

Spreadsheets for Legal Reasoning: The Continued Promise of Declarative Logic Programming in Law

by

Jason Patrick Morris

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Laws

In Faculty of Law and Department of Computing Science

University of Alberta

© Jason Patrick Morris, 2020

Abstract

The legal services market is one in which there is too much demand, and too little supply. One method of increasing supply in a market is to increase efficiency by automating. Automated legal services require the automation of legal reasoning. Declarative logic programming (DLP) has long been recognized as well-suited to the automation of legal reasoning. This dissertation reviews the legal academic literature surrounding the automation of legal services using DLP, which has been discussed primarily with regard to how it can be used to build “expert systems”. This dissertation argues that most of the criticisms of the use of expert systems for automating legal services can be addressed by using modern DLP technologies, conceiving of the encoding as being representative of an interpretation of the law rather than the law’s correct meaning, and increasing ease-of-use for non-programmers.

The dissertation then proposes a set of 7 criteria of suitability for DLP tools developed from a legal services automation perspective. A survey of available tools is performed, comparing the available tools to these criteria. The dissertation advocates for the development of open source DLP tools that have both accessibility features (ease of use and price), and at least one of the five technical features. The dissertation concludes with the description of an open-source tool developed by the author as an ABA Innovation Fellowship project, which is open-source, free, aims to be easy to use, and implements case-based reasoning to allow users to automate reasoning around open-textured legal concepts.

Preface

Portions of this dissertation are adapted from work submitted for credit in LAW 696 and CMPUT 605 as a part of this degree program. A demonstration paper based on the work in Chapter 7 was published in the 2019 Proceedings of the International Conference on Artificial Intelligence and Law (Jason Morris, “User-Friendly Open-Source Case-Based Legal Reasoning” [2019] Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law 270).

None of the work in this dissertation has been submitted for credit in any other degree program.

Unless otherwise noted this dissertation is my own work.

*For Maja, Liam, Gabe, and Oliver
for allowing me to be myself.*

Acknowledgements

I am grateful for the assistance of my advisor in the Faculty of Law, Dr. Cameron Hutchison, and my advisor in the Department of Computing Science, Dr. Randy Goebel, and the additional members of my review committee.

I am grateful to the Faculty of Law and the Department of Computing Science, as well as the Faculty of Graduate Studies and Research for their unflinching support of my interdisciplinary course of study.

I am grateful also to the University of Alberta Faculty of Law for awarding me a scholarship to pursue these studies.

Coherent Knowledge provided valuable feedback and insights to encoding written legal rules in the ErgoAI and ErgoLite languages.

Thank you to the American Bar Association Center for Innovation, in particular Chase Hertel, Sarah Hoffmeyer, and Jordan Furlong. My being awarded the 2018/2019 ABA Innovation Fellowship allowed me to go from an idea about what ought to be done to actually doing it.

Thank you also to Clio, and in particular Jack Newton and Joshua Lennon, who recognized the potential of that fellowship project and whose sponsorship of it allowed it to expand into the project it now is.

Thank you to Jonathan Pyle, author of Docassemble, for his availability and support over the course of my research.

Thank you to Dr. Kevin Ashley for his openness and generosity.

Thank you to Dr. Matthias Grabmair for his expertise and contribution of OpenLCBR.

No words could express the gratitude I feel for my wife Maja, and my children Liam, Gabriel, and Oliver, whose support of their husband and dad has been absolutely unwavering.

Contents

1	Introduction	1
2	Declarative Logic Programming (DLP) in Law	4
2.1	What are “Written Legal Rules”?	4
2.2	What is “Declarative Logic Programming”?	6
2.2.1	What is Declarative Programming?	6
2.2.2	What is Logic Programming?	7
2.2.3	Rules, Facts, and Questions	7
2.3	Why Declarative Logic Programming?	8
2.3.1	Translation, not Reformulation	9
2.3.2	Increased Efficiency for Encoding and Maintaining	11
2.3.3	Explainability	12
2.4	Applications of Encoding Legal Rules in DLP	13
2.5	Conclusion	14
3	Using DLP To Find Legislative Bugs: An Experiment	15
3.1	Introduction	15
3.2	Kraft	16
3.2.1	Background of the Case	16
3.2.2	Our “Bug” in the Code	17
3.3	Encoding Kraft	19
3.3.1	The ErgoAI Language	19
3.3.2	The Selected Scope	19
3.3.3	Encoding Counterfactual Conditions	21
3.3.4	Using ErgoAI to Test for the Bug	23
3.4	The Results	25
3.5	Conclusions	27
4	Legal Scholarship on DLP in Law	29
4.1	Expert Systems and DLP	29
4.2	Why Legal Scholarship?	29
4.3	Susskind	31
4.4	Popple	34
4.5	Leith	35
4.6	Ashley	36
4.7	McCarty	43
4.8	Addressing the Criticisms	45
4.8.1	The Legal Services Supply Perspective	45
4.8.2	Standards of Appropriateness for Automated Legal Services	46
4.8.3	We Encode Interpretations, not “the Law”	47
4.8.4	Concerns with Statutory Interpretation Can be Mitigated, and Do Not Apply Universally	48

4.8.5	DLP Technology Has Improved	52
4.9	The Real Challenge	52
4.9.1	The Solution: The Legal Expert Is the Programmer	53
4.9.2	Spreadsheets for Legal Reasoning	53
4.10	Conclusion	56
5	Desirable Qualities in DLP Tools for Automation of Legal Services	57
5.1	Introduction	57
5.2	Affordability	58
5.3	Uncertainty	58
5.4	Explainability	59
5.5	Case-Based Reasoning	59
5.6	Temporal Reasoning	60
5.7	Defeasibility	61
5.8	Usability	62
5.9	Conclusion	62
6	Survey of Selected Tools for Automating Legal Reasoning with DLP	63
6.1	Criteria for Inclusion	63
6.2	Selected Tools	64
6.2.1	ErgoAI/ErgoLite	64
6.2.2	Neota Logic	65
6.2.3	Oracle Policy Automation	66
6.2.4	Regulation as a Platform	68
6.2.5	Docassemble	70
6.2.6	DataLex	73
6.3	Summary of Available Options	75
6.4	What is Missing?	77
6.5	What Should We Build?	77
6.6	Conclusion	78
7	User-Friendly Legal Case-Based Reasoning	79
7.1	Introduction	79
7.2	OpenLCBR	79
7.3	The IBP Algorithm In Brief	80
7.4	Docassemble-OpenLCBR	83
7.5	Results	84
7.6	Temporal Reasoning in Procedural Languages	88
7.7	Impressions	90
8	Conclusion	92
8.1	Summary	92
8.2	The Question of Scale	93
8.3	Future Work	94
	References	96
	Appendix A Oracle Policy Automation Encoding of Adult Interdependent Partnership Act	99
	Appendix B ErgoAI Encoding of Adult Interdependent Partnership Act	107

List of Tables

3.1	Variables used in ErgoAI Code	23
3.2	Bugs Found Automatically in Copyright Law	26
6.1	Summary of available DLP tools and their features	76
7.1	Results of Leave-One-Out Testing on Relationship of Interdependence Reasoner	87

List of Figures

6.1	RaaP's Rule-Browsing Interface.	69
6.2	RaaP's Graph Visualization Interface.	70
6.3	An interview created in Docassemble.	71
6.4	An interactive consultation generated by DataLex.	74
7.1	Expandable reasons for a prediction are displayed to the user in docassemble-openlcb.	85
7.2	How the list of factors are displayed to the user in docassemble-openlcb.	86
7.3	How the details of a factor are displayed to the user in docassemble-openlcb.	86
7.4	How the details of an issue are displayed to the user in docassemble-openlcb.	87

Listings

2.1	Example of DLP Rule	7
2.2	Example of Encoding Socrates is Mortal	8
3.1	ErgoAI Code for section 27(2)(e) Copyright Act	21
3.2	Code To Find “Bug” in Copyright Act	24
6.1	Code as entered into DataLex tool.	73
7.1	Example of Python Code Block in Docassemble Interview	88
7.2	Excerpt of Python code for Calculating the Start of an AIP	89
B.1	ErgoAI Encoding of Adult Interdependent Partnerships Act	107
C.1	ErgoAI Encoding of Kraft	118

Chapter 1

Introduction

This dissertation seeks to contribute an interdisciplinary perspective to the academic conversation surrounding the use of declarative logic programming in the automation of legal reasoning around written legal rules. Specifically, it seeks to approach the topic from the perspective of the following question: How can the potential of declarative logic programming in law be brought to bear to automate legal services in a responsible way?

In Chapter 2 this dissertation defines “written legal rules” and “declarative logic programming”. It then provides a brief introduction to declarative logic programming, and answers the question of why these tools in particular are deserving of attention, arguing that they have a great deal of potential for automating reasoning around written legal rules.

As an example of the potential applications discussed in Chapter 2, Chapter 3 describes an experiment to demonstrate the potential use of these technologies for scenario testing. Statutory and common law rules surrounding Canadian copyright are encoded in the ErgoAI¹ programming language to determine whether the encoding of those rules would be capable of predicting scenarios in which the outcome arrived at in the Supreme Court of Canada decision in *Euro Excellence v Kraft*, 2007 SCC 37 (CanLII) [*Kraft*] would be obtained. By generating a wide spectrum of possible fact scenarios the code was able to find three distinct fact scenarios in which the result in *Kraft* would

¹Over the course of the writing of this dissertation, the names of some of the software products changed. The programming language offered by Coherent Knowledge called Ergo was renamed ErgoAI. The open source version of this language called Flora-2 was renamed ErgoLite. I have attempted to use only the newer names.

occur, of which the facts of *Kraft* are one.

Chapter 4 is a brief review of and response to legal and interdisciplinary academic writing on the viability of expert systems in law. This chapter argues that most of the difficulties identified with declarative logic programming tools for the automation of legal services are misconceived, have been solved by technical advances, or can be solved by improving the ease of use of the tools in order to facilitate their use by legal subject matter experts. It argues for a focus, in future tool development, on ease of use for non-programmers.

In Chapter 5, the dissertation sets out a list of seven desirable features for a tool designed to facilitate the use of declarative logic programming to automate reasoning around legal rules.

Chapter 6 is a survey of legal DLP tools available commercially or on an open-source basis, addressing how each of the tools satisfies the requirements set out in Chapter 5. This chapter then identifies significant gaps in the tools that are available, and argues that development efforts should focus on tools that are open source, easy to use, and feature one or more of the other 5 features listed in Chapter 5.

Chapter 7 then describes an effort by the author to develop a tool that is open source, easy to use, and includes the additional desirable features of explainability and case-based reasoning. Docassemble-OpenLCBR was an ABA Innovation Fellowship project undertaken by the author as a part of his research. The project demonstrates that lawyer-friendly interfaces to features like case-based reasoning in open source tools are possible.

The dissertation then concludes in Chapter 8, reiterating its major arguments. First, the only remaining obstacle to the adoption of DSL tools in the automation of legal reasoning is what has been called the “knowledge acquisition bottleneck”. Second, a solution to the knowledge acquisition bottleneck is to make the person who has the knowledge and the person who uses the DLP tool the same person. Third, this solution does not require legal subject matter experts to learn to code as that is traditionally understood, but rather we need a step forward in usability in logic programming analogous to the step forward in usability for mathematics programming that occurred with

the advent of electronic spreadsheets. That is to say, we need Spreadsheets for Legal Reasoning. Fourth, there are significant gaps in the available tools for the automation of legal reasoning around written rules, and these gaps should be filled with free, open-source, easy-to-use DLP tools that feature one or more of uncertainty, defeasibility, temporal reasoning, case-based reasoning, and explanation. And lastly, the experiment in Chapter 7 demonstrates that this is a realistic goal.

Chapter 2

Declarative Logic Programming (DLP) in Law

This dissertation examines only declarative logic programming tools, which are declarative logic programming languages, or tools based on those languages. It is also limited to an examination of the usefulness of these tools in automating legal reasoning with regard to written legal rules. This Chapter will set out what exactly “written legal rule” means in this dissertation. It will then explain what “declarative logic programming” is, and set out why declarative logic programming tools are worthy of particular scrutiny with regard to automating legal reasoning around written legal rules.

2.1 What are “Written Legal Rules”?

This dissertation does not address the ability to automate reasoning about “the law” writ large. It focuses only on automating reasoning around “written legal rules.” And as we will see in later Chapters, not all written legal rules are appropriate for encoding in these tools (or perhaps at all).

For the purpose of this dissertation, a statement is a written legal rule if it is written in natural language, it is intended to have legal effect, and it is drafted not in a narrative or persuasive way, but in the succinct, specific, general and prescriptive or proscriptive manner of legal rules. Clearly included in this category would be most legislation, most regulations, some policies, and most contracts. Excluded from the category would be most holdings of law

that appear in judgements.

The distinction is not black and white as between statutes and case law. If a judgment sets out a statement of law that is sufficiently general and specific as to be comparable to the format of legislation, portions of judgements may also be caught within the definition of “written legal rules.” Or, someone may write a legal rule that is derived from the ratio decendi of one or more cases that fits in the category.

For example, in the recent Supreme Court Case of *Canada (Minister of Citizenship and Immigration) v Vavilov*, 2019 SCC 65 (CanLII) [*Vavilov*], the Court sets out circumstances in which the correctness standard of review might apply. One such circumstance is where the legislation explicitly requires a correctness standard. The ratio decidendi “if a law states that review is on the standard of correctness then the standard of correctness applies”, once written, constitutes a written legal rule that would fall into the scope of this dissertation.

On the other hand, *Vavilov* also stands for the proposition that in the case of a person whose citizenship is denied because their parents were foreign spies at the time of their birth in Canada, they nevertheless can obtain citizenship by birth in Canada. That may be a “rule” of law that comes from *Vavilov*, but it is not expressed as a rule in the judgement in those succinct and general terms. The succinct statement above is derived from the judgement, not contained in it. As such, this “rule”, despite having the same force of law and coming from the same primary source, is not a written legal rule in the sense that term is used in this dissertation.

Portions of legislation, regulation, and contract will be written in language that fails to satisfy the requirements. In particular, purpose statements will generally not be written legal rules because they lack prescriptive or proscriptive content.

At this point, the dissertation does not distinguish between written legal rules that are useful to encode, and those that are difficult or unwise to encode. For example, a subjective legal written rule, is nevertheless a written legal rule. Issues of subjectivity and vagueness will be dealt with in detail later in the

dissertation.

2.2 What is “Declarative Logic Programming”?

A programming language is an interface between a computer and a human being, the programmer. In the same way that there are many different user interfaces for desktop computers, there are many different programming languages that have been developed for different purposes.

2.2.1 What is Declarative Programming?

There are many different ways of categorizing programming languages, but one of the most significant divisions is between “imperative” programming languages and “declarative” programming languages.

An imperative programming language is a language which sets out the steps that a computer should follow in order to obtain a certain objective, and how each step should change the state of the data the program is manipulating.

For example, imagine that you were trying to instruct a computer on how to fill a grocery bag. In an imperative language you would give sequential instructions, like this: “First, sort all the items according to their size. Starting with the largest object, examine each object to see if it is heavier than all the other objects that have not been bagged. If it is, bag it and then examine the next object. If it is not, return it to the table and examine the next object.”

You can see that the instructions are expected to be followed in a certain order. That order can be complicated because some instructions are repeated (e.g. “examine each object”), and not all instructions happen every time (e.g. “if it is heavier than all the other objects that have not been bagged”). In an imperative programming language these sorts of basic structures are called looping and conditional statements.

Declarative programming languages, instead of specifying a set of instructions for the computer to follow, set out an objective that the computer should seek. There is then another piece of software that interprets the objectives and the rules to follow in order to generate a procedure like the one above. In this

way, a declarative programming language allows you to give the computer a set of instructions on how to write its own imperative program to solve a problem.

Using declarative language to program a computer to fill a bag of groceries, you might say something like this: “All Objects should be placed in the Bag. Usually, large Items should go in before smaller items. If a large item is lighter than other smaller items, it should go in after those smaller items.”

A computer receiving these instructions might then generate its own imperative procedure for filling grocery bags similar to the one above.

You can see that these instructions are not expected to be followed in a particular order. They do not set out a process for the computer to follow that will achieve the desired result. They merely state the desired result.

So declarative programming can be understood in a simplified way as programming in a language that sets out objectives, not processes, with the help of another piece of software that will generate the required processes.

2.2.2 What is Logic Programming?

Logic programming languages are a category of declarative programming languages. What distinguishes logic programming from other forms of declarative programming is that the basic unit of programming is a logical “rule”, as opposed to an “instruction”. The software that generates a process to follow those rules and achieve the objective, called a “reasoner”, implements the rules of a formal logic like *modus ponens*.

I refer to these languages as declarative logic programming languages (or DLP languages) in this dissertation.

2.2.3 Rules, Facts, and Questions

A declarative logic programming language typically uses three primary statement types: rules, facts, and questions. A rule is stated as follows:

Listing 2.1: Example of DLP Rule

```
1 Conclusion :- Conditions.
```

This rule states that the conclusion is true, if all of the conditions are true. The symbol “:-” is used as an approximation of the logical implication symbol \vdash .

In addition to rules, declarative logic programming languages allow you to state facts, and to ask questions, by omitting one or the other of the two parts of a rule. A fact omits the conditions, and a question omits the conclusion. To distinguish between the two, a question may include an identifier at the start of the line, such as “?-”.

As an example, one might ask the computer to determine whether or not Socrates is mortal by giving it the following expressions:

Listing 2.2: Example of Encoding Socrates is Mortal

```
1 isMortal(X) :- isHuman(X) .
2 isHuman(socrates) .
3 ?- isMortal(socrates) .
```

The first line is a rule, which states that a thing “X” is mortal if that thing is human. The second line says that Socrates is human. The third line asks whether Socrates is mortal.

If you ran this logic program, the computer would answer “Yes.”

2.3 Why Declarative Logic Programming?

Written legal rules can be, and regularly are, automated in a variety of tools. But declarative logic programming languages have always been recognized as being particularly well suited to the task of automating legal reasoning.

One of the first and most influential declarative logic programming languages is Prolog, which was developed in the 1970s.¹ Very shortly after its development, it was recognized that Prolog had utility in automating legal reasoning.² One early experiment involved encoding the British Nationality Act of 1981 in Prolog.³

¹M Van Emden and R Kowalski, “The Semantics of Predicate Logic as a Programming Language” (1976) 23(4) *Journal of the ACM (JACM)* 733.

²Marc A Borrelli, “Prolog and the law: Using expert systems to perform legal analysis in the uk” (1989) 3 *Software LJ* 687.

³Marek J Sergot et al., “The British Nationality Act as a logic program” (1986) 29(5) *Communications of the ACM* 370.

While the potential of declarative logic programming for modeling legal rules was recognized early on, declarative logic programming is not commonly used, even for automating legal services. So why should we examine declarative logic programming tools in particular?

There are at least three arguments that declarative logic programming is preferable to imperative programming for the purpose of automating legal reasoning specifically with regard to written legal rules.

2.3.1 Translation, not Reformulation

“Imperative” and “declarative” are called “paradigms” of programming languages. In the same way that programming languages have paradigms, so does writing in natural languages. Writing in a natural language can be imperative, procedural, declarative, logical, narrative, persuasive, and fall into any number of other categories.

Written legal rules tend to be written in a paradigm that is very similar to the logical declarative paradigm. The statements tend to set out outcomes that are to be achieved, and outcomes that are to be avoided, as opposed to setting out the only process by which those outcomes should be achieved or avoided. And they can be analysed in a deductive manner.⁴

The stereotypical example of “keep off the grass” demonstrates this. This is a declarative statement that states that an objective, that at any point in time the number of people on the grass should be zero. It is not an imperative or procedural statement. If it were, it would read something more akin to “if you are on the grass, get off the grass, and if you are not on the grass, don’t move onto the grass.” This process of converting the declarative statement “keep off the grass” to the imperative statements “if you are on the grass, get off the grass, and if you are not on the grass don’t move onto the grass” I refer to as “reformulation.”

I use the word “reformulation” here to distinguish between it and “translation.” Translation changes only the language, reformulation changes the mode

⁴Cameron Hutchison, *The Fundamentals of Statutory Interpretation* (LexisNexis Canada 2018) at p. 20.

of expression, such as from declarative to imperative. For example, drawing a copy of a picture is a type of translation. The model, a two dimensional representation, doesn't change. Describing a photo in words is a reformulation. It requires that a two-dimensional image be converted to a verbal description.

We tend to write legal written rules declaratively, and in ways that can be modeled in deductive reasoning. Modelling a legal rule written in declarative natural language in a declarative computer language requires translation, but it does not require reformulation. It is argued here that the absence of a need to reformulate a rule when modelling it in a DLP tool simplifies the task of modelling that rule. It also results in a structure to the model of the rules that more closely mirrors the structure of the rules themselves. Reformulating a rule in procedures generates something that is typically much more complicated than the rule was, and may be completely unrecognizable compared to its source.

This increased simplicity and greater similarity in structure have two positive effects. First, it increases the amount of confidence that users of the programming language (i.e. legal subject matter experts building automated systems) can have in the finished product. For example, if a lawyer familiar with the law can read the code, and the code is structurally similar to the source written legal rules, and expressed in a similar paradigm, that will increase the confidence the lawyer will have that they understand the meaning of the code, and can advise as to whether the code is a reasonable model of the law.

Secondly, the similarity in structure and paradigm between the source rules and the encoded model of those rules increases the likelihood that legal subject matter experts will be able to learn to do the encoding themselves. This dissertation will argue below that the opportunity to democratize access to these tools by making them easier for non-programmers to learn is a critical component of overcoming the problems that have prevented the adoption of DLP tools more widely.

2.3.2 Increased Efficiency for Encoding and Maintaining

To the extent declarative logic programming language tools allow a greater one-to-one relationship between the source written legal rules and the code, it will simplify the task of developing and maintaining the code.⁵ The degree to which the original written legal rules and the representation in the programming language have matching structures is called “isomorphism”. Declarative logic programming languages, because they allow the programmer to write rules, are better suited to maintaining that one-to-one relationship.

Without isomorphism, every time a legal written rule is added, or changed, the developer needs to review everywhere in the imperative code that might have been affected by that addition or change. By contrast, where isomorphism exists, the developer in a declarative logic programming language would only need to review the parts of code that encode the portion of the rule that was changed. This problem grows exponentially if code needs to be written to do more than one thing with the rule-set.

An intuitive way to understand this problem is by analogy to a person who is responsible for programming self-driving cars to find routes to their destinations. We would not expect that person to write a separate program for each combination of starting and ending location, and then to modify all of those programs whenever the roads change. Instead, we would expect that person to write one tool that is capable of searching for routes on its own, and then focus on providing that software with up-to-date digital maps.

By analogy, a person attempting to implement legal reasoning around a rule in an imperative language is writing a set of directions from A to B. A person encoding those rules in a DLP language is creating a digital map that the computer can search for solutions once you give it a specific problem. The search algorithms, in this case, are the generic logic programming solvers provided for use with DLP languages.

The benefit of isomorphism is seen most clearly when the written legal

⁵Kevin D Ashley, *Artificial Intelligence and Legal Analytics* (Cambridge University Press June 2017) at p. 63.

rules change, and the code needs to be updated to represent the changes. In a non-isomorphic representation of the rules, the programmer will need to ask themselves where in the code the current rule is represented. And it will likely be in more than one place, particularly if the rule is an exception that applies to more than one determination. So one change to the rule becomes multiple changes to the code. And it is impossible to predict from where the amended rule exists in the written legal rules where the corresponding piece of code will be in the model.

But more problematically, the programmer will also need to find where in the code the prior rule is not represented, because it previously didn't matter, and where the new rule needs to be represented, because now it does matter. This means that in a non-isomorphic language, one change to the written legal rules can require the programmer to survey the logic of a substantial part of their application in order to see whether and where their code needs to be changed.

