logic programming. In *Proceedings of the First International Workshop on Juris-Informatics (JURISIN 2007)*, pages 76–85, 2007.

[61] Ken Satoh, Masahiro Kubota, Yoshiaki Nishigai, and Chiaki 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*, pages 162–171, Amsterdam, The Netherlands, 2009.

[62] Shigeo Ito. *Basis of Ultimate Facts*. 2001.

[63] Ken Satoh, Kento Asai, Takamune Kogawa, Masahiro Kubota, Megumi Nakamura, Yoshiaki Nishigai, Kei Shirakawa, and Chiaki Takano. PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology. In *New Frontiers in Artificial Intelligence*, Lecture Notes in Computer Science, pages 153–164, Berlin, Heidelberg, 2011.

[64] Ken Satoh, Takamune Kogawa, Nao Okada, Kentaro Omori, Shunsuke Omura, and Kazuki Tsuchiya. On generality of proleg knowledge representation. In *Proceedings of the 6th International Workshop on Juris-informatics (JURISIN 2012), Miyazaki, Japan*, pages 115–128, 2012.

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

[66] Trevor JM Bench-Capon. Hypo' s legacy: introduction to the virtual special issue. *Artificial Intelligence and Law*, 25(2):205–250, 2017.

[67] Kristijonas Cyras, Ken Satoh, and Francesca Toni. Abstract argumentation for case-based reasoning. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 243–254, CA, USA, 2016.

[68] Duangtida Athakravi, Ken Satoh, Mark Law, Krysia Broda, and Alessandra Russo. Automated inference of rules with exception from past legal cases using ASP. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 83–96, Cham, 2015.

[69] Edward H Levi. *An introduction to legal reasoning*. Chicago, USA, 2013.

[70] Kevin D Ashley. *Artificial intelligence and legal analytics: new tools for law practice in the digital age*. Cambridge, England, 2017.

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

[72] Latifa Al-Abdulkarim, Katie Atkinson, and Trevor Bench-Capon. Accommodating change. *Artificial Intelligence and Law*, 24(4):409–427, 2016.

[73] John Henderson and Trevor Bench-Capon. Describing the development of case law. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law*, ICAIL '19, pages 32–41, New York, NY, USA, 2019.

[74] Antonino Rotolo and Corrado Roversi. Constitutive rules and coherence in legal argumentation: The case of extensive and restrictive interpretation. In *Legal Argumentation Theory: Cross-Disciplinary Perspectives*, pages 163–188. Dordrecht, The Netherlands, 2013.

[75] Ryuta Arisaka. A belief revision technique to model civil code updates. In *JSAI International Symposium on Artificial Intelligence*, pages 204–216, Cham, 2015.

[76] Tokyo High Court 1994 (O) 693. Case to seek removal of a building and surrender of lands, October 1996. URL http://www.courts.go.jp/app/hanrei_en/detail?id=273.

[77] Henry Prakken and Giovanni Sartor. A dialectical model of assessing conflicting arguments in legal reasoning. In *Logical models of legal argumentation*, pages 175–211. 1996.

[78] Michał Araszkiewicz and Krzysztof Płeszka. *The Concept of Normative Consequence and Legislative Discourse*, pages 253–297. Legisprudence Library. 2015. ISBN 978-3-319-19575-9.

[79] Ronny Frith. Preventing and avoiding loopholes and unintended consequences in legislation, October 2012. URL http://www.ncsl.org/documents/lsss/spotloophole1.pdf.

[80] Douglas Walton, Giovanni Sartor, and Fabrizio Macagno. *Statutory Interpretation as Argumentation*, pages 519–560. Dordrecht, 2018.

[81] Rafael Caballero, Adrián Riesco, and Josep Silva. A Survey of Algorithmic Debugging. *ACM Computing Surveys*, 50(4):60:1–60:35, August 2017.

[82] Claus Zinn. Algorithmic Debugging for Intelligent Tutoring: How to Use Multiple Models and Improve Diagnosis. In *KI 2013: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 272–283, 2013.

