COMP4805 – Honours Thesis

Topic:	**Induction of Defeasible Logic Theories in the Legal Domain**
Student:	Benjamin Johnston
Number:	33551715
Email:	s355171@student.uq.edu.au
Supervisor:	Guido Governatori
Degree:	Bachelor of Information Technology
Course:	COMP4805

Statement of Originality

I declare that the work presented in the thesis is, to the best of my knowledge and belief, original and my own work, except as acknowledged in the text, and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

Benjamin Johnston

Acknowledgements

I'd like to thank the following people who are among those who have helped make this thesis what it is;

- **My Saviour: Jesus Christ**
  For making me who I am, and inspiring me to try to be a better person.
- **My Supervisor: Guido Governatori**
  For encouraging me to explore all sorts of interesting tangents, helping me get more out of Honours than I would have otherwise expected.
- **My Family: Graham, Sue, Michael**
  For providing the loving support through all the years of my life.
- **My Closest Friends: Jeremy Jones, Owen Thomas, Andrew Whitby**
  For being there for me.
- **My Fellow Students**
  For making the honours labs *the* place to be when you need a break from work.
- **My Second Examiner: Ralf Muhlberger**
  For being the supremely intelligent, inspirational, witty and cultured person that he is.
- **The ITEE Information Systems Group**
  For entertaining a thesis that ended up having barely tenuous connections to database technology, and for offering helpful suggestions during my seminars.

Publication Arising from this Work

- Benjamin Johnston and Guido Governatori. An Algorithm for the Induction of Defeasible Logic Theories from Databases. Accepted for the *Fourteenth Australasian Database Conference*, 2003.

Abstract

The market for intelligent legal information systems remains relatively untapped, but it is a domain for which information systems could offer dramatic cost reductions whilst simultaneously improving accuracy. Even though there does not appear to be any technical or ethical reasons that might make the development of such systems infeasible, none of the recorded attempts in this domain have reported overwhelming successes. An analysis suggests that the fundamental failure of these prior attempts is a poor balance between powerful knowledge representations and efficient methods of encoding knowledge – a problem that can be solved with defeasible logic.

Defeasible logic is a non-monotonic logic with proven successes in representing legal knowledge that overcomes many of the deficiencies of prior efforts. Unfortunately, an immediate application of technology to the challenges in this domain is an expensive and computationally intractable problem. So, in light of the potential benefits, we have developed a practical algorithm that uses heuristics to discover approximate solutions to the challenges. The algorithm integrates defeasible logic into a decision support system by automatically deriving its knowledge from databases of precedents. Experiments with the new algorithm are very promising – delivering results comparable to and exceeding other approaches.

Integrating this work into a meaningful operational context is a difficult challenge, but that can be solved using semi-structured data formalisms such as XML. Some work has been done in this area, but much of the routine implementation remains. However, the future of this work is extremely promising and worthwhile of further research and development.

Contents

List of Figures

Chapter 1

Introduction

Law is still a domain with relatively low penetration of intelligent information systems, but clearly one in which such systems would be immediately beneficial. The growing pressure on the legal system to reach conclusions with accuracy and expediency, the need to increase public confidence in the consistency and equality of the legal process, and also the need for providing quality advice to those considering beginning legal proceedings are the motivations for producing legal information systems. The legal domain has several peculiarities, namely the importance of extremely high accuracy, the value of judicial independence and the need for presenting both the decisions and the rationale in a language that is natural to its users. These peculiarities are constraints on the development of information systems that, to date, have yet to be adequately overcome.

While technology has made significant in-roads to the legal profession, much of its application is in text storage and retrieval as opposed to intelligent information systems. While some projects have achieved minor successes in attempting to produce more powerful systems that aid the legal decision making process, the legal community has not whole-heartedly taken up these technologies (they do not form a common part of a lawyer's tool-set). This project seeks to produce a system that provides valuable assistance to the decision making process of law-makers – this is achieved by providing a tool that offers advice, and reasoning that supports the advice, given information about current legal proceedings.

1.1 Intended Outcomes

The development of intelligent legal information systems should offer the legal community benefits that are difficult to realise without computer technologies. Thus, we direct our efforts to achieve three major goals:

1. **To Assist in Accurate and Expedient Decision Making**
   The importance of a case's outcome and the influence it might have on an individual's life, mean that the utmost care is required in reaching any decision. Unfortunately, there are contradicting pressures on the legal system (and on any modern organisation) to produce accurate outcomes faster and at lower costs.

2. **To Help Reach Consistent Decisions**
   A challenge in the legal system is to address the public's growing disillusionment regarding the equality and consistency of decisions. A legal information system could assist decision makers in ensuring the consistency of the decision making process across different cases, judges and states or districts. This consistency could be achieved by presenting the rationale and outcome of similar cases, or by allowing for the analysis of trends in the adjudications of the decision-making processes. Of course, law is an evolutionary process – this means that two virtually identical cases that occur in at different times (and therefore in different social contexts) might result in entirely opposite adjudications. A legal information system should therefore not seek to prevent all change, but it is important that when law workers deviate from established practice and precedent they do so in full knowledge of that fact.

   Another possible application of a legal information system is to aid decision makers's introspective analysis of their own possible biases or prejudices. A decision maker could attend to the undesirable trends (such as race or gender bias) that the information system may have detected within precedents.

3. **To Provide Good Advice and Information at a Low Cost**
   Whether information is needed by legal professionals or by a member of the public considering legal action, the unstructured nature of existing legal

databases makes querying and inference with such data difficult and labour-intensive. By structuring legal information, and developing information systems that can make inferences about new cases, the legal system could use these tools to provide better and quicker adjudications than otherwise possible.

Even though legal information systems are to assist in the decision making process, judicial independence and the need for compassion and humanity in legal decision making is such that we do not intend to replace the role of the judge – but simply assist him or her in producing a more accurate and equitable decision. For this reason, the ultimate goal of this project is best termed as the construction of a legal *decision support* system (LDSS).

1.2 Assumed Requirements

While a law worker using a legal decision support system would recognise that no system can be faultless and would therefore carefully consider any conclusions that the system reaches, it is still important that the system does indeed produce fair decisions and operates with high accuracy so that it provides value to its users. We therefore have to consider the unusual "requirements" in the legal domain:

1. **Conclusions Presented with Justifications**
   Unlike domains where a correct answer "speaks for itself", in law it is necessary to provide an answer as well as the justification and/or the reasoning behind the answer. A decision maker can use the justifications to verify the correctness of a decision, and users considering entering legal proceedings can consider what factors were used in the decision making process and thereby possibly invest time and effort into rectifying deficiencies in their argument.

2. **Justifications Presented in Lay-man's Terms**
   Whereas in technical fields, such as engineering or mathematical sciences, there can be a reasonable expectation of proficiency and technical understanding of computers, programming and mathematics, such assumptions

can not be made of the typical user of an LDSS. Even though it is desirable to build an LDSS on a strong theoretical basis[1], it is inappropriate to require that users understand the theory before they can correctly use the tool. Lawyers and judges do not undertake courses in formal logic (and it is much less likely for a member of the general public to have undergone such training), so presentation of arguments must be carefully considered so as to be clear and natural to the users of the LDSS.

3. **A Verifiable and Comprehensible Internal Representation**
   In order that its users understand the strengths and weaknesses of the application of an LDSS, it is necessary to provide some means by which the internal representation of the system can be inspected, verified and validated. Ideally, it should be possible for the users of the system to understand the representation without technical assistance.

4. **A Conservative Mode of Operation**
   The importance of correctness is such that where the system is unable to predict an answer correctly, it is better to give *no answer* than to incorrectly and inconsistently produce *an answer*.

5. **A Low Cost Method of Construction**
   For a system to be adopted by a rational user, its benefits must outweigh its costs. By reducing production costs, the barrier to entry is reduced and with sufficiently low costs it is possible to regularly reconstruct or maintain the information system so as to remain consistent with current legal practice.

The difficulty in producing legal decision support systems is that these requirements are somewhat contradictory. For example, while automatically generating knowledge bases might alleviate the problem of expense (as a result of less need for expert knowledge in generating and structuring the data), this method of construction typically reduces the confidence that users have in the soundness of the system. Similarly, heuristics and large sets of training examples might improve the accuracy and applicability of the system at the cost of a sound and verifiable reasoning method or the ability to examine and comprehend the underlying reasoning involved in reaching a given conclusion.

[1]This is because a theoretical basis allows us to reason (and prove theorems) about a tool in the abstract theory, without concern for the trivial details of the specific form of implementation.

In spite of the inherent difficulties, the potential benefits to the legal community make LDSSs a worthwhile area for research. This project therefore aims to investigate these issues and develop technical solutions to the problem of processing and reasoning with legal databases to produce legal decision support systems.

Chapter 2

Foundation and Context

We are faced with three serious questions that must be addressed when attempting to create a legal decision support system:

1. Is it meaningful, from a jurisprudential perspective, to create a legal decision support system?
2. Can we structure input and data in a way that makes a legal decision support system possible?
3. Is it even ethical to create a legal decision support system?

2.1 Jurisprudential Justification

While there is some contention in the jurisprudential context as to whether law operates in a way that can even be encoded as rules and knowledge that a computer can work with [48], there is (to date) no way of proving or disproving the possibility of constructing such systems without actually attempting to construct them.

Admittedly, the need for compassion and humanity, and the difficulty of interpreting law gives rise to cases that are far beyond the capabilities of any existing computer systems to reason about. But even though we cannot handle every such case, it is understood that in many fields of legal practice, (particularly in statutory domains such as taxation and traffic law, but also common law such as contracts) legislation and precedents are well understood and the typical cases before the courts are "routine". In a case involving traffic law, for example, many

common attributes can be readily identified: the blood-alcohol level of the driver, the speed of the car, the crime committed, the experience of the driver and the importance of a drivers licence for the driver's livelihood. Of the thousands of traffic cases before the courts, there might only be 100 or so different classes of 'incidents' and the relevant features of these are likely to be readily identifiable.

Thus, we have made this assumption that most cases are not "landmark" decisions, but that much of the decision making process of legal practice is routine and potential scope for at least partial automation. While we cannot expect a legal decision support system to draw accurate conclusions for every case, we believe it will suffice to have the tool offer recommendations where it recognises the case is within its scope (i.e., the tool has a conservative mode of operation, that offers suggestions when it would be accurate and helpful). The decision maker can use the recommendations that the tool generates as additional considerations in an adjudication.

2.2 Structuring Input

Given this routine domain of application, it seems natural that a legal decision support system that takes an encoded form of the facts that are currently before a court (or the situation of a user considering legal action) and returns suggestions that are consistent with precedents along with justifications for the suggestions would prove to be beneficial to the legal community.

Implicit in this claim, is the assumption of the availability of such structured datasets that encode the facts of precedents. Unfortunately, there are presently few structured datasets available[1] in the legal domain, but we do not see this as a problem for two reasons:

1. Generating datasets is a low cost task that can be performed accurately by relatively unskilled workers who can read the unstructured description of a decision and identify key attributes of the relevant parties. This is cheaper and the resultant system is likely to be more accurate than that of having

[1] And the structured datasets that are available tend to describe a few landmark cases, as opposed to large datasets of routine cases that would be more useful for automatically inferring the underlying reasoning.

an expert manually infer the reasoning used by the decision maker (this will be addressed further in the next chapter).

2. As the value of legal decision support systems are recognized, the legal system is likely to respond by entering data into structured or semi-structured formats suitable for machine processing. Efforts such as the Oasis' LegalXML project[2] are already exploring such options.

We admit that the assumption of structured relational datasets is slightly unrealistic. The value of a legal decision support system is such that we have decided to maintain the assumption in order to drive the creation of such datasets based on demand. We will explore this assumption further, and suggest some practical solutions in Chapter 7.

2.3 Ethical Considerations

Any legal technology, by its nature, has the potential to dramatically affect the lives of people involved in the decisions. Thus, we deviate from the technical aspects to consider the ethical implications of what we are doing:

2.3.1 Possible Elimination of Compassion

Whenever regulation is read with a strictly literal interpretation and followed precisely, nonsensical situations arise. Consider, for example, the ridiculous cases of utility companies that send sternly worded letters demanding compensation for a bill 5 cents short, or mandatory sentencing that sends a youth to jail for shoplifting a chocolate bar. When constructing an LDSS, care has to be taken to ensure that we give the system 'common-sense', or that its users are trained to recognise that sometimes human-compassion or special circumstances demand that precedent is ignored.

An extension of this concern is that if a legal decision support system proved to be too successful, there might be moves to replace judges entirely in the name of efficiency. If such a situation were to arise, there would be serious implications for the compassion of the legal system (and also for the system to accommodate changing social contexts).

[2]Project homepage at `http://www.legalxml.org/`

Unfortunately, the capabilities of modern technology are often misunderstood – if we were proposing to create a printed table that can be used to suggest outcomes of a case there would be far less concern for its misuse. But, as a computer program, there is a danger of people who do not belong to the information and communication technology industries to overrate the intelligence of such an LDSS.

Because there are so many exceptional cases in law, we do not believe that these concerns are likely to materialize. No tool can be successfully applied to every single case because of the massive (and increasingly growing) domain over which it would have to operate. Through the fallability of the machine, and its conservative answers (i.e., it would indicate it could not give an answer under some circumstances), users will develop an understanding that the tool can be used for the routine, but that it will always require a human to handle the exceptions. In fact, it will always require a human to identify that a case is indeed routine[3]. Furthermore, with careful attention to marketing the tool as a *tool*, as opposed to a *solution*, users can be taught realistic expectations of the capabilities of a legal decision support system: providing the benefits of low cost advice without destroying the existing legal processes that we value.

2.3.2 Possible Prevention of Evolutionary Change

Social conditions change; while homosexuality is no longer considered a crime, to carry a pocketknife on an airplane is becoming increasingly punishable. For this reason, it is essential that an LDSS that draws inferences based on precedent does not limit the ability of the legal system to adapt to the changing perceptions and needs of the community. This concern can be addressed, in part, by placing greater significance on more recent precedents, and by educating users in the proper use of the system so they pay attention to the timeliness of advice of the system. In other words, we do not believe that a legal decision support system will prevent evolutionary change, in fact, it will empower decision makers with a far greater understanding of how their decisions are deviating from precedent

[3]Even though a human is required to identify the applicability of the tool, it does not necessarily mean that the tool is useless – a decision maker can readily recognise where the input parameters of the LDSS do not adequately describe the case, and in those cases where the input parameters do adequately describe it, the LDSS provides assistance in reaching consistent and rapid decisions.

and whether this is appropriate.

2.3.3 Possiblity of Unequal Availability and Subversion

Like virtually all technologies, a legal decision support system could be used for devious ends. Not only can innocent parties use the strategic information provided by an LDSS in a positive way to structure their case to maximize their success, but also guilty parties could potentially use the same information to subvert the legal process. This is a concern, but it is a reasonable price to pay:

1. Our modern legal system is such that every person has every opportunity to demonstrate his or her innocence. An LDSS is perfectly within the rights and fair use of any party. At worst, an LDSS would simply increase the burden of proof that lies with the accuser.

2. While a guilty party could use the tool to subvert the legal process, it seems more likely that the widespread availability of tools that can analyse reasoning in precedents would assist a decision maker in recognising this subversion and in reaching a decision that is consistent with the entire established body of precedents. That is, while a guilty party might be somewhat benefited by the information provided by an LDSS, decision makers and innocent persons would receive at least the same benefits, but in all likelihood, significantly more dramatic benefits.