By contrast, in a perfectly isomorphic representation, when there is a change to the written legal rules in one place, the code that represents those rules also needs to be changed only in one place. Isomorphism therefore reduces exponentially the amount of work involved in maintaining encodings of written legal rules once they are developed. Declarative logic programming tools increase the isomorphism of encoded written legal rules, and reduce the cost of developing and maintaining them.

2.3.3 Explainability

Declarative logic programming also has one significant advantage over the current generation of machine learning and neural network techniques, which are also used in legal applications. Unlike 'modern' artificial intelligence methods, declarative logic programming allows the computer to explain how it used the rules and facts in order to come to a conclusion.⁶

The ability to explain reasons is also a very important feature in developing systems, as it allows the developer to understand the "source" of erroneous

⁶Ashley (see n. 5) at p. 64.

output.⁷ It is also a very important aspect of the automation of legal services by governmental or judicial bodies which are obliged to meet strict standards of transparency in their reasons for making certain decisions. For example, such tools could be used to provide explanations for how court websites determine whether electronic filings can be accepted, or to provide explanations for why a person was determined not to be entitled to a government benefit.⁸

This should not be understood to mean that all DLP systems feature the capacity to generate explanations for conclusions. As the survey in Chapter 6 reveals, that is not the case. But fundamentally, generating explanations for conclusions reached by DLP tools is technically feasible.

2.4 Applications of Encoding Legal Rules in DLP

It is beyond the scope of this dissertation to review in detail all the potential applications of DLP programming in the automation of legal reasoning with regard to written rules. The possibilities include the creation of expert systems that power automated legal services; a drastic modernization of legal knowledge management allowing recorded legal knowledge to be used by computers in addition to other human legal experts; the application of formal quality assurance methods from computing science to laws, contracts, and other rulesets; improvements in legislative and contract drafting methodology; the simultaneous drafting of natural language and digital legislation, as proposed by the growing international “Rules as Code” movement; and more.⁹

These tools are already used in large national governments. One tool included in the Survey in Chapter 6, Oracle Policy Automation, is used by the

⁷Giridhar Pemmasani et al., “Online Justification for Tabled Logic Programs” [2004] *Functional and Logic Programming* (Yukiyoshi Kameyama and Peter J Stuckey eds. 24.

⁸The term “explain” here is used colloquially. There are distinctions that can be drawn between concepts of explanation, proof, or justification. For the purposes of this dissertation, I refer only here to the capacity of the tool to generate something that would explain in a way intelligible to a human being how the human being might come to the same conclusion.

⁹Service Innovation Lab, Government of New Zealand, *Better Rules for Government Discovery Report* (<https://www.digital.govt.nz/assets/Uploads/Better-Rules-for-Government-Discovery-Report.pdf>, Accessed: July 25, 2019).

government of the UK to process applications for legal aid, by the government of the United States to process applications for health insurance, and by the government of New Zealand for processing applications for child benefits. But most of these uses involve web interviews, which is only one of the possible uses of the technology.

As an example of some of the other possible applications of DLP tools in automating reasoning around written legal rules, I performed an experiment using a DLP tool to do scenario testing on Canadian copyright law. That experiment is described in detail in the next Chapter.

2.5 Conclusion

Declarative logic programming is a style of programming in which the programmer, instead of writing procedures, writes rules, facts, and questions. The declarative paradigm is the same paradigm in which written legal rules are written, allowing those rules to be translated without their needing to also be reformulated from one paradigm to another, and increasing the confidence that legal subject matter experts can have in the result. It also increases the isomorphism of the resulting code, making the code easier to maintain when the written legal rules change. Unlike many other modern techniques, declarative logic programming is also inherently explainable, making it more appropriate for use in public law purposes. For those reasons, it is a very promising technology, and to the extent that promise has not been realized the reasons why, and whether there are solutions to them, is an important area of research.

There are a wide number of possible uses of declarative logic programming in automating reasoning with regard to written legal rules. A survey of those purposes is outside the scope of this dissertation. But as a demonstration, the next Chapter will demonstrate a use of a DLP tool to automate analysis around Canadian copyright law.

Chapter 3

Using DLP To Find Legislative Bugs: An Experiment

3.1 Introduction

This Chapter describes an experiment using the ErgoAI programming language to encode legislation and common law rules in order to demonstrate something analogous to detecting bugs in legislation. The ErgoAI programming language is described in Chapter 6.

The experiment revolves around the unusual Supreme Court of Canada decision in *Euro-Excellence Inc v Kraft Canada Inc*, [2007] 3 SCR 20, 2007 SCC 37 (CanLII) [*Kraft*]. In that decision, it was determined that sales of works may not infringe copyright, despite the existence of an exclusive license, if the works were purchased from the copyright owner overseas.

This creates an injury to the owner of the exclusive license, with no means of redress. An injury with no means of redress is a stereotypical example of an error in legislation, making *Kraft* a good example. In the software development technique called “regression testing” the developer can write an automated test to ensure that a known bug does not reappear in the code as the code changes in the future.

In this experiment, I took the idea of a person with an injury and no remedy as the bug, encoded the rules as they were understood after the *Kraft* decision, and asked the software to detect whether the bug existed in those rules.

The resulting software was not provided with the description of the facts in *Kraft*. Rather, the experiment was to determine whether, given the description of the bug, and an encoding of the law, the software could discover, using scenario testing, the facts such as those in *Kraft* that would give rise to the problem.

In the result, the software was able to predict that the fact scenario in *Kraft* would result in the bug. It also found two additional fact scenarios in which the problem would arise, neither of which were known to the author before running the software.

3.2 Kraft

3.2.1 Background of the Case

For the purposes of this dissertation, the following simplified summary of *Kraft* will suffice.

The plaintiff Kraft Canada was the exclusive distributor of Toblerone chocolate bars in Canada. The defendant Euro-Excellence was a company which purchased Toblerone bars in Europe, imported them to Canada, and resold them in Canada. The Toblerone bars were manufactured by parent companies of Kraft located in Europe. The parent companies made the plaintiff an exclusive licensee within Canada of the Toblerone logo. The plaintiff thereafter sued the defendant for breach of their copyright in the Toblerone logo by importing infringing works. That copyright action made its way to the Supreme Court of Canada.

It was agreed that the defendant was being accused of violating the rule in the *Copyright Act*, s 27(2)(e), RSC 1985, c C-42, [in this Chapter, the *Act*] against importing into Canada a copyrighted work from another jurisdiction in circumstances where had the manufacturer of the product manufactured and sold it in Canada, the manufacturer would be in breach of the exclusive licensee's copyright. This rule is designed to prevent a company from breaching Canadian copyright by simply manufacturing an infringing copy in another jurisdiction and then importing it for sale.

There were two legal issues discussed in the case, but this experiment deals only with the first - namely, whether the owner of a copyright can be sued by an exclusive licensee of that copyright for copyright infringement in Canada.

Whether or not it was possible for an exclusive licensee to sue a copyright owner for copyright infringement turned on the interpretation of section 27(2)(e) of the *Copyright Act*, RSC 1985, c C-42 [in this Chapter, the *Act*]. It was the opinion of five of the nine justices of the Court that an owner of a copyright could not be sued by an exclusive licensee for copyright infringement.

3.2.2 Our “Bug” in the Code

For the purposes of this experiment, we are assuming two things: that the interpretation of the five members of the Court who found that an owner was not liable to an exclusive licensee for copyright infringement under the *Act* was correct, and that the outcome of *Kraft* was an unintended effect for the legislature.

These assumptions are not justified here on legal or policy grounds. That it was an unintended effect for the legislature means, for our purposes, that had the legislature been aware that the legislation as drafted precluded the enforcement of a copyright by an exclusive licensee against the copyright owner, the legislature would have chosen to draft the legislation differently.

The interpretation of the *Act* which we adopt as having been unintentionally given effect by the words of the *Act* is that interpretation set out in the minority judgement of Justice Rothstein, at paragraphs 1-56 of *Kraft*. The relevant reasons are summarized as follows:

1. In order to find an infringement under section 27(2)(e) of the *Act*, it is necessary to find that the person who produced the imported good would have been in violation of copyright had they produced it in Canada.
2. By default, a person without property rights in a copyright cannot sue for infringement of that copyright, and a person with property rights in a copyright can.

3. An owner of copyright and an assignee of copyright have a property right in the copyright.
4. An owner loses their property rights to an assignee when they assign those rights.
5. There is an exception in the legislation for exclusive licensees to be able to sue for infringement.
6. A person with a right to sue for infringement but without a property right cannot sue a person with a property right.
7. The *Act* does not give property rights to exclusive licensees.
8. Therefore, an exclusive licensee cannot sue a person with a property right in the copyright for infringement.
9. Therefore, an exclusive licensee cannot sue the owner of a copyright for hypothetical infringement.
10. The inability to maintain an action with regard to the hypothetical infringement means the importation is not infringing.

These rules come from the *Act* and the common law. Again, this understanding of Rothstein's reasons is not defended, here. It is merely provided to be explicit about the interpretation of those reasons that will be encoded. However, accurately modelling the law of copyright is not necessary for this experiment. Including common-law elements in our analysis, whether established or novel, enhances the realism of our experiment, as statutory interpretation may consider common law principles.

From the perspective of the dissertation, we are treating Justice Rothstein's opinion and the relevant sections of the *Act* as our "written legal rules".

3.3 Encoding Kraft

3.3.1 The ErgoAI Language

ErgoAI is a declarative logic programming language in the Prolog family of languages, published by Coherent Knowledge, and marketed for use in legal applications.¹ ErgoAI was selected as the tool for this experiment for two primary reasons.² First, this experiment did not require a user-friendly interface. Second, ErgoAI is an advanced logical programming language that provides a number of features anticipated to be valuable in this experiment, and that were not available in the other tools. Specifically, ErgoAI is capable of defeasible reasoning, which allows the encoding of a law to remain more isomorphic with the source material.

3.3.2 The Selected Scope

For the purpose of this experiment it was not necessary to encode the entirety of the *Act*. This experiment requires only encoding elements of the *Act* and jurisprudence sufficient to reveal our “bug.” Through a trial and error process that began from the “bug” and worked backwards to the elements we wanted to be able to test and distinguish, the following ontological entities were included in the scope:

- Parties;
- Works;
- Sales;
- Exclusive Licenses;
- Assignments (for comparison purposes);

¹Coherent Knowledge, ErgoAI (<https://coherentknowledge.com/>, Accessed: July 25, 2019).

²The ErgoAI programming environment was downloaded onto a Intel i7-5500U dual-processor laptop running at 2.4GHz with 12GB of installed ram, running Windows 10. The ErgoAI 1.2 (Solon) release was used. Coherent Knowledge provided the author with an academic license for the software for this purpose, and advice on programming technique, which assistance is gratefully acknowledged.

- Products;
- Places of Manufacture;
- Rights to Sue for Infringement;
- Infringements including infringing products, sales, and importations; and
- Property Rights.

The rules that were encoded for this experiment are listed here:

1. Section 27(2)(e) of the *Copyright Act*
2. The rule that by default you cannot sue for copyright infringement unless you have a property interest.
3. The rule that owners have a property right by default.
4. The rule that owners lose their property rights when they assign them, which defeats the rule above.
5. The rules that gives assignees a property interest in copyright.
6. The rule allowing exclusive licensees to sue.
7. The rule that if a law gives a type of person a right to sue and requires them to join owners in those lawsuits, that type of person does not have a property right.
8. The rule that a person without a property right cannot sue someone with a property right, which defeats any rule to the contrary.
9. The rule that the absence of a right to sue for hypothetical infringement implies the absence of a right to sue for the actual importation, which defeats any rule to the contrary.
10. Rules about what constitutes an infringement of a copyright.

It's worth noting briefly that this code reformulates slightly the decision in *Kraft*. In the decision, the Court held that the absence of an ability to sue with regard to the hypothetical infringement means that the importation is not infringing. This is encoded as if the absence of an ability to sue for the hypothetical infringement implied only a similar lack of ability to sue with regard to the importation. This modification allowed the code to use the presence of an infringement and the absence of a remedy as the definition of our "bug".

3.3.3 Encoding Counterfactual Conditions

The ErgoAI code which was used to represent the ontology and rules in this experiment is attached as Appendix C. As an example, the following excerpt is the section of code that represents section 27(2)(e) of the *Act*. This section is included because section 27(2)(e) is an example of a statutory provision with a counterfactual condition, as discussed in Chapter 4.

A counterfactual condition is a condition that depends for its calculation on the software being able to imagine that something that is false is true, and consider the results if it were true. In this case, whether a person is liable for an infringement depends on whether the same thing would have been infringing if it had been done by someone else somewhere else.

This is dealt with in the code is by creating an additional type of product, called a Hypothetical Product, and then asking whether the Hypothetical Product is infringing, in order to determine whether there is an infringing importation. You can see from the code below that while the use of counterfactual contingencies does complicate the code slightly, and injures the isomorphism between the code and the legislation, it is not an insurmountable obstacle.

Listing 3.1: ErgoAI Code for section 27(2)(e) Copyright Act

```
1 HypoProduct :: Product .
2 HypoProduct [| real_sale=>Sale |] .
3 HypoProduct (? _product) : HypoProduct [
4     origin ->Canada ,
5     manufacturer ->?_manufacturer ,
6     original_work ->?_original ,
7     real_sale ->?_P] :-
8     ?_P: Sale [
```



```

9           product_sold ->?_product : Product [
10             origin ->NotCanada ,
11             manufacturer ->?_manufacturer ,
12             original_work ->?_original
13           ]
14         ].
15
16 ?X: InfringingImportation :-
17     ?X: Sale ,
18     ?Y: HypoProduct [ manufacturer ->?_accused , real_sale ->?X ] ,
19     ?Y: InfringingProduct .

```

Read from top to bottom, this code states that a Hypothetical Product is a type of product, with the added information of being associated with a real sale. It then states that a hypothetical product exists with an origin of Canada for every sale of a product originating from outside of Canada. Thirdly, it states that if an object is a sale, if there is a hypothetical product associated with that sale, and if the hypothetical product would be an infringing product, then the sale is an infringing importation.

Compare this to the text of the *Act* itself, which does not find it necessary to spell out the nature of the hypothetical infringement in that level of detail:

Secondary infringement

(2) It is an infringement of copyright for any person to

(a) sell or rent out,

(b) distribute to such an extent as to affect prejudicially the owner of the copyright,

(c) by way of trade distribute, expose or offer for sale or rental, or exhibit in public,

(d) possess for the purpose of doing anything referred to in paragraphs (a) to (c), or

(e) import into Canada for the purpose of doing anything referred to in paragraphs (a) to (c),

a copy of a work, sound recording or fixation of a performer's performance or of a communication signal that the person knows or

Variable Name	Variable Type
Owner	Party
Manufacturer	Party
Seller	Party
Licensor	Party
Licensee	Party
Assignor	Party
Assignee	Party
Place of Manufacture	Boolean (Canada, or Not Canada)

Table 3.1: Variables used in ErgoAI Code

should have known infringes copyright or would infringe copyright if it had been made in Canada by the person who made it.

3.3.4 Using ErgoAI to Test for the Bug

The process used in this Chapter for testing for bugs is a brute force search, which involves two steps. The first is to have the software generate a large number of possible fact situations, and the second is to have the software search the implications arising from those fact scenarios for evidence of the “bug.” Having the software generate possible fact scenarios and then testing those fact scenarios for certain implications is one of the many possibilities opened up by declarative logic programming for the law.

In our model of the law there are 8 relevant variables, show in table 3.1. In order to address all possible scenarios, including those scenarios where the same party fills more than one of the various roles, we need to create 7 possible parties, and two possible locations, and generate test data for each partition of parties across the various roles, combined with each possible location. We also want to be able to consider those scenarios in which there is no licensee, scenarios in which there is no assignee, and scenarios in which there is neither.

In order to do that code was written to

- generate facts for each possible unique combination of a unique partition of parties and a location,
- test those facts for whether or not they reveal the bug we are looking

for,

- report the result,
- delete the generated facts, and
- move on to the next unique combination.

Four of these code blocks were written, one each for each of the four possibilities with regard to the existence or non-existences of licenses and assignments.³

Choosing how to search for the bug is a challenging part of the design of the experiment, because it would be very easy to design the query to search for the problem we already know is there. The question is therefore what might the legislative drafter have been seeking to avoid before they knew about this bug? One answer might be that they would seek to avoid infringements for which the authorized party does not have a right of action against an infringing party, for a reason other than that the plaintiff and defendant would have been the same person.

That query can be expressed as follows in ErgoAI:

Listing 3.2: Code To Find “Bug” in Copyright Act

```
1 isabug(?_I) :-
2   ?_I:Infringement, // there is an infringement
3   ( // either
4     // the infringement is an infringing sale
5     ?_I:InfringingSale [
6       seller ->?_S,
7       product_sold ->?_P:Product [
8         original_work ->?_W
9       ]
10    ],
11    // the product sold is not hypothetical.
12    \naf ?_P:HypoProduct,
13    // there is a person authorized with regard to the work
14    ?_W[authorized ->?_A],
15    ( // either
16      // the right of action has been overridden
17      ?_R:RightOfAction [ plaintiff ->?_A, claim ->?_I ],
18      Overridden(?_R),
19      // in which case the right of action is the bug
20      ?bug = ?_R
21    ) \or (
```

³The assistance of Keith Morris and Steven Taschuk is gratefully acknowledged in designing an efficient algorithm for generating only legally-unique scenarios.

```

22         // it never existed
23         \neg ?_R:RightOfAction[plaintiff->?_A,claim->?_I],
24         // the impugned seller is not also the plaintiff
25         ?_A !== ?_S
26     )
27 ) \or (
28     // the infringement is an infringing product
29     ?_P:InfringingProduct:Product[
30         manufacturer->?_M,
31         original_work->?_W
32     ],
33     // the infringing product is not hypothetical.
34     \naf ?_P:HypoProduct,
35     // there is a person authorized with regard to the work
36     ?_W[authorized->?_A],
37     ( // either
38         // the right of actions has been overridden
39         ?_R:RightOfAction[plaintiff->?_A,claim->?_P],
40         Overridden(?_R),
41         // in which case the right of action is the bug
42         ?bug = ?_R
43     ) \or (
44         // it never existed
45         \neg ?_R:RightOfAction[plaintiff->?_A,claim->?_P],
46         // the impugned manufacturer is not also the plaintiff
47         ?_A !== ?_M
48     )
49 ).

```

3.4 The Results

A summary of the results is provided in table 3.2. The software was run and asked to find the bug in the automatically-generated fact scenarios. This began with the least complicated scenario, i.e. those involving no licensee or assignee. The software completed its search in under 3 seconds and found no instances of the bug among the 10 possible scenarios.

The code was then run against all possible scenarios involving a licensee but no assignee. The software completed its run in approximately 30 seconds, and among the 104 possible scenarios, it found three which resulted in a bug. One scenario was the scenario in *Kraft*. The two other scenarios were as follows:

1. A work owned by a party A is manufactured by a party B outside of Canada. The owner A then purchases that product from manufacturer B and sells the product in Canada. If the owner A has given an exclusive

Variables	Possible Scenarios	Running Time	Bugs Found
Manufacturers Owners Sellers Place of Manufacture	10	<3 seconds	0 bugs
Manufacturers Owners Sellers Place of Manufacture Licensee Licensor	104	~30 seconds	3 bugs
Manufacturers Owners Sellers Place of Manufacture Assignee Assignor	104	~30 seconds	0 bugs
Manufacturers Owners Sellers Place of Manufacture Licensee Licensor Assignee Assignor	8280	unknown	n/a

Table 3.2: Bugs Found Automatically in Copyright Law

license to party C, party C cannot sue party A for the sale, despite the fact that it is the first sale of an infringing product imported into Canada. This is different from the scenario in *Kraft*, because in *Kraft* the seller and the owner were not the same party.

2. A work is owned by Party A, manufactured anywhere, and sold in Canada by party A. If the party A has given an exclusive license to a party B, the party B is prohibited from enforcing any copyright against A despite the exclusive license. This is different from *Kraft*, and the scenario described above in paragraph 1, because it does not require hypothetical infringement. The product can be owned, manufactured, and sold in Canada, and the “bug” still happens.

The code was then run to detect bugs in scenarios involving no exclusive license, but a valid assignee of copyright. The code completed its search of all 104 possible scenarios in approximately 30 seconds, finding no bugs.

Unfortunately, a search of all 8280 scenarios involving both an assignment and a license took an inordinate amount of time (more than several hours), suggesting that there was an error in how the code for that search had been drafted.⁴ There was not sufficient time to resolve that error, and this portion of the experiment was abandoned.

3.5 Conclusions

Automated scenario testing is just one of many possible applications of declarative logic programming with regard to written legal rules. But this experiment demonstrates that it is possible to encode legal rules in a way that will allow for their automated analysis in order to discover things which might not have been apparent otherwise. This could be done when a law is being drafted, to improve its quality. It could be done at the time of amendment to see

⁴The number of scenarios to test in this model can be calculated as the Bell number (the number of possible unique partitions of a set of length N) for integer N, where N is the number of roles in the scenario, multiplied by 2 for the number of possible locations of manufacture.

whether a proposed amendment has any unanticipated and undesired effects. And there are many other possible applications.

Having discussed what declarative logic programming is, why it is relevant with regard to the automated analysis of written legal rules, and having demonstrated one possible use of the technology, the next Chapter will now survey the treatment of DLP tools in legal research, and respond to the common criticisms.

The ErgoAI code that was used in this experiment is included in this dissertation as Appendix C.

Chapter 4

Legal Scholarship on DLP in Law

4.1 Expert Systems and DLP

There are varying definitions of the term “expert system.” Many of the current uses of the term describe applications that do not rely on DLP as an underlying technology. But as the term was understood in the 1980s, and as it has been used in legal scholarship, it refers to technologies that allow for the encoding of rules in systems that use a semantics of formal logic. In reviewing the literature surrounding expert systems in law, the term “expert system” still seems to be used in this way, and so it used in that meaning, which is usefully analogous to DLP tools, in this chapter.

4.2 Why Legal Scholarship?

In this Chapter I will review some of the most significant legal academic scholarship on the topic of expert systems in the law. Most of what has been written on the topic comes from the field of computer science. It is important, therefore, to ask why this review should focus on the contributions of legal academics.

The need for legal scholarship with regard to DLP technologies is illustrated by an example. Consider a future in which more and more contractual relationships are being documented and implemented using declarative language programming techniques for automating the rules agreed to between

the parties.¹ Our laws of contractual construction are built on the premise that the contracts are written in a natural language. What happens when the parties agree that their intent is reflected primarily in the encoded version of the contract, and there is a disagreement over what the encoded contract was intended to mean?

Questions arising from that possibility, all of which are fundamentally legal questions, include: Do we have judges who are capable of “reading” an encoded contract? Do we need them? Does it still make sense to suggest that a contractual clause can have more than one meaning if it is written in a language with strict semantics, and the parties tested it before signing it? What impact will parol evidence as to the intent of the parties have when a contract is incapable of being interpreted automatically in more than one way, and it can only possibly have been incorrectly drafted? If an encoded contract is interpreted by a court, and the court finds that it was drafted in a way that was inconsistent with the intent of the parties in fact, and should be “read” differently than it was encoded, will the judge be expected to provide the new code? How do we equip the profession for these tasks? If a lawyer provides advice on the meaning of an encoded contract, but the lawyer cannot read the language in which the contract was encoded, relies on the advice of a technical person, and that advice is flawed, is the lawyer liable in professional negligence?