[83] Kizysztof Kuchcinski, Wlodzimierz Drabent, and Jan Maluszynski. Automatic diagnosis of vlsi digital circuits using algorithmic debugging. In *Automated and Algorithmic Debugging*, Lecture Notes in Computer Science, page 350–367, 1993. ISBN 978-3-540-48141-6.

[84] Jiro Naganuma, Takeshi Ogura, and Tamio Hoshino. High-level design validation using algorithmic debugging. In *Proceedings of European Design and Test Conference EDAC-ETC-EUROASIC*, page 474–480, Feb 1994.

[85] Chiaki Sakama. Some properties of inverse resolution in normal logic programs. In *International Conference on Inductive Logic Programming*, pages 279–290. Springer, 1999.

[86] Luc De Raedt and Maurice Bruynooghe. On negation and three-valued logic in interactive concept-learning. In *Proceedings ECAI90: 9th European Conference on Artificial Intelligence*, pages 207–212, 1990.

[87] Martin Brain and Marina De Vos. Debugging logic programs under the answer set semantics. In *Answer Set Programming*, 2005.

[88] R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez. A theoretical framework for the declarative debugging of datalog programs. In *Semantics in Data and Knowledge Bases*, Lecture Notes in Computer Science, page 143–159, 2008. ISBN 978-3-540-88594-8.

[89] Tommi Syrjänen. Debugging inconsistent answer set programs. In *Proc. NMR*, volume 6, page 77–83, 2006.

[90] Claudia Schulz, Ken Satoh, and Francesca Toni. Characterising and explaining inconsistency in logic programs. In *Lecture Notes in Computer Science*, page 467–479, 2015. ISBN 978-3-319-23264-5.

[91] Markus Ulbricht, Matthias Thimm, and Gerhard Brewka. Measuring inconsistency in answer set programs. In *Logics in Artificial Intelligence*, Lecture Notes in Computer Science, page 577–583, 2016. ISBN 978-3-319-48758-8.

[92] Martin Caminada and Chiaki Sakama. On the existence of answer sets in normal extended logic programs. In *ECAI 2006: 17th European Conference on Artificial Intelligence*, volume 141, page 743–744, 2006.

[93] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Catching the ouroboros: On debugging non-ground answer-set programs. *Theory and Practice of Logic Programming*, 10(4–6):513–529, 2010.

[94] Andrew Cropper and Sebastijan Dumancic. Inductive logic programming at 30: a new introduction. *CoRR*, abs/2008.07912, 2020.

[95] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[96] Domenico Corapi, Alessandra Russo, Marina De Vos, Julian Padget, and Ken Satoh. Normative design using inductive learning. *Theory and Practice of Logic Programming*, 11(4-5):783–799, 2011.

[97] Michael P Healy. Communis opinio and the methods of statutory interpretation: Interpreting law or changing law. *Wm. & Mary L. Rev.*, 43:539, 2001.

[98] Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[99] James Wogulis and Michael J Pazzani. A methodology for evaluating theory revision systems: Results with audrey ii. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1128–1134, San Francisco,CA,USA, 1993.

[100] Bart Verheij. About the logical relations between cases and rules. In *Proceedings of the 2008 Conference on Legal Knowledge and Information Systems: JURIX 2008: The Twenty-First Annual Conference*, page 21–32, Amsterdam, The Netherlands, 2008.

[101] Henry Prakken. A tool in modelling disagreement in law: preferring the most specific argument. In *Proceedings of the 3rd international conference on Artificial intelligence and law*, pages 165–174, New York, NY, USA, 1991.

[102] Jorge Fandinno and Claudia Schulz. Answering the "why" in answer set programming - a survey of explanation approaches, 2018.

[103] English Court R v Allen (1872). Case of bigamy, 1872.

[104] Critical Analysis of the Literal, Golden & Mischief Rule, October 2019. URL https://www.lawteacher.net/free-law-essays/administrative-law/critical-analysis-of-the-literal-golden-and-mischief-rule-law-essay.php.