3. A low cost LDSS can help "level the playing field" by giving both sides the same access to information. When legal knowledge is widely available through low cost legal information systems, there is a reduction in the advantage that one party can gain in having the finances to afford top lawyers.

Point 3 above addresses to the concern that an LDSS might only be available to the wealthy. In fact, underlying this concern is one of the strongest motivations for this work: wealthy parties can already afford the top lawyers to represent their case so that an LDSS would do little to improve their advantage, but because an LDSS can be built at a low cost, there is real potential for offering high-quality advice to less wealthy parties and thereby making the legal system more equitable.

2.3.4 Possibility for Invasion of Privacy and Security

The concerns of privacy and security, while always important, would not be exacerbated by the introduction of LDSSs. Precedents are freely available in unstructured forms; converting them to structured formats, while giving due care to existing privacy regulations is not likely to be a detriment to privacy. Finally, the need for security – that decision makers should not be influenced by systems that have had their precedent data tampered with – is indeed a concern, but is somewhat alleviated by the fact that a human decision maker should use the LDSS as a tool, and should only accept a suggestion of the LDSS when the rationale it presents is sound and sensible. By offering legal decision support systems to public scrutiny (and this is certainly more possible if our requirement for transparent and verifiable internal representation is satisfied), the confidence in the system can only improve.

2.3.5 Conclusions

Undeniably, the introduction of legal decision support systems brings to mind a minefield of ethical considerations. While there is potential for subversion, the benefits are such that it is still worthwhile to cautiously proceed – taking care to educate decision makers and the general public in how to correctly interpret and use the suggestions of the LDSS.

Chapter 3

Related Work

Existing approaches to constructing legal expert systems fall into two broad categories: manual encoding of rules in expert systems, or automatic induction via machine learning techniques. Manually encoded expert systems have the luxury of being able to use expressive and verifiable internal representations, but it is an expensive and delicate operation to produce such expert systems. In fact, manually encoding knowledge into an expert system may result in a system that is inconsistent with legal practice due to the difficulty in having experts accurately encode their knowledge. While there have been many successful implementations of legal expert systems [48] some of which can efficiently draw inferences from over 10,000 rules, we have ruled out manual construction for its exorbitant costs and focus our attention to automatic and semi-automatic means. But, while machine learning can reduce the construction cost, existing systems based on such technologies appear to be unable to reconcile automatic construction of knowledge bases with knowledge representations that suitably encode knowledge for reasoning. The following sections explore some of these existing approaches.

3.1 Case Based Reasoning

Case based reasoning systems (such as Hypo [48]) attempt to find related cases under the assumption that a similar case is likely to have a similar conclusion. While such an information system certainly would benefit a lawyer in preparing an argument, the systems do not focus on the rationale of a decision but on the similarity between the cases, so it is difficult for a case based reasoning system to

produce a justification for its predictions other than by analogy. Some systems attempt to resolve this difficulty by manually inferring the principles underlying the case, but this is really no more than producing an expert system with analogical reasoning capabilities (and so is as expensive as other approaches to manually constructing expert systems).

3.2 Attribute-Value and Propositional Learners

Propositional learners such as the ID3 and C4.5 decision tree induction algorithms or ASSISTANT have had some success when applied in the legal domain [17]. Unfortunately, propositional rules are difficult to interpret by a lawyer without a computer science background, and even with training, the resultant trees or propositional expressions are still difficult to interpret. Furthermore, the output of such algorithms are typically expressions that seek to classify cases with as few attributes as possible, but do not adequately handle the possibility that those attributes may not even be available at all (i.e., propositional learners do not work satisfactorily with partial information) [47].

Modern algorithms such as C4.5 often have additional modes of operation that output a structured set of rules more suitable for "human interpretation" [45], but these are based on monotonic[1] logics that do not have a natural correspondence to law, and in practice we have found that they result in unwieldy rules that are challenging to interpret.

3.3 Neural Networks, Bayesian Networks and Other Continuous Model Fitting

Projects such as the Split-Up project [49] which applied neural networks to family law, often achieve a high degree of success. But unfortunately, while approaches

[1] A monotonic logic is a logic that does not allow for tentative conclusions. For example, it is possible to prove that $2 + 2 = 4$ in the standard monotonic first order logic with the axioms of formal arithmetic, and no additional information will nullify the proof. In contrast, we might say that law is a non-monotonic logic, because given no information we can "prove" innocence of an accused (i.e., the presumption of innocence), but if video evidence is presented to show guilt, then it is necessary to revise this conclusion (in fact, the new conclusion of guilt can still remain tentative because it may be revealed, for example, that the video was doctored).

such as neural networks that perform model fitting over continuous variables give very high accuracy by learning from examples and have proven successes in disparate fields, they do not apply as successfully in a legal context. The reasoning behind such methods is simply an adaptive non-linear function and, as such, it is extremely difficult to generate explanations from their output, or even to verify their correctness.

Some researchers have attempted to extract knowledge as **if** − **then** rules from neural networks [13, 9] by analysing connection strengths and by random sampling of outputs. Such analysis helps improve the confidence in the soundness of a neural network, but the researchers acknowledge that a neural network still remains a black-box and it is difficult to estimate the applicability of a network to new cases.

3.4 Association Rules

Some work has been done towards using association rules (mined by the Apriori algorithm [1]) as a basis for an LDSS. While this is currently work-in-progress (and, in fact, was the initial motivation for this project [31]), it seems unlikely that taking this approach alone is going to result in significant success. Association rule mining finds "general associations" between properties, but does not produce rules. A specific field in the legal domain might have several guiding rules with many exceptions and so is guided by not only the associations between assumptions and a conclusion, but also exceptions to the associations – because of this, association rule miners are unlikely to be a general solution to the problem of learning the reasoning principles underlying law. Furthermore, care must be taken to ensure that mined associations are not simply common properties of the domain, but are indeed deciding factors. These difficulties with interpreting association rules, and the challenge of integrating them into a formal model suggests that association rules might be better suited as a heuristic device as opposed to an end in itself.

3.5 Inductive Logic Programming

Inductive logic programming (ILP) is concerned with producing pure negation-free Prolog programs (theories in Horn clause logic) from example sets of inputs and outputs, and usually operate in either a bottom-up approach by generalising examples or a top-down approach specialising the empty program into a suitable theory [34]. In the literature, there appears to be limited successful application of ILP in the legal domain, which may be due to the monotonicity of Horn clause logic being unsuitable for law. The simplicity of theories induced by ILP systems in their typical domains is significant, so we have drawn from the body of work in ILP for our research, but have adapted some of the benefits into a framework that handles the defeasibility of law directly.

There is some existing work with non-monotonic ILP that seeks to overcome some of the limitations of the closed world assumption and monotonicity of the logic used within ILP. ILP approaches have been used [23, 40] to induce theories in Reiter's Default logic [43] – a similar strategy [33] has also been used to learn Extended Logic programs (which represent a subset of default logic). While these approaches are near what we are attempting to achieve, finding the extension of a Default logic theory is an NP-complete problem so even the application of Default logic in predicting new cases gives rise to computational problems. This is compounded by the fact that Default logic is a form of expression that is difficult for untrained persons to interpret [43] – we believe that our work fits more naturally in a legal context.

Chapter 4

Defeasible Logic

While the existing approaches described in Chapter 3 have experienced a certain degree of success, that none has received widespread acceptance is indicative that they are deficient in some way. The challenge seems to be finding a balance between having a representation for efficient reasoning and the cost of encoding knowledge into that representation. In [43], Prakken describes several non-monotonic logics that appear to have a more natural correspondence to legal reasoning than other forms of formal expression. Furthermore, defeasible logic, a form of non-monotonic reasoning invented by Donald Nute (and related to one of the logics presented by Prakken) has a language that permits the expression of regulations with an almost one-to-one correspondence between plain-language expression and its encoded form [2, 20]. Having a domain expert annotate the rules of a defeasible logic theory with plain-English expressions during a validation phase can further enhance the clarity of a defeasible logic theory to an untrained user: the annotations can be presented to the user as an "explanation" of the conclusion drawn from the theory.

Defeasible logic, being a non-monotonic logic, handles partial knowledge well [43, 30], has a sceptical reasoning process [3, 5], is equivalent to or subsumes many other non-monotonic logics [3, 6] and has an algorithm for computing its extension in linear time and space [4, 35]. These favourable properties are important for the formalisation of legal reasoning. Law abounds in cases involving partial knowledge (there simply is not enough time for every minute detail to be presented), and any reasonable decision support system must act in a conservative and sceptical way (for it is better to give *no* answer than *an* incorrect

Rule				Explanation
$r1$:		$\Rightarrow$	$\neg guilty$	Innocence is presumed
$r2$:	$evidence$	$\Rightarrow$	$guilty$	Evidence can show guilt
$r3$:	$\neg motive$	$\rightsquigarrow$	$\neg guilty$	Lack of motive can suggest innocence
$r4$:	$alibi$	$\Rightarrow$	$\neg guilty$	An alibi can prove innocence
$r4 \succ r3$, $r3 \succ r2$, $r2 \succ r1$				

Figure 4.1: Hypothetical Criminal Law Theory

answer). The computational efficiency is also important, not only for drawing conclusions from a theory, but also because it lends itself to efficient algorithms for the induction of theories from datasets. While the variant of defeasible logic that we have used does not allow for meta-level reasoning about the purpose or backing of rules, which is possible in some logics [43], the logic remains a powerful [2, 20] and natural form of expression that is relevant for routine legal practice and a form for which rules in a defeasible logic theory correspond to the untrained understanding of a "rule".

For these reasons it seems that defeasible logic would be a suitable representation for knowledge in an LDSS. The challenge that remains is the automatic or semi-automatic creation of such knowledge from precedents. This will be addressed in Chapter 5.

In the remainder of this chapter we will present a formal explanation of defeasible logic. Because we are focussing our attention to a specific application of defeasible logic, for simplicity our terminology slightly deviates from that used in other works. More complete definitions of propositional defeasible logic appear in [3, 5, 41].

4.1 An Example

A defeasible logic theory is a collection of rules that permit us to reason about a set of facts, or known truths, to reach a set of defeasible conclusions. Because multiple conflicting rules may be applicable in any given situation, a defeasible logic theory additionally includes a relation for resolving these conflicts.

For example, consider the theory about criminal law in Figure 4.1. The theory consists of two components:

- a set of rules that can be used to conclude the guilt or innocence of the defendant in the event of certain facts being presented in the court of law, and
- an acyclic transitive[1] relation (called the superiority relation) that indicates the relative strength of each rule.

Suppose that we are given the theory of Figure 4.1, and that the set of facts $\{evidence, alibi\}$ have been presented to the court of law and are assumed true, then we can defeasibly prove the innocence of the defendant, because:

- We note that $r4$ permits us to conclude $\neg guilty$, and
- The necessary conditions for the application of $r4$ hold, namely it is known that *alibi* is a fact (or has been defeasibly proven true), and
- Of the remaining rules, the only one that reaches the contradictory conclusion *guilty* and for which its necessary conditions are satisfied, is $r2$, but $r4$ is stronger than $r2$ (i.e., $r4 \succ r2$) so $r2$ does not override the conclusion.

4.2 Formal Definition

We now formalize the ideas and terminology in the example, and present a proof theory.

A defeasible logic theory T is a pair $(R, \succ)$ where R is a finite set of rules and $\succ$ is a partially ordered[2] relation defining superiority over R.

Rules are defined over *literals*, where a literal is either an atomic propositional variable a or its negation, $\neg a$. Given a literal, p, the complement, $\sim p$ of that literal is defined to be a if p is of the form $\neg a$, and $\neg a$ if p is of the form a.

There are two kinds of rules, *defeasible rules* and *defeaters*. *Defeasible rules* can be used to defeasibly prove some conclusion, but *defeaters* can only be used to prevent a conclusion being reached. Typically a third kind of rule is permitted, *strict rules*, which have a more classical meaning in that they are monotonic and

[1] We have only denoted the relevant mappings, the actual superiority relation would in fact be the least acyclic transitive relation containing the mappings denoted – in this case, the transitive closure of these mappings.

[2] Though, in the general case, the superiority relation is simply a binary relation over the set of rules.

cannot be defeated. We disregard strict rules in application to the automatic induction of defeasible theories because it is impossible to conclude a strict correlation with only the partial knowledge possible with finite datasets (in any case, strict rules can be simulated with defeasible rules that are 'high' in the superiority relation such that they can rarely be defeated).

A *defeasible rule* is denoted by $Q \Rightarrow p$ where Q is a set of literals denoting the premises of the rule, and p is a single literal denoting the conclusion upon application of the rule. A rule of this form can be interpreted to say that whenever the literals in Q are known to be facts or to be defeasibly provable, then we can defeasibly prove p (by "defeasibly", we mean that we can only prove p tentatively, and is subject to possible defeat by other, stronger, rules).

A *defeater* is a rule that is likewise denoted by $Q \rightsquigarrow p$ where Q is a set of literals denoting the premises of the rule, and p is a single literal denoting the counter-conclusion that can be used upon application of the rule. A rule of this form can be interpreted to say that whenever the literals in Q are known to be facts or to be defeasibly provable, then we can only reach a conclusion that is consistent with p (subject to defeat by other rules); that is, we cannot prove $\sim p$, but *may* prove p if there are other rules supporting this position, otherwise we may reach no conclusion at all. Note that with a pair of rules of the form $Q \rightsquigarrow a$ and $Q \rightsquigarrow \neg a$, each at the same superiority, we can block any conclusion with respect to a.

We define $Ante(r) = Q$ where r is a rule of the form $Q \Rightarrow p$ or $Q \rightsquigarrow p$; that is, $Ante(r)$ is the set of premises or antecedents of the rule r (i.e., $Ante(r)$ is the left hand side of the rule).

We reason about a set of facts (of a given case) F with respect to a defeasible theory T to reach defeasible or tentative conclusions of that particular case. A conclusion of a defeasible theory T and facts F is conventionally a tagged literal of one of the following forms:

- $+\partial p$, which is intended to mean that p is defeasibly provable in T over F
- $-\partial p$, which is intended to mean that p is not defeasibly provable in T over F.

We define an entailment relation, $T, F \vdash c$, which indicates that c is a conclusion of the set of facts F with respect to theory T. The entailment relation is

defined by the proof mechanism expounded in [3, 5], and which is briefly presented here for completeness.

A proof within a defeasible logic theory T given a set of facts F is a finite sequence $P = \langle p_1, p_2, \ldots \rangle$ of tagged literals satisfying the two inference rules that follow. $P(1..i)$ denotes the initial part of the sequence P, of length i, and $P(i)$ denotes the ith element of P. R_d denotes the set of defeasible rules in R (i.e., those rules that are not defeaters) and $R[q]$ denotes the set of rules in R with conclusion q.

$+\partial$:
If $P(i+1) = +\partial p$ then either
$p \in F$ or
(1) $\exists r \in R_d[p] \forall q \in Ante(r) : +\partial q \in P(1..i)$ and
(2) $\forall s \in R[\sim p]$ either
(a) $\exists q \in Ante(s) : -\partial q \in P(1..i)$ or
(b) $\exists t \in R_d[p]$ such that
$\forall q \in Ante(t) : +\partial q \in P(1..i)$ and $t > s$.