This is just one non-exhaustive set of questions arising from one possible application of DLP technology in law. The current state of legal scholarship and jurisprudence on the construction of contracts expressed in languages with strict semantics is beyond the scope of this dissertation. But it will hopefully be uncontroversial to suggest that the law is unsettled with regard to all of these questions. Settling the law will require a dialogue including legal academia,

¹This is not a fanciful proposition. DLP tools like Accord (Accord Project, Accord Project (<https://www.accordproject.org/>, Accessed: December 1, 2017)) are currently being used to implement smart contracts, and methodologies for evaluating these contracts are being developed (Florian Idelberger et al., Evaluation of logic-based smart contracts for blockchain systems [2016] (In: Alferes J, Bertossi L, Governatori G, Fodor P, Roman D (eds), Rule Technologies. Research Tools, and Applications, RuleML 2016. Lecture Notes in Computer Science, vol 9718, Springer, Cham.)).

and founded in a strong understanding of the technologies involved. This dissertation aspires to contribute to that conversation.

4.3 Susskind

The legal academic discussion surrounding expert systems begins with Richard Susskind's *Expert Systems in Law*.² Susskind's text calls for a jurisprudential study of expert systems in order to promote their adoption, and improve their quality.³ Susskind's research was the jurisprudential sibling to computer science work on the development of an expert system dealing with latent damages.⁴ That system was implemented in Prolog, which is a DLP language. Susskind's text, therefore, is an examination of precisely the same technology that this dissertation examines.⁵

Susskind's definition of expert system is relatively constrained. He is interested in examining specifically systems that are designed to operate as an alternative to one lawyer calling on the expertise of another lawyer who is an expert in a given field.⁶ That definition would eliminate many concerns associated with expert systems, but his text also addresses a number of arguments for and against the adoption of expert systems that extend past that limited context.

The main thrust of the text is that expert systems in law have great potential, but that interdisciplinary work must continue to make them more generally applicable.⁷

Susskind argues that the knowledge collection process for the development of expert systems in law ought to be simpler than in other fields of study, because law has human-created and readily-available primary sources of data,

²Richard E Susskind, "Expert systems in law: A jurisprudential approach to artificial intelligence and legal reasoning" (1986) 49(2) *The modern law review* 168.

³Richard E Susskind, *Expert systems in law: a jurisprudential inquiry* (Clarendon; New York 1987) at pp. 18, 26.

⁴Phillip Capper and Richard E Susskind, *Latent Damage Law: The Expert System:[a Study of Computers in Legal Problem Solving]* (Butterworths 1988).

⁵Van Emden and Kowalski (see n. 1).

⁶Susskind, *Expert systems in law: a jurisprudential inquiry* (see n. 3).

⁷*ibid.* at p. 288.

and other fields of study require experimentation.⁸ He also argues that the purpose of an expert system is not to encode “the law”, but rather to encode the expert’s formulation of the law.⁹ He does suggest, however, that areas of law in which there are significant disagreements between experts as to the correct formulation should be avoided.¹⁰

Throughout, he insists that the use by lawyers of computer programming languages is to be rejected. He calls tools that require a lawyer to learn a programming language “no more than a hindrance”, and even describes controlled natural language interfaces as requiring a prohibitive degree of “preparatory effort on the part of the prospective user.”¹¹ Here, Susskind is talking about the lawyer as the user of the expert system, not as the subject matter expert in the development of the system.

When he turns his gaze to the other lawyer in the process, the legal subject matter expert, he equally anticipates that it is not feasible for lawyers to learn the programming languages in which expert systems are manufactured. He states that “one of the most important goals of researchers in AI and legal reasoning should be the development of a shell for expert systems in law.”¹² This is undoubtedly also a result of his evident opinion that it is not practicable to ask lawyers to learn to use programming languages the way software developers do.

The word “shell” refers to a usually text-based interface between a user and a computer to make it easier to accomplish certain tasks. Operating system command-line prompts, and interactive interpreters for programming languages are examples of such shells. Susskind is writing at a time when a text-based interface was the primary method of interacting with a computer. So his call for a “shell” for the development of legal expert systems can be better understood today as a call for an application designed to make it realistic that lawyers could learn and use these technologies.

⁸Susskind, *Expert systems in law: a jurisprudential inquiry* (see n. 3) at pp. 49, 61.

⁹*ibid.* at Ch. 2.

¹⁰*ibid.* at p. 53.

¹¹*ibid.* at pp. 65-66.

¹²*ibid.* at p. 156.

Susskind considers and responds to a number of counter-arguments against the use of expert systems in law. Many of these objections he manages to avoid by having specified that the end users of his concept of expert systems are other lawyers. He argues that lawyers should be aware of basic legal principles, and should be aware if it seems like there is relevant information that the expert system is not requesting. This mitigates certain risks. He does, however, acknowledge what he calls “boundaries” to what expert systems in law can accomplish.

One such boundary is the argument from open texture and vagueness.¹³ Typical expert systems do not have a mechanism to deal with uncertainty or subjectivity as to what words in law mean (e.g. ”reasonable”), or uncertainty or subjectivity as to whether a given fact falls into a particular category where the extremes might be uncontroversial, but the boundaries are fuzzy (e.g. ”bald”).

In response to these issues Susskind suggests that the conclusions of an expert system should be considered contingent on whether the formulation of the law encoded is accurate, whether there are not implied exceptions on the grounds of principle or purpose, and other similar factors.¹⁴ The text does not assert that expert systems can be perfect, but rather takes a pragmatic approach suggesting that there are circumstances in which they can nevertheless be valuable.

We are faced with the choice between ... expert systems in law built by human beings upon whose skill we rely and upon whom great responsibility is bestowed, or ... no further research and development into expert systems in law at all. It is submitted that the former option is preferable.¹⁵

This dissertation can be understood as an examination of how far we have come toward developing Susskind’s “shell” for developing legal expert systems.

¹³Susskind, *Expert systems in law: a jurisprudential inquiry* (see n. 3) at p. 191.

¹⁴*ibid.* at pp. 191, 198.

¹⁵*ibid.* at p. 151.

This dissertation attempts to survey what has happened in the meantime, what remains to be done, and what steps we might want to take next.

4.4 Popple

In contrast to Susskind's insistence that a jurisprudential point of view be taken to how to develop expert systems, James Popple's PhD dissertation argued for a pragmatic approach.¹⁶ Popple's SHYSTER legal expert system tool was a case-based reasoning tool, and was later adapted to an existing MYCIN rules-based expert system to create a hybrid DLP and case-based reasoning tool. SHYSTER was published in 1993, and it was integrated with MYCIN in 2003.¹⁷ Popple's pragmatic argument, which is compelling on its face, is that many lawyers operate at the level of expertise expected of a lawyer without any jurisprudential knowledge. I take this to mean that Popple believes a typical lawyer might review of similarities in fact scenarios and outcomes, without a deeper analysis of the reasons for each decision, if they have no grounds to believe that the case before them "breaks the mold". SHYSTER is built to mimic what a lawyer would do in the same circumstances, without concerning itself with jurisprudential models of the law.

I understand Popple's pragmatic argument to be a response to the idea that encoding laws in an expert system allows the system to predict an outcome, but does not give the system an understanding the underlying jurisprudential theory that has been applied so as to arrive at that interpretation of the law. The risk from this would be that if the facts of a new case engage the underlying theory in a way that isn't obvious from the precedent, the lawyer may make a prediction that is likely based only on past outcomes, but ultimately incorrect. Popple's reply seems to be "that is no worse than a lawyer would do."

¹⁶James Popple, *A pragmatic legal expert system* (Dartmouth (Ashgate) 1996).

¹⁷Thomas A O'Callaghan, "A Hybrid Legal Expert System" (PhD thesis, 2003).

4.5 Leith

Where Susskind is a high-minded proponent of expert systems, and Popple is a more pragmatic proponent of them, Phillip Leith has positioned himself as an opponent to the use of expert systems in law.^{18,19,20} He does so from a place of knowledge and experience, having done PhD work in the field.

Leith argues that it was always a naive belief that law could be reduced to a set of rules, the automatic manipulation of which would result in legal advice. Writing in 2016, he calls the experimentation with expert systems in law a failure.²¹ He argues that there was an illusion of early success for expert systems driven by commercial motivations, but that they were never actually adopted even in the medical sphere, where the first commercially available applications existed.²²

He suggests that a misunderstanding of what the formalism of logic represents, and a misunderstanding of what a law actually is, led people to encode laws in a way that “gutted” them.²³ I understand this critique to be grounded in the view that law is not deterministic, and cannot be forced into deterministic models without losing something critical in its nature.

Leith also suggests that law was approached by technicians as a test problem for the purpose of demonstrating the usefulness of logic programming, as opposed to a means of providing something that would be of value to practicing lawyers.²⁴ Essentially, Leith suggests that these early legal expert systems were academic experiments, and never demonstrated real-world usefulness because they were not intended to. They were useful if you wanted to do a particular thing in a particular way envisioned by the makers of the tool, but in reality that is not how that thing was done, and so they were not used.

¹⁸Philip Leith, “The rise and fall of the legal expert system” (2016) 30(3) *International Review of Law, Computers & Technology* 94.

¹⁹Philip Leith, “The Emperor’s New Expert System” (1987) 50(1) *The Modern Law Review* 128.

²⁰P Leith, “Fundamental Errors in Legal Logic Programming” (1986) 29(6) *The Computer Journal* 545.

²¹Leith, “The Emperor’s New Expert System” (see n. 19) at p. 98.

²²*ibid.* at p. 99.

²³*ibid.* at p. 100.

²⁴*ibid.* at pp. 100-101.

He also points out that lawyers do not typically deal with agreed-upon legal interpretation of written rules. He suggests that proponents of expert systems in law have adopted a view of the law as knowable, known, and unchanging, and that none of these things are true in practice. He colourfully describes “two users of expert systems going into the courtroom and waiving their respective printouts at the judge, claiming that theirs states that they should win” as what proponents of expert systems were offering.²⁵

Leith argues that there is little sign of life in the legal expert systems research community now, and sees it as unlikely that what he views as the mistakes of the expert system proponents in the past will be repeated. He notes that logic programming, formal proofs, natural language processing are not widely adopted in computing, and that much of the academic enthusiasm for the project came not from proven utility, but from the availability of funding.²⁶

The huge support for the expert systems movement in the UK was partly that the idea of producing such systems was attractive to the academic IT community, partly that there was a lack of critical perspective on the nature of law within the academic legal community, but also because there was a massive influx of funding from a programme which was to be the saviour of UK computing.²⁷

4.6 Ashley

In his 2017 text *Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age*, Kevin Ashley makes an argument in favour of what he calls a new paradigm for intelligent technology in legal practice.²⁸ The text espouses a belief in something Ashley calls “Cognitive Computing”, which he describes as a paradigm in which

human users are ultimately responsible for customizing their own solution using a legal app, but the commoditized legal service tech-

²⁵Leith, “The Emperor’s New Expert System” (see n. 19) at p. 103.

²⁶ibid. at pp. 104-105.

²⁷ibid. at p. 104.

²⁸Ashley (see n. 5).

nology should apprise the humans of the need for customization and support them with customized access to relevant legal information to help them construct a solution.²⁹

Ashley also talks about the potential of argument retrieval (AR) technology.³⁰ AR is imagined as an evolution of the existing information retrieval (IR) systems that are used to help lawyers access statute and case law relevant to their legal issue. In AR, the items that are being searched through are not documents, or paragraphs of text, but abstract representations of legal arguments that might be relevant to a situation. This would be achieved by automatically identifying issues in dispute inside decisions, and the structure of the arguments that were used to resolve those disputes. It would then be possible, for example, to search for cases in which a specific issue was argued (instead of "a specific word was used"), or a specific argument with regard to that issue was adopted or rejected.

Ashley contrasts cognitive computing and argument retrieval, the new paradigm, with what he calls the "former paradigm" of expert systems.³¹ In that sense, much of the text can be read as setting out a new and better alternative to expert systems, though he also acknowledges that the new systems will not do what was expected of expert systems.

The text is an optimistic and detailed look at a variety of artificial intelligence techniques that can be used to extract legal meaning from texts without (or with minimal) human intervention. Extracting legal information from text is the "legal analytics" of the title of his book. The fundamental distinction that Ashley draws between the old and new paradigms is that in the old paradigm knowledge was acquired from experts, and in the new paradigm it is - to the degree possible - extracted from data.³² Human experts are expensive, but machine learning provides an opportunity to extract meaning from data with a speed and scale that human beings cannot match, and that requires no human intervention. This new paradigm, therefore, promises greater efficiency

²⁹Ashley (see n. 5) at p. 13.

³⁰ibid. at p. 11.

³¹ibid. at p. 8.

³²ibid. at p. 13.

in automating the digitization of legal knowledge.

It is interesting that Ashley suggests that cognitive computing should involve the collaboration of humans and machines in solving legal problems. Yet in the task of acquiring knowledge of the law, he suggests that people should be left out of it as much as possible. But this is not without reason. He repeatedly notes what he calls the “bottleneck of knowledge acquisition” in the old paradigm, where the subject matter expert is the only source of legal knowledge.³³

The prevailing model for how expert systems should be developed is the combination of the skills of two different people. One person is the “subject matter expert” (or SME) who knows the things that need to be represented in code. The other person is the “programmer” who knows how to express those ideas in a programming language, allowing them to be used by the computer. The SME attempts to teach the subject to the programmer, the programmer attempts to encode their understanding, and the program is tested by the SME. Where the program does not behave properly the two people attempt to determine whether that is because the SME’s understanding is inconsistent, the SME explained their understanding poorly, the programmer grasped it poorly, the programmer coded it poorly, the data used to represent the situation is wrong, or inadequate, the SME misunderstands what the software is trying to do, or some combination of the above.

It is a challenging communication task that Ashley describes as: “the manual process of acquiring rules is cumbersome, time-consuming, and expensive, a knowledge acquisition bottleneck that has limited to utility of expert systems in law and many other fields.³⁴”

He also notes that logic-based representations of statutory rules is “a kind of model that probably is *not* yet ready to automatically connect directly to legal texts”.³⁵ So Ashley is saying that using people to get legal knowledge into an automated system is cumbersome, and at the same time, it is a task

³³Ashley (see n. 5) at pp. 4, 11, 226, 355.

³⁴*ibid.* at p. 11.

³⁵*ibid.* at p. 11.

that is beyond the capabilities of machine learning.

The implication of these two facts - getting knowledge into logic-based systems from experts has been inefficient, and it is not yet possible to automate the process of extracting the logical representations from text - is that the old paradigm is to be disfavoured compared to those techniques that might be able to chew through more source material with less human involvement.

In chapter 2 of his text Ashley focuses on DLP technologies under the heading of “modeling statutory reasoning.” This is his discussion of the use of DLP for modelling written legal rules. While he notes that there is a “pressing necessity” to be able to reason with statutes, he notes a number of areas that he says pose challenges to the logical encoding of statutory rules.³⁶

The restraints he notes are vagueness, semantic ambiguity, syntactical ambiguity, the difficulty of statutory interpretation, the need to address disagreements as to legislative meaning, and the practical difficulty of maintaining logical representations of statutes alongside textual statutes.³⁷

Semantic ambiguity is defined as uncertainty as to which of a limited number of possible meanings was intended by the legislature for a given word or phrase. An example might be a word like “cleave” which might mean either “separate” or “combine.” Vagueness refers to uncertainty about whether a particular entity is included in the meaning of a phrase. An example might be when the phrase “reasonable efforts” is used, and there is some uncertainty as to whether a given set of efforts is “reasonable.” Ashley notes that vagueness in particular may be used as a technique of legislative drafting, with the purpose of delegating some high fidelity interpretation questions to the court, or to facilitate legislative compromise, and that as a result, this problem is likely endemic to legislation.³⁸

Syntactical ambiguity, by contrast, is a situation in which it is not clear what the syntactical meaning of the elements of the law actually are. The celebrated example of syntactical ambiguity is the case of *State v. Hill*, 245

³⁶Ashley (see n. 5) at p. 38.

³⁷ibid. at pp. 42-48.

³⁸ibid. at p. 40.

La 119 (1963) where in a Louisiana statute the word “and” was interpreted by the Louisiana Supreme Court to mean “or.” The statutory language was as follows:

No person shall engage in or institute a local telephone call, conversation or conference of an anonymous nature and therein use obscene, profane, vulgar, lewd, lascivious, or indecent language, suggestions or proposals of an obscene nature and threats of any kind whatsoever.

In *State v. Hill* the accused had been obscene, but not threatening, and argued that the word “and” syntactically required both.

Unlike vagueness or semantic ambiguity, which can sometimes be intentional, there is no conceivable benefit from syntactical ambiguity. But it is a challenge identified in encoding statutes. Effectively, it creates a situation in which there is more than one possible meaning of the words of the statute, and it needs to be decided which of those meanings should be encoded. Who, Ashley asks, should make that decision, given that legal experts might disagree?

Ashley then reviews the difficulties that were encountered in the encoding of the British Nationality Act, 1948 in the Prolog programming language, one of the first published attempts at using a declarative logic programming tool to automate legal reasoning.³⁹ The problems he notes are reformulation, negation, default reasoning, counterfactual conditions, and open-textured terms.⁴⁰

Reformulation is described as the process of encoding a clause of a law and then later coming back to it to change it later when later clauses in the same act change what needs to be considered in the earlier encoding. In natural language the new text is implicitly included in the definition created in the original clause, but Prolog had no ability to include it implicitly, so the encoding of the original clause had to be amended, which injured isomorphism.

An example of reformulation can be seen in Appendix A. The second rule in that code sets out how to calculate whether “the parties have a valid adult

³⁹Sergot et al. (see n. 3).

⁴⁰Ashley (see n. 5) at pp. 49-52.

interdependent partnership agreement”. But that rule is not isomorphic to any one part of the *Adult Interdependent Partnership Act*. That is to say, there is no one-to-one relationship between a section of the code and a section of the *Act*. Instead, the code combines elements from sections that deal with age of consent, capacity, whether the parties were married, and termination, among others.

The issue identified with negation is that while legal reasoning requires that certain things be uncertain until they are known to be true, or false, Prolog did not have that capability, and instead treated anything that was not either known to be true or provable as true, as false.

To illustrate the problem with negation, consider a rule that states ”if your facial hair is longer than 5mm, you have a beard.” The encoded rule knows how to figure out whether or not the answer to the question ”do you have a beard” is true. But it does not know how to figure out whether or not the answer to the question ”do you have a beard” is false. Is it if you know that their facial hair is 5mm or less, or if you don’t know anything about their facial hair, or either, or both?

Prolog uses a “closed-world assumption”, which means that anything not provable as true or known as true is false. So by default Prolog would say that a person for whom it was unaware of the length of their facial hair, could be shown to not have a beard, which is intuitively incorrect. Our intuitive understanding of the logic of a rule is that if we don’t know whether the conditions are satisfied, we also don’t know whether the conclusion is satisfied. But that was not the logic of Prolog.

Default reasoning is the fact that legislation is frequently drafted in ways where something is presumed to be true until it is made false (or vice versa) by some other condition. This is non-monotonic reasoning, where the addition of information to the logical formula can change a result that already existed. Prolog did not have the ability to represent non-monotonic reasoning, and so default reasoning was difficult to model.

As a simple example, if you wanted to say ”a thing can fly if it is a bird, but a wounded bird can’t fly”, you could not. Saying something along the

lines of "but" or "unless" requires default reasoning. Prolog forced you to reformulate this statement into a single statement that "a thing can fly if it is a bird and it is not wounded." So the lack of default reasoning would have made the reformulation problem worse.

Counterfactual conditionals refers to a difficulty in encoding rules that make decisions about what might have been true, but isn't. Again, because Prolog implemented formal logic, it was not possible from Prolog to consider both that something was true and false at the same time, and it became necessary to add concepts to the code that did not exist in the law such as "the manner in which a person would have become a citizen if they had become a citizen despite the fact that they died first."

He quotes Berman and Hafner⁴¹ on the inappropriateness of formal logical models for modeling statutory reasoning as a result:

It is logically impossible to begin with a set of premises, and create a valid argument for both a conclusion and its opposite. This restriction certainly makes sense - but in the law, such a "logical impossibility" seems to be precisely what happens!⁴²

Ashley very briefly notes "[l]ogicians have developed some alternative logics that can deal with inconsistency subject to various constraints."⁴³ He does not mention what they are, or where they have been implemented. Later in the chapter he notes the need for rules systems to support defeasible reasoning, which is reasoning about defaults and exceptions.⁴⁴

Defeasible reasoning is reasoning that allows for statements to contradict one another, and provides a mechanism for determining which of the contradictory statements applies in a given situation. The form of logic implemented in Prolog is not defeasible, which created problems because defeasibility is often used in written legal rules. Exceptions, defaults, and presumptions are

⁴¹Donald H Berman and Carole D Hafner, "Indeterminacy: A challenge to logic-based models of legal reasoning" (1987) 3(1) *International Review of Law, Computers & Technology* 1.

⁴²Ashley (see n. 5) at p. 55.

⁴³ibid. at p. 56.

⁴⁴ibid. at p. 63.

all forms of defeasible reasoning used in law, which became more difficult to express when written in Prolog.

Open-textured terms refer, essentially, to instances of vagueness. In the BNA project, the developers dealt with vagueness by simply asking the user whether or not the vague term was satisfied, and providing no other guidance.

Ashley then refers to larger issues of statutory interpretation, pointing to the fact that when judges interpret laws they consider linguistic meaning, systemic coherence, purpose of the law, the intent of the legislator, and implicit requirements of things like constitutionality and coherence with public policy or maxims of law.⁴⁵ He also points to the difficulties that arise where even if parties agree on the meaning of the relevant law, they dispute the relevant facts.⁴⁶

Ashley also discusses the benefits of isomorphism, the need for a one-to-one relationship between pieces of legislation and pieces of code, in order to facilitate development and maintenance of the code. With regard to maintenance, he notes that isomorphism is difficult to maintain given the complexity of statutes and the fact that they change.⁴⁷

4.7 McCarty

L. Thorne McCarty recently wrote an article⁴⁸ providing his theory of the balance to be had between different conceptions of artificial intelligence and law. In the article he comments on the recent writings of Susskind^{49,50} and Ashley⁵¹.

McCarty has been a part of the field of expert systems in law for as long as the field has existed. He published his work on the TaxMan system in a paper

⁴⁵Ashley (see n. 5) at p. 53.

⁴⁶ibid. at p. 54.

⁴⁷ibid. at pp. 63-64.

⁴⁸LThorne McCarty, "Finding the right balance in artificial intelligence and law" [2018] Research Handbook on the Law of Artificial Intelligence 55.

⁴⁹Richard E Susskind and Daniel Susskind, *The future of the professions* (First published in paperback, Oxford University Press 2017).

⁵⁰Richard E Susskind, *Tomorrow's lawyers* (Second edition, Oxford University Press 2017).

⁵¹Ashley (see n. 5).

in 1977 that was a rules-based approach to tax law, similar in that sense to the work that Susskind did in latent damages nearly a decade later.⁵²

McCarty argues that very quickly the focus of the work shifted away from what could be calculated with rules and facts to what couldn't. A later paper of McCarty's espoused the idea that law is incurably open-textured, and that laws are better represented as a combination of invariant requirements, prototypical examples of scenarios that meet those requirements, and then deformations from those prototypes to other examples.⁵³

In his Balance paper, he argues that the important point for representing laws is not to apply a theory to facts, but to generate a theory from them.⁵⁴ These theories can be represented in traditional logical terms, which he calls a logical template, but they can also be represented by these prototypes and deformations.

McCarty's recent paper then sets out the history of his work toward being able to automatically generate structured case notes from judicial decisions. He describes this process as involving elements of machine learning, elements of natural language processing, and also elements of knowledge representation, which is analogous to DLP.

He seems to suggest that there is a place for expert system style knowledge representation in these technologies of the future. But his project also seems to hope for more than this dissertation is examining. Indeed, he is looking for ways to automatically translate judicial materials into his Language for Legal Discourse⁵⁵, which he says is a domain-specific logic programming language, and points to ErgoAI⁵⁶ as an alternative to it.⁵⁷

He also explicitly criticizes Ashley for failing to have spoken to the new

⁵²LThorne McCarty, "Reflections on "Taxman": An Experiment in Artificial Intelligence and Legal Reasoning" (1977) 90(5) Harvard Law Review 837.

⁵³LThorne McCarty, "An implementation of *Eisner v. Macomber*" [1995] ICAIL '95 276.

⁵⁴McCarty, "Finding the right balance in artificial intelligence and law" (see n. 48).

⁵⁵LThorne McCarty, "How to ground a language for legal discourse in a prototypical perceptual semantics" [2015] ICAIL '15 89.

⁵⁶ErgoAI (see n. 1).

⁵⁷McCarty, "Finding the right balance in artificial intelligence and law" (see n. 48) at p. 33.

breed of DLP tools such as ErgoAI.⁵⁸

He notes that there are applications for KR in smart contracts, though he expresses uncertainty about how open-textured questions will be dealt with in that context. He also notes that the systems which are useful for interpreting judicial decisions are not useful for understanding statutes, and that there is a place for the manual annotation of statute law, which presumably could be done in DLP tools. He also expects that to some extent knowledge representation will be a part of attempts to predict legal outcomes in the future.⁵⁹

4.8 Addressing the Criticisms

In this section I will address each of the criticisms raised by the authors above about the feasibility of the legal expert system project. But first, it is necessary to contrast the perspective of this dissertation from the perspective of some of the critics that makes some of these criticisms not unfair, but unimportant.

4.8.1 The Legal Services Supply Perspective

Taken from the perspective of people who want to automate legal services to increase their supply, the fundamental question is do legal expert systems realistically have the potential to increase affordable access to appropriate legal services. The emphasis here is on practicality, and appropriateness.

Practicality requires us to consider whether it is viable to use DLP technology to automate legal services. Reasons it might not be viable, even if technically feasible, include whether it is prohibitively expensive, or the people with access to the tools do not have any motivation to use them in the service of automating legal services.

Appropriateness requires us to consider whether what can be provided by an expert system (or with the help of one) is a net improvement for the consumer of the service. Appropriateness considers both issues of quality and cost. Consistently incorrect legal advice is not appropriate. Overly expensive

⁵⁸McCarty, “Finding the right balance in artificial intelligence and law” (see n. 48) at pp. 39-40.

⁵⁹*ibid.* at pp. 34-38.

legal advice is not appropriate. But legal services that are less reliable than a lawyer and also considerably less expensive than a lawyer may be appropriate.

The various critiques of expert systems in law will be addressed from this dual perspective.

4.8.2 Standards of Appropriateness for Automated Legal Services

The question of whether automated legal services are appropriate needs to consider both the risks and the benefits of automated legal services and the risks and the benefits of the best alternatives. Examples of the best alternatives may be self-representation, asking a friend for help, or relying on publicly available legal education services.

It is useful to make explicit and distinguish between three different standards of appropriateness. In order from least strict to most strict I will call these “better than nothing”, “no worse than a lawyer”, and “perfection”.

If an automated legal service can be provided that is significantly better than the realistic alternatives, and it can be provided sustainably, and affordably, then it will increase the supply of legal services, and is appropriate. This is the “better than nothing” standard of appropriateness, because “nothing” is often the alternative against which the automated legal services should be compared. Many people seek to solve their problems with no legal advice or information whatsoever.

There is an intermediate standard of appropriateness, which I call “no worse than a lawyer”. If an automated legal service is no worse than the service that would have been provided by *the lawyer who built the automated version of that service*, and it is less expensive than receiving the service directly from the same lawyer, that automation will also serve to increase the supply of legal services. It is important to note that the best way to know that a service is “no worse than a lawyer” is if it does the same thing a given lawyer would do in the same situation. And the best way to know that is the case is if the person who automated it was the lawyer themselves. I will return the importance of that feature of the “no worse than a lawyer” standard.

It is worth noting that as Chapter 3 demonstrates, the “no worse than a lawyer” standard also admits of the real possibility that there may be automated legal services that are superior to the service that would have been provided by lawyers. The “no worse than a lawyer” standard could also be called “no worse than a lawyer and potentially much better than one”.

There is a third standard that most often occurs in the literature, but only implicitly. That is the standard of “perfection”. That is a standard not applied in any other context, and applying it exclusively in the realm of automated legal services only serves to badly injure the supply of legal services.

We assert that we must be consciously aware of the apparently sub-conscious tendency to apply the “perfection” standard of reasonableness when discussing the automation of legal services. The “better than nothing” or “no worse than a lawyer” standards can be argued for, but the “perfection” standard must be made explicit, and rejected.

4.8.3 We Encode Interpretations, not “the Law”

I will start by addressing the concerns expressed about whether it is possible to accurately reflect the meaning of law in DLP tools.

The use of DLP tools should be understood to involve, as Susskind suggests, not an encoding of a categorically correct representation of the meaning of the relevant law, but an encoding of the internally coherent understanding of the law that a responsible legal professional believes would be appropriate for people receiving automated legal services to rely upon, in all the relevant context.⁶⁰

If what we are encoding is not “the law”, but one person’s understanding of it, all the abstract concerns about whether expert systems can accurately represent the “true” meaning of a law disappear. Leith’s concerns that representing laws in logical terms “guts” the law can therefore be ignored. If gutting the law in this way serves increasing the supply of legal services through automation, then taking a cue from Pople’s pragmatism, we should have no compunctions.

⁶⁰Susskind, *Expert systems in law: a jurisprudential inquiry* (see n. 3) at Ch. 2.

The idea, which is repeatedly expressed in the literature, that we cannot be sure that our encoding of a law represents what the law really means, is an example of an application of the implicit standard of perfection. No human lawyer is required to know, with certainty, that they are giving advice on the basis of a law's true meaning. The absence of perfection in that regard is not an obstacle to the responsible provision of legal services by lawyers.

Another way of expressing the idea that we encode interpretations, and not the law itself, is by considering it as a sequence of events. A law is written, the law is read by a lawyer, and then the lawyer gives advice about the law. Similarly, a law is written, the law is read by that same lawyer, and then the lawyer encodes the law, and the code provides advice. In both of these situations, the advice that was given, either by the human lawyer or the code, was based on that particular lawyer's understanding of the law. Because the same understanding was used both times, we can expect that the result from the encoding might be "no worse than a lawyer".

Difficulties involved in determining what a law means must be overcome before either the lawyer gives advice, or they encode their understanding. Interpretation is necessary in both circumstances, and so the need for interpretation is not a critique of expert systems at all, but a critique of laws.

Ashley mentions concerns about disagreements as to the meaning of statutes. But in the same way, the fact that the interpretation that would be interpreted by two different lawyers might result in two different sets of code that did not agree with one another is not a criticism of expert systems. This inconsistency exists as between lawyers who do not encode their understandings, too.

4.8.4 Concerns with Statutory Interpretation Can be Mitigated, and Do Not Apply Universally

The difficulty of interpreting statutes is only relevant with regard to the use of expert systems if there are difficulties that are exacerbated with regard to automated understandings. In his text on statutory interpretation, Cameron Hutchison breaks down the categories in which there may be uncertainty as to the meaning of a law into three: incompleteness, error, and circumstantial

change.⁶¹ The text also sets out four causes for incompleteness: the imprecision of language, the possibility of unanticipated cases, the intentional use of vagueness by legislators, and the necessity to consider implicit rules. These incompleteness factors overlap to an extent with the concerns Ashley raises about vagueness, semantic ambiguity, and syntactical ambiguity.

It is helpful to look at which, if any, of these problems is worse with regard to understandings of the law that have been encoded. First, we can say that if a lawyer is aware of a statutory error, they can encode their corrected understanding of the law as easily as they can advise using it. If they are not aware of the error, they would provide incorrect advice in any case. The encoded version will still meet the “no worse than a lawyer” standard. With regard to changes to the meaning of laws that arise due to changed circumstances, the same thing is true. A lawyer may anticipate that the change in general circumstances will require a change in the meaning of a provision, or they won’t. If they do, they can encode that understanding. Legislative error and changes in circumstances are difficulties with statutory interpretation that have no particular impact on the use of expert systems.

The fact that the meaning of statutes can change over time, even in the absence of explicit amendment, suggests that the maintenance of automated systems will be an important factor in whether they continue to be reasonable.

The case of legislative incompleteness is more problematic. First, we can look at cases where the legislation is incomplete because it either intentionally or unavoidably uses imprecise language. The source and motivation of the vagueness is not important. It is still the case that where the lawyer cannot give advice in person, they ought not automate it. Non-provision of a service may still be “no worse than a lawyer”. But what if the lawyer does give advice about vague terms? It may be that the lawyer cannot encode how they do that task satisfactorily, in which case they ought not encode their understanding of that topic. If they can set out the factors that they personally would use in determining whether or not the requirements of a vague term are satisfied, and can set out how those factors should be used, it might be reasonable to

⁶¹Hutchison (see n. 4) at p. 7.

encode those factors and perform the same analysis in code.

As Ashley notes in Chapter 3 of his text, significant strides have been made in using case-based reasoning in order to address a wider range of open-textured legal issues in automated systems. This can be a potential solution where statutory language is vague.⁶²

However, this sort of automated advice about vague terms has a flaw in its automated form that may not exist when performed by a person.

New fact scenarios may change the lawyers mind about how to determine the correct answer. A client may appear with an unusual combination of facts that the lawyer had never anticipated, and so could not have added to their encoded process for determining whether vague terms are satisfied. This is a problem with automated legal services of any variety, and is not specific to expert systems in particular, but it is nevertheless a situation in which the encoded understanding is not “as good as a lawyer”. Because the lawyer can come to a new and better way of giving the advice based on the client’s scenario, and the code cannot.

This same problem exists with regard to cases in which there is an application of implicit rules. It is possible that a fact scenario may implicate a different additional source of law not anticipated by the lawyer when they encoded their understanding.

With regard to cases of incompleteness caused by a failure of the legislators to consider certain fact scenarios, there is a similar problem. If you are dealing with a lawyer, the lawyer can take a fact scenario from a client and will perhaps recognize that it was not anticipated by the law, and the straightforward application of the understanding the lawyer previously had of the law will result in an absurd result. Again, this is not a realization that the automated version of the advice can achieve. So in this way, advice from an automated system may not be “as good as a lawyer”.

The question then becomes whether it is possible to achieve the standard of “better than nothing” with regard to advice that deals with these difficult forms of interpretation, and if so, how.

⁶²Ashley (see n. 5).

There are ways of mitigating these problems. The first and most obvious is to simply not encode problematic provisions, but this does not achieve even the “better than nothing” standard, and is not to be preferred. Second, disclaimers can be used to advise the user of the system’s limitations, and allow them to adopt the risk that the system might be flawed with regard to their own situation. This is the approach that is used, for example, in tax filing preparation software, and is probably to be recommended even where the developer is not aware of any specific concerns, as there may be problems they do not anticipate. Third, human advisors can be involved in the process only where they are required, or on the user’s request, and the remainder of the process automated. There are undoubtedly other options.

There will be situations in which no combination of these mitigations will result in a system that meets even the standard of “better than nothing”. This may be because the risks to the client from incorrect advice are so large. In those situations, expert systems should not be used. But this will not be all cases of uncertainty. Far more importantly, there will be cases in which despite incompleteness in the law, the automated version of the lawyer’s understanding will be either “better than nothing” or “no worse than a lawyer”. And in those situations, expert systems are still appropriate.

We cannot justify refusing to use DLP tools for what they *can* do because there remain things they *cannot* do. Responsible use of these tools will always include deciding when not to use them, and issues of open-texture, vagueness, or uncertainty may remain good reasons to come to that conclusion.⁶³

Even if uncertainty as to the meaning of the law can be mitigated in some circumstances, there are two other categories of concerns that Ashley raises that apply to all possible uses of expert systems: technical shortcomings of the tools, and the knowledge acquisition bottleneck.⁶⁴

⁶³The Better Rules conversation (Better Rules for Government Discovery Report (see n. 9)) proposes a fascinating possible resolution to this problem: that public rules ought to be drafted with an eye to how easily they could be automated, encouraging the avoidance of open-textured terminology except where the vagueness serves an explicit policy objective. Such a change in how legal rules are drafted would be a sea change for the applicability of DLP tools, and statutory interpretation itself.

⁶⁴I rely on Ashley here not because he is particularly critical, but because his text provides

4.8.5 DLP Technology Has Improved

In the category of concerns with the technology, Ashley mentions the practical difficulty of maintaining logical representations of statutes alongside textual statutes, reformulation, negation, default reasoning, and counterfactual conditions. But as McCarty correctly notes, Ashley fails to seriously consider how the available technology for DLP has changed since the British Nationality Act, 1948 was encoded.⁶⁵

The problems of reformulation and default reasoning can be dealt with by using tools that include defeasible reasoning. Defeasible reasoning also increases isomorphism, which makes the task of maintaining encodings of the law easier, as exceptions in the law can be represented by exceptions in the code. The problems that Ashley describes with negation are solved with tools that offer three-valued logic and both classical negation and negation as failure.

Counterfactual conditions are complicated to implement in DLP tools, because they require creating additional entities in code to represent both the actual legal condition and the hypothetical. The complication, however, taken in the context of the complexity of implementing DLP tools generally, is minor. It is a hurdle, not a wall. An example of implementing counterfactuals is included in Chapter 3, where the legal question required consideration of whether something would have been valid had it been done by someone else in a different place.

4.9 The Real Challenge

The knowledge acquisition bottleneck is the only common criticism left unaddressed. It applies to all possible uses of expert systems. It is not resolved by using modern tools. It cannot be resolved by merely avoiding the standard of perfection. The viability of expert systems as a tool for increasing the supply of legal services may legitimately turn on whether there is a realistic and appropriate solution to this problem.

such a comprehensive survey of the problems that have been expressed throughout the literature.

⁶⁵McCarty, “Finding the right balance in artificial intelligence and law” (see n. 48).

To reiterate, the knowledge acquisition bottleneck refers to the high cost and low reliability of the method of having a legal subject matter expert and a programmer work side by side to develop expert systems.

4.9.1 The Solution: The Legal Expert Is the Programmer

The knowledge acquisition problem disappears entirely when the person who holds the subject expertise and the person who understands the programming language are the same person. There is no risk of anything being lost in translation, missed, or misunderstood when the process of legal encoding involves only one person.

The solution to the knowledge acquisition bottleneck problem, therefore, and potentially the key to getting the benefits of DLP for the purpose of increasing the supply of legal services through automation, is making it realistic for legal subject matter experts to use DLP tools without the assistance of a programmer.

Note that this also allows the person who is both the legal expert and the developer to satisfy themselves that what the tool they have built does is “no worse than a lawyer”, because it does precisely what they themselves would do.

As a person with formal training in both technology and law, I might be accused of unrealistic optimism in this regard. Ashley and Susskind are both either explicitly or implicitly fatalistic about the possibility of lawyers learning to use DLP tools.

I argue that pessimism is based on a false assumption - that if lawyers are to begin using DLP tools, because the most effective of those tools today are programming languages, that lawyers would need to become programmers. If that assumption were true, the pessimism might be well-founded. It is not.

4.9.2 Spreadsheets for Legal Reasoning

Making legal subject matter experts such as lawyers the people who use DLP tools is possible without requiring those people to go through the difficulty

currently associated with learning to write software in a programming language.

For this argument, I point to a natural experiment that can be observed by comparing the legal profession today to the accounting profession of approximately 40 years ago.

Prior to the advent of electronic spreadsheets in the late 1970s, it was technically possible for an accountant who was also a programmer to get a computer to do complicated math.⁶⁶ We can safely presume that very few accountants actually did so.

That state of affairs is analogous to the legal profession of today. It is possible for lawyers who are also programmers to get computers to do complicated legal reasoning. Very few, the author included, actually do so.

The controlled variable in this natural experiment is a technological advancement that changed the difficulty of learning to use computers to automate the respective field. The late 1970s saw the advent of electronic spreadsheets, starting with VisiCalc. By the early 1990s research showed that the vast majority of professional accountants were using spreadsheets for financial modelling without the assistance of a computer professional.⁶⁷ By the mid-90s, the ability to use electronic spreadsheets was considered a basic competence for members of the profession.

After the advent of electronic spreadsheets, demand for accounting services increased, because accountants were able to do more sophisticated analysis, faster, more accurately, and less expensively than ever before.⁶⁸

What changed? How did electronic spreadsheets so significantly change what we expect an accountant to know?

Spreadsheets used an interface, values in the cells of a grid, with which accountants were already familiar, making spreadsheets feel like something

⁶⁶B Grad, "The Creation and the Demise of VisiCalc" (2007) 29(3) IEEE Annals of the History of Computing 20.

⁶⁷Bob Berry and Alyson McLintock, "Accountants and financial modelling" (1991) 4(4) OR Insight 11.

⁶⁸Jacob Goldstein, How The Electronic Spreadsheet Revolutionized Business "All Things Considered" (<https://www.npr.org/2015/02/27/389585340/how-the-electronic-spreadsheet-revolutionized-business>).

accountants already knew, making them easier to learn, driving adoption.

The key difference between the two stories is accountants have an application or interface which takes what is possible, and makes it easy. Lawyers do not have that application.

The analogy between spreadsheets and declarative logic programming may not immediately be obvious, but it is strong. Spreadsheets are, in fact, a declarative tool for math programming. In the same way that DLP technologies allow you to specify rules without specifying the order in which they are applied, spreadsheets allow you to express the relationships between numerical values without setting out the order in which those calculations will be performed.

And in the same way that spreadsheets mimic an interface that was in use by accountants before it was made electronic, it is possible to make declarative logic programming look very much like statutory drafting. Two of the products included in the survey later in this dissertation are proof. The constrained natural language interface of Oracle Policy Automation can be seen in Appendix A, and the similar interface of DataLex is shown in Chapter 6.

The knowledge acquisition bottleneck is the only real challenge remaining. It disappears entirely when the legal subject matter expert and the programmer are the same person. That can and will happen when someone builds a tool for declarative logic programming that is as accessible, as powerful and as easy to learn as electronic spreadsheets.

This “spreadsheets for legal reasoning” will not be a literal spreadsheet. But it will do for legal reasoning and legal service providers what spreadsheets did for math and accountants. It will make lawyers faster, more accurate, increase demand for legal services, increase the supply of legal services, and make legal services possible, including automated legal services, that were never possible before.

4.10 Conclusion

Legal academics have considered the viability of expert systems in law for decades, and have expressed a consistent set of concerns. Most of these concerns can be dispensed with by using modern technology, by understanding that what we are attempting to encode faithfully is not the meaning of the law, but what an expert understands the meaning of the law to be, and by concerning ourselves with what can be done instead of what cannot. Of the criticisms in the literature, only one stands as a true obstacle to increasing the supply of legal services with DLP technologies: the knowledge acquisition bottleneck.

That problem has a solution that has not been seriously considered in the literature: make DLP tools possible for legal subject matter experts to use without the assistance of programmers. Indeed, the suggestion has been dismissed out of hand, to the extent it has even been discussed, for decades. But the experience of the accounting profession at the advent of electronic spreadsheets provides strong evidence that easy-to-use declarative programming tools can quickly and dramatically change the face of a profession, and there is no reason to believe the same is not true of law.

The open question, then, is only whether the things that it is already possible to do with DLP tools can be made easier.

The rest of this dissertation will examine what would be required of these tools, including ease of use, where the existing tools stand, and what steps should be taken next.

Chapter 5

Desirable Qualities in DLP Tools for Automation of Legal Services

5.1 Introduction

This chapter sets out a short list of qualities or features that would be desirable in a modern DLP tool for automating legal reasoning: Affordability, Uncertainty, Explainability, Case-Based Reasoning, Temporal Reasoning, De-feasibility, and Usability.

An effort has been made to prioritize features by their potential impact on automation of legal services. An effort has also been made to keep the list to as few items as possible.

This list is generated in the light of attempts to encode Alberta's *Adult Interdependent Partnerships Act*, SA 2002, c A-4.5, [AIPA] in two of the tools included in the survey in Chapter 6, ErgoAI and Oracle Policy Automation. The encoding of AIPA in ErgoAI is included as Appendix B to this dissertation. The encoding of AIPA in Oracle Policy Automation is included as Appendix A.

The list draws also from the literature reviewed in Chapter 4, the bibliography, and informal observations of public conversations online among people pursuing research or work in the automation of legal reasoning. Of the seven criteria listed, the first six appear repeatedly in existing legal and interdisciplinary writings on the topic of DLP tools. Only the criteria of usability by

non-programmers is novel, and then only perhaps in the priority given to it.

5.2 Affordability

The importance of affordability for the use of these technologies in the automation of legal services cannot be overstated. Organizations and individuals who provide the sorts of legal services that can most benefit from automation are chronically underfunded, and cannot afford large investments in technology. A high price tag prevents the tool from being tried.

Affordability obviously includes the price of the software, if there is one. But it should also take into account the maintenance and operating costs of using a particular technology, as well as the expenses associated with training employees to use that technology.

In the best case scenario, DLP tools for automating legal reasoning would be open source, and free. They would have excellent freely available documentation, tutorials, and active mutually-supportive user communities to assist in getting users up to speed.

5.3 Uncertainty

Formal logic, which is implemented in DLP tools, operates on truth values of true and false.¹ DLP tools have also typically implemented monotonic logic, which means that the addition of new information cannot change any previously known conclusions.² These are two of the ways in which traditional forms of formal logic are not, by themselves, well suited to encoding legal reasoning. A great deal of legal reasoning occurs in situations where not all the relevant facts are known, and we need to be able to draw contingent conclusions, or conclusions based on presumptions. DLP tools, to be effective, need to be able to distinguish between something that is not known, and something that is known to be false. That requires the ability to reason about uncertainty.

¹Ashley (see n. 5) at p. 44.

²ibid. at pp. 50-51.

It would also be helpful to have tools which are capable of answering questions by relative value instead of only by absolute value. A great deal of legal reasoning occurs in situations in which facts or conclusions may not be known specifically, but an unspecific answer may be sufficient. For example, the dates of distant facts are often estimated. It would be convenient if DLP tools could accept a relative value instead of an absolute value, so that the user could either specify a day, or use a statement such as “more than 5 years ago”. I am aware of no tools with this feature.

5.4 Explainability

There is a general distrust of legal algorithmic reasoning today, raised by valid concerns about the incorporation of human bias into machine learning algorithms which have no ability to justify their conclusions. In the judicial and quasi-judicial context there are administrative law requirements for sufficiency of reasons that will need to be met by systems which make determinations. The ability to have the tool explain its answers is therefore a critical component of developing with DLP tools. The ability of the code to explain an incorrect answer provides the tool developer with the opportunity to understand very quickly where and how the code went wrong, speeding up development and enhancing testing.³ The ability to explain conclusions will also provide greater confidence to subject matter experts in how tools are coming to the conclusions that they reach, and may provide better confidence to users of those tools.

5.5 Case-Based Reasoning

There are elements of written legal rules that are ambiguous. One of the approaches taken in the past to deal with these sorts of problems is to attempt to encode “heuristic” rules in DLP tools. Heuristic rules can be thought of as “rules of thumb”, or perhaps more accurately as an imperfect model, in rule form, of how a human expert would make the same decision. A typical example of the use of heuristics in an expert system for diagnosing car problems might

³Pemmasani et al. (see n. 7).

be something like "if the noise does not change with speed, the problem is most likely not in the drive train". In practice this is an inadequate solution to the problem. Heuristic rules are difficult to formulate, and of questionable reliability. They are also not how lawyers actually analyze legal problems that depend on knowledge of written legal rules.

Machine learning techniques can be used in this realm, but by themselves lack the ability to generate an explanation for the conclusion or prediction reached.

There is a third sort of technology, case-based reasoning, that can be used to compare a fact scenario to a database of previously decided cases and come to an explainable prediction of the legal outcome.⁴ CBR is analogous to the process that a lawyer undertakes in deciding ambiguous questions. This similarity makes the task of building of CBR tools more intuitive for legal experts than drafting heuristic rules. CBR also employs algorithms that make it possible to generate explanations for their predictions.