$-\partial$:
If $P(i+1) = -\partial p$ then
$p \notin F$ and
(1) $\forall r \in R_d[p]\ \exists q \in Ante(r) : -\partial q \in P(1..i)$ or
(2) $\exists s \in R[\sim p]$ such that
(a) $\forall q \in Ante(s) : +\partial q \in P(1..i)$ and
(b) $\forall t \in R_d[p]$ either
$\exists q \in Ante(t) : -\partial q \in P(1..i)$ or $t \not> s$.

If we consider only theories for which the set of all possible premises is disjoint from the set of all conclusions, then it is the case that all inferences can be performed in a single step. Because a single-step mode of operation precludes the need for recursive evaluation of the backing of a rule (i.e., there is no need to defeasibly prove the truth of the antecedents, beyond checking facts), we can simplify the above inferences rules to give the following simpler proof mechanism:

$+\partial$:
If $T, F \vdash +\partial p$ then

(1) $\exists r \in R_d[p] \forall q \in Ante(r) : q \in F$ and
(2) $\forall s \in R[\sim p]$ either
    (a) $\exists q \in Ante(s) : \sim q \in F$ or
    (b) $\exists t \in R_d[p]$ such that
        $\forall q \in Ante(t) : q \in F$ and $t > s$.

$-\partial$:
If $T, F \vdash -\partial p$ then
(1) $\forall r \in R_d[p]$ $\exists q \in Ante(r) : \sim q \in F$ or
(2) $\exists s \in R[\sim p]$ such that
    (a) $\forall q \in Ante(s) : q \in F$ and
    (b) $\forall t \in R_{sd}[p]$ either
        $\exists q \in Ante(t) : \sim q \in F$ or $t \not> s$.

Or, in plain English, we use the conclusion of the strongest of the defeasible rules that have all premises satisfied. If no such rule exists, or if there is any defeater with the opposite conclusion that is not stronger, then we have no conclusion.

For simplicity, we disregard the tagging, and instead represent $T, F \vdash +\partial p$ as simply $T, F \vdash p$, and use $T, F \vdash ?a$ to denote the case that both $T, F \vdash -\partial a$ and $T, F \vdash -\partial \neg a$ holds (that is, $T, F \vdash ?a$ denotes the case that nothing can be defeasibly proven with respect to a).

This formal definition of defeasible logic has a natural mapping to a Prolog meta-program [36]. To a fluent Prolog programmer, this meta-program elegantly conveys the proof theory in a far more intuitive fashion. A summary of the work in [36] appears as the Prolog meta-program in Appendix C.

Chapter 5

Induction of Theories

Given the apparent similarities of defeasible logic with legal expression and reasoning, we turn our attention to the problem of inducing a defeasible logic theory from precedent data (i.e., a set of training examples). That is, we require an algorithm that, given a training dataset of cases and their corresponding conclusions, will produce a theory that:

- When applied to each element of the training dataset, will reach the same conclusions as the training set, and
- When applied to an unseen case, will reach a 'good' conclusion.

Two results are important here; that it is possible to find a theory that describes a dataset, but that finding the optimal solution is unfortunately an NP optimisation problem. In this chapter, we consider these theoretical challenges and an algorithmic solution that is a compromise between theory and practice.

5.1 Theoretical Foundation

We begin with some definitions first.

We define a *dataset* D as a set of records d, where each d is tuple of the form (F, c), where F is a possibly empty set of literals denoting the known truths or facts of the particular precedent, and where c is either a literal (a or $\neg a$) or the term $?a$ indicating that no conclusion is reached with respect to propositional variable a. We say a dataset is *consistent* if every conclusion is formed from the same propositional variable, and if for all records $d_1 = (F_1, c_1) \in D$ and

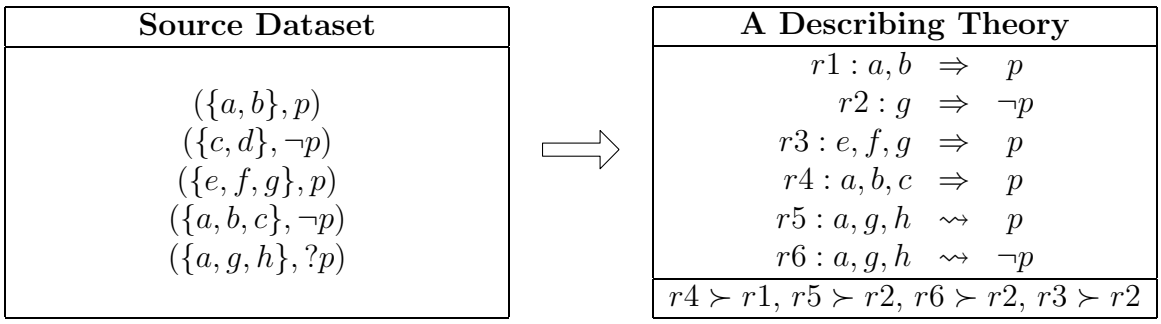

Figure 5.1: Example Construction of a (Poor) Describing Theory

$d_2 = (F_2, c_2) \in D$, if $F_1 = F_2$ then $c_1 = c_2$. In other words, a dataset is consistent if no two identical cases have different conclusions. Given a consistent dataset D we define $Var(D) = a$, where a is the propositional variable that appears in every conclusion.

For convenience we will assume that all datasets are consistent. Whenever this assumption is invalid, it can be corrected by pre-processing the dataset. Depending on the particular context of the application this might mean deleting the records that are causing the inconsistency, using the result from the more recent record, or replacing all such inconsistent records with a new record of the form $(F, ?a)$ to indicate that given the particular facts F that are causing the inconsistencies, we do not have any known conclusion.

It should be noted that the definition of a dataset only allows for a single propositional variable to be used over all the conclusions. In applications where multiple conclusions are necessary, this limitation can be trivially overcome by processing each class of conclusions with a different dataset and repeated application of the algorithm presented later, in Section 5.3.

We say a defeasible logic theory, T, has an *accuracy* of $x\%$ with respect to a given dataset, D, if for $x\%$ of the records $d = (F, c) \in D$, it can be proven that $T, F \vdash c$. We say a theory *describes* a dataset if it is 100% accurate, that is for each record $d = (F, c) \in D$, it can be proven that $T, F \vdash c$.

Theorem 1 *Given any consistent dataset D, there exists a theory that describes the dataset.*

Proof. By Construction.

Let $a = Var(D)$, we construct a theory $T = (R, \succ)$, from rules

$$\begin{aligned} R &= \{F \Rightarrow c | (F, c) \in D \wedge c \neq ?a\} \cup \\ &\quad \{F \rightsquigarrow a | (F, c) \in D \wedge c = ?a\} \cup \\ &\quad \{F \rightsquigarrow \neg a | (F, c) \in D \wedge c = ?a\} \end{aligned}$$

and superiority relation

$$\succ = \{(r1, r2) | r1, r2 \in R \wedge Ante(r2) \subset Ante(r1)\}.$$

This theory describes the dataset (an example of such a theory appears in Figure 5.1), for given any record $r = (F, c) \in D$, we have $T, F \vdash c$. By the reasoning mechanism of defeasible logic, we know that only the rules in

$$\begin{aligned} &\{r | r \in R \wedge Ante(r) \subseteq F\} \\ &= \{r | r \in R \wedge Ante(r) = F\} \cup \{r | r \in R \wedge Ante(r) \subset F\} \end{aligned}$$

can be applied in the reasoning process. The dataset is consistent (by assumption), and the theory has been constructed from each element of the dataset, so we know that the theory T will contain either exactly one defeasible rule with $Ante(r) = F$ and conclusion c, or exactly two defeaters with $Ante(r) = F$ which will give conclusion $?a = c$. This means that applying the rules in the left set of the above union

$$\{r | r \in R \wedge Ante(r) = F\}$$

will give us the correct conclusion. It is clear that the rules in the right set of the above conclusion

$$\{r | r \in R \wedge Ante(r) \subset F\}$$

will have no effect on this outcome, for the superiority relation defines those rules where $Ante(r) \subset F$, to be weaker than the rules where $Ante(r) = F$. □

In fact, it turns out that for any given dataset D, there can be many theories that describe the dataset. We could use the theory generated by the construction that appears in the proof of Theorem 1 for predicting new cases, this is unsatisfying because using such a theory becomes no more than a primitive case-based reasoning system. Instead, we look to find the theory that describes the dataset

and has the *minimal* number of rules. Because a minimal theory has few rules, we would expect each rule to carry more significance and have greater predictive power than might the rules of a larger theory describing the same dataset. For this reason we expect that a minimal theory would give the best generalisation of the dataset and would be most likely to perform well on unseen cases. Finding a minimal theory also simplifies the comprehension effort required to have a human expert verify a theory, and more closely matches our expectations that human experts in their own practice would themselves typically avoid reasoning with convoluted rules or on an individual case-based-reasoning approach. Unfortunately, though, finding the minimal theory turns out to be intractable, we show this by transformation from the hitting set problem.

Garey and Johnson [29] report that the hitting set problem is NP-complete (by transformation from the vertex cover problem due to Karp in 1972). The problem is as follows:

Input Given a collection C of subsets of a finite set S, and a positive integer $k \leq |S|$

Problem Is there a subset $S' \subseteq S$ with $|S'| \leq k$ such that S' contains at least one element from each subset in C?

Theorem 2 *Given a dataset D and positive integer $k' \leq |D|$, the problem of deciding whether there exists a theory $T = (R, \succ)$ of size $|R| \leq k'$ that describes D is NP-hard. (We refer to this problem as the Describing Theory Problem.)*

Proof. If we assume the existence of an algorithm for solving the describing theory problem, we can solve the hitting set problem (SP8 in [29]) via transformation to the inputs of such an algorithm:

- The collection of the hitting set problem can be encoded in polynomial time to a dataset, then
- An algorithm that solves the describing theory problem can be used to reach a decision for the hitting set problem.

The describing theory problem is therefore NP-hard, because if there exists a polynomial-time algorithm that can solve the describing theory problem, we can solve the hitting set problem in polynomial time, and therefore *all* NP-complete

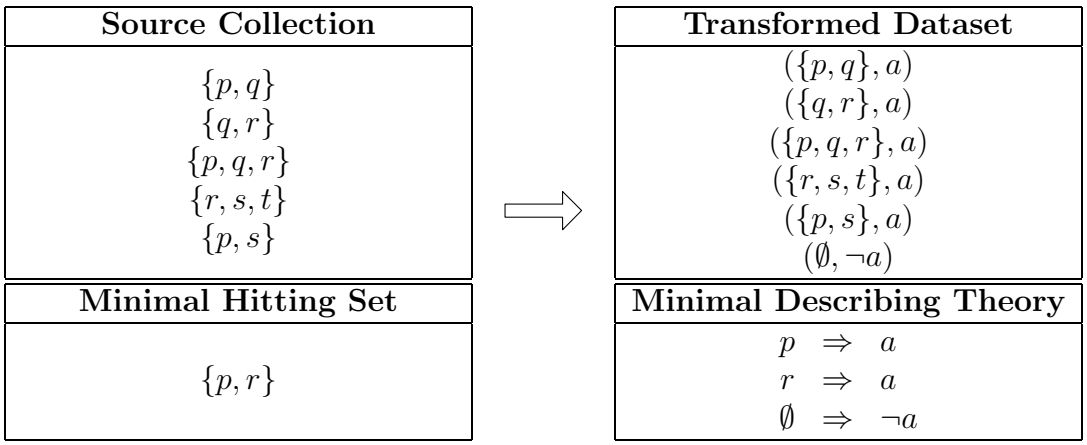

Figure 5.2: Example Construction for Proof of NP-Hardness

problems in polynomial time. The transformation we use is possible because of a correspondence between a minimum hitting set and a minimum describing theory.

Given a collection C of subsets of a finite set S, and a positive integer $k \leq |S|$, these inputs are encoded to a form suitable for theory induction as follows:

- Create a new propositional variable, a, not appearing in S,
- Construct a dataset

$$D = \{(E, a) | E \in C \wedge E \neq \emptyset\} \cup \{(\emptyset, \neg a)\},$$

- Execute an algorithm that solves the describing theory problem, with input D and $k' = k+1$, if the algorithm returns "true", then there exists a hitting set for C of size k.

Noting the correspondence between a minimum hitting set and a minimum describing theory, the correctness of the above encoding can be seen:

The dataset, D, of the above construction (an example appears in Figure 5.2) has several interesting properties. Any theory that describes the dataset must contain the rule $\emptyset \Rightarrow \neg a$. This is because when reasoning about the record $d = (\emptyset, \neg a) \in D$, the only applicable rules are those with $Ante(r) = \emptyset$, and because we need to reach conclusion $\neg a$ given the facts $F = \emptyset$, we must have a rule of the form $\emptyset \Rightarrow \neg a$. Furthermore, any other rule in the theory will be

stronger than the rule $\emptyset \Rightarrow \neg a$ and will be of the form $F \Rightarrow a$ for $F \neq \emptyset$ because all the remaining records $d \in D \backslash \{(\emptyset, \neg a)\}$ have the conclusion a. While it is possible for a theory to describe the dataset with additional rules such as defeaters or rules of the form $F \Rightarrow \neg a$ for $F \neq \emptyset$, a simpler theory can be produced by eliminating such redundant rules and for this reason a minimal theory would not contain such rules.

Now, we note that when reasoning about some facts, $F \neq \emptyset$, with the rules of a minimal theory $T_{min} = (R, \succ)$ as above, a given rule $r \in R$ is applicable if $Ante(r) \subseteq F$. In fact, a minimal describing theory of a dataset D is a theory such that for each record $d = (F, c) \in D$ where $F \neq \emptyset$, it holds that

$$\exists r \in R \mid Ante(r) \subseteq F$$

(this property is quite close to the definition of a hitting set). Finally, by noting that if we have a rule $r = (Q \Rightarrow a)$ that applies to a record $d = (F, c) \in D$, we can choose any element $s \in Q$ to produce a new rule $r' = (\{s\} \Rightarrow a)$ so that it is still the case that

$$Ante(r') \subseteq F \wedge Ante(r') \neq \emptyset.$$

Thus, if we are given a minimal theory T_{min} for a dataset that has been constructed from a hitting set problem, we can simplify each rule (with the positive conclusion a) in the theory T_{min} to give a new theory

$$T'_{min} = (R', \succ')$$

so that the premise of each rule is a singleton[1]. This simplification neither adds nor removes rules, so we still have a minimal theory; but by taking the union of the premises of each rule we have a minimum hitting set

$$S' = \cup\{Ante(r) | r \in R'\}.$$

We see that this is indeed the case because, for each record $d = (F, c) \in D$, it holds that

$$\exists r \in R' \mid Ante(r) \subseteq F$$

[1] A *singleton* is set with one element

and since each $Ante(r)$ is a singleton, this is equivalent to

$$\exists s \in S' \mid s \in F$$

(i.e., S' is a hitting set). By the same reasoning, we can construct a minimal describing theory from a minimum hitting set by creating a rule with singleton premise set for each element of the hitting set (of course, in addition to the rule $\emptyset \Rightarrow \neg a$).