For its ability to deal with ambiguous legal texts in a way that is analogous to how legal experts answer the same sort of question, the ability to resort to CBR when ambiguity requires it is a critical feature for automating legal reasoning around written legal rules.

5.6 Temporal Reasoning

Written legal rules very frequently deal with events and periods of time, and provisions will relate to one another in terms of how those events or periods of time overlap, if at all.⁵ Words like "while" and "until" are very intuitively understood by people when written into legislation, but become very difficult to encode unless the DLP provides specific mechanisms for doing it.⁶

There are different forms and depths of temporal reasoning that might be implemented in a DLP tool, and various purposes for which they might be

⁴Ashley (see n. 5) at Ch. 3.

⁵ibid. at p. 65.

⁶Listing 7.6 shows an example of what it becomes necessary to do to make date calculations without the benefit of temporal reasoning features.

used, including calculating a legal state at a point in time based on a series of legal events, and determining the rules applicable to a specific event based on proclamation dates.

Given how frequently these sorts of problems arise in legislation, and how difficult they are to solve in the absence of sophisticated temporal reasoning features, a high quality tool for automating legal reasoning would need to have the ability to simplify this sort of reasoning for the user.

5.7 Defeasibility

Isomorphism is a word used to describe the degree to which a piece of software, written in a DLP language, can be divided into sections of code, where each section of code has a one-to-one relationship to a section of the written legal rule.⁷ All DLP languages will have some greater degree of isomorphism than most imperative languages, because they are structured as sets of rules. But the degree of isomorphism varies.

In terms of the underlying logic implemented by the language, a tool that implements “defeasibility” is a tool in which you can write rules that are exceptions to other rules.⁸ This is an important capability, because it greatly increases how isomorphic the code will be to the source material.

That is the case because writing rules as exceptions to other rules is a much more efficient way of expressing them, and so is used widely in written legal rules. Expressing written legal rules in a DLP tool that does not feature defeasibility requires you to re-implement the same rule in any other rule to which it might apply, causing a loss of isomorphism, increased difficulty in encoding the rules, and complicating the task of maintaining them when the written rules inevitably change.

An excellent tool for encoding legal reasoning would therefore need to make it possible for the user to express rules using exceptions.

⁷Ashley (see n. 5) at p. 63.

⁸ibid. at p. 63.

5.8 Usability

Usability is a difficult factor to define or quantify, but it is used here to describe to the degree to which users find the tool easy to learn and use. Users, in this case, are the subject matter experts who are using these tools to encode legal knowledge.

Chapter 4 argues that one of the solutions to the problems identified with the implementation of DLP tools in the legal realm is to eliminate the division between the programmer and the legal subject matter expert, and make both the same person. Achieving that requires vastly improved usability of DLP tools from the perspective of non-programmer legal subject matter experts.

5.9 Conclusion

These constitute a “top 7” list of features that would make a DLP tool for automating legal reasoning highly effective.

The first six of these features are not novel. Usability, in a sense, is also not novel. Most legal scholarship with regard to the use of DLP tools for automating legal reasoning seems to have simply presumed that usability could never be achieved in sufficient measure to eliminate the need for lawyers to rely on programmers to encode legal rules.

In the next Chapter the dissertation will review several existing tools against these seven criteria, and discuss the state of the currently available options.

Chapter 6

Survey of Selected Tools for Automating Legal Reasoning with DLP

6.1 Criteria for Inclusion

In order to limit the number of products that needed to be considered for this dissertation, the review was limited to products that are commercially available, or are available as mature open-source projects, and which are built for, marketed toward, or used by individuals who are automating reasoning around legal rules.

For people familiar with the legal technology space, this may lead to some surprising exclusions. There are a number of tools designed to allow people to build legal expert systems of some variety, but which do not use a declarative programming paradigm. Examples include A2JAuthor¹, and QnA Markup². These tools are excluded.

Similarly, there are a large number of “generic” applications of declarative logic programming such as Prolog and related programming languages, and Business Rule Management Systems, such as InRule³ in the commercial space or Drools⁴ in the open source space. These, too, are excluded.

¹A2J Author (<https://www.a2jauthor.org/>, Accessed: July 25, 2019).

²David Colarusso, QnA Markup (<https://www.qnamarkup.org/>, Accessed: July 25, 2019).

³InRule Technology, Inc, InRule (<https://www.inrule.com/>, Accessed: July 25, 2019).

⁴Drools (<https://www.drools.org/>, Accessed: July 25, 2019).

Every effort was made to include as wide a variety of tools as possible over the course of the research, but as with any survey, examples have most likely been unintentionally omitted.

6.2 Selected Tools

6.2.1 ErgoAI/ErgoLite

ErgoAI⁵ is an example of a modern declarative logic programming language, a descendent of Prolog. There are many such languages, but ErgoAI and ErgoLite are included here because they include features specifically useful in automating legal reasoning, and because the commercial version has been marketed for use in that realm.⁶ ErgoAI is the name given to the commercial version of the language, which is offered by Coherent Knowledge Inc. Licenses for the use of ErgoAI, at the time of writing, were available free for academic use, and in the range of several hundred to several thousand dollars for commercial use, depending on how the software was deployed.⁷ ErgoLite is the name given to the open-source version of the language, which was previously known as Flora-2.

ErgoLite is a programming language with a simple text-based interpreter. ErgoAI adds a number of visual interface elements to the development environment, adds features for obtaining explanations for the reasoner's conclusions, and includes features that allow the software to be more easily integrated with other systems. Most of these interface elements are also available for the open-source version of Prolog, named XSB, on which Ergo operates.⁸ However, the same user interface elements are not available for ErgoLite. Both version of the software include support for defeasibility, allowing the user to specify rules that override other rules.⁹ Neither product has any built-in support for

⁵ErgoAI (see n. 1).

⁶Coherent Knowledge, Financial Domain Application (<https://coherentknowledge.com/financial-domain-application/>, Accessed: July 25, 2019).

⁷Coherent Knowledge, Ergo Pricing (<https://coherentknowledge.com/pricing/>, Accessed: July 25, 2019).

⁸interProlog Consulting, Prolog Studio (<http://interprolog.com/interprolog-studio/>, Accessed: July 25, 2019).

⁹Michael Kifer, Defeasible Reasoning in Ergo (<https://docs.google.com/document/>

temporal reasoning, or deals with uncertainty of facts.

6.2.2 Neota Logic

Neota Logic is a tool that is for generating web-based applications which ask users for information, and provide users with information or documents based on the responses to those questions. These interfaces, which ask the user for one or a small number of pieces of information at a time, and progress based on logical deductions about what other information is required in order to calculate some goal, will be referred to as "interviews".¹⁰ Unlike any of the other tools in this survey, the makers of Neota Logic have specifically targeted the legal industry in their efforts to market the tool. This has been done in part by promoting the adoption of the tool and a program of education on its use in various law schools throughout North America.¹¹

Recently, however, Neota has announced that their focus on the legal industry will be broadened. Subsequent to that announcement, and consistent with it, Neota Logic announced the creation of an advisory board that includes representatives of both law firms and insurance companies.¹²

Neota Logic advertises itself as a "no code" solution for building legal expert system websites. The encoding of declarative rules and procedures is done in a visual coding environment in which the rules and procedures are depicted as flowchart-style diagrams. Neota's rules system does not include defeasibility. All relevant criteria for a given conclusion must be included in the same rule.

However, because Neota Logic requires that the rules be encoded as diagrams, the benefit of a 1:1 relationship between the law and the diagrams is less significant. Arguably, text and diagrams are so ill suited for side-by-side comparison that isomorphism, to the degree it exists in Neota Logic, does not

d/1aPUUUzkBgtdQ8PMvyIRGgA-qGYRowbqZBeaUBUXFH3k/edit, Accessed: July 25, 2019).

¹⁰Neota Logic, Neota Logic (<https://www.neotalogic.com/>, Accessed: December 1, 2017).

¹¹Neota Logic, Neota Logic: University Programs (<https://www.neotalogic.com/pro-bono/law-schools/>, Accessed: July 25, 2019).

¹²Neota Logic, Neota Logic reveals new Client Advisory Board [] (<https://www.neotalogic.com/2019/06/03/neota-logic-reveals-new-client-advisory-board/>, Accessed: July 25, 2019).

confer the advantages that it has in other products that use a text-based interface. Any further injury to isomorphism by virtue of the lack of defeasibility is probably less of a strike against Neota Logic, as a result.

Neota Logic does not publish its pricing, but the author understands that in 2017 Neota Logic offered commercial licenses of its product starting in the mid four-figure range for one year, making it likely the second most expensive offering in this survey.

6.2.3 Oracle Policy Automation

Oracle Policy Automation (OPA) is a tool offered by Oracle Corporation for the automation of web-based expert systems based on encoded rules.¹³ OPA is actually a suite of products that work together for this purpose. Oracle Policy Automation is currently marketed as a part of Oracle's suite of products for customer service delivery. It is primarily marketed as a tool to provide customer service representatives with an interactive interview that will allow them to answer customer questions in compliance with a centralized database of policy rules, allowing for a change in policy in one location to instantly change the advice provided by any number of phone or automated customer service agents.

OPA is the latest evolution of a product that was originally designed with specifically legal applications in mind, called SoftLaw. That software went through a number of different owners and names, including "Ruleburst", "Hayley Office Rules", and "STATUTE Expert", before being purchased by Oracle in 2007 and eventually rebranded as Oracle Policy Automation.^{14,15}

OPA has a number of features that make it unique. First, rule authoring in OPA happens in a combination of an installed application called Oracle Policy Modelling, which deals with specifying the ontology, and the layout of the interview pages, and in Microsoft Word and Excel, with the assistance of

¹³Neota Logic reveals new Client Advisory Board (see n. 12).

¹⁴Wikipedia, Oracle Policy Automation (https://en.wikipedia.org/wiki/Oracle_Policy_Automation, Accessed: July 25, 2019).

¹⁵Charles Lindop, Oracle to Acquire Hayley/Ruleburst/Softlaw for A\$150m (<http://tmt-transactions.com/oracle-to-acquire-hayleyruleburstsoftlaw-for-a150m/>, Accessed: July 25, 2019).

plug-ins that connect those applications to the data in the Policy Modelling app.

Drafting of rules, specifically, happens in Microsoft Word or Microsoft Excel. In Word, the rules use a constrained natural language, allowing the rule developer to encode the rules using language which is similar to natural English. Rules which can be more easily described in a table, (e.g. tax rates given for different ranges of income) can be specified in a Microsoft Excel sheet, or in a table in a Microsoft Word document. An example of what legislation encoded in OPA looks like is included as Appendix A.

While by default it is presumed that you will be developing rules for generating web-based expert systems, it is also possible to redeploy these rules for other purposes. The OPA Reasoner and rules can be added to any other application via an application programming interface. Oracle also provides software that allows for the management of your database of rules as a resource that is available over a network, allowing multiple applications and users of those applications to operate off of a centralized set of rules.

OPA includes a number of functions that can be used in the rule modelling in order to convert a set of date-stamped entries into a single list of different values for the same variable over time. If this value is used as a condition in any rule, OPA automatically converts the result of that rule into a temporal list of changing variables over time. It also provides mechanisms by which you can visualize the changing values of any temporal variable. This makes the task of writing rules capable of determining whether a legal conclusion was valid at a give point in time only marginally more complicated than writing rules capable of determining whether it is valid now.

OPA's ontology model includes all of the information that is required to recognize when a concept is being referred to in the rules, and that information is also used to explain the outcome of a decision. An interview developed in OPA will have a tree-structured explanation of its outcome, allowing the user to dig down into reasons for conclusions and sub-conclusions until they arrive at the data they originally entered. That explanation is generated automatically based on the ontology and the rules.

OPA also treats all conclusions as unknown until it has the information necessary to determine their value, making it possible to deal with scenarios where some pieces of information are missing. It also includes features that allow the developer to run an interview in a test environment and save the data entered in that interview as a test that can be automatically loaded to test the database again later.

OPA is probably the most widely used of the tools on this list, and the best supported in terms of available documentation, training, and support. Because the rule authoring happens in Microsoft Word and Microsoft Excel, and uses something similar to the language in which written rules might originally be written, the difficulty of learning to develop tools in OPA can be expected to be less than that in all of the other tools listed.

OPA is currently licensed on a server-usage basis, with one unit of usage representing up to a certain number of minutes of time for an interview. There is a minimum purchase of such units. When I investigated the cost of the minimum purchase of these units of purchase in 2017, the cost was in the range of \$50,000USD.¹⁶ This places OPA well outside the range of the sorts of tools that can easily be adopted by most legal service organizations unless it can be shown to have very significant productivity gains. Proving those gains, however, would require owning the license. For educational purposes, Oracle Policy Modelling is free to use.

6.2.4 Regulation as a Platform

Regulation as a Platform (RaaP) is a tool under development by Data61, a division of CSIRO, in cooperation with the government of Australia.¹⁷ RaaP is an example of an attempt to capture the benefits of DLP tools that may be available for the drafters of legislation and for governments announcing legal rules.

¹⁶Oracle Corporation, Oracle e-Business Suite Applications Global Price List (<http://www.oracle.com/us/corporate/pricing/applications-price-list-070574.pdf>, Accessed: September 21, 2017).

¹⁷Commonwealth Scientific and Industrial Research Organization, Data61 Digital Legislation & Regulation as a Platform (<https://digital-legislation.net/>, Accessed: July 25, 2019).

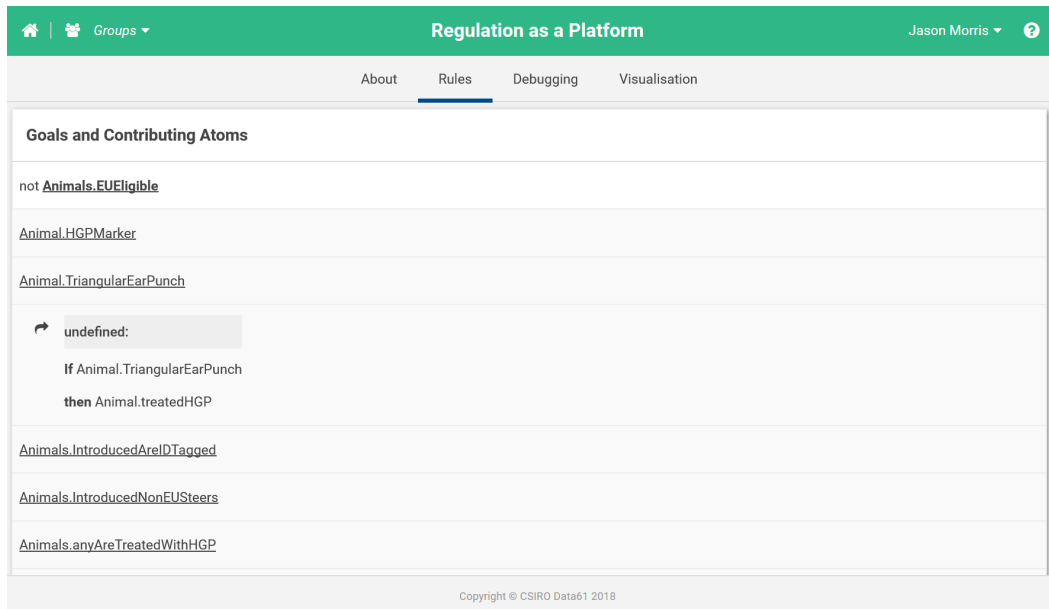


Figure 6.1: RaaP’s Rule-Browsing Interface.

RaaP is a web-based interface on which anyone can encode a piece of legislation using a tool that implements a defeasible deontic logic designed for encoding laws. The tool is based on the work of logician Guido Governatori.¹⁸

Deontic logics are a sub-set of formal logics that include deontic “modes” for statements. These deontic modes are evolved from the modes in logics of necessity, which allow for distinctions between statements of necessity, possibility, and impossibility. Deontic logic uses similar modes to represent the moral or legal concepts of obligation, permission, and prohibition.

Unlike Neota Logic, Oracle Policy Automation, and Docassemble, which are intended to be used to create end-user applications, RaaP is intended to be used by knowledge engineers to create application programming interfaces (APIs). An API is a resource on the Internet that can be accessed by other pieces of software to do specific tasks. A knowledge engineer could encode legislation in RaaP, and then publish what they have encoded. That would then make it possible for anyone creating an end-user focussed application that needs to be able to make decisions on the basis of the legislation to delegate the job of answering the legal questions to the RaaP server, and simply show

¹⁸Ho-Pun Lam and Guido Governatori, *The making of SPINdle* (Springer 2009).

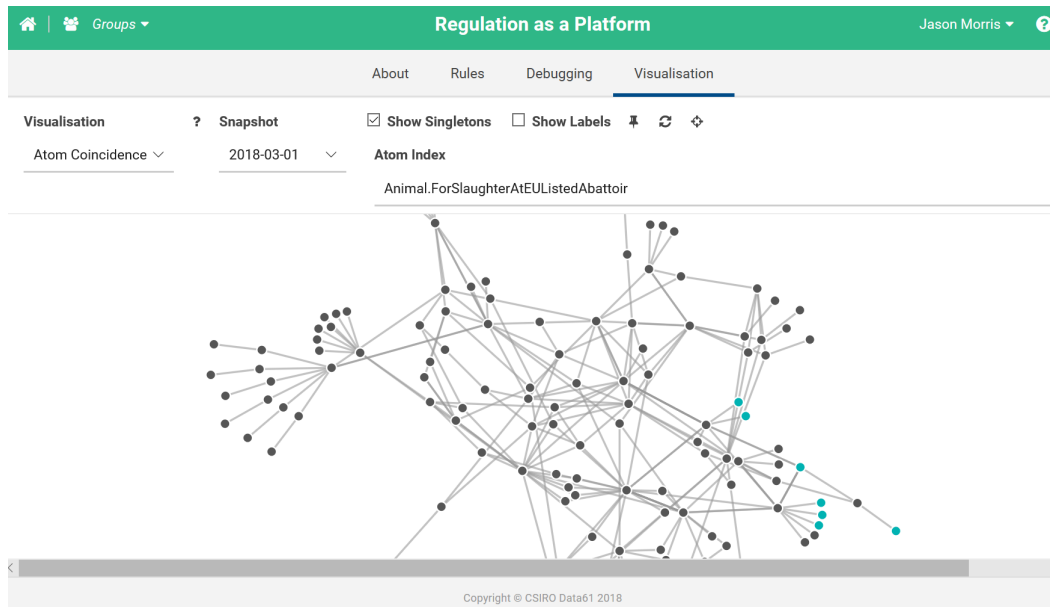


Figure 6.2: RaaP's Graph Visualization Interface.

the user the results.

The interface to RaaP is more user friendly than that of a programming language, but is not yet what someone might call straightforward. An example of the rulebrowsing interfaces is shown at Figure 6.1. It does include some unique features designed to ease the work of the knowledge engineer, including the ability to view a graph representation of how the rules in the system relate to one another, and navigate the rules using that visual interface. An example of that graph representation is shown at Figure 6.2.

6.2.5 Docassemble

Docassemble is an open-source software developed by Jonathan Pyle.¹⁹ Remarkably, Pyle is a lawyer and computer programmer who works for Philadelphia Legal Assistance, a pro bono legal organization. Pyle wrote Docassemble to assist him in his own work attempting to develop web based interviews for the generation of legal documents more efficiently.

The motivation for Docassemble was to generate a tool that could gather as a much information as possible about how to build a web interview from

¹⁹Jonathan Pyle, Docassemble (<https://docassemble.org>, Accessed: January 25, 2019).

[Introduction](#)[▼ Interview](#)[Payment](#)[Documents](#)[Conclusion](#)

What is a Personal Directive?

A Personal Directive is a legal document that says what will happen when you can't make decisions for yourself.

It usually tells people:

1. Who gets to decide when someone else should make choices for you,
2. Who makes those choices,
3. How they should make those choices, and
4. Who should care for your minor children.

This interview will help you create a basic Personal Directive for someone in the Province of Alberta.

[Continue](#)

Figure 6.3: An interview created in Docassemble.

the annotations that are used for document assembly. In this way it is possible to have the document be the source code for your interview. An image of an interview generated in Docassemble is included at Figure 6.3.

Docassemble is not advertised as declarative logic programming tool. It is included in this survey because the way that Docassemble has been implemented allows for something that approximates declarative logic programming.

Docassemble operates with the user providing the software with an “interview” file, written in the YAML markup language. The interview file contains a number of “blocks”, and each block defines a question, a piece of code, or another configuration for the interview. Docassemble evaluates these interview blocks idempotently. Any time Docassemble’s code would be forced to stop because it encounters a variable that has not been previously defined, instead of displaying an error and stopping, it treats that variable as though it was another field to fill in the document, and searches the interview to see if there is a code block or a question block that might give it the answer for what that

variable should be. If it can find one, it asks that question, or runs that code, and starts again at the beginning.

This idempotent design means that “rules” can be implemented as “code blocks” in a Docassemble interview. The way in which the software searches for blocks of code that can fill in the blanks is analogous to the way a logic programming language reasoner uses backward-chaining to find a rule that might trigger the condition it is looking for. In Docassemble the rules themselves are written as small blocks of generalized Python code.

It is therefore possible to implement written legal rules in a quasi-declarative logic programming way by using the Docassemble configuration language, which is a declarative configuration language. This method has its limits, and while it would be possible to integrate Docassemble with a true rules-based reasoning engine, that has not yet happened.

Despite the fact Docassemble has existed for only a few years, it has seen significant adoption. Two annual user conferences for Docassemble users have been held, at the time of writing. The American Bar Association in January of 2019 released a list of the top 20 web tools for lawyers.²⁰ Docassemble was on that list, and two of the other 19 were pieces of software developed using Docassemble. In 2019, the Legal Services Corporation, a funding body for pro bono civil law organizations in the United States, has advised funded organizations seeking Technology Initiative Grants that they should be seriously considering using Docassemble for their projects.²¹

Because Docassemble provides the user with the ability to include arbitrary Python code in their interview, almost anything that can be done with the Python programming language is possible to achieve in Docassemble. However, Docassemble provides no features beyond the ability to write and execute Python code that would facilitate defeasibility, explainability, temporal reasoning, or uncertainty.

Docassemble’s strengths as a tool for encoding legal rules is in its extensibil-

²⁰Stephen Rynkiewicz, Best Web Tools of 2018 (http://www.abajournal.com/magazine/article/best_legal_apps_2018/, Accessed: July 25, 2019).

²¹Legal Services Corporation, Technology Initiative Grant Program (<https://www.lsc.gov/grants-grantee-resources/our-grant-programs/tig>, Accessed: July 25, 2019).

ity, its affordability, and in its ability to very efficiently generate a high-quality user interface for the desired application.

6.2.6 DataLex

DataLex is an offering of the Australasian Legal Information Institute.²² It is a web-based version of a text-based tool developed in the 80s, very much as Richard Susskind anticipated in his text *Expert Systems in Law*.²³

DataLex provides two interfaces, one a more fully-featured interface for developing knowledge bases which is available on approval of a request for access and is licensed only for non-profit and educational use. A second, limited version provides a single window into which DataLex code can be written, and allows the user to run consultations, or perform two types of debugging tests on that code. That is the interface that was reviewed for this survey.

DataLex allows you to mark one or more of the rules in your knowledge base as a “goal” rule. When you run a consultation using the code you are taken to a text-chat style interface where you are asked questions one at a time, in much the format anticipated by Richard Susskind. For boolean criteria, the possible answers are “yes,” “no”, and “uncertain”, indicating that by default DataLex operates with uncertainty. But unlike Susskind’s specification, DataLex is capable of dealing with numerical values, dates, strings of text, and other forms of answers.

When the rules are formed in DataLex, they are formed using a controlled natural language that allows DataLex to use that formulation to generate questions, answers, and explanations. As an example of how this works, Listing 6.1 shows the code that was used to generate the conversation that appears in Figure 6.4.

Listing 6.1: Code as entered into DataLex tool.

```
1 GOAL RULE Section 1 of the Rules PROVIDES
2 section 1 applies ONLY IF
3 section 1(a) applies OR
4 section 1(b) applies
```

²²Australasian Legal Information Institute, DataLex (<http://austlii.community/foswiki/DataLex/>, Accessed: July 25, 2019).

²³Susskind, *Expert systems in law: a jurisprudential inquiry* (see n. 3).

1) Does section 1(a) apply ?

Uncertain

2) Does section 1(b) apply ?

Yes

Here is your report:

DataLex Consultation Report

This application developed using AustLII's DataLex software is to be used only for educational and testing purposes only, and any other use is unauthorised, whether for commercial or non-commercial purposes.

Section 1 applies because section 1(b) applies.



This application developed using AustLII's DataLex software is to be used only for educational and testing purposes only, and any other use is unauthorised, whether for commercial or non-commercial purposes.

Another problem?

What if?

Input text here

Yes

No

Uncertain



Figure 6.4: An interactive consultation generated by DataLex.

We can see that DataLex has extended the basic structure of the rule in declarative logic programming by requiring the user to specify the rule's provenance (e.g. "Section 1 of the Rules"), before providing the rule's conclusion (e.g. "section 1 applies"). This is designed to facilitate explanations.

Similar to OPA, DataLex is able to generate the phrases "section 1 does not apply" and "does section 1 apply" from the phrase "section 1 applies." However, if the automatic translations are inadequate, again like OPA, it is possible to correct them specifically. The tool also has the ability to include links to source materials within the natural language elements of the rules, making it possible to have a certain section of legislation link to a web resource

for that legislation.

DataLex also implements PANNDATA, which is a case based reasoning tool, allowing the user to specify a set of example cases and receive an explanation for whether or not a given proposition is true based on the similarity of the user's fact scenario to the propositions listed as having been true in those cases.

Where DataLex excels is the code is in a controlled natural language that is very intuitive for a legal expert to read, and the fact that a sophisticated user interface can be generated for a specific application merely by typing the word "GOAL" in front of one of the legal provisions.

Where DataLex falls short of the other tools listed is that it features only propositional logic. This makes it difficult to model scenarios that involve multiple instances of a type of object, or require determining whether something is true for any or all of those individual instances.

Another place that DataLex is slightly less sophisticated than some of the other tools is how it deals with nested conjunctions and disjunctions. Where for example OPA will allow you to specify multiple nested conjunctions and disjunctions in a single rule, DataLex seems to allow for only one. Given that a single section of legislative code may have nested conjunctions or disjunctions without the use of numbered subsections, it might then prove difficult in DataLex appropriately set out the provenance of the nested lists.

This restriction may be to ensure that each conjunction or disjunction has a name that can be used to describe it when it is true or false, making it easier to display explanations for conclusions. If that suspicion is correct, the benefit comes with the cost of less isomorphism in how the legislation is encoded.

6.3 Summary of Available Options

A summary of the products described above indicating the features of those products that are considered relative strengths is set out in Table 6.1.

A few caveats are worth noting. ErgoAI, Neota Logic, and Oracle Policy Automation all have different levels of cost associated with their use, varying at

(* indicates commercial)	Price	Uncertainty	Explanations	Case-Based Reasoning	Temporal Reasoning	Defeasibility	Ease of Use
ErgoAI*			X			X	
ErgoLite	X					X	
Neota Logic*		X	X				X
Oracle Policy Automation*		X	X		X		X
Regulation as a Platform	X	X				X	
Docassemble	X						
DataLex	X	X	X	X			X

Table 6.1: Summary of available DLP tools and their features

times by degrees of magnitude. But none has a price so low that it wouldn't be considered a disincentive to adoption among people operating in the pro bono legal services space. RaaP is listed as having an advantage on price because it is currently free to use. It is not, however, open source software, and its availability may change. The same is true of DataLex, which is not open-source, but is currently licensed at no cost for non-profit and educational use. The gold standard for price is permanently open source and free, a standard met only by Docassemble and ErgoLite.

With regard to uncertainty, some of the tools listed as not having this feature do allow for reasoning using three-valued logic, but either this doesn't happen by default, requires extra steps, or is initially configured for a different purpose. For example, ErgoAI and ErgoLite use truth values of true, false, and undefined, but undefined in this context refers to the inability of the reasoner to determine the truth value of the question, not to the idea that there is some uncertainty as to the facts that might prove it.

6.4 What is Missing?

The obvious conclusion from Table 6.1 is that there is no one product that “has it all.” Particularly noteworthy are the many features that are not available in any open-source product. None of the open source products have explanations as a built-in feature. Not even ErgoLite, the open source version of ErgoAI, includes a functioning explanation system.²⁴ It has been noted that implementing justifications in modern DLP tools that use tabling to speed up reasoning time is a “non-trivial” task.²⁵ However, all the commercial products surveyed include an explanation feature, reflecting how critical it is both to providing a tool that will be trusted by end users, and as a debugging tool during development.²⁶

None of the open source tools have temporal reasoning features built in, or can claim ease-of-use.²⁷

The product with the most of these features is DataLex, but DataLex is restrained in that it may not be available for commercial use at all, and it does not allow the user to express rules in terms of predicate logic.

From the perspective of automating legal reasoning, the ideal tool with all of the qualities described in the previous Chapter does not yet exist.

6.5 What Should We Build?

People interested in developing tools for the automation of legal reasoning using DLP ought to focus their efforts in those places where the technological solutions are known, and it is realistic to make progress.

²⁴ErgoLite is built on top of a Prolog-like language called XSB. XSB includes a module entitled ‘justify’ which seems intended to provide explanations for answers generated in XSB. The XSB justify module is not referred to in the ErgoLite documentation, and I was unable to find a way to take advantage of it from within ErgoLite.

²⁵Pemmasani et al. (see n. 7).

²⁶ibid.

²⁷Researchers at the University of Luxembourg ran a tutorial at ICAIL 2019 demonstrating the use of NAI (University of Luxembourg, NAI: Normative Reasoner [<http://nai.uni.lu>, Accessed: July 25, 2019]), a normative declarative logic coding tool designed for ease of use for lawyers. It is not included in this survey because it does not qualify as a “mature” open source tool. It utilizes an interface in which logical representations of rules are generated by annotating the text of the actual rule, which is a promising approach for ease-of-use.

Ease of Use and price go to the accessibility of the tool. Tools which are not used will not help, and so accessibility features should be primary in all future work. As can be seen in the case of DataLex, where the tool is free, but not available for commercial use, licensing is also an accessibility concern, and so open source should be the gold standard.

If we look only at tools in Table 6.1 that are both open source and easy to use, there are none. The target for developers of DLP tools for legal reasoning, therefore, should be anything that is easy to use, open source, and has at least one of the other features listed.

6.6 Conclusion

While high quality tools exist, the ideal tools required for using DLP for automating legal reasoning do not. For the reasons set out in Chapter 4, work going forward must focus on ease of use. In order to assure accessibility, the ideal tools will also be open source and free to use.

Chapter 7

User-Friendly Legal Case-Based Reasoning

7.1 Introduction

The author proposed a project to the American Bar Association Center for Innovation to develop a tool that would seek to create an example of one of the tools argued for in Chapter 6. The project proposed was to build a tool that was open source, easy for lawyers to use, and featured both case-based reasoning and explanations.

That application was accepted, and was generously funded by Canadian legal practice management software vendor Clio.¹

The project was a combination of two open source projects, Docassemble and OpenLCBR. Docassemble is described in Chapter 6.

7.2 OpenLCBR

Prior to making the application, the author contacted the authors of IBP to ask about the availability of open-source implementations of that algorithm.² IBP was selected for two primary reasons. First, it uses a terminology and a data structure that is intuitive and familiar to lawyers, which would make it more intuitive to lawyer-users, and which would make the output of the tool analogous to how legal arguments are frequently expressed. Second, it

¹Themis Solutions Inc, Clio (<http://www.clio.com>, Accessed: July 28, 2019).

²Kevin D Ashley and Stefanie Brüninghaus, “Automatically classifying case texts and predicting outcomes” (2009) 17(2) Artificial Intelligence and Law 125.

had shown remarkable success in developing high quality predictions. The original paper reported 91% accuracy and 100% coverage with their trade secret database.³

At the time, attempts to find an open-source implementation of IBP, or any other legal case-based reasoning algorithm, were fruitless. At my request the authors of IBP consented to the development and release of an open-source implementation of their algorithm. Some time later, OpenLCBR was developed and released by Matthias Grabmair.⁴ OpenLCBR is a re-implementation of IBP in the Python programming language.

7.3 The IBP Algorithm In Brief

IBP is an algorithm that works with a data structure which includes legally relevant factors, issues, and cases. Factors are those facts that may or may not be true about a given case, along with the side of the argument they favour if they are true. In IBP, the side of the argument is described as being the plaintiff's side, or the defendant's side. The IBP algorithm works on a database generated by an expert, who determines what the relevant factors are, and which side they favour.

OpenLCBR includes a demonstration database, and here are some examples of factors in that database:

1. "plaintiff disclosed its product information to outsiders" — favours Defendant
2. "plaintiff's disclosures to outsiders were subject to confidentiality restrictions" - favours Plaintiff

The IBP algorithm also requires data about the issues relevant to the open-textured legal question. The issues are set out in a tree. The root of the tree is the main legal question to be answered. The branches of each node are the conjunctive or disjunctive issues, which, if predicted, are sufficient

³Ashley and Brüninghaus (see n. 2).

⁴Matthias Grabmair, OpenLCBR (<https://github.com/mgrabmair/openlcb>).

to predict the node issue. Each issue node must indicate whether there is a presumption that applies in the absence of relevant factors, and if so, whether the presumption is for the plaintiff or the defendant.

For leaf issue nodes (issues with no sub-issues), the algorithm uses a list of the factors that are relevant to deciding that issue. The structure of the issue tree, the presumptive outcomes if any, whether sub-issues are conjoint or disjoint, and the factors relevant to each issue must all be set out by a human expert.

Below is a representation of the issue structure included in the OpenLCBR package (with relevant factors excluded for brevity).

1. There was a misappropriation of a trade secret if (all of)
 - (a) The information was a trade secret, which is true if (all of)
 - i. The information was valuable, which will be found for the plaintiff if unraised and depends on the following factors...
 - ii. The information was kept secret, which will be found for the plaintiff if unraised and depends on the following factors...
 - (b) The information was misappropriated, which will be found for the plaintiff if unraised, and is true if (any of)
 - i. There was a breach of confidentiality by the defendant, which is true if (all of)
 - A. The information was used, which is found for the plaintiff if unraised, and depends on the following factors...
 - B. There was a relationship of confidentiality, which is found for the plaintiff if unraised, and depends on the following factors...
 - ii. There was improper means used by the defendants, which depends on the following factors...

The expert may also specify a list of factors called “knock-out” factors. These are factors are used to justify ignoring precedential cases that do not

share those factors with the test case. In the database provided with OpenL-CBR there are two knock-out factors. The first is “whether the plaintiff disclosed the alleged trade secret in a public forum”, and the second is “whether the plaintiff completely failed to protect their trade secret.”

Lastly, the IBP algorithm requires the expert to specify a database of cases. Each case consists of two pieces of data used by the algorithm, who won on the base issue (defined as either plaintiff or defendant), and the list of factors that were present in that case.

The algorithm that IBP uses in order to predict the outcome of a test case, which is specified as a list of factors, starts at the root of the issue tree. It then goes through the issue tree following this recursive 7-step procedure:

1. If the current issue was not raised, and there is no default, the algorithm “abstains” with regard to that issue. If the current issue was not raised, and there is a default, it predicts the default. “Raised” means either a relevant factor for the current issue exists in the test case, or the same is true of a sub-issue.
2. If the current issue was raised and has sub-issues all of which must be true, it repeats the process from 1 for each sub-issue, and predicts for the plaintiff only if the prediction for all the sub-issues is also the plaintiff. It predicts for the defendant if the defendant is predicted for any of the sub-issues. If it abstains from predicting any sub-issues, it abstains from predicting this issue.
3. If the current issue was raised and has sub-issues one of which must be true, it repeats the process from 1 for each sub-issue, and predicts for the plaintiff if any sub-issue is predicted for the plaintiff. If none is predicted for the plaintiff but any are abstained, this issue is also abstained. And if they are all predicted for the defendant, this issue is predicted for the defendant.
4. If the current issue has no sub-issues, and all of the factors relevant to that issue are to the favour of the same party, the prediction is for that

party.

5. If the current issue has no sub-issues, and there are factors relevant to that issue going in both directions, the software collects all of the cases which share any of the factors of the test case. If it finds any precedents, and all of those cases were decided for the same party, that party is the prediction for this issue. If it finds precedents, but not all those cases were decided for the same party, it checks to see if any of the cases decided for the defendant can be explained away because they do not have the same knock-out factors as in the test case. If any cannot be explained away, it predicts for the defendant. If they can all be explained away, it predicts for the plaintiff.
6. If the current issue has no sub-issues, and there are factors relevant to that issue going in both directions, and there are no precedents which share all of the factors in the test case, then it will search for cases with all but one of the relevant factors favouring the plaintiff, dropping each factor in turn. For each of those sets of factors, it re-does the analysis in 5 as if those were the only factors in the test case. If the result of all of those analyses is a prediction for the plaintiff, then a prediction for the plaintiff is made for the current issue. If the result of any of those predictions is for the defendant, the current issue is abstained.
7. If it is a raised issue, with factors favouring the plaintiff, but there are no precedential cases even after dropping one factor at a time, the prediction for the current issue is abstained.

7.4 Docassemble-OpenLCBR

The objectives for the Docassemble-OpenLCBR project were as follows:

1. Develop software that will allow OpenLCBR to be used from within Docassemble interviews.

2. Develop a Docassemble interview (a type of web application) that would allow a non-technical user to generate an IBP database for use in OpenLCBR.
3. Develop a tool that will allow a person generating an IBP database to perform leave-one-out testing on that database to test its predictive strength.
4. Using the tools in 2 and 3, generate an IBP database capable of predicting an open-textured legal issue in an access-to-justice area of law, and capable of explaining that prediction.
5. Implement the database in 4 in a Docassemble interview which provides legal advice about a legal issue in an access-to-justice area of law, and explains that advice.

7.5 Results

All of the objectives of the experiment were met. The source code for Docassemble-OpenLCBR was released under the MIT open-source license, and is available at <https://github.com/Gauntlet173/Docassemble-OpenLCBR>.⁵

As a part of that effort to make the tools as user friendly as possible, I refer to the combination of the IBP algorithm and a database as described above as a “reasoner”. As such, the objective 1 above was to generate the software that allowed a “reasoner” to be used in Docassemble. Objective 2 was to create a “reasoner builder” web application. Objective 3 was to create a “reasoner tester” web application.

Tools for generating IBP “reasoners” and for testing them were developed as Docassemble interviews. An example of how these reasons generated by Docassemble-OpenLCBR are displayed to the user in the resulting web tool is provided in Figure 7.1.

⁵Jason Morris, Docassemble-OpenLCBR (<https://github.com/Gauntlet173/Docassemble-OpenLCBR>, Accessed: July 25, 2019).

Your issue would be decided for the plaintiff

▼ It is true that there was a misappropriation of a trade secret, as all of the requirements for that finding are met.

▼ It is true that the information is a trade secret, as all of the requirements for that finding are met.

▶ A review of the relevant cases shows the issue of whether it is true that the the plaintiff maintained secrecy of the information would be decided for the plaintiff.

▶ A review of the relevant cases shows the issue of whether it is true that the the information is valuable would be decided for the

Figure 7.1: Expandable reasons for a prediction are displayed to the user in docassemble-openlcbr.

The tools were then used to generate an IBP reasoner capable of predicting whether two individuals are in a relationship of interdependence as that term is defined in the *Adult Interdependent Relationships Act*, SA 2002, c A-4.5 in Alberta. That reasoner, in turn, was used in developing a Docassemble interview designed to advise whether two people are common law partners under the terms of that Act.

An image of the screen in which the list of factors is presented is included as Figure 7.2 and an image of the screen used to edit one of the factors is included as Figure 7.3.

An image showing how the issue review screen is displayed to the user is included as Figure 7.4.

Due time time limitations, the development of the reasoner was limited to a database of 50 promising cases. Of those, 28 were eventually encoded in the reasoner, the remainder having been unsuitable for various reasons. After the data entry process was completed, it was necessary to do some analysis of the output of the tool to determine its performance, and tune the reasoner to improve it. After redefining several factors by changing the side for which they were relevant, and setting appropriate knock-out factors, the results shown in Table 7.1 were obtained.

Introduction

Factors







Cases


Issues

Conclusion

Review Your Factors

Use the table below to review your factors and make any required changes.

ID	Name	Side	Actions
F1	cohabiting-gt-year	Plaintiff	 Edit  Delete
F2	children	Plaintiff	 Edit  Delete
F3	joint-accounts	Plaintiff	 Edit  Delete

 Add another

Continue

Figure 7.2: How the list of factors are displayed to the user in docassemble-openlcb.

Introduction

Factors

Cases

Issues

Conclusion

What are the details of the first factor?

Internal ID *

Which Side Does This *
Factor Favour? Defendant Plaintiff

Short Description *

Long Description *

Continue

Figure 7.3: How the details of a factor are displayed to the user in docassemble-openlcb.

Introduction

Factors

Cases

Issues

Conclusion

Review Issue: "Relationship of Interdependence"

Field	Value	Actions
ID	Relationship of Interdependence	Edit
Name	the parties are in a relationship of interdependence	Edit
Winner-if-Unraised	d	Edit
Join Type	conjunctive	Edit

Sub-Issues

Sub-Issue	Actions
Shared Life	Review Sub-Issue
children of relationship	Review Sub-Issue
+ Add a sub-issue to this issue	

Figure 7.4: How the details of an issue are displayed to the user in docassemble-openlcb.

Cases	28
Abstentions	4
Predictions	24
Correct Predictions	22
False Positives	1
False Negatives	1
Coverage (predictions/cases)	86%
Accuracy (correct predictions/predictions)	92%

Table 7.1: Results of Leave-One-Out Testing on Relationship of Interdependence Reasoner

The total amount of time spent developing this reasoner was less than 30 hours. Again, that is unlikely to be generalizable across different legal issues, or different subject-matter experts, and will scale with the size of the database and the complexity of the legal issue. However, it does give some impression as to the relative investment of subject-matter expert time required in order to generate a case-based reasoning tool of relatively strong accuracy as measured in leave-one-out testing.⁶

As mentioned in Chapter 6, Docassemble requires that deductive elements of the legislation are encoded in a series of blocks of code in the Docassemble configuration language. An example of a code block for implementing the other elements of the Act is included at listing 7.5

Listing 7.1: Example of Python Code Block in Docassemble Interview

```
1  _____
2  |
3  code: |
4  |     if first_party.has_other_aip or second_party.has_other_aip:
5  |         aip_invalid_no_multiples = True
6  |     else:
7  |         aip_invalid_no_multiples = False
8  |     _____
```

7.6 Temporal Reasoning in Procedural Languages

A challenge encountered in encoding this legislation was the fact that there are two different ways of initiating a common law relationship under the legislation. The first is a cohabitation of not less than three years, where the parties do not have children, and the second is a cohabitation of “some permanence” in the case that they do.⁷ The relationship begins at the end of the required period of time. Knowing when exactly the adult interdependent partnership begins is necessary in order to determine whether any terminating events happened

⁶No claim is made with regard to the actual strength of this reasoner. Leave-one-out testing is less than ideal for measuring prediction performance, which is better done against data that was not used in training the reasoner, or by comparison to human predictions.

⁷I might have treated “some permanence” as an open-textured phrase, but impermanence of the relationship is so seldom found as a factor in the presence of a child of the relationship, that I decided instead to arbitrarily treat one month as sufficient for “some permanence.”

before or after that time. And determining the start date is complicated in that a couple who began cohabiting without a child and then have a child may become partners immediately upon the date of the child's birth. Also, the cohabitation may have been interrupted by a separation, meaning that for purposes of the Act, the cohabitation is not deemed to have begun until the separation ended.

All of this is very typical of attempting to encode the provisions of legislation that deal with sequences of events over time. These sorts of problems are why temporal reasoning is an important feature of DLP tools for automating legal reasoning.

Docassemble does not have temporal reasoning capabilities built in, and so it was necessary to implement a procedural algorithm for determining when an adult interdependent partnership began, and whether, after that date it had ever been terminated.

The process of developing, coding, and testing that algorithm took nearly two business days. As an illustration of the complexity involved in dealing with temporal reasoning without specific features to support that work, an excerpt of the code used to do this is included as Listing 7.6.

Listing 7.2: Excerpt of Python code for Calculating the Start of an AIP

```
1 # Go through all the events
2 for e in events:
3     if is_cohab_start(e):
4         # Birth, then cohabitation, one month from start.
5         if child and is_after_birth_and_more_than_month_ago(e['
6             date'], child_date):
7             if not has_termination_within_one_month_after(e['date
8                 ']):
9                 aip_start_dates.append(e['date'] + one_month)
10
11         # Cohabitation, then birth in less than 35 months,
12         # one month from birth.
13         elif child and is_within_35_months_of_birth(e['date'],
14             child_date):
15             # First, if the cohabitation lasted 3 years,
16             # it might be the start date.
17             if is_more_than_three_years_ago(e['date']):
18                 if not has_termination_within_three_years_after(e
19                     ['date']):
20                     aip_start_dates.append(e['date'] + three_years
21                         )
```

```

18         # Second, if there was no termination within 1 month
19         of
20         # the birth, the birth might be the start date.
21         if is_more_than_one_month_ago(e['date']):
22             if not has_termination_within_one_month_after(
23                 child_date):
24                 aip_start_dates.append(child_date + one_month)
25
26         # No Birth before Or within 35 months of cohab start ,
27         # three years from start.
28         else:
29             if is_more_than_three_years_ago(e['date']):
30                 if not has_termination_within_three_years_after(e
31                     ['date']):
32                     aip_start_dates.append(e['date'] + three_years
33                         )
34
35     # make the list of aip termination dates
36     for e in events:
37         if is_termination(e):
38             if is_more_than_one_year_ago(e['date']):
39                 if not has_reconciliation_within_one_year_after(e['
40                     date']):
41                     aip_end_dates.append(e['date'] + one_year)
42
43     return earliest_unterminated_aip_start(aip_start_dates ,
44         aip_end_dates)

```

Had Docassemble offered a DLP language with temporal reasoning features such as pre-defined meanings for dates, durations, qualities such as “before”, “during”, and “after”, and the ability to convert a series of events into a truth function over time, the process would have been significantly easier.

7.7 Impressions

Based on my anecdotal experience of using the tools created as part of the Docassemble-OpenLCBR project, I believe they are drastically easier for lawyers to use and understand than most open source DLP tools available to lawyers today. It is not possible to compare Docassemble-OpenLCBR to the usability of other open-source case-based reasoning tools, because there are none in the survey in Chapter 6. In fact, Docassemble-openLCBR may be the first.

I submit that this experiment gives reason to believe that there is an opportunity to make case-based reasoning something that lawyers are comfortable

using.

Similarly, while this experiment was not designed to measure the quality of the IBP algorithm, this reasoner, or CBR tools generally, it does provide some evidence that CBR has potential for expanding the realm of legal services that can be automated to include more open-textured matters.

This experiment also shows that it is possible to generate explanations which are intelligible to human users for the predictions that arise from an IBP algorithm or other case-based reasoning algorithms.

Chapter 8

Conclusion

8.1 Summary

Declarative Language Programming technologies have been around for nearly four decades, and were immediately recognized as having significant utility in automating legal reasoning with regard to written legal rules.