That is, given a minimal theory T_{min}, with size $k' = |R|$, of such a dataset, we can produce a minimum hitting set size,

$$k = S' = k' - 1,$$

of the corresponding collection C, by selecting from the premise of each rule one element. And likewise, we can produce a minimal theory from a minimum hitting set. So that if there is a theory of size $|R| \leq k'$ that describes the dataset, then there will exist a hitting set of size

$$|S'| \leq k = k' - 1$$

for the collection. □

Theorem 3 *Given a dataset D and positive integer $k' \leq |D|$, the Describing Theory Problem is NP-complete.*

Proof. The decision procedure can be solved in NP time. An algorithm is as follows:

1. Non-deterministically, generate a theory T of size $|R| \leq k'$ over the propositional variables in D.
2. For each record $(F, c) \in D$, use the linear time algorithm in [35] to check $T, F \vdash c$.
3. If the theory T describes the dataset D, succeed.

The correctness of this algorithm can be seen immediately, for it simply checks that the nondeterministically generated theory of Step 1 describes the dataset.

Clearly, this algorithm runs in nondeterministic polynomial time. Given that the decision procedure is also NP-hard (Theorem 2) we conclude that the decision procedure is NP-complete. □

While the above results pertain to a decision procedure, the problem of finding the *smallest* theory that describes a dataset belongs to the class NPO (defined in [7]), and is consequently NP-hard. These results follow immediately from the definition of NPO, or can be proven via transformation from SP7 in [7] using a similar mapping as that used Theorem 2.

The important consequence of these results is that it is unlikely that there is a tractable algorithm that finds the global optimum, but that it is necessary to use heuristics to find an approximate solution.

5.2 Inductive Logic Programming

Much of the existing work on generation of logical theories from examples is in the context of inductive logic programming (ILP), it is therefore natural to expect that some of the techniques of ILP could be applied to induction of defeasible logic theories.

Inductive logic programming is concerned with producing pure Prolog programs (theories in Horn clause logic) from example sets of inputs and outputs. It usually operates in either a bottom-up approach by generalising examples or a top-down approach specialising the empty program into a suitable theory [34]. It turns out that the problem of inducing theories in defeasible logic from examples is very much related to inductive logic programming where the hypothesis space is restricted to clauses where the body of each clause consists of only unary predicates (this approach will produce a theory, but it will not be minimal because an ILP search finds theories without considering a superiority relation and so does not include the possibility of smaller theories that make use of a superiority relation).

Inductive logic programming can be further categorised by their fundamental approaches: direct generalisation, meta-queries, inverse resolution, top-down refinement or transformation. We have considered each approach, but have found all of them unsuitable or lacking in some respect.

5.2.1 Direct Generalisation from Examples

By defining the concept of least general generalisation (*lgg*), which forms a generalisation lattice, ILP systems are able to produce rules that generalise given sets of examples. When a similar concept is applied to defeasible logic, we have a bottom-up refinement process that simply computes intersections of the assumptions of examples with the same conclusions. Unfortunately, finding the intersections is related to the hitting set problem – and so this technique which is suitable for small ILP problems is likely to be unsuitable for application to defeasible logic.

5.2.2 Meta-queries

The concept of a meta-query – a second order template that specifies the type of patterns to be discovered [24] – as used in ILP initially appears to be a particularly attractive approach to mining defeasible logic theories. Given that legal texts and decisions are likely to have similar patterns of formulation, by identifying these patterns of formulation it is possible to simplify a model search into that of a parameter search over the second-order template.

However, defeasible logic already has a very high correspondence with legal expression [3] and so it would be expected that virtually all theories would be possible for a given dataset, and that limiting a search with meta-queries would be unduly restrictive or would require meta-queries to be so general that no benefit is gained from the approach. The most relevant deciding factor in whether a particular theory is likely to represent a realistic legal framework (beyond the simple constraint that circular rules would likely be considered undesirable in practical situations) is likely to be the simplicity of the theory (though this is always going to be a subjective measure).

5.2.3 Inverse Resolution

Inverse resolution is an approach to inductive logic programming whereby unification is performed "in reverse" – replacing atoms that appear in examples with variables. This approach applies in the context of inductive logic programming over binary (or higher arity) predicates and so is of limited benefit to inducing defeasible theories (which is roughly equivalent to ILP over only unary predicates).

```
set program P, the program containing the empty clause
while P has not yet converged
    read a new example, e
    if the conclusion of e is inconsistent with P
        with the refinement operator, ρ(c),
            specialise P, or add exceptions to P
    if e has no conclusion, and P gives a conclusion for e
        with the refinement operator, ρ(c),
            add defeaters to P, specialise P, or simplify the priority relation
output P
```

Figure 5.3: Simplified Model Inference System

5.2.4 Top Down Refinement

In the process of top-down search or refinement, the empty theory is initially assumed, which is then refined to match the example data [34]. A refinement operator $\rho(c)$ is defined over the clauses of a theory, and this is typically used in a refinement process such as the model inference system by Shapiro [46], summarised in [28], and refined here in Figure 5.3 for defeasible logics.

The particular refinement operator $\rho(c)$ is carefully chosen for efficiency, and to ensure that the refinement process converges on some solution [38] (that is, it results in "identification in the limit"). In our research, the top-down refinement approach proved to offer the most promising results. Unfortunately, a literal translation of the above algorithm results in extremely poor execution and theories due to the mismatch between the montonicity of Prolog and the nonmonotonicity of defeasible logic. In Section 5.3, a new algorithm that draws from this approach is presented.

5.2.5 Problem Transformation

Some ILP systems such as LINUS [34] begin by transforming the problem to a format suitable for a propositional learner, running a propositional learning algorithm, and transforming the result back into a logic program. While this is applicable to a monotonic logic such as Horn clause logic, it does not apply to

```
set theory, T = (∅, ∅)
do
    invoke Rule Search Routine (Figure 5.5), to find a new rule r
    if r ≠ nil
                set T, to T + r
while r ≠ nil
```

Figure 5.4: Defeasible Theory Search Algorithm

inducing defeasible theories for the same reasons that propositional learners are unsuitable for inducing knowledge in LDSSs (see Section 3.2).

5.3 A New Algorithm: HeRO

Given the inherent difficulty of the induction problem for defeasible logic, it is necessary to take a pragmatic approach in seeking an algorithm that produces "reasonable" theories in "reasonable" time. A new algorithm, HeRO[2], that uses a greedy, branch-and-bound, best-first search strategy, suits these criteria (a good description of these techniques in a general sense appears in [32]), producing meaningful output on realistic data.

The algorithm starts with an empty theory and iteratively adds rules to the theory in a greedy fashion so as to improve the accuracy of the theory. With every iteration the search space of possible rules is explored using a branch and bound algorithm to select the rule with the highest gain. This greedy mode of operation is not unrealistic because rules that offer a high degree of predictive power should naturally offer the greatest degree of improvement in accuracy of the theory (and indeed, practice confirms that this is the case or at least a suitable approximation of reality). Pseudocode for this high-level operation of the algorithm is detailed in Figure 5.4.

Each invocation of the outer loop invokes the rule search routine to select the next rule (and the position of the rule in the superiority relation) to add to the current theory. We now turn our attention to this algorithm.

[2] **He**uristic **R**ule **O**ptimisation

If a rule $r = (Q \Rightarrow c)$ or $r = (Q \rightsquigarrow c)$ is added at some position in the superiority relation $\succ$ of a theory $T = (R, \succ)$, to give a new theory T', we define the gain, $gain_{T,r,T'}$, of that rule to be the difference between the number of records, $d = (F, c) \in D$, for which $T', F \vdash c$ and the number of records for which $T, F \vdash c$. That is, the gain of a rule is the increase in the accuracy of a theory that is a result of adding the rule to the theory.

This definition of gain can be equivalently stated in terms of "incorrect conclusions that are corrected by adding the new rule", and "correct conclusions that are blocked by adding the new rule", or formally:

$$\begin{aligned} gain_{T,r,T'} =& \#\{(F,c) \in D \mid T', F \vdash c\} \\ & - \#\{(F,c) \in D \mid T, F \vdash c\} \\ =& \#\{(F,c) \in D \mid T', F \vdash c \ \wedge \ T, F \nvdash c\} \\ & - \#\{(F,c) \in D \mid T', F \nvdash c \ \wedge \ T, F \vdash c\} \end{aligned}$$

We derive an upper bound for $gain_{T,r,T'}$ by noting that if the rule r is refined by adding literals to the premises to make the rule more specific, then the number of "incorrect conclusions that are corrected by adding the new rule" must decrease because a subset (but no more) of these "corrections" will still be applicable after refining the rule, and the number of "correct conclusions that are blocked by adding the new rule" will also decrease, for the same reason: only a subset of these "blocks" will still be applicable after refining the rule. Under ideal circumstances, the refinement of a rule would result in no reduction of "corrections", but would eliminate all "blocking". It is this ideal condition that leads us to the upper bound, $maxgain_{T,r,T'}$ for any refinement of the rule r:

$$maxgain_{T,r,T'} = \#\{(F,c) \in D | T', F \vdash c \wedge T, F \nvdash c\}$$

These expressions for $gain_{T,r,T'}$ and $maxgain_{T,r,T'}$ can be further refined if required to support a legal practice that is known to evolve over time. Instead of simply counting records with the # operator, it is possible to compute a weighted sum, with the contribution of each record inversely proportional to the age of the record or proportional to some user-specified parameter of the case's importance. This approach places greater emphasis on more recent conclusions, and allows theories to be generated for datasets that may contain evolutionary change.

Now, by either best-first or simply breadth-first search, we can explore the search space by maintaining a variable that holds the best rule found so far, and only exploring those branches of the search space where the upper bound, $maxgain_{T,r,T'}$, is strictly greater than the value of $gain_{T,r,T'}$ for *bestgain*. The *bestgain* can then be added to the current theory T (if it would result in a positive gain), before repeating the search again for the next iteration (or halting if no more rules exist that result in positive gain).

By only considering totally ordered superiority relations, it is possible to obtain an efficient implementation of this algorithm. For each position in the total order, the weaker rules are immediately applied to the dataset to give tentative conclusions and the records in the dataset to which stronger rules apply are discarded (because if we added a rule at this position in the superiority relation it would have no effect on the conclusions of records for which one of the stronger rules is applicable). This initial processing allows the $gain_{T,r,T'}$ and $maxgain_{T,r,T'}$ to be efficiently computed in a single pass over the dataset. Furthermore, additional performance gains are possible by associating with each set of premises, the records in the dataset that are applicable (and maintaining this set during each set-based computation). Restricting the algorithm to only totally ordered superiority relations does not appear to result in poorer theories, and in fact, produces a theory that is easier for a human expert to comprehend since such a theory represents an ordered list of rules, as opposed to a digraph of rules that is more difficult to interpret and display.

Pseudocode for an implementation of the greedy rule search appears in Figure 5.5. A best-first search is shown, but this can be trivially modified to a breadth-first search by replacing the priority queue with a standard queue. Because it is possible to compute the accuracy gain of a rule $r = (Q \Rightarrow p)$ and its negation $r' = (Q \Rightarrow \neg p)$ in a single pass, we compute both simultaneously for a given premise set Q, and return the conclusion, *gain* and *maxgain* of the rule that has greater accuracy gain.

While HeRO can be implemented in languages such as Prolog, a significantly more efficient implementation is possible by using hash-based data structures available in imperative languages in order to improve the speed of set based operations (particularly set membership). The source code for an efficient breadth-first C# implementation of HeRO appears in Appendix A.

```
set best gain so far, bg ← 0
set best premises, bp ← nil
set best conclusion, bc ← nil
foreach position in the totally ordered superiority relation
        set weaker ←the existing rules that are weaker than the current position
        set stronger ←the existing rules that are stronger than the current position
        set priority queue, q ← ∅ using q enqueue ∅ with priority 0
        while q ≠ ∅ set current premise p = q.dequeue()
                compute preferred conclusion, c, of p,
                        gain, g, of p, and
                        maxgain, mg, of p
                if g > bg
                        set bg ← g
                        set bp ← p
                        set bc ← c
                if mg > bg
                        foreach refinement p' of p
                                q.enqueue(p')
if bg > 0
        return (bg ⇒ c) and current position
else
        return nil
```

Figure 5.5: Rule Search Routine

Chapter 6

Experimental Results

In order to evaluate HeRO, we have used the algorithm on datasets that have been generated from known theories, and we have also compared the algorithm with other approaches in the literature.

6.1 HeRO on Known Theories

HeRO can be evaluated by noting how faithful a theory induced by HeRO is, when compared to the known theory underlying the dataset. We do this by manually creating a theory, randomly generating a dataset of 1000 records (including "noise" variables) that is consistent with the theory, and running HeRO on the dataset. HeRO gives impressively accurate results on such theories, and executes on all datasets within 0.5 seconds. We analyse an important outcome on each common pattern identified in [31]:

6.1.1 Simple Exception

Source Theory
$r1$: $a \Rightarrow p$
$r2$: $a, b \Rightarrow \neg p$
$r3$: $a, c \rightsquigarrow \neg p$
$r4$: $e \Rightarrow p$
$r2 \succ r1, r3 \succ r1$

Induced Theory
$r1$: $a \Rightarrow p$
$r2$: $a, b \Rightarrow \neg p$
$r3$: $e \Rightarrow p$
$r4$: $a, b \rightsquigarrow \neg p$
$r5$: $a, c \rightsquigarrow \neg p$
$r5 \succ r4, r4 \succ r3, r3 \succ r2, r2 \succ r1$

Though the induced theory above is not identical to the source theory, there is an extremely high correspondence (and the two theories are logically equivalent). The only significant difference is the additional rule $r4$ that appears in the induced theory – this rule compensates for the fact that the superiority relation of the source theory is a partial order, because HeRO considers only totally ordered superiority relations.

6.1.2 Aggregate vs Separate

Source Theory
$r1$: $a \Rightarrow p$
$r2$: $b \Rightarrow p$
$\emptyset$

Induced Theory
$r1$: $a \Rightarrow p$
$r2$: $b \Rightarrow p$
$r2 \succ r1$

The problem of separating aggregate rules ($a, b \Rightarrow p$) from separate rules ($a \Rightarrow p$ and $b \Rightarrow p$) was identified as a challenge in [31], and yet even if a and b appear together extremely often, HeRO operates perfectly.

6.1.3 General vs Specific

Source Theory
$r1$: $a \Rightarrow p$
$r2$: $a, b \Rightarrow p$
$r5 \succ r4$

Induced Theory
$r1$: $a \Rightarrow p$
$\emptyset$

The problem of distinguishing between a general ($a \Rightarrow p$) and a specific ($a, b \Rightarrow p$) rule is relevant because in some contexts the more specific (and complex) rule is preferred, and in others the more general rule is preferred [31]. This problem contradicts our assumption that a minimal theory is best, and we believe that contexts requiring the more specific rule are somewhat atypical. In the source theory above, the second rule is in fact redundant, and HeRO has correctly identified the minimal equivalent theory. If, for example, $r1$ in the source theory was never exercised – that b always occurred with a – then it might be argued that we should not generalize to cases in which b does not occur with a. On real

datasets, we have not encountered this problem, but it is possible to alter HeRO to accommodate this requirement.

6.1.4 Conclusions

HeRO faithfully generates theories consistent with the underlying theory. While the generated theory may not be identical or optimal, the differences are due to the assumption that the superiority relation is totally ordered – an assumption that does not significantly reduce comprehensibility of the induced theories. Furthermore, even though the induction process was executing over 1000 records, runtimes are more than satisfactory[1].

6.2 HeRO Versus Other Approaches

In this section we compare the HeRO algorithm with two other approaches to machine learning in the legal domain.

6.2.1 DefGen

The approach used by DefGen [8] follows the intent of original work by Governatori and Stranieri [31]. This work is motivated by the syntactic similarities, and the certain degree of semantic overlap between mined association rules and defeasible logic theories. The intent of their work is that by post-processing an efficient association rule-mining algorithm, a defeasible logic theory could be generated. Even though HeRO also follows from the work by Governatori and Stranieiri [31], an association rule based approach seemed inappropriate for HeRO because it is difficult to use association rules to capture the exceptions or "disassociations", and because exhaustively post-processing the output of an association rule mining algorithm appears to be at least as difficult as generating a theory directly from the dataset (and possibly more difficult because association rules lose too much of the information about the original dataset).

DefGen was implemented to identify common patterns of regulation formulation within the output of a fast Apriori [1] mining algorithm. The algorithm

[1] Curiously enough, it was faster to induce the theories than to generate the original datasets! This was because the dataset generation was performed in Prolog, but the induction was with a C# implementation.

Algorithm	DefGen	DefGen	HeRO	HeRO
Minimum support	5	10		
Minimum confidence	100	75		
Target theory size			8	no limit
Accuracy	87%	84%	90%	97%
Rules generated	46	55	8	16
Runtime	0.4s	0.5s	2s	20s

Table 6.1: Comparison of Results for Credit Application Dataset

was tested against the Japanese Credit Application dataset, available as part of the UCI Machine Learning Repository[2]. The dataset contains 125 records describing 10 attributes (in total, 27 attributes after "binning" the continuous attributes into discrete categories) of credit applicants and the outcome of their applications. Avery reports that different parameters of minimum support and confidence give differently sized theories (a trade-off between theory size and accuracy), but recommends two parameters that offer a good balance. In contrast, HeRO uses only a single parameter that specifies the size of the theory to be generated before halting.

A comparison of the algorithms appears in Table 6.1. Clearly, while DefGen has faster runtimes, HeRO offers far superior accuracy and smaller theories within sufficient run times. Even though both of these techniques are still in their infancy, HeRO is showing much promise.

6.2.2 Neural Networks and Association Rules

In [9], Bench-Capon justifies the development of legal information systems such as that discussed in this thesis, and then trains neural networks on existing cases. Indeed, Bench-Capon faces the same problem of black box mode of operation we discussed in Section 3.3. He admits that a trained neural network is difficult to comprehensively verify, and it is clear that a neural network offers little justification for its answers – much work is necessary to explain to non-technical law workers how to interpret and use the output of a neural network.

Bench-Capon used synthetic datasets describing a fictional welfare benefits scheme paid to pensioners that suffer expenses visiting a spouse in hospital. The

[2]Freely Available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`

decision of whether or to pay the pensioner is subject to the following conditions:

1. The person should be of pensionable age (60 for a woman, 65 for a man);
2. The person should have paid contributions in four out of the last five relevant contribution years;
3. The person should be a spouse of the patient;
4. The person should not be absent from the UK;
5. The person should not have capital resources amounting to £30,000; and
6. If the relative is an in-patient the hospital should be within a certain distance: if an out-patient, beyond that distance.

These six conditions immediately translate into 12 features (age, gender, 5 contributions, spouse, absence, capital resources, in-patient, distance), which form the dataset along with 52 additional "noise" features that have no influence on the conclusion.

The neural networks that Bench-Capon trained on 2400 such records obtained the following success rates [9]:

One hidden layer:	99.25%
Two hidden layers:	98.90%
Three hidden layers:	98.75%

Even though this accuracy is quite high, it is of course difficult to judge how faithfully the conditions for payment are encoded within the network, and so it is hard to rationalize the use of such technology in legal practice if alternatives exist.

In a later work, Bench-Capon [10] analysed the same datasets using association rule mining algorithm, while this work did not result in any specific strategy for predicting or describing future cases, we follow the methodology and justification used to transform the continuous features of the welfare benefit dataset into propositional variables. When applying HeRO to this transformed dataset, the following theory with accuracy 99.8% is produced (in order of superiority, from weakest to strongest):

$$
\begin{aligned}
&\Rightarrow && grant \\
distance\_short,\ inpatient &\Rightarrow && \neg grant \\
\neg spouse &\Rightarrow && \neg grant \\
absent &\Rightarrow && \neg grant \\
age\_lt\_60 &\Rightarrow && \neg grant \\
capital\_gt\_3000 &\Rightarrow && \neg grant
\end{aligned}
$$

It is trivial to alter the algorithm to prefer scepticism[3] when two rules have equal accuracy gain; doing so results in an even simpler theory with the same accuracy (in order of superiority, from weakest to strongest):

$$
\begin{aligned}
&\Rightarrow && \neg grant \\
spouse,\ \neg absent,\ \neg age\_lt\_60,\ \neg capital\_gt\_3000 &\Rightarrow && grant \\
distance\_short,\ inpatient &\Rightarrow && \neg grant
\end{aligned}
$$

Clearly, this theory has a very high correspondence to the original conditions – it would be easy for a non-technical law worker to understand this theory (and, in fact, could be manually annotated by a domain expert so that plain-English explanations for conclusions can be presented to the user when the theory is used). Even if we ignore the fact that defeasible logic allows such clearly interpreted theories to be generated, the high accuracy (99.8%) is competitive with the results produced by Bench-Capon's neural networks.

A positive outcome of Bench-Capon's work with Apriori [10] in this context is an approach to identifying and eliminating noise variables. Even though HeRO works satisfactorily on the 2400 records and 64 features of this dataset, by pre-processing the dataset to eliminate noise variables, the scalability of HeRO could be dramatically improved further.

6.2.3 Conclusions

It is quite clear that HeRO has competitive (or greater) accuracy when compared to existing approaches in the legal domain that solve similar problems. But, by inducing defeasible logic theories, HeRO generates a far more transparent knowledge representation and so is significantly more applicable to legal domains. Of

[3]This is simply a matter of setting the conclusion to $\neg p$ in the event of a rule that has equal gain in both its positive and negative form.

course, HeRO can never achieve guarantees of 100% accuracy, but the theories that the algorithm generates have demonstrated correspondences with the underlying theories and thus we can expect the theories induced in realistic situations to accurately reflect the underlying reasoning in precedents.

Chapter 7

Application of HeRO

7.1 The Knowledge Discovery Process

While the technical approaches to induction from examples is a key part of knowledge discovery, the full process consists of much more than simply applying algorithms to data [27, 26]. An important consideration in producing legal decision support systems is the creation of datasets that might be used for "training". To do so, we must work within the structured framework of a knowledge discovery process typically performed in 5 steps [26]:

1. Selection (selecting the relevant data to mine)
2. Preprocessing (the process of cleansing the data – removing unnecessary data, contradictory data and data that may slow down queries)
3. Transformation (translating the data to a suitable format for a data mining algorithm)
4. Extraction (extracting the patterns from the data)
5. Interpretation (evaluating the patterns for correctness and interpreting the "discovered" knowledge)

Large legal databases such as AUSTLII[1] are freely available, but do not have a structured data-model – outcomes of cases are stored in plain-English texts.

[1] Accessible at `http://www.austlii.edu.au/`

Throughout this project, we have focussed on the extraction stage of the knowledge discovery process and have assumed the existence of structured datasets for which selection, preprocessing and a certain degree of the transformation stages have already been completed. We now turn our attention to generating these datasets.

7.2 Dataset Generation

As discussed in Section 2.1, one way of structuring the dataset generation (the selection, preprocessing and transformation) would be to have an expert identify as many attributes as possible that might have a bearing on the outcome of a case, and employ low cost labour to identify and enter these attributes that occur within the precedent cases. Costs remain low because each case need only be entered once, and a suitable form based interface can facilitate and encourage the entry of data at the resolution of each case.

Selecting attributes to record might ordinarily be a difficult problem, but because the defeasible logic induction algorithm allows values to be marked as unknown, it is possible to add additional attributes to the dataset at a later date. Retrospectively recording a new attribute as unknown for the precedents of prior cases, and entering the attribute for each case that occurs henceforward, evolutionary changes in the factors involved in decisions for future cases can be supported (e.g., the use of DNA evidence would not have even been conceived 30 years ago).

Unfortunately, manually encoding cases to a prescribed set of attributes is a risky activity because it may not be clear whether enough attributes for induction have been selected up front. After data-entry, if execution of HeRO gives a theory that is dramatically different from established practice (e.g., reasoning with attributes such as hair colour where this is a completely unrelated attribute) then it is likely that the precedents were described with the incorrect attributes. Unfortunately, the only recourse in such a situation would be to return to the original cases and enter the new attributes from scratch. This risk can be mitigated by careful selection of attributes, and sampling the induced theories early on in the data entry process.

A more effective approach might be to describe every case in full detail, ac-

cording to a grammar that is both machine readable and sufficiently descriptive to capture all attributes considered by the court. This is an extremely challenging problem, but some approximations of the ideal vision are possible, such as the works presented in [15, 14, 42, 37]. An example of how XML might be used to represent a case appears in Figure 7.1.

The attraction of using a semi-structured representation such as XML [16] is three-fold:

1. The notation is flexible enough to allow cases to be comprehensively described (as can be clearly seen in Figure 7.1),

2. The underlying schema (in the case of XML, it is likely to be XML Schema [25]) can be readily *extended* to allow the description of new attributes that are brought to the courts attention, and

3. In addition to being readily understood by human beings, semi-structured notations such as XML can be processed and queried by machine.

Given a set of precedents that are comprehensively described in a notation such as XML, an expert or knowledge engineer can generate a structured dataset for HeRO by writing a query for each attribute. Although we still require an expert to identify attributes that could potentially be considered in a decision, a semi-structured representation allows attributes to be redefined or added at any time, without requiring the precedents to be manually inspected again. Given that HeRO can induce a theory in seconds – it feasible to incorporate the induction algorithm and the attribute query and definition language into a single unified environment that allows an expert to define attributes and immediately gauge the improvement in the accuracy or the clarity of the induced theory and accordingly refine the queries. Possible approaches to specifying these queries include the use of XML query languages such as SQL/XML, XMLQuery or XPath, and the use of description logics.

7.2.1 XML Query Languages

The world wide web consortium (W3C), has specified two languages for querying XML documents: XPath [11] which offers a basic syntax for addressing and querying nodes in an XML document, and XQuery [12] that extends the power

```
<case id="A113-22-182-7783">
    <person id="joey" type="defendant">
        <name>Joey Jo-Jo Junior Shabadoo</name>
        <citizen>Australia</citizen>
        <gender>Male</gender>
        <age>20</age>
        <prior-convictions>
            <ref id="B19-67-21-5573"/>
        </prior-convictions>
        ...
    </person>
    <accident third-party-damage="none">
        <driver person="joey" passengers="none">
            <vehicle>
                ...
            </vehicle>
        </driver>
        <evidence type="blood-alcohol-sample" disputed="no">
            <measure type="alcohol">0.13</measure>
        </evidence>
        ...
    </accident>
    <judgement type="guilty">
        <licence-loss duration="3 months"/>
    </judgement>
    ...
</case>
```

Figure 7.1: Sample XML document describing a drink-driving accident

of XPath with a powerful query language that bears resemblance to SQL. Additionally, direct extensions to the ISO/IEC 9075 standard for SQL (SQL-99) are currently under investigation by the SQLX workgroup[2]. While XQuery, XPath2 and SQL/XML are still progressing through the standardisation process, prototypes from major vendors such as Oracle and Microsoft are already available. Even if XML is not chosen as a representation for semi-structured data, many other tree or graph based semi-structured representations could be mapped into XML and then queried using these powerful languages.

XPath2 offers a simple syntax that resembles file-system path names, but also has powerful querying features including universal and existential quantifiers (`some` and `every`) that allow simple joins. Using XPath2, an expert might define a dataset in a manner similar to Figure 7.2.

In lieu of a suitable development environment, it is possible to conveniently generate datasets with existing technology via an XSLT template (XSLT is an XML transformation language that is based on the XPath query language). An example of such a template appears in Appendix B.

A deficiency present in this XPath approach is the mismatch between the two-valued nature of the XPath logical expressions and the three-valued nature of defeasible logic (it is possible in defeasible logic to have no conclusion with respect to a particular variable). Of course, HeRO will work perfectly and induce a theory over a dataset created from queries performed using XPath, but it might be desirable to use a three valued logic so that unknown values are explicitly marked as *unknown*. This could be resolved by directly modifying the semantics of XPath to create a new language, or using two expressions for each variable – the first expression defines the truth value of the variable, and the second expression defines whether the value of the variable is in fact unknown (i.e., with two Boolean expressions we can define up to four truth-values of a variable).

7.2.2 Description Logics

Description logics are subsets of first order logic with desirable computability properties that make them useful for knowledge representation and reasoning. Research into description logics has developed a broad range of logics with well-understood complexity trade-offs – this body of literature is sufficiently cata-

[2]Homepage at `http://www.sqlx.org/`

Attribute	XPath2 Expression
Male	`person[attribute::type = 'defendant']/gender = 'Male'`
Youth	`person[attribute::type = 'defendant']/age < 21`
PreviousIncident	`some $prior in person/prior-convictions/ref/attribute::id satisfies (/precedents/case[attribute::id = $prior]/judgement/attribute::type = 'guilty')`
Drunk	`.//evidence/measure[attribute::type = 'alcohol'] > 0.05`
VeryDrunk	`.//evidence/measure[attribute::type = 'alcohol'] > 0.15`
CausedDeath	`(some $passenger in accident/passenger satisfies $passenger/injury = 'death') or (some $onlooker in accident/pedestrian satisfies $onlooker/injury = 'death')`

Figure 7.2: Example Dataset Definition using XPath2

logued that selecting a description logic for a given application is a matter of selecting desired 'features' of the logic, selecting a computational upper bound (i.e., from the entire spectrum of polynomial to undecidable) and then using the appropriate tableaux decision algorithm in the literature for that logic [19, 22].

Description logics can be applied to law by either direct representation of the precedents in a description logic, or indirectly, by storing precedents in a semi-structured format such as XML and reasoning about this data with a description logic. While the former approach is an interesting possibility that is consistent with other efforts that have created knowledge bases on a description logic foundation [39], we have not explored this in detail because of the challenge involved in presenting advanced logical formalisms to relatively unskilled data entry clerks. The latter option of using XML proved to be more pragmatic considering our time constraints – with this approach we assumed a direct mapping from XML nodes to description logic individuals and use the roles in description logic to connect nodes in accordance with the tree-structure of the XML document[3]. With this mapping, formulas in description logic bear extremely close resemblance to XPath2 expressions, but offers dramatically greater expressive powers.