The legal academic literature on expert systems in law reveals a longstanding skepticism about the viability of DLP tools. Of the many concerns stated, most can be resolved by conceptualizing of the encodings differently, and by using the features of modern DLP tools. The remaining hard problems are open-textured concepts, for which case-based reasoning provides a partial solution, and which do not affect all applications of DLP tools, and the knowledge acquisition bottleneck problem.

This dissertation argues that the solution to the knowledge acquisition bottleneck problem is to focus on improving the ease of use of DLP tools until we have something that is analogous to spreadsheets for legal reasoning. Tools such as Oracle Policy Automation show that it is possible to create user-friendly interfaces that use metaphors that are familiar to legal subject matter experts.

When viewed from an access to justice perspective, many of the concerns commonly expressed fade in importance compared to the potential benefits to people for whom the next best alternative is nothing.

The most important qualities for tools that bring the power of DLP to automated legal reasoning are affordability, uncertainty, defeasibility, ease of

use, temporal reasoning, case-based reasoning, and explainability. A review of the tools currently available shows that most of these features do not exist in an open source alternative, and none of the open source alternatives that exist feature ease of use. While explanation remains a difficult feature to implement, there are no technological obstacles to tools that feature all of these capabilities.

People who wish to develop technology to improve automated legal reasoning should focus on building DLP tools that are open source and free, are easy enough for lawyers to learn and use, and have at least one of the other technological features listed. Because defeasibility is already implemented in the open source tool ErgoLite, an easy-to-use open source tool implementing defeasible reasoning would seem like a realistic next step.

8.2 The Question of Scale

Professor Ashley's text suggests that DLP technologies for human beings, which are designed to allow one person to do one thing at a time, are inadequate to the scale of the problem of automating legal services. In discussing why expert systems are not the "killer app" for the legal domain, Ashley argues "... text analytics cannot solve this particular knowledge acquisition bottleneck. While the new text analytics can extract certain kinds of semantic legal information from text, they are not yet able to extract expert system rules."¹

Reviewers of this dissertation, too, have also asked whether the solution proposed to the knowledge acquisition bottleneck can "scale." There are three answers to this criticism.

First, this dissertation does not attempt to answer the question of how we can digitize the entire statute book. It is asking whether and how we can increase the efficiency of the provision of legal services using DLP.

Second, high-quality DLP tools for encoding written legal rules aimed at human beings encoding one rule at a time are a necessary first step toward our best hope of automation on that larger scale.

¹Ashley (see n. 5) at p 11.

Machine learning has made amazing strides, particularly in the task of translation. The task of taking written legal rules in a natural language and encoding them can certainly be thought of as a translation task. There is every reason to believe those same technologies will be able to do a great deal of this work for us at some point in the future.

But machine learning requires data. For translation, it requires human translations between the source and target language. The translation data needs to be high-quality, and it needs to be voluminous. For translating between written legal rules and declarative logic code there is no viable database, and no obvious route to getting one.

That is, no obvious route “yet.” Giving human beings a tool that makes encoding written legal rules worthwhile is our best hope of creating a database of sufficient size and quality to take advantage of machine learning, and achieve the scale of digitization to which Ashley and others aspire.

Third, even if machine learning were to progress to the point that it could generate semantically-meaningful representations of written legal rules and use them, unless those representations can be shared with human beings, and tested for adherence to their expectations, they will not be trusted. Which means that tools such as this dissertation argues for will be necessary to have people learn what the computer thinks the rule means, if not the other way around.

In that sense, regardless of the progress on other fronts, human-focused tools for working with DLP encodings of written legal rules are a necessary step toward practical solutions at scale.

8.3 Future Work

We make a number of assumptions which, with further work, it would be possible to objectively test.

There is a presumption that lawyers will find it easier to encode written legal rules in DLP tools than in procedural tools. There is a presumption that encoding and maintaining the encoding of written legal rules is simplified

by the use of declarative tools. There is a presumption that interfaces which appear similar to statute law, as opposed to alternatives like drag-and-drop language interfaces, text annotation, or flowcharts, will be easier for lawyers to learn to use.

If DLP tools are to have the beneficial effect in automated legal reasoning that they are capable of having, unrelenting focus must be placed on ease of use. It is worth noting that because ease of use and ease of learning are so closely linked, legal education institutions are ideally placed to be able to do this sort of research.

References

- A2J Author (<https://www.a2jauthor.org/>, Accessed: July 25, 2019).
- Accord Project, Accord Project (<https://www.accordproject.org/>, Accessed: December 1, 2017).
- Ashley KD, *Artificial Intelligence and Legal Analytics* (Cambridge University Press June 2017).
- Ashley KD and Brüninghaus S, “Automatically classifying case texts and predicting outcomes” (2009) 17(2) *Artificial Intelligence and Law* 125.
- Australasian Legal Informatmon Institute, DataLex (<http://austlii.community/foswiki/DataLex/>, Accessed: July 25, 2019).
- Berman DH and Hafner CD, “Indeterminacy: A challenge to logic-based models of legal reasoning” (1987) 3(1) *International Review of Law, Computers & Technology* 1.
- Berry B and McLintock A, “Accountants and financial modelling” (1991) 4(4) *OR Insight* 11.
- Borrelli MA, “Prolog and the law: Using expert systems to perform legal analysis in the uk” (1989) 3 *Software LJ* 687.
- Capper P and Susskind RE, *Latent Damage Law: The Expert System:[a Study of Computers in Legal Problem Solving]* (Butterworths 1988).
- Coherent Knowledge, Ergo Pricing (<https://coherentknowledge.com/pricing/>, Accessed: July 25, 2019).
- ErgoAI (<https://coherentknowledge.com/>, Accessed: July 25, 2019).
- Financial Domain Application (<https://coherentknowledge.com/financial-domain-application/>, Accessed: July 25, 2019).
- Colarusso D, QnA Markup (<https://www.qnamarkup.org/>, Accessed: July 25, 2019).
- Commonwealth Scientific and Industrial Research Organization, Data61 Digital Legislation & Regulation as a Platform (<https://digital-legislation.net/>, Accessed: July 25, 2019).
- Drools (<https://www.drools.org/>, Accessed: July 25, 2019).
- Goldstein J, How The Electronic Spreadsheet Revolutionized Business “All Things Considered” (<https://www.npr.org/2015/02/27/389585340/how-the-electronic-spreadsheet-revolutionized-business>).
- Grabmair M, OpenLCBR (<https://github.com/mgrabmair/openlcbr>).
- Grad B, “The Creation and the Demise of VisiCalc” (2007) 29(3) *IEEE Annals of the History of Computing* 20.

- Hutchison C, *The Fundamentals of Statutory Interpretation* (LexisNexis Canada 2018).
- Idelberger F et al., Evaluation of logic-based smart contracts for blockchain systems [2016] (In: Alferes J, Bertossi L, Governatori G, Fodor P, Roman D (eds), Rule Technologies. Research Tools, and Applications, RuleML 2016. Lecture Notes in Computer Science, vol 9718, Springer, Cham.).
- InRule Technology, Inc, InRule (<https://www.inrule.com/>, Accessed: July 25, 2019).
- interProlog Consulting, Prolog Studio (<http://interprolog.com/interprolog-studio/>, Accessed: July 25, 2019).
- Kifer M, Defeasible Reasoning in Ergo (<https://docs.google.com/document/d/1aPUUUzkBgtdQ8PMvyIRGgA-qGYRowbqZBeaUBUXFH3k/edit>, Accessed: July 25, 2019).
- Lam H and Governatori G, *The making of SPINdle* (Springer 2009).
- Legal Services Corporation, Technology Initiative Grant Program (<https://www.lsc.gov/grants-grantee-resources/our-grant-programs/tig>, Accessed: July 25, 2019).
- Leith P, “Fundamental Errors in Legal Logic Programming” (1986) 29(6) *The Computer Journal* 545.
- Leith P, “The Emperor’s New Expert System” (1987) 50(1) *The Modern Law Review* 128.
- “The rise and fall of the legal expert system” (2016) 30(3) *International Review of Law, Computers & Technology* 94.
- Lindop C, Oracle to Acquire Hayley/Ruleburst/Softlaw for A\$150m (<http://tmt-transactions.com/oracle-to-acquire-hayleyruleburstsoftlaw-for-a150m/>, Accessed: July 25, 2019).
- McCarty LT, “Reflections on “Taxman”: An Experiment in Artificial Intelligence and Legal Reasoning” (1977) 90(5) *Harvard Law Review* 837.
- “An implementation of *Eisner v. Macomber*” [1995] *ICAIL ’95* 276.
- “How to ground a language for legal discourse in a prototypical perceptual semantics” [2015] *ICAIL ’15* 89.
- “Finding the right balance in artificial intelligence and law” [2018] *Research Handbook on the Law of Artificial Intelligence* 55.
- Morris J, “User-Friendly Open-Source Case-Based Legal Reasoning” [2019] *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law* 270.
- Docassemble-OpenLCBR (<https://github.com/Gauntlet173/Docassemble-OpenLCBR>, Accessed: July 25, 2019).
- Neota Logic, Neota Logic (<https://www.neotalogic.com/>, Accessed: December 1, 2017).
- Neota Logic reveals new Client Advisory Board [] (<https://www.neotalogic.com/2019/06/03/neota-logic-reveals-new-client-advisory-board/>, Accessed: July 25, 2019).
- Neota Logic: University Programs (<https://www.neotalogic.com/pro-bono/law-schools/>, Accessed: July 25, 2019).

- O'Callaghan TA, "A Hybrid Legal Expert System" (PhD thesis, 2003).
- Oracle Corporation, Oracle e-Business Suite Applications Global Price List (<http://www.oracle.com/us/corporate/pricing/applications-price-list-070574.pdf>, Accessed: September 21, 2017).
- Pemmasani G et al., "Online Justification for Tabled Logic Programs" [2004] Functional and Logic Programming (Kameyama Y and Stuckey PJ eds. 24).
- Popple J, *A pragmatic legal expert system* (Dartmouth (Ashgate) 1996).
- Pyle J, Docassemble (<https://docassemble.org>, Accessed: January 25, 2019).
- Rynkiewicz S, Best Web Tools of 2018 (http://www.abajournal.com/magazine/article/best_legal_apps_2018/, Accessed: July 25, 2019).
- Sergot MJ et al., "The British Nationality Act as a logic program" (1986) 29(5) Communications of the ACM 370.
- Service Innovation Lab, Government of New Zealand, Better Rules for Government Discovery Report (<https://www.digital.govt.nz/assets/Uploads/Better-Rules-for-Government-Discovery-Report.pdf>, Accessed: July 25, 2019).
- Susskind RE, "Expert systems in law: A jurisprudential approach to artificial intelligence and legal reasoning" (1986) 49(2) The modern law review 168.
- *Expert systems in law: a jurisprudential inquiry* (Clarendon; New York 1987).
- *Tomorrow's lawyers* (Second edition, Oxford University Press 2017).
- Susskind RE and Susskind D, *The future of the professions* (First published in paperback, Oxford University Press 2017).
- Themis Solutions Inc, Clio (<http://www.clio.com>, Accessed: July 28, 2019).
- University of Luxembourg, NAI: Normative Reasoner (<http://nai.uni.lu>, Accessed: July 25, 2019).
- Van Emden M and Kowalski R, "The Semantics of Predicate Logic as a Programming Language" (1976) 23(4) Journal of the ACM (JACM) 733.
- Wikipedia, Oracle Policy Automation (https://en.wikipedia.org/wiki/Oracle_Policy_Automation, Accessed: July 25, 2019).

Appendix A

Oracle Policy Automation Encoding of Adult Interdependent Partnership Act

the parties are in an Adult Interdependent Relationship if

an adult interdependent partnership was commenced
the parties have a valid adult interdependent partnership agreement
or
the parties have lived in a relationship of interdependence for a continuous period of not less than 3 years
or
the parties have lived in a relationship of interdependence of some permanence and there is a child of the relationship by birth or adoption.
and
the parties have not terminated their adult interdependent partnership

the parties have a valid adult interdependent partnership agreement if

ExistsScope(all instances of AIP Agreement)
the agreement is between the two parties
and
the AIP agreement has not been terminated
and
the agreement was not induced by fraud, duress, or undue influence
and
the agreement was signed before today
the date the AIP agreement was signed < CurrentDate()
and
ForAllScope(all instances of party to the agreement for AIP Agreement)
the party to the agreement was not married when the agreement was signed
and
the party to the agreement was competent at the time the agreement was signed
and
the party to the agreement was of age or had consent
and
the parties were living together or intended to when the agreement was entered into
the parties lived together in a relationship of interdependence when the agreement was entered into
or
the parties intended to live together in a relationship of interdependence when the agreement was entered into

the agreement is between the two parties if

in the case of First Party (FP)
ExistsScope(all instances of party to the agreement for AIP Agreement, target party)
ForScope(the person who is target party, target person)
InstanceEquals(FP, target person)
and
in the case of Second Party (SP)
ExistsScope(all instances of party to the agreement for AIP Agreement, target party)
ForScope(the person who is target party, target person)
InstanceEquals(SP, target person)

the AIP agreement has been terminated if

for at least one of all instances of party to the agreement for AIP Agreement , the party to the agreement got married after the agreement was signed
or
the agreement was terminated by something else

the party to the agreement was married when the agreement was signed if

forScope(the AIP agreement for the party to the agreement)

ForScope(the person who is party to the agreement, testparty)
ExistsScope(all instances of Marriage)
testparty is a member of spouse
and
the date the AIP agreement was signed >= Marriage Date Start
and
one of
the date the AIP agreement was signed <= Marriage Date End
or
the marriage is not terminated

the party to the agreement got married after the agreement was signed if

forScope(the AIP agreement for the party to the agreement)
ForScope(the person who is party to the agreement, testparty)
ExistsScope(all instances of Marriage)
testparty is a member of spouse
and
Marriage Date Start > the date the AIP agreement was signed

the party to the agreement was of age or had consent if

the party to the agreement was of age
or
the party to the agreement had consent

the party to the agreement was of age if

ForScope(the AIP agreement for the party to the agreement, AGREEMENT)
ForScope(the person who is party to the agreement, TRYPerson)
the number of years between the date the AGREEMENT was signed and the
TRYPerson's birth date >=18

the party to the agreement had consent if

ForScope(the AIP agreement for the party to the agreement, TRYAGREEMENT)
ForScope(the person who is party to the agreement, TRYPerson)
the number of years between the date the TRYAGREEMENT was signed and the
TRYPerson's birth date >=16
and
the party to the agreement had the prior written consent of their guardians to
enter into the agreement

the parties lived together in a relationship of interdependence when the agreement was entered into if

ExistsScope(all instances of cohabitation)
ForAllScope(all instances of party to the agreement for AIP Agreement)
ForScope(the person who is party to the agreement, agreement party)
agreement party is a member of Parties to the Cohabitation
and
Cohabitation Start Date < the date the AIP agreement was signed
and
one of
Cohabitation End Date is uncertain
or
Cohabitation End Date > the date the AIP agreement was signed
and
ValueAt(the date the AIP Agreement was signed, the cohabitation involves a relationship
of interdependence)

the parties intended to live together in a relationship of interdependence when the agreement was entered into if

ExistsScope(all instances of cohabitation)

ForAllScope(all instances of party to the agreement for AIP Agreement)
ForScope(the person who is party to the agreement, agreement party)
agreement party is a member of Parties to the Cohabitation
and
Cohabitation Intent Date < the date the AIP agreement was signed
and
one of
Cohabitation End Date is uncertain
or
Cohabitation End Date > the date the AIP agreement was signed
and
the cohabitation was intended to involve a relationship of interdependence

the cohabitation involves a relationship of interdependence if

the parties to the cohabitation share one another's lives
and
the parties to the cohabitation are emotionally committed to one another
and
the parties to the cohabitation function as an economic and domestic unit
and
the cohabitation does not involve an employee presuming false
and
the parties to the cohabitation are not married to one another presuming false
and
the cohabitation does not involve a related minor

Marriage is a member of marriages in which all spouses are parties to the cohabitation if

for each of Parties to the Cohabitation (cohab party)
cohab party is a member of spouse

Marriage is a member of marriages in which any spouse is a party to the cohabitation if

for at least one of Parties to the Cohabitation (cohab party)
cohab party is a member of spouse

IsMemberOf(Marriage, marriages to which either party is a spouse) if

in the case of First Party (FP)
ExistsScope(spouse, S)
InstanceEquals(FP,S)
or
in the case of Second Party (SP)
ExistsScope(spouse, S)
InstanceEquals(SP,S)

**the parties to the cohabitation are married to one another =
TemporalFromRange(marriages in which all spouses are parties to the cohabitation,
Marriage Date Start, Marriage Date End, True)**

**the cohabitation involves an employee = TemporalFromRange(all instances of employment
period for Cohabitation, employment period start date, employment period end date,
True)**

the cohabitation involves a related minor if

the parties to the cohabitation are related
and
for at least one of Parties to the Cohabitation (cohab party)
ValueAt(Cohabitation Start Date,the cohab party is a minor)

the person's age in years = TemporalYearsSince(the person's birth date, the current date)

the person is a minor if

the person's age in years < 18

IsMemberOf(Cohabitation, cohabitations between the parties in a relationship of interdependence) if

in the case of First Party (FP)

ExistsScope(Parties to the Cohabitation, CP)

InstanceEquals(FP,CP)

and

in the case of Second Party (SP)

ExistsScope(Parties to the Cohabitation, CP)

InstanceEquals(SP,CP)

and

the cohabitation involves a relationship of interdependence

the parties are cohabiting in a relationship of interdependence = TemporalFromRange(cohabitations between the parties in a relationship of interdependence, Cohabitation Start Date, cohabitation end date allowing for open ended cohabitations, True)

IsMemberOf(Period of Separation, periods of separation between the parties) if

Intent of one or both parties was that relationship would terminate

and

in the case of First Party (FP)

ExistsScope(separation parties, SepP)

InstanceEquals(FP, SepP)

and

in the case of Second Party (SP)

ExistsScope(separation parties, SepP)

InstanceEquals(SP, SepP)

the parties are separated = TemporalFromRange(periods of separation between the parties, Period of Separation Start Date, Period of Separation End Date allowing for open ended periods of separation, True)

the parties are cohabiting uninterrupted in a relationship of interdependence if

the parties are cohabiting in a relationship of interdependence presuming false

and

the parties are not separated presuming false

the parties have lived in a relationship of interdependence for a continuous period of not less than 3 years if

IntervalConsecutiveDays(Earliest(),the current date,365*3 ,the parties are cohabiting uninterrupted in a relationship of interdependence presuming false)

the parties have lived in a relationship of interdependence of some permanence and there is a child of the relationship by birth or adoption if

the parties have lived in a relationship of interdependence of some permanence assuming false

and

there is a child of the relationship by birth or adoption

there is a child of the relationship by birth or adoption if

ForScope(First Party, FP)

ForScope(Second Party, SP)

ExistsScope(all instances of Person)

SP is a member of parents

and

FP is a member of parents

the parties have lived in a relationship of interdependence of some permanence if

IntervalConsecutiveDays(Earliest(), the current date, 30, the parties are cohabiting in a relationship of interdependence)

the parties have terminated their adult interdependent partnership if

AIP Common Law Rules.docx

10

11/05/2018 2:28 PM

one of

the adult interdependent partnership is terminated by a written agreement regarding separation

or

the adult interdependent partnership is terminated by separation

or

the adult interdependent partnership is terminated by marriage

or

the adult interdependent partnership is terminated by an AIP Agreement

or

the adult interdependent partnership is terminated by a declaration of irreconcilability

the adult interdependent partnership is terminated by a written agreement regarding separation if

ExistsScope(all instances of Written Agreement re Separation)

ForScope(First Party, FP)

FP is a member of WArS parties

and

ForScope(Second Party, SP)

SP is a member of WArS parties

and

Written Agreement re Separation Provides evidence of intent to live separate and apart without possibility of reconciliation

and

Written Agreement re Separation Date > WhenLast(the current date, an adult interdependent partnership was commenced)

the adult interdependent partnership is terminated by separation if

IntervalAtLeastDays(earliest(), latest(), 365, the parties are separated)

and

WhenLast(the current date, an adult interdependent partnership was not commenced) < WhenLast(the current date, the parties are not separated)

one of the parties got married = TemporalFromRange(marriages to which either party is a spouse, Marriage Date Start, AddDays(Marriage Date Start, 1), True)

the adult interdependent partnership is terminated by marriage if

WhenLast(the current date, an adult interdependent partnership was not commenced) < WhenLast(the current date, one of the parties got married)

the adult interdependent partnership is terminated by an AIP agreement if

one of

WhenLast(the current date, one of the parties signed an AIPA) < WhenLast(the current date, the parties have not lived in a relationship of interdependence of some permanence and there is a child of the relationship by birth or adoption)

or

WhenLast(the current date, one of the parties signed an AIPA) < WhenLast(the current date, the parties have not lived in a relationship of interdependence for a continuous period of not less than 3 years)

one of the parties signed an AIPA = TemporalFromRange(AIPAs signed by either party, the date the AIP agreement was signed, AddDays(the date the AIP agreement was signed, 1), True)

IsMemberOf(AIP Agreement, AIPAs signed by either party) if

in the case of First Party (FP)

ExistsScope(all instances of party to the agreement for AIP Agreement)

ExistsScope(the person who is party to the agreement, TP)

InstanceEquals(FP, TP)

or

in the case of Second Party (SP)

ExistsScope(all instances of party to the agreement for AIP Agreement)

ExistsScope(the person who is party to the agreement, TP)

InstanceEquals(SP, TP)

the adult interdependent partnership is terminated by a declaration of irreconcilability if

ExistsScope(all instances of Declaration of Irreconcilability)

FP is a member of DoI Parties

and

SP is a member of DoI Parties

and

Declaration of Irreconcilability Date > WhenLast(the current date, an adult interdependent partnership was not commenced)

cohabitation end date allowing for open ended cohabitations	
Cohabitation End Date	<u>Cohabitation End Date</u> is certain
Latest()	otherwise

the parties are cohabiting in a relationship of interdependence presuming false	
the parties are cohabiting in a relationship of interdependence	<u>the parties are cohabiting in a relationship of interdependence</u> is certain
False	otherwise

the parties are separated presuming false	
the parties are separated	<u>The parties are separated</u> is certain
False	otherwise

the parties have lived in a relationship of interdependence of some permanence assuming false	
the parties have lived in a relationship of interdependence of some permanence	<u>the parties have lived in a relationship of interdependence of some permanence</u> is certain
False	otherwise

the cohabitation involves an employee presuming false	
the cohabitation involves an	<u>the cohabitation involves an employee</u> is certain

employee	
False	otherwise

the parties to the cohabitation are married to one another presuming false	
the parties to the cohabitation are married to one another	<u>the parties to the cohabitation are married to one another is certain</u>
False	otherwise

the parties are cohabiting uninterrupted in a relationship of interdependence presuming false	
the parties are cohabiting uninterrupted in a relationship of interdependence	<u>the parties are cohabiting uninterrupted in a relationship of interdependence is certain</u>
False	otherwise

Period of Separation End Date allowing for open ended periods of separation	
Period of Separation End Date	<u>Period of Separation End Date is Certain</u>
Latest()	otherwise

Appendix B

ErgoAI Encoding of Adult Interdependent Partnership Act

Listing B.1: ErgoAI Encoding of Adult Interdependent Partnerships Act

```
1 // AIRA.ergo
2 // An Ergo Implementation of the Adult Interdependent
  Relationships Act, SA 2002, c A-4.5
3 // As accessed May 16, 2018 at https://www.canlii.org/en/ab/laws/
  stat/sa-2002-c-a-4.5/latest/sa-2002-c-a-4.5.html
4 // Implemented as coursework for CMPUT 605
5 // Course Supervisor: Randy Goebel
6 // University of Alberta
7 // Jason Morris
8 // LLM Candidate
9 // Student ID #0353905
10
11 // The "expert" compiler option allows for certain sort of frame
  syntax expressions in the conclusion portion of rules,
12 // and allows for other features which tend to be mis-used by
  inexperienced coders.
13 :- compiler_options{expert=on}.
14
15 // The use argumentation theory setting allows for the use of the
  defeasibility features of the Ergo language.
16 :- use_argumentation_theory.
17
18 // ONTOLOGY
19 Relationship []
20   first=>Person,
21   second=>Person
22   []. // A Relationship has a first person and a second person.
23
24 AIP::Relationship. // An Adult Interdependent Partnership is a
  type of Relationship.
25 FAIP::Relationship. // A Former Adult Interdependent Partnership
  is a type of Relationship.
26 AIR::AIP. // An Adult Interdependent Relationship is a type of
  Adult Interdependent Partnership.
```

```

27 ROI::Relationship. // A Relationship of Interdependence is a type
   of Relationship.
28 ROI[|
29   continuous_period_of_not_less_than_3_years ,
30   of_some_permanence
31   |].
32
33 Person [|
34   married_to=>Person ,
35   shares_life_with=>Person ,
36   is_emotionally_committed_to=>Person ,
37   functions_as_economic_and_domestic_unit_with=>Person ,
38   has_child_with=>Person ,
39   related_by_blood_or_adoption_to=>Person
40   |].
41
42
43 // ENCODING OF ACT
44
45 /*
46 Section 8(2), 8(3), 9, are note encoded because they do not
   address the question of when an adult interdependent
47 partnership exists. Section 2 is not encoded because it merely
   states that it applies retroactively.
48 */
49
50 /*
51 Interpretation
52 1(1) In this Act,
53
54           (a) "adult interdependent partner"
55               means an adult interdependent
56                   partner within the meaning of
57                   section 3, but does not include a
58                   former adult interdependent partner
59                   ;
60
61 */
62 \opposes(faip(?_A,?_B):FAIP,aip(?_A,?_B):AIP). // If A is a FAIP
   of B, A is not an AIP of B. The two are opposites.
63
64 /*
65 Interpretation
66 1(1) In this Act,
67
68           (c) "adult interdependent
69               relationship" means the
70                   relationship between 2 persons who
71                   are adult interdependent partners
72                   of each other;
73
74 */
75 // If A is an AIP of B and B is an AIP of A, then there is an AIR
   between A and B.
76 // Note that this will prove both AIR(A,B) and AIR(B,A), so either
   notation can be used to test for the existence of an AIR.