Calvanese et al [18] have also explored the correspondence between XML and

[3]In order to more closely match the tree-oriented tableaux algorithms usually used in description logics, references in trees such as the prior-convictions element of Figure 7.1 are removed by replacing the reference with the entire sub-tree that can be found at the target of the reference.

description logics, and though we have chosen a different mapping from XML, we follow their reasoning in selecting the description logic $\mathcal{DL}$ that provides similar functionality to that of XPath2. Even though the decision problem for $\mathcal{DL}$ is EXPTIME-hard, we have used the mapping from description logics to propositional dynamic logics of [21] and implemented a variant of the EXPTIME tableaux for propositional dynamic logics due to Pratt [44] to find very satisfactory performance on all the formulas we considered. The fact that worst-case constructions for these logics are rather atypical, and that description logics are regularly applied in large scale ontological databases gives confidence that legal reasoning with description logics will generally give responsive results.

Whilst many XPath2 expressions closely correspond to $\mathcal{DL}$ formulae, the power of using $\mathcal{DL}$ lies in the ability to provide more general 'common sense' knowledge. For example, if we have asserted the formula

$$FEMALE \sqcap \exists CHILD \doteq MOTHER$$

to the knowledge base, then the system knows that a particular person is a mother whether it has been explicitly specified that the person is a mother or if it is known that the person is female and has a child. An assertion such as this also gives the system the knowledge that if a person is a mother, than the person must be female – the kind of 'common sense' knowledge that can make an expert system appear truly intelligent[4] and simplify the task of data entry for users. This power also allows information about a case to be supplied in flexible ways, or even to allow the partial specification of a case to be supplied – for example, stating that somebody is a mother merely implies the existence of children and does not require the children to be explicitly stated. An example of how $\mathcal{DL}$ might be used to describe a dataset appears in Figure 7.3.

An additional advantage of description logic is that provability of these expressions is effectively a three-valued truth notion that corresponds closely to that of defeasible logics:

[4]Admittedly, this particular example that a mother is female is hardly an insightful observation, but description logics such as $\mathcal{DL}$ offer a decidable mechanism that can derive any conclusion that logically follows from a knowledge base (including expressions that require reasoning about quantifiers). To a human being, most of these derivations would appear 'obvious', but that is exactly the point of giving machines 'common sense' and can be very difficult to achieve using other formalisms.

Attribute	$\mathcal{DL}$ Expression
Male	$OneDefendant \sqcap$ $(=1)PERSON.(\exists TYPE.Defendant \sqcap \forall GENDER.Male)$
Youth	$OneDefendant \sqcap$ $(=1)PERSON.(\ \exists TYPE.Defendant\ \sqcap\ (\leq 21)AGE)$
PreviousIncident	$OneDefendant\ \sqcap\ (=1)PERSON.($ $\exists PRIORCONVICTION.GuiltyConviction)$
Drunk	$\exists EVIDENCE.(\ (>5)CMEASURE.(\exists TYPE.Alcohol)\ )$
VeryDrunk	$\exists EVIDENCE.(\ (>15)CMEASURE.(\exists TYPE.Alcohol)\ )$
CausedDeath	$\exists ACCIDENT.(\ \exists PASSENGER.(\exists INJURY.Death)\ \sqcup$ $\exists PEDESTRIAN.(\exists INJURY.Death)\ )$

Background Knowledge		
$OneDefendant$	$\doteq$	$(=1)PERSON.(\exists TYPE.Defendant)$
$GuiltyConviction$	$\doteq$	$\exists JUDGEMENT.(\exists TYPE.Guilty)$

Figure 7.3: Example Dataset Definition using $\mathcal{DL}$

- **True** if the expression can be proved,
- **False** if the negation of the expression can be proved, and
- **Unknown** if neither the expression nor its negation can be proved.

The above truth classifications allow a dataset to be generated with a single expression that accurately and naturally captures the decision support system's knowledge of a particular attribute.

7.3 Discussion

Throughout the development of HeRO, we have assumed the existence of structured datasets. While this assumption does hold under some domains, in realistic applications it is necessary to take a more complete view of the knowledge discovery process. Semi-structured data representation and query languages such as XPath or expressive description logics such as $\mathcal{DL}$ offer significant flexibility in producing input datasets for HeRO. A semi-structured approach not only allows

for meaningful application and development for HeRO, but opens the opportunity for more advanced legal knowledge processing such as semantic querying of precedents or the use of algorithms other than HeRO.

In this project, we have only presented a cursory glance of a possible approach to the knowledge discovery process. An obvious extension of this work is in the development of an integrated knowledge discovery environment – that allows a knowledge engineer to define grammars (e.g., in XML Schema), encode precedents, submit queries, analyse reasoning with HeRO and build interfaces for end-user interaction. Careful attention to usability and to interfaces that allow for flexible and evolving interaction is necessary, but it is a straightforward progression from this work to a tool that allows for powerful encoding, reasoning and querying.

Chapter 8

Future Work

Clearly HeRO, and the framework of a knowledge discovery process that we outlined, shows significant promise for further development to a commercial product. An extension of some of the theoretical groundwork of this thesis would also lead to benefits to other domains of academic interest such as ILP.

8.1 HeRO as an Algorithm for ILP

Given the partial inspiration we drew from the inductive logic programming (ILP) community in creating HeRO, it is worthwhile to further explore the correspondences between Horn clause logic and defeasible logic that we might offer new techniques for ILP. The similarity between the defeasible logic theory minimization problem and ILP with unary predicates is one such indication of this potential for correspondence. Antoniou et al [6] have demonstrated that defeasible logic subsumes logic programming without negation as failure, and so our algorithm might provide new approaches for the ILP community and an improvement over existing attempts at nonmonotonic ILP.

Drawing on the parallel with ILP over unary predicates and comparing this with full ILP, it might be worthwhile exploring the development of quantified defeasible logics, how these might further extend the expressiveness of defeasible logic in a legal context, and whether it is worth sacrificing computational efficiency for increased expressiveness.

Other extensions include the development of extensions that allow reasoning in multiple steps (i.e., a justification that involves multiple sub-conclusions) or

with background knowledge. We have disregarded this because simple strategies to solve these problems are well known and understood in the ILP community [34] – these techniques can be directly applied in the same way when using HeRO. A more advanced approach might automatically infer sub-conclusions – this would require modifications to the algorithm and a significant amount of additional research.

8.2 HeRO over Continuous Variables

Like many other approaches to machine learning, our algorithm only works with truth-valued attributes. This is not a serious problem, for it is possible to turn a discrete domain into a set of attributes by converting each value of the domain to a new truth-valued attribute, and turn a continuous domain into a set of attributes by selecting discrete ranges and again creating new truth-valued attributes for the ranges. Other algorithms such as C4.5 [45] handle continuous domains and finite domains directly, negating the need for a knowledge engineer to select suitable ranges at the right level of granularity or with the right thresholds. Techniques such as those used in C4.5 that select the best threshold with lowest entropy might be applicable to HeRO, and is a worthwhile avenue for further exploratory work.

8.3 Development of HeRO as a Product

In order to turn HeRO into a product suitable for industrial use, several implementation concerns need to be further addressed:

8.3.1 Scalability

While we have found HeRO to offer very satisfactory performance and indications suggest that the algorithm is reasonably scalable to moderate databases, before risking significant sums of money on commercial development, a more thorough analysis of the particular demands of a working LDSS is necessary to ensure that the runtime performance is suitable. Even in the face of poor scalability, it should be relatively straightforward to use the techniques briefly mentioned in Section

6.2.2 to identify and eliminate noise attributes and thereby reduce the size of the dataset processed by HeRO (the complexity of the algorithm, as it stands, is primarily in the number of attributes as opposed to the number of records).

8.3.2 User Interface

In order to encourage the use of the LDSS by the public or decision makers, it is necessary to provide an interface that does not make overwhelming use of formal notation, and that allows the interactive exploration of the reasoning behind the conclusions that the LDSS reaches. A comprehensive survey of the significant body of work in user interfaces for expert systems, as it applies to the legal domain, appears in [48]. Options include a conversational style of interactions (whereby users can ask the user interface, "Why?", to explore the details of the rationale), graph-based representations that display the rules in an argument and how they support each other, and collapsible trees that allow the user to drill-down into the structure of the mechanical reasoning. We have already suggested that an expert can annotate rules with "natural language" explanations to allow for the development of friendly forms of interaction, but adapting the established techniques of creating interfaces for expert systems was primarily considered out of scope of this project, to date.

Since the success of the LDSS relies on the quality of data entry, it is also necessary to undergo research to development a flexible interface that allows accurate, consistent and rapid description of cases at a level of simplicity suitable for relatively untrained data entry clerks. This interface may be a simple case of guided generation of an XML document by filling in templates that are automatically formed from XML Schemas (similar to the technique used in Microsoft Visual Studio .NET).

Therefore, we see that further work is necessary to experiment with combinations of formal notation, plain English explanations or even graphical representations that reduce the comprehension effort and make users feel comfortable. But, even before this work begins it is absolutely critical to mitigate some of the risks of this kind of development through market research that identifies the needs and expectations of users and anticipate their likely response to the tool. Even though the approach presented in this thesis is intended to be comparatively low cost, some investment is still necessary to bring the technology to fruition. Feasibility

research is therefore a responsible use of investor funds, and user profiling that would be a result of these studies is only going to improve the quality and the likelihood of acceptance of the product.

8.3.3 Integrated Knowledge Discovery Environments

Not only is it necessary to pay attention to the interface for end-users, it is also important to develop a powerful and integrated environment to assist knowledge engineers and domain experts encode knowledge. This environment would need to incorporate:

- A schema/grammar design tool, for specifying the semi-structured representation and the user interfaces that data-entry clerks interact with when encoding precedent cases,
- A facility for creating queries and maintaining a 'common-sense' knowledge base,
- An end-user GUI designer for specifying a user interface that users can supply factual information about a case, or even an interview-style questioning process,
- A front end for HeRO that allows a knowledge engineer to select attributes, induce theories, and examine the generated rules,
- A rule annotation tool that allows a domain expert to add plain-English explanations to defeasible logic rules,
- A visualization tool that supports multidimensional analysis of precedent cases and of evolutionary change within these precedents,
- An update wizard that provides support for quickly evaluating and updating the state of the knowledge base in the face of evolutionary changes that demand alterations to annotations and queries,
- A deployment wizard that 'compiles' the legal decision support system's design into a format suitable for deploying to legal workers.

Implementing each of these tools should be a reasonably straightforward task – the major challenge is in providing an interface that integrates these features to maximise the ability of the knowledge engineer to design effective legal decision support systems.

8.3.4 Field Experiments

Finally, while our experiment has included real and synthetic datasets, we have not had the time to actually explore the application of HeRO through the entire knowledge discovery process, in a real setting. In experimentally applying HeRO in such settings, the algorithm can be further "tweaked" with pragmatic ends in mind.

Another motivation for field experimentation is the need to further refine the processes of defining schema for precedent cases, and entering these cases. The challenge is in permitting enough flexibility to comprehensively describe every case, while simultaneously ensuring that the semantics are sufficiently well defined (and that the system has enough common-sense to reason about the schema) so that it is not necessary to analyse each case in isolation to extract the value of an attribute. An overly flexible schema is virtually impossible to reason with (the extreme of this being a schema for plain-English unstructured data), but on the other hand an overly restrictive schema is easy to reason with but makes it difficult to capture a full description of the precedents. Field experimentation is the key to finding a suitable balance between the two extremes.

Chapter 9

Conclusions

The lack of success in the development of intelligent legal decision support systems provides an exciting opportunity for commercialisation, and a dire warning of the challenges inherent in the domain. An analysis reveals that there does not appear to be any specific reason why legal decision support systems *cannot* be made, and moreover, there does not appear to be any strong reason why legal decision support systems *should not* be made (provided we do so in cautious steps). This leads us to some surprise at the limited successes of such information systems. But, by examining the existing approaches to constructing LDSSs with a fresh perspective, we see two common failings:

1. Expensive, manual construction methods, or

2. Poor knowledge representations.

Defeasible logic would appear to resolve both of these issues, provided that we have an algorithm for automatic induction of theories. While the theoretical treatment of such an algorithm reveals that the underlying problem belongs to the class of NP optimisation problems and therefore likely to be intractable, inductive logic programming has managed to successfully apply heuristics to give "good enough" theories in "good enough" runtimes. We have applied and extended some of these techniques to create a new algorithm, HeRO that has extremely impressive performance in terms of both runtimes and accuracy.

In the long term, work remains in integrating the algorithm into a knowledge discovery suite and in refining the techniques for representing and processing the

semi-structured representations that give an LDSS the flexibility and 'intelligence' to provide meaningful answers in broad legal domains.

Fortunately, there is no indication that any of this future work is insurmountable. Given the demand for this kind of technology, the potential applications of HeRO are extremely encouraging and are tremendously worthwhile subjects of further research and development.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Michael Stonebraker and Joseph Hellerstein, editors, *Readings in Database Systems*, chapter 7, pages 580–592. Morgan Kaufmann Publishers, 3rd edition, 1998.

[2] Grigoris Antoniou, David Billington, Guido Governatori, and Michael Maher. On the modelling and analysis of regulations. In *Proceedings of the 10th Australasian Conference on Information Systems*, pages 20–29, 1999.

[3] Grigoris Antoniou, David Billington, Guido Governatori, and Michael Maher. A flexible framework for defeasible logics. In *AAAI/IAAI*, pages 405–410, 2000.

[4] Grigoris Antoniou, David Billington, Guido Governatori, and Michael Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287, April 2001.

[5] Grigoris Antoniou, David Billington, Guido Governatori, Michael Maher, and Andrew Rock. A family of defeasible reasoning logics and its implementation. In *Proceedings of the 14th European Conference on Artificial Intelligence*, Amsterdam, 2000. IOS Press.

[6] Grigoris Antoniou, Michael Maher, and David Billington. Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming*, 42(1):47–57, January 2000.

[7] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggio Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability*

Properties, chapter Appendix B, page 426. Springer-Verlag, 1999. See also http://www.nada.kth.se/ viggo/problemlist/.

[8] John Avery, Andrew Stranieri, Guido Governatori, and John Zeleznikow. The identification of defeasible rule sets from databases using association rules. Submitted to The Fourteenth Australasian Database Conference.

[9] Trevor Bench-Capon. Neural networks and open texture. In *Proceedings of the Fourth International Conference on Artificial Intelligence and Law*. ACM Press, 1993.

[10] Trevor Bench-Capon, Frans Coenen, and Paul Leng. An experiment in discovering association rules in the legal domain. In *Proceedings of the Eleventh International Workshop on Database and Expert Systems Applications*, pages 1056–1060, Los Alamitos, California, 2000. IEEE Computer Society.

[11] Anders Berglund, Scott Boag, Don Chamberlin, Mary Fernandez, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML path language (XPath) 2.0. `http://www.w3c.org/TR/2002/WD-xpath20-20020816`, August 2002. W3C Working Draft.

[12] Scott Boag, Don Chamberlin, Mary Fernandez, MichaelDaniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language. `http://www.w3.org/TR/2002/WD-xquery-20020816/`, August 2002. W3C Working Draft.

[13] Laurent Bochereau, Danièle Bourcier, and Paul Bourgine. Extracting legal knowledge by means of a multilayer neural network: Application to municipal jurisprudence. In *The Third International Conference on Artificial Intelligence and Law*, pages 288–296. ACM Press, 1991.

[14] Alexander Boer, Rinke Hoekstra, Radboud Winkels, Tom van Engers, and Frederik Willaert. Proposal for a dutch legal xml standard. In *Proceedings of the eGOV2002 Conference: State of the Art and Perspectives*. Springer Verlag, 2002.

[15] Karl Branting. A generative model of narrative cases. In *The Seventh International Conference on Artificial Intelligence and Law*, pages 1–8, 1999.

[16] Tim Bray, Jean Paoli, Michael Sperberg-McQueen, and Eve Maler. Extensible markup language (XML) 1.0 (second edition). `http://www.w3.org/TR/REC-xml`, October 2000. W3C Working Draft.

[17] Stefanie Brüninghaus and Kevin Ashley. Toward adding knowledge to learning algorithms for indexing legal cases. In *The Seventh International Conference on Artificial Intelligence and Law*, pages 9–17, 1999.

[18] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on xml documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[19] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in expressive description logics. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1581–1634. Elsevier Science Publishers, 2001.

[20] Michael Covington. Logical control of an elevator with defeasible logic. *IEEE Transactions on Automatic Control*, 45(7):1347–1349, 2000.

[21] Giuseppe De Giacomo. *Decidability of Class-based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

[22] Francesco Donini, Maurizio Lenzerinie, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Principles of Knowledge Representation and Reasoning*, pages 193–238. CLSI Publications, 1996.

[23] Béatrice Duval and Pascal Nicolas. Learning default theories. In Anthony Hunter and Simon Parsons, editors, *ECSQARU*, number 1638 in LNCS, pages 148–159, London, 1999. Springer.

[24] Sašo Džeroski. Inductive logic programming and knowledge discovery in databases. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhr Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 117–152. AAAI Press/The MIT Press, 1996.

[25] David Fallside. XML Schema part 0: Primer. `http://www.w3c.org/TR/xmlschema-0/`, May 2001. W3C Recommendation.

[26] Usama Fayyad, Gregory Piatetsky-Shapiro, and Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, November 1996.

[27] William Frawley, Gregory Piatetsky-Shapiro, and Christopher Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13(3):57–70, 1992.

[28] Peter Gammie. COMP9417 project: Shapiro's model inference system. http://www.cse.unsw.edu.au/ cs9417ml/MIS/doc/html/top.html, November 2001.

[29] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, chapter Appendix A, page 222. W. H. Freeman and Company, 1979.

[30] Matthew Ginsberg. AI and nonmonotonic reasoning. In Dov Gabbay, Christopher Hogger, and John Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3. Oxford University Press, 1993.

[31] Guido Governatori and Andrew Stranieri. Towards the application of association rules for defeasible rule discovery. In Bart Verheij, Arno Lodder, Ronald Loui, and Antoinette Muntjewerff, editors, *Frontiers in Artificial Intelligence and Applications*, volume 70. IOS Press, 2001. Proceedings of JURIX 2001.

[32] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, 2001.

[33] Katsumi Inoue and Yoshimitsu Kudoh. Learning extended logic programs. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 176–181. Morgan Kaufmann, 1997.

[34] Nada Lavrăc and Sažo Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.

[35] Michael Maher. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming*, 1(6):691–711, November 2001.

[36] Michael Maher and Guido Governatori. A semantic decomposition of defeasible logics. In *Proceedings of American National Conference on Artificial Intelligence*, pages 299–305. AAAI/MIT Press, 1999.

[37] Marie-Francine Moens, Caroline Uyttendaele, and Jos Dumortier. Abstracting of legal cases: The SALOMON experience. In *The Sixth International Conference on Artificial Intelligence and Law*, pages 114–122, 1997.

[38] Stephen Muggleton. *Inductive Acquisition of Expert Knowledge*. Turing Institute Press, 1990.

[39] Daniele Nardi and Ronald Brachman. An introduction to description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors, *The Description Logic Handbook*, pages 47–100. Cambridge University Press, 2002.

[40] Pascal Nicolas and Béatrice Duval. Representation of incomplete knowledge by induction of default theories. In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczyríski, editors, *Logic Programming and Nonmonotonic Reasoning*, number 2173 in LNAI, pages 160–172. Springer, 2001.

[41] Donald Nute. Defeasible logic. In Dov Gabbay, Christopher Hogger, and John Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3. Oxford University Press, 1993.

[42] Daniel Poulin, Guy Huard, and Alain Lavoie. The other formalization of law: SGML modelling and tagging. In *The Sixth International Conference on Artificial Intelligence and Law*, pages 82–88, 1997.

[43] Henry Prakken. *Logical Tools for Modelling Legal Argument*. Kluwer Academic Publishers, 1997.

[44] Vaughan Pratt. A practical decision method for propositional dynamic logic: Preliminary report. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, pages 326–337. ACM Press, 1978.

[45] John Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, San Mateo, California, 1993.

[46] Ehud Shapiro. *Algorithmic Program Debugging.* PhD thesis, Yale University, 1982. Printed as part of ACM Distinguished Dissertations Series, 1983.

[47] Sebastian Thrun et al. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[48] John Zeleznikow and Dan Hunter. *Building Intelligent Legal Information Systems.* Computer/Law Series. Kluwer Law and Taxation Publishers, 1994.

[49] John Zeleznikow and Andrew Stranieri. Knowledge discovery in the split up project. In *The Sixth International Conference on Artificial Intelligence and Law*, 1997.

Appendix A

C# Implementation of HeRO

The HeRO algorithm has been implemented in the Microsoft .NET programming language, C#. The project consists of the following files:

1. **Home.cs**
   The main entry point of the implementation, including the functionality required to parse a comma separated value (CSV) dataset.

2. **Literal.cs**
   A C# mapping of our definition of literal that appears in Section 4.2.

3. **Record.cs**
   A C# mapping of a record as per the formal definition that appears in Section 5.1.

4. **Rule.cs**
   A C# mapping of a defeasible logic rule (as defined in Section 4.2), including the functionality required to compute the accuracy gain of the rule (as per Section 5.3).

5. **Theory.cs**
   A C# mapping of a theory (as defined in Section 4.2) and the core of the HeRO algorithm.

The source for each of these files appears throughout the remainder of this appendix (in the same order as above).

A.1 Home.cs

```
using System;
using System.IO;
using System.Collections;

namespace DssInduce
{
    /// <summary>
    /// This class is the main entry point of the application.
    /// </summary>
    class Home
    {

        /// <summary>
        /// The collection of training records.
        /// </summary>
        static ArrayList dataset;

        /// <summary>
        /// A description of each attribute (including the conclusion), in the same
        /// order as attributes that appear in the dataset.
        /// </summary>
```

```
static string[] titles;

/// <summary>
/// The name of the attribute that is the conclusion of each record.
/// </summary>
static string conclusionKey;

/// <summary>
/// The main entry point for the application. Reads the datafile, and induces
/// a theory.
/// </summary>
[STAThread]
static void Main(string[] args)
{
    string conclusion = "p";
    ReadDataset(@"c:\honours\dataset.csv", conclusion);
    Theory theory = new Theory(titles, conclusionKey, dataset);
    theory.GenerateTheory(30);
    theory.DumpToConsole(); // show the generated theory
    Console.ReadLine(); // wait for user confirmation, before exiting
}

/// <summary>
```

```
        /// Reads a comma-separated data file, and populates the dataset with
        /// <c>Record</c>s for each object.
        /// </summary>
        /// <param name="fileName">The name of the file to read.</param>
        /// <param name="conclusion">The attribute to use as the conclusion of
        ///        each record.</param>
        static void ReadDataset(string fileName, string conclusion)
        {
            conclusionKey = conclusion;
            TextReader input = File.OpenText(fileName);
            char[] split = new char[] {','};
            titles = input.ReadLine().Split(split);
            dataset = new ArrayList();
            string line;
            while ((line = input.ReadLine()) != null)
            {
                dataset.Add(new Record(titles, line.Split(split), conclusion));
            }
            input.Close();
        }

    }
}
```

A.2 Literal.cs

```
using System;
using System.Collections;

namespace DssInduce
{
    /// <summary>
    /// An instance of <c>Literal</c> corresponds to the usage of literals within
    /// the core of the thesis. A literal is either an atomic propositional variable
    /// or its negation and is used to represent the truth value of an attribute in
    /// rules and in records.
    /// </summary>
    public class Literal
    {

        /// <summary>
        /// A cache of positive literals -- avoids having to construct thousands of
        /// identical <c>Literal</c> objects.
        /// </summary>
        static Hashtable literalsPositive = new Hashtable();

        /// <summary>
```

```
/// A cache of negative literals -- avoids having to construct thousands of
/// identical <c>Literal</c> objects.
/// </summary>
static Hashtable literalsNegative = new Hashtable();

/// <summary>
/// Basically a Factory Pattern -- instead of constructing thousands of
/// identical <c>Literal</c>s, we cache and reuse previously created literals.
/// This gives measurable performance and memory benefits.
/// </summary>
/// <param name="name">The name of the atom.</param>
/// <param name="isPositive">Whether the literal is to be the atom (true), or
///     its negation (false).</param>
/// <returns></returns>
public static Literal getLiteral(String name, bool isPositive)
{
    if (isPositive)
    {
        Literal literal = (Literal)literalsPositive[name];
        if (literal == null)
            literalsPositive[name] = literal = new Literal(name, true);
        return literal;
    }
```

```
    else
    {
        Literal literal = (Literal)literalsNegative[name];
        if (literal == null)
            literalsNegative[name] = literal = new Literal(name, false);
        return literal;
    }
}


/// <summary>
/// The name of the atom underlying the literal.
/// </summary>
string name;

/// <summary>
/// Whether the literal represents the truth value of the atom (true) or
/// of its negation (false)
/// </summary>
bool isPositive;

public Literal(String name, bool isPositive)
{
```

```
    this.name = name;
    this.isPositive = isPositive;
}

// Pass through properties ---

public bool Positive
{
    get
    {
        return isPositive;
    }
}

public string Name
{
    get
    {
        return name;
    }
}

/// <summary>
```

```
/// We override equality based on either:
/// <list type="bullet">
///        <item>Object identity (since we've cached most instances,
///              we're working with canonical versions of objects,
///              and this test should usually be very quick), or</item>
///        <item>Value equality (for the sake of completeness).</item>
/// </list>
/// </summary>
public override bool Equals(Object obj)
{
    if (this == obj)
        return true;
    if (obj.GetType() != this.GetType())
        return false;
    Literal other = (Literal)obj;
    return this.name.Equals(other.name) && this.isPositive == other.isPositive;
}

/// <summary>
/// We override <c>GetHashCode</c>, for the obvious performance benefits of
/// hash based data-access.
/// </summary>
/// <returns></returns>
```

```
        public override int GetHashCode()
        {
            return isPositive ? name.GetHashCode() : -name.GetHashCode();
        }

        public override string ToString()
        {
            return isPositive ? name : "-" + name;
        }

    }
}
```

A.3 Record.cs

```
using System;
using System.Collections;

namespace DssInduce
{
    /// <summary>
    /// An instance of <c>Record</c> corresponds exactly to the theoretical concept
    /// of a record in the thesis.
```

```
/// </summary>
public class Record
{

    /// <summary>
    /// Allows fast access to the literal corresponding to a particular attribute.
    /// </summary>
    Hashtable dataset;

    /// <summary>
    /// Used as a hash-based set to tell whether a given <c>Literal</c> is
    /// present in the <c>Record</c>
    /// </summary>
    Hashtable lookups;

    /// <summary>
    /// The attribute that is the <c>Record</c>'s conclusion.
    /// </summary>
    string conclusionKey;

    /// <summary>
    /// The value of the conclusion as it appears in the dataset.
    /// </summary>
```

```
Literal conclusion;

/// <summary>
/// An annotation used by the induction algorithm. This variable is used
/// as a placeholder for the conclusion that can be derived for this record
/// by the current theory.
/// </summary>
Literal derivedConclusion;

public Record(string[] titles, string[] record, string conclusionKey)
{
    // populate the instance variables
    this.conclusionKey = conclusionKey;
    dataset = new Hashtable(titles.Length);
    lookups = new Hashtable(titles.Length);
    for (int i = 0; i<titles.Length; i++)
    {
        if (record[i] != Theory.Nothing)
        {
            Literal literal = Literal.getLiteral(titles[i], record[i] == Theory.True);
            dataset[titles[i]] = literal;
            lookups[literal] = literal;
        }
```

```
        }
        conclusion = (Literal)dataset[conclusionKey];
    }

    /// <summary>
    /// Resets the annotations to no conclusion.
    /// </summary>
    public void Reset()
    {
        derivedConclusion = null;
    }

    /// <summary>
    /// Checks if the supplied literal appears in the facts of the record.
    /// </summary>
    /// <param name="literal">The literal that might be present in the facts.</param>
    /// <returns><c>true</c>, if the literal is present.</returns>
    public bool HasLiteral(Literal literal)
    {
        return lookups[literal]!=null;
    }

    /// <summary>
```

```
/// This property hides the details of the particular atom of the conclusion
/// but simply concerns itself with the truth value of the conclusion.
/// </summary>
public bool DerivedConclusion
{
    get
    {
        return derivedConclusion.Positive;
    }

    set
    {
        derivedConclusion = Literal.getLiteral(conclusionKey, value);
    }
}

/// <summary>
/// This property indicates whether the <c>DerivedConclusion</c> property
/// is set or not.
/// </summary>
public bool DerivedSet
{
    get
```

```
        {
            return derivedConclusion != null;
        }

        set
        {
            if (!value)
                derivedConclusion = null;
        }
    }

    // Pass through properties ---

    public bool ActualConclusion
    {
        get
        {
            return conclusion.Positive;
        }
    }

    public bool ActualSet
    {
```

```
            get
            {
                return conclusion != null;
            }
        }

    }
}
```

A.4 Rule.cs

```
using System;
using System.Collections;
using System.Text;

namespace DssInduce
{
    /// <summary>
    /// This class represents a rule in a theory, and candidate rules for inclusion
    /// into a theory.
    /// </summary>
    public class Rule
    {
```

```
/// <summary>
/// The list of literals that form the antecedent of the rule.
/// </summary>
ArrayList guard;

/// <summary>
/// A list of records to which the rule is applicable (used for performance
/// reasons).
/// </summary>
ArrayList records;

/// <summary>
/// Used as a hash-based set to determine whether the rule applies to a
/// particular record. (ie. the set-based representation of the
/// <c>records</c> instance variable.
/// </summary>
Hashtable appliesTo;

/// <summary>
/// The names of the atoms that appear in the guard/antecedent of the rule.
/// </summary>
Hashtable names;
```

```
/// <summary>
/// Whether the conclusion of the rule is positive (true) or negative (false).
/// </summary>
bool conclusion;

/// <summary>
/// Whether the rule is a defeater (true) or simply a defeasible rule (false).
/// </summary>
bool isDefeater;

/// <summary>
/// The gain to the accuracy of the theory, that is given by adding this rule.
/// </summary>
int gain;

/// <summary>
/// The name of the (lexicographically) greatest atom that occurs in the
/// antecedent. This attribute is used to ensure that we cannot refine
/// both of a=>p and b=>p into a,b=>p. (ie. this variable eliminates repeats
/// from the search space, and does not destroy the algorithm's correctness).
/// </summary>
string maxGuard;
```

```
/// <summary>
/// Initialises the rule.
/// </summary>
/// <param name="guard">The list of literals that form the guard/antecedent
///     of the rule.</param>
/// <param name="conclusion">Whether the rule is positive or negative.</param>
/// <param name="records">A list of records that is a superset of the records
///     that the rule applies to.</param>
public Rule(ArrayList guard, bool conclusion, ArrayList records)
{
    // set up the instance variables.
    this.guard = guard;
    this.conclusion = conclusion;
    this.records = new ArrayList();
    this.appliesTo = new Hashtable();
    if (guard.Count > 0)
        maxGuard = ((Literal)guard[guard.Count - 1]).Name;
    else
        maxGuard = null;
    names = new Hashtable();
    foreach (Literal a in guard)
        names[a.Name] = a.Name;
```

```
    // work out which records are applicable to this rule
    foreach (Record r in records)
    {
        foreach (Literal a in guard)
        {
            if (!r.HasLiteral(a))
                goto NotPresent;
        }

        //Add the record to the set
        this.records.Add(r);
        appliesTo[r] = r;

    NotPresent:
        continue;
    }
}

// Pass through properties ---

public bool Conclusion
{
    get
```

```
        {
            return conclusion;
        }

        set
        {
            conclusion = value;
        }
    }

    public int Gain
    {
        get
        {
            return gain;
        }

        set
        {
            gain = value;
        }
    }
```

```
public bool IsDefeater
{
    get
    {
        return isDefeater;
    }

    set
    {
        isDefeater = value;
    }
}

/// <summary>
/// Determines whether the rule is applicable to a given record
/// </summary>
/// <param name="record">The record for which the rule might be applicable.</param>
/// <returns><c>true</c>, if the rule is applicable to the record.</returns>
public bool AppliesTo(Record record)
{
    // "lookup" the set.
    return appliesTo[record] == record;
}
```

```
/// <summary>
/// Annotates each rule that the record applies to, with an annotation that is
/// consistent with this rule's conclusion.
/// </summary>
public void Apply()
{
    foreach (Record r in records)
    {
        if (isDefeater)
        {
            if (r.DerivedSet && r.DerivedConclusion != conclusion)
                r.DerivedSet = false;
        }
        else
            r.DerivedConclusion = conclusion;
    }
}

/// <summary>
/// Computes the gain in accuracy that is due to this rule.
/// </summary>
/// <param name="best">The gain of the rule.</param>
```

```
/// <param name="bound">An upper bound on the gain of any refinements (via
///     <c>GetChildren</c>) of this rule.</param>
/// <param name="winningConclusion">The conclusion that lead to best gain.</param>
/// <param name="isDefeater">Whether the best gain was acheived by a
///     defeater.</param>
public void ComputeGain(out int best, out int bound, out bool winningConclusion, out bool isDefeater)
{
    int trueGain = 0;
    int trueGainMax = 0;
    int trueDefeaterGain = 0;
    int trueDefeaterGainMax = 0;
    int falseGain = 0;
    int falseGainMax = 0;
    int falseDefeaterGain = 0;
    int falseDefeaterGainMax = 0;
    // Compute the score, as per the thesis
    foreach (Record r in records)
    {
        if (!r.DerivedSet)
        {
            if (!r.ActualSet)
            {
                trueGain--;
```

```
            falseGain--;
        }
        else if (r.ActualConclusion)
        {
            trueGain++;
            trueGainMax++;
        }
        else
        {
            falseGain++;
            falseGainMax++;
        }
    }
    else
    {
        bool derivedConclusion = r.DerivedConclusion;
        if (!r.ActualSet)
        {
            if (derivedConclusion)
            {
                falseDefeaterGain++;
                falseDefeaterGainMax++;
            }
```

```
        else
        {
            trueDefeaterGain++;
            trueDefeaterGainMax++;
        }
    }
    else
    {
        bool actualConclusion = r.ActualConclusion;
        if (derivedConclusion && actualConclusion)
        {
            falseGain--;
            falseDefeaterGain--;
        }
        else if (derivedConclusion && !actualConclusion)
        {
            falseGain++;
            falseGainMax++;
        }
        else if (!derivedConclusion && actualConclusion)
        {
            trueGain++;
            trueGainMax++;
```

```
            }
            else //(!derivedConclusion && !actualConclusion)
            {
                trueGain--;
                trueDefeaterGain--;
            }
        }
    }
}

// Now that we've computed the score, work out which conclusion gives the
// best gain.
best =  Math.Max(Math.Max(trueGain, falseGain),
                 Math.Max(trueDefeaterGain, falseDefeaterGain));
bound = Math.Max(Math.Max(trueGainMax, falseGainMax),
                 Math.Max(trueDefeaterGainMax, falseDefeaterGainMax));
if (trueGain == best)
{
    winningConclusion = true;
    isDefeater = false;
}
else if (falseGain == best)
{
```

```
            winningConclusion = false;
            isDefeater = false;
        }
        else if (trueDefeaterGain == best)
        {
            winningConclusion = true;
            isDefeater = true;
        }
        else //if (falseDefeaterGain == best)
        {
            winningConclusion = false;
            isDefeater = true;
        }
    }

    /// <summary>
    /// Finds all refinements of the current rule
    /// </summary>
    /// <param name="titles">The set of propositional attributes that can be used
    ///     to extend the guard/antecedent.</param>
    /// <param name="conclusionKey">The conclusion of the rule (so that the
    ///     conclusion is not added as an guard/antecedent of any refinement.</param>
    /// <returns>A list of new rules, that represent the refinements of the
```

```
///     current rule.</returns>
public ArrayList GetChildren(string[] titles, string conclusionKey)
{
    ArrayList children = new ArrayList();
    foreach (string s in titles)
    {
        if (s.CompareTo(maxGuard) > 0 && s != conclusionKey && names[s] == null)
        {
            Literal yes = Literal.getLiteral(s, true);
            ArrayList yesList = new ArrayList(guard);
            yesList.Add(yes);
            children.Add(new Rule(yesList, true, records));
            Literal no = Literal.getLiteral(s, false);
            ArrayList noList = new ArrayList(guard);
            noList.Add(no);
            children.Add(new Rule(noList, true, records));
        }
    }
    return children;
}

public override string ToString()
{
```

```
            StringBuilder sb = new StringBuilder();
            sb.Append("[");
            sb.Append(gain.ToString().PadLeft(3, ' '));
            sb.Append("] ");
            bool first = true;
            foreach (Literal a in guard)
                if (!first)
                    sb.Append(", ").Append(a);
                else
                {
                    first = false;
                    sb.Append(a);
                }
            if (isDefeater)
                sb.Append(" ~> ");
            else
                sb.Append(" => ");
            sb.Append(conclusion);
            return sb.ToString();
        }

    }
}
```

A.5 Theory.cs

```
using System;
using System.Collections;

namespace DssInduce
{
    /// <summary>
    /// A <c>Theory</c> is a collection of rules that describe a dataset.
    /// This class includes the core of the HeRO algorithm.
    /// </summary>
    public class Theory
    {

        // Simple, global, constants ---

        public static readonly string True = "true";
        public static readonly string False = "false";
        public static readonly string Nothing = "";

        /// <summary>
        /// The current list of rules in the theory (a list of <c>Rule</c>s).
        /// </summary>
```

```
ArrayList rules;

/// <summary>
/// The training dataset (a list of <c>Record</c>s).
/// </summary>
ArrayList dataset;

/// <summary>
/// The queue of rules to be considered in the search.
/// </summary>
Queue queue;

/// <summary>
/// The names of the propositional attributes that appear in the dataset.
/// </summary>
string[] titles;

/// <summary>
/// The name of the attribute that is the conclusion of the dataset.
/// </summary>
string conclusionKey;

public Theory(string[] titles, string conclusionKey, ArrayList dataset)
```

```
{
    rules = new ArrayList();
    this.dataset = dataset;
    this.titles = titles;
    this.conclusionKey = conclusionKey;
}

/// <summary>
/// Induces a theory of (up to) a given size.
/// </summary>
/// <param name="size">The target size of the theory (the algorithm halts
///     after <c>size</c> rules are generated).</param>
public void GenerateTheory(int size)
{
    for (int i=0; i<size; i++)
    {
        Rule next;
        int pivot;
        ChooseRule(out next, out pivot);
        if (next == null)
            break;
        rules.Insert(pivot, next);
    }
```

```
}

/// <summary>
/// Selects the rule with teh best gain to add to the theory.
/// </summary>
/// <param name="found">The new rule to add.</param>
/// <param name="rulePivot">The pivot -- the position in the totally ordered
///     superiority relation that the rule should be added.</param>
public void ChooseRule(out Rule found, out int rulePivot)
{
    int best = 0;
    found = null;
    bool conclusion = false;
    bool isDefeater = false;
    rulePivot = 0;
    // Select a pivot
    for (int pivot = 0; pivot <= rules.Count; pivot++)
    {
        // Compute the base image
        foreach (Record rec in dataset)
            rec.Reset();
        foreach (Rule rule in rules.GetRange(0, pivot))
            rule.Apply();
```

```
// Add only the records that are applicable
ArrayList view = new ArrayList(rules.Count);
foreach (Record rec in dataset)
{
    bool putIn = true;
    foreach (Rule rule in rules.GetRange(pivot, rules.Count - pivot))
        if (rule.AppliesTo(rec))
        {
            putIn = false;
            break;
        }
    if (putIn)
        view.Add(rec);
}

// Reset the search
ResetRule(view);

// And look for a better rule
for (Rule next = NextRule(); next != null; next = NextRule())
{
    int nextbest;
```

```
                int nextbound;
                bool nextconclusion;
                bool nextIsDefeater;
                next.ComputeGain(out nextbest, out nextbound, out nextconclusion, out nextIsDefeater);
                if (nextbest > best)
                {
                    best = nextbest;
                    found = next;
                    conclusion = nextconclusion;
                    isDefeater = nextIsDefeater;
                    rulePivot = pivot;
                }
                if (best < nextbound)
                    AddRules(next.GetChildren(titles, conclusionKey));
            }
        }
        if (found != null)
        {
            found.Conclusion = conclusion;
            found.Gain = best;
            found.IsDefeater = isDefeater;
        }
    }
```

```
/// <summary>
/// Clears the current search space, and adds the empty rule as a
/// new starting point.
/// </summary>
/// <param name="records">The records that describe the dataset.</param>
void ResetRule(ArrayList records)
{
    if (queue == null)
        queue = new Queue();
    queue.Clear();
    queue.Enqueue(new Rule(new ArrayList(), true, records));
}

/// <summary>
/// Retreives and removes the next rule in the search space.
/// </summary>
Rule NextRule()
{
    if (queue.Count == 0)
        return null;
    return (Rule)queue.Dequeue();
}
```

```
/// <summary>
/// Adds a list of rules to the search space.
/// </summary>
void AddRules(ArrayList rules)
{
    foreach (object o in rules)
        queue.Enqueue(o);
}

/// <summary>
/// Adds a single rule to the search space.
/// </summary>
public void AddRule(Rule rule)
{
    rules.Add(rule);
}

/// <summary>
/// Displays the induced theory on the user's console.
/// </summary>
public void DumpToConsole()
{
```

```
            foreach (Rule r in rules)
                Console.WriteLine(r);
        }
    }
}
```

Appendix B

XSLT Template for Dataset Generation

In lieu of an interactive environment for generating and interacting with queries, it is possible to create datasets from XPath2 attribute definitions, by inserting the name and query appropriately within the following sample template. XSLT stylesheets (such as this example) can be directly executed on XSLT processors with XPath2 support, such as Michael Kay's SAXON[1].

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="text"/>

    <!-
           Set-up the Titles
           -----------------
           Each attribute should appear as a separate item in the
           comma-separated list that appears at the top of the generated
           dataset.
    -->
    <xsl:template match="/">
        <xsl:text>Male,Youth,PreviousIncident,...</xsl:text>
        <xsl:apply-templates/>
```

[1] Available at `http://saxon.sourceforge.net/`

```
    </xsl:template>

    <!-
           Populate Each Record
           --------------------
           Each of the XPath2 expressions for the attributes appear in the
           select queries of separate value-of operations. These are then
           comma-separated.
    -->
    <xsl:template match="case">
        <xsl:value-of
               select="person[attribute::type = 'defendant']/gender = 'Male'"/>
        <xsl:text>,</xsl:text>
        <xsl:value-of
               select="person[attribute::type = 'defendant']/age &lt; 21"/>
        <xsl:text>,</xsl:text>
        ...
    </xsl:template>

</xsl:transform>
```

Appendix C

Prolog Defeasible Logic Meta-program

In [36], a meta-program for logic programming languages such as Prolog is presented. This work is simplified for our notation, and summarized here, because the meta-program is an intuitive formalisation of the proof theory of defeasible logic, for an experienced Prolog programmer.

We begin by declaring dynamic predicates:

```
:- dynamic fact/1.
:- dynamic defeasible/3.
:- dynamic defeater/3.
:- dynamic sup/2.
```

These predicates, are used to represent a defeasible logic theory as follows:

`fact(`*Literal*`)`
The facts of the case (as individual literals), are asserted in separate `facts`.

`defeasible(`*Name*`, `*Ante*`, `*Conc*`)`
Each defeasible rule is mapped into an equivalent assertion, consisting of: *Name*, a unique identifier for the rule (used in the superiority relation); *Ante*, a list of assumptions (or antecedents) of the rule; and *Conc*, a literal representing the

conclusion of the rule.

`defeater(`*Name*`,` *Ante*`,` *Conc*`)`
In the same way as defeasible rules, each defeater is mapped into an assertion.

`sup(`*NameStronger*`,` *NameWeaker*`)`
Each pair of rules that appears in the superiority relation is mapped by asserting the names of the rules in separate tuples of the `sup` predicate. The first argument is the name of the stronger rule.

Assuming the above predicates have been asserted to the Prolog database, we define the following auxiliary predicates:

```
%
% negate(+LiteralIn, -LiteralOut)
%     Negates the literal in the first argument.
%
negate(Atom, not(Atom)) :- atom(Atom).
negate(not(Atom), Atom).

%
% rule(-Name, -Ante, -Conc)
%     A rule is either defeasible or a defeater.
%
rule(Name, Ante, Conc) :- defeasible(Name, Ante, Conc).
rule(Name, Ante, Conc) :- defeater(Name, Ante, Conc).
```

And finally, the defeasible predicate can be defined:

```
%
% defeasibly(+Conc)
%  Succeeds if Conc follows from the theory.
%
defeasibly(Conc) :-
      fact(Conc).
```

```
defeasibly(Conc) :-
      defeasible(_Name, Ante, Conc),
      forall(member(Conj, Ante), defeasibly(Conj)),
      \+ overruled(Conc).

overruled(Conc) :-
      negate(Conc, NegConc),
      rule(CounterArgument, Ante, NegConc),
      forall(member(Conj, Ante), defeasibly(Conj)),
      \+ defeated(CounterArgument, Conc).

defeated(CounterArgument, Conc) :-
      sup(Stronger, CounterArgument),
      defeasible(Stronger, Ante, Conc),
      forall(member(Conj, Ante), defeasibly(Conj)).
```

Given the above meta-program and a suitable set of Prolog assertions P, of a defeasible logic theory T and facts F, we can define provability in defeasible logic in terms of the provability relation of Prolog, `?-`, as follows:

$T, F \vdash +\partial p$ iff P `?- defeasibly(`p`).`
$T, F \vdash -\partial p$ iff P `?- \+defeasibly(`p`).`