```

```

65 // Note also that this definition is not used anywhere in the act.
    It exists in the code solely to allow it to be queried.
66 @{AIRbetweenTwoAIP_Section_1.1.c}
67 air(?A,?B):AIR :- // There is an AIR between A and B if
68   aip(?A,?B):AIP, // A is an AIP of B and
69   aip(?B,?A):AIP. // the other way around.
70
71 /*
72 Interpretation
73 1(1) In this Act,
74
75           (f) "relationship of
76               interdependence" means a
77               relationship outside marriage in
78               which any 2 persons
79               (i) share one another's lives
80               (ii) are emotionally committed
81                   to one another, and
82               (iii) function as an economic
83                   and domestic unit.
84 */
85 // Two people are in a relationship of interdependence if i, ii,
86   and iii, and they are not married.
87 @{ROIdefined_Section_1.1.f}
88 roi(?A,?B):ROI[ first ->?A, second ->?B] :-
89   ?A:Person, ?B:Person, // A and B are people.
90   \naf ?A[married_to->?B],
91   \naf ?B[married_to->?A], //A and B and not married.
92   ?A[shares_life_with ->?B],
93   ?B[shares_life_with ->?A], // A and B share one another's lives.
94   ?A[is_emotionally_committed_to->?B],
95   ?B[is_emotionally_committed_to->?A], // A and B are emotionally
96       committed to one another, and
97   ?A[functions_as_economic_and_domestic_unit_with->?B],
98   ?B[functions_as_economic_and_domestic_unit_with->?A]. // A and B
99       function as an economic and domestic unit.
100
101 // Section 3
102 /*
103 Adult interdependent partner
104 3(1) Subject to subsection (2), a person is the adult
105       interdependent partner of another person if
106           (a) the person has lived with the
107               other person in a relationship of
108               interdependence
109               (i) for a continuous period
110                   of not less than 3 years, or
111               (ii) of some permanence, if
112                   there is a child of the
113                   relationship by birth or
114                   adoption,
115 or
116           (b) the person has entered into an
117               adult interdependent partner

```



```

                                agreement with the other person
                                under section 7.
101 (2) Persons who are related to each other by blood or adoption
    may only become adult interdependent partners of each other by
    entering into an adult interdependent partner agreement under
    section 7.
102 */
103 @{AIPDefined_Section_3}
104 aip(?A,?B):AIP :- // A is an AIP of B if (one-directional)
105   roi(?A,?B):ROI, // there is a relationship of interdependence
    between the parties (presumed bidirectional) , and
106   (
107     roi(?A,?B):ROI[continuous_period_of_not_less_than_3_years] //
    3(1)(a)(i)
108     \or
109     ( // 3(1)(a)(ii)
110       roi(?A,?B):ROI[of_some_permanence] ,
111       ?A[has_child_with->?B] // relationship is presumed
    bidirectional.
112     )
113   )
114   \or
115   (
116     aipa(?A,?B):AIPA; aipa(?B,?A):AIPA // there is an AIPA
    between them.
117   ).
118
119 @{AIPBloodAdoption_Section_3_2}
120 \neg aip(?A,?B):AIP :- // A is not an AIP of B if
121   ?A:Person[related_by_blood_or_adoption_to->?B], // A is related
    by blood or adoption to B, and
122   \naf aipa(A,B):AIPA,
123   \naf aipa(B,A):AIPA.
124 \overrides(AIPBloodAdoption_Section_3_2 , AIPDefined_Section_3).
125
126 // Section 4
127 /*
128 Relationship of interdependence
129 4(1) A relationship of interdependence may exist between 2
    persons who are related to each other by blood or adoption
    except where one of the persons is a minor.
130 (2) A relationship of interdependence does not exist between 2
    persons where one of the persons provides the other with
    domestic support and personal care for a fee or other
    consideration or on behalf of another person or organization ,
    including a government.
131 */
132
133 @{NoROIMinorBlood_Section_4_1}
134 \neg roi(?A,?B):ROI :- // there is no ROI if
135   ?A[related_by_blood_or_adoption_to->?B] ,
136   (
137     ?A[is_minor] ;
138     ?B[is_minor]

```

```

139   ).
140 \overrides(NoROIMinorBlood_Section_4_1 , ROIDefined_Section_1_1-f).
141
142 @NoROIEmployment_Section_4_2}
143 \neg roi(?A,?B):ROI :- // there is no ROI if
144   ?A[provides_care_for_consideration_to->?B];
145   ?B[provides_care_for_consideration_to->?A].
146 \overrides(NoROIEmployment_Section_4_2 , ROIDefined_Section_1_1-f).
147
148 // Section 5
149 /*
150 Restrictions
151 5(1) A person cannot at any one time have more than one adult
      interdependent partner.
152 (2) A married person cannot become an adult interdependent
      partner while living with his or her spouse.
153 */
154 // Note that these provisions are explicitly temporal. Given the
      difficulty of implementing temporal reasoning in
155 // ergo without developing a complicated library , those temporal
      aspects have been subsumed into the predicates ,
156 // which specify "when formed". Note that "when formed" is a
      guess at what this section means in 5(1). 5(2) is
157 // explicit about "becoming" an AIP "while". So the period of the
      AIP cannot commence during the period of cohab.
158 // But section 5(1) does not say what the resolution is if there
      are two processes that would both result in overlapping
159 // AIPs. The presumption is that only the first comes into effect
      , which is approximated by "had other AIP when formed."
160 // However, it is also possible for a person to meet the
      requirements for an AIP with two different people at the same
161 // time, and there is no indication of how to address that
      circumstance in the legislation.
162
163 @OneAIPAtATime_Section_5_1}
164 \neg aip(?A,?B):AIP :- // there is no AIP between a and B if
165   aip(?A,?B):AIP, // there is any AIP between A and B, and
166   ( //either
167     aip(?A,?B):AIP[first_had_other_AIP_when_formed];
168     aip(?A,?B):AIP[second_had_other_AIP_when_formed]
169   ).
170 \overrides(OneAIPAtATime_Section_5_1 , AIPDefined_Section_3).
171
172 @NoAIPWhileMarriedAndCohab_Section_5_2}
173 \neg aip(?A,?B):AIP :- // there is no AIP between A and b if
174   aip(?A,?B):AIP, // there is any AIP between A and B, and
175   ( // either
176     aip(?A,?B):AIP[first_lived_with_spouse_when_formed];
177     aip(?A,?B):AIP[second_lived_with_spouse_when_formed]
178   ).
179 \overrides(NoAIPWhileMarriedAndCohab_Section_5_2 ,
      AIPDefined_Section_3).
180
181 // Section 6

```

```

182 /*
183 Minors
184 6 Subject to sections 4(1) and 7(2), a minor may be an adult
interdependent partner.
185 */
186 // This does not need to be encoded.
187 // It state that a minor may be an adult interdependent partner.
There is no way to encode the possibility of
188 // something being true. If this is intended in the deontic sense
of "is permitted to", this encoding is not attempting
189 // to encode deontic concepts.
190 // This is also not a defeasibility statement.
191 // Section 4(1) states that a minor cannot be party to an ROI.
That is not an exception
192 // to the statement a minor may be an AIP. Section 7(2) states a
minor must have consent or be 16 to enter
193 // into an AIPA. That is not an exception to the statement that a
minor may be in an AIP. That a minor
194 // can be an AIP is implicit from the possibility of entering into
the AIPA.
195
196 // Section 7
197 /*
198 Adult interdependent partner agreement
199 7(1) Subject to subsection (2), any 2 persons who are living
together or intend to live together in a relationship of
interdependence may enter into an adult interdependent partner
agreement in the form provided for by the regulations.
200 (2) A person may not enter into an adult interdependent partner
agreement if the person
201 (a) is a party to an existing adult
interdependent partner agreement,
202 (b) is married, or
203 (c) is a minor, unless
204 (i) the minor is at least 16
years of age, and
205 (ii) the minor's guardians have
given their prior written
consent.
206 */
207 // Again, we are not encoding deontic concepts, so "may enter" and
"may not enter" are being translated to determine
208 // whether or not the AIPA is "valid". Note that negating the
existence of the AIPA at this point eliminates the need
209 // to encode section 8(d).
210 // Note that the semantics of 7(1) is unclear. Not clear whether
"in a relationship of interdependence" applies to both
211 // "are living together" and "intend to live together". I
presumed that it applies to both.
212
213 @{AIPADefined_Section_7_1}
214 aipa(?A,?B):AIPA :- // there is an adult interdependent
partnerhsip agreement between A and B if

```

```

215     (?_form [ first ->?A, second ->?B ] : AIPAFORM; ?_form [ first ->?B, second
216         ->?A ] : AIPAFORM), // the parties fill out a form, and
217     ( // either
218         ?_form [ parties_cohabited_in_ROI_when_entered_into ]; // they
219             cohabited when they signed the form, or
220         ?_form [ parties_intended_to_cohabit_in_ROI_when_entered_into ]
221     ).
222 @ { Ineligibility_for_AIPA_Section_7_2 }
223 \neg aipa (?A, ?B) : AIPA :- // there is NO AIPA between A and B if
224     (?_form [ first ->?A, second ->?B ] : AIPAFORM; ?_form [ first ->?B, second
225         ->?A ] : AIPAFORM), // the parties filled out the form, and
226     ( // any of
227         (
228             ?_form [ first_already_has_AIPA_when_signed ]; ?_form [
229                 second_already_has_AIPA_when_signed ]
230         ) \or (
231             ?_form [ first_is_married_when_signed ]; ?_form [
232                 second_is_married_when_signed ]
233         ) \or (
234             ( // either
235                 ?_form [ first_is_minor_when_signed ],
236                 \naf ?_form [ first_is_at_least_16_when_signed ],
237                 \naf ?_form [ first_has_parental_consent_for_signing ]
238             ) \or (
239                 ?_form [ second_is_minor_when_signed ],
240                 \naf ?_form [ second_is_at_least_16_when_signed ],
241                 \naf ?_form [ second_has_parental_consent_for_signing ]
242             )
243         )
244     ).
245 \overrides ( Ineligibility_for_AIPA_Section_7_2 ,
246     AIPADefined_Section_7_1 ).
247
248 // Section 8
249 /*
250 Validity of adult interdependent partner agreement
251 8(1) An adult interdependent partner agreement is invalid if
252     (a) one of the parties was induced
253         by fraud, duress or undue influence
254         to enter into the agreement,
255     (b) one of the parties lacked the
256         mental capacity to understand the
257         nature of the agreement,
258     (c) the parties were neither living
259         together nor intending to live
260         together in a relationship of
261         interdependence when the agreement
262         was entered into, or
263     (d) one of the parties was
264         prohibited by section 7(2) from
265         entering into the agreement.
266 */

```

```

252 // Note that 8(1)(c) is entirely duplicative of the same
      requirement in 7(1). It is omitted. 8(1)(d) is already
      implemented
253 // by the conclusion of 7(2), which also negates the truth of the
      AIPA fact, and is not repeated here.
254
255 @{{Invalid_AIPA_Section_8}}
256 \neg aipa(?A,?B):AIPA :- // there is no AIPA between A and B if
257   ?_form[first->?A,second->?B]:AIPAForm, // there is an agreement
      between the parties, and
258   ( // one of
259     ( // section 8(1)(a)
260       ?_form[first_was_induced_by_fraud];
261       ?_form[second_was_induced_by_fraud]
262     ) \or ( // section 8(1)(b)
263       ?_form[first_lacked_mental_capacity];
264       ?_form[second_lacked_mental_capacity]
265     )
266   ).
267 \overrides(Invalid_AIPA_Section_8, AIPADefined_Section_7_1).
268
269 // Section 10
270 /*
271 Former adult interdependent partner
272 10(1) Unless another enactment provides otherwise, an adult
      interdependent partner becomes the former adult interdependent
      partner of another person when the earliest of the following
      occurs:
273
      (a) the adult interdependent
          partners enter into a written
          agreement that provides evidence
          that the adult interdependent
          partners intend to live separate
          and apart without the possibility
          of reconciliation;
274
      (b) the adult interdependent
          partners live separate and apart
          for more than one year and one or
          both of the adult interdependent
          partners intend that the adult
          interdependent relationship not
          continue;
275
      (c) the adult interdependent
          partners marry each other or one of
          the adult interdependent partners
          marries a third party;
276
      (d) in the case of an adult
          interdependent partner referred to
          in section 3(1)(a), the adult
          interdependent partner enters into
          an adult interdependent partner
          agreement with a third party;
277
      (e) one or both of the adult
          interdependent partners have

```

obtained a declaration of
irreconcilability under section 83
of the Family Law Act.

278 (2) For the purposes of subsection (1)(b), a period of living
separate and apart is not considered interrupted or terminated
279 (a) by reason only that either adult
interdependent partner has become
incapable of forming the intention
to live separate and apart, or
280 (b) by reason only that the adult
interdependent partners have
resumed living together during a
single period of not more than 90
days with reconciliation as its
primary purpose.

281 (3) An adult interdependent partner agreement expires when the
parties become former adult interdependent partners under
subsection (1).

282 */
283 // note, again, that this section has explicit temporal aspects "the
earliest of the following occurs", and implicitly temporal
aspects, in that the events which cause an adult
interdependent partnership to arise must have happened "before
"

284 // the factors in 10(1) can be satisfied, since they all apply to
adult interdependent partners.

285
286 @{\Written_Agreement_Re_Separation_Section_10_1_a}
287 faip(?A,?B):FAIP :- // A is a former adult interdependent partner
of B if
288 (?_agreement[first->?A,second->?B]:WARS; ?_agreement[first->?B,
second->?A]:WARS), // there is a written agreement regarding
separation between the parties
289 ?_agreement[evidences_intent_to_live_separately_and_apart]. //
which evidences intent.

290
291 // necessary because FAIP(?A,?B) is defined as the opposite of AIP
(?A,?B), and so rules that prove a FAIP are in conflict with
rules that prove an AIP.

292 \overrides(Written_Agreement_Re_Separation_Section_10_1_a,
AIPDefined_Section_3).

293
294 @{\Separation_Section_10_1_b}
295 faip(?A,?B):FAIP :- // A is a former adult interdependent partner
of B if
296 (?_separation:Separation[first->?A,second->?B]; ?_separation:
Separation[first->?A,second->?B]), // there is a separation
between the two parties, and
297 ?_separation[more_than_one_year], // the separation is for more
than one year and
298 ?_separation[one_or_more_intend_to_terminate]. // one or both
intend it not continue.

299

```

300 // necessary because FAIP(?A,?B) is defined as the opposite of AIP
    (?A,?B), and so rules that prove a FAIP are in conflict with
    rules that prove an AIP.
301 \overrides(Separation-Section-10-1-b , AIPDefined-Section-3).
302
303 @{Marriage-Section-10-1-c}
304 faip(?A,?B):FAIP :- // A is a former adult interdependent partner
    of B if
305     ( // either
306       aip(?A,?B):AIP[one_party_married_after_coming_into_effect];
307       aip(?A,?B):AIP[parties_married_after_coming_into_effect]
308     ).
309
310 // necessary because FAIP(?A,?B) is defined as the opposite of AIP
    (?A,?B), and so rules that prove a FAIP are in conflict with
    rules that prove an AIP.
311 \overrides(Marriage-Section-10-1-c , AIPDefined-Section-3).
312
313 // We need to be able to distinguish an AIP under the common law
    rules from an AIP under the agreement rules.
314 // We will approximate it by requiring the presence of an AIP and
    the absence of an AIPA between the parties.
315 // In a more accurate model, the idea of having an AIPA when you
    already have an AIP would violate the rule against having two
    AIPs. However, we have implemented that rule as a predicate
    of the AIP (person has another AIP),
316 // which is not derived, so this rule will not conflict with that
    rule. When describing fact scenarios of this sort, AIP
    created by AIPAs after a prior AIP for the same person should
    have that predicate set to false.
317 // Because it doesn't override any implications of any rules, we
    can ignore the requirement that the date of the AIPA be later
    than the date of the AIP.
318 // This is clearly sub-optimal, and requires the person describing
    the fact scenario to understand how the model works in a way
    that is unintuitive. But without robust temporal options,
    getting to the point where the rules
319 // can derive whether or not an AIP exists for one of the parties
    who enter into an AIPA is beyond the scope of the coursework.
320 @{AIPA_Overrides_AIP-Section-10-1-d}
321 faip(?A,?B):FAIP :- // A is a former adult interdependent partner
    of B if
322     \naf aipa(?A,?B):AIPA, // they do not have an AIPA
323     \naf aipa(?B,?A):AIPA,
324     aipa(?_first,?_second):AIPA, // there is an AIPA between one of
    them and someone else
325     (
326       (?_first == ?A, ?_second !== ?B)
327       \or
328       (?_first == ?B, ?_second !== ?A)
329     ).
330
331 // necessary because FAIP(?A,?B) is defined as the opposite of AIP
    (?A,?B), and so rules that prove a FAIP are in conflict with

```

332 rules that prove an AIP.
 \overrides(AIPA_Overrides_AIP_Section_10_1_d , AIPDefined_Section_3)
 .

Appendix C

ErgoAI Encoding of Kraft

Listing C.1: ErgoAI Encoding of Kraft

```
1 // New_Copyright.ergo
2 // by Jason Morris, student ID #0353905
3 // as part of research paper "Encoding Kraft"
4 // for Law 696, Professor Muir, University of Alberta, December 8,
   2017
5 :- compiler_options{production= on,expert=on,omni=on}.
6 :- use_argumentation_theory.
7 ?- setmonitor{10,extended}.
8 ?- setruntime{memory(24)}.
9
10 // ONTOLOGY
11
12 // A party is a type of thing.
13 Party::Thing.
14
15 // Work is a type of object, which includes an owner, which has a
   value which is of the Party type.
16 Work[|owner=>Party|].
17
18 // Canada and NotCanada are Places of Manufacture.
19 {Canada,NotCanada}:PlaceOfManufacture.
20
21 // A Product has a manufacturer, an origin, and an original work.
22 Product[|manufacturer=>Party,origin=>PlaceOfManufacture,
   original_work=>Work|].
23
24 // An Infringing sale is a type of Sale.
25 InfringingSale::Sale.
26
27 // An Infringing Importation is a type of infringing sale.
28 InfringingImportation::InfringingSale.
29
30 // An Infringing Product is a type of Product
31 InfringingProduct::Product.
32
33 // An infringing sale, and infringing importation, and an
   infringing product are all infringements.
```

```

34 {InfringingSale , InfringingImportation , InfringingProduct }::
    Infringement .
35
36 // Property Rights adhere to a work with regard to a party .
37 PropertyRight [| property=>Work , owner=>Party |] .
38
39 // A Right of action is owned by a party with regard to an alleged
    infringement .
40 RightOfAction [| plaintiff=>Party , claim=>Infringement , accused->Party
    |] .
41
42 // A Sale includes a product sold , and a seller .
43 Sale [| product_sold=>Product , seller=>Party |] .
44
45 // An assignment of copyright is from an assignor to an assignee
    with regard to a work .
46 Assignment [| assignor=>Party , assignee=>Party , work->Work |] .
47
48 // An exclusive license of copyright is from an licensee to a
    licensor with regard to a work .
49 License [| licensee=>Party , licensor=>Party , work->Work |] .
50
51 // COMMON LAW RULES
52
53 // Owners have a property right by default .
54 @ {OwnersPropertyRight}
55 PR(?_owner , ?_property) : PropertyRight [ property -> ?_property , owner -> ?
    _owner ] :-
56     ?_property : Work [ owner -> ?_owner ] .
57
58 \opposes ( Overridden (?_X) , ?_X : RightOfAction ) . // If a thing is
    Overridden , that thing is not a right of action , and vice
    versa .
59
60 // A person without a property interest cannot sue someone who has
    a property interest . This rule overrides all other rules .
61 @ {CantSueOwner}
62 Overridden (?_roa) :- // a right of action is overridden if
63     ?_roa : RightOfAction [ plaintiff -> ?_plaintiff , claim -> ?
        _infringement , accused -> ?_accused ] , // there is a
        right of action
64     ( //either
65         ( //It is a sale , the seller has a
            property right and the plaintiff does not
66             ?_infringement : InfringingSale [ seller -> ?_accused ,
                product_sold -> ?_product : Product [ original_work
                    -> ?_work ] ] ,
67             ?_sellersright : PropertyRight [ property -> ?_work ,
                owner -> ?_accused ] , // the seller has a
                property right in the work
68             \naf ?_plaintiffright : PropertyRight [ property -> ?
                _work , owner -> ?_plaintiff ] // the plaintiff has
                no property right in the work ..
69         ) \or (

```

```

70         //it is a product, the manufacturer has a property
71         right and the plaintiff does not
72         ?_infringement:Product[manufacturer->?_accused,
73         original_work->?_work],
74         ?_manufacturersright:PropertyRight[property->?
75         _work,owner->?_accused], // the manufacturer
76         has a property right in the work
77         \naf ?_plaintiffright:PropertyRight[property->?
78         _work,owner->?_plaintiff] // the plaintiff has
79         no property right in the work.
80     )
81     ).
82 \overrides(CantSueOwner,?-). // Because this rule causes a
83     conflict because overridden() opposes :RightOfAction, this
84     rule defeats all other rules.
85
86 // If a hypothetical right of action against a hypothetical
87 infringer would be overridden by Can't Sue Owner above, then
88 the right of action with regard to the actual infringement
89 // to which it relates is also overridden.
90 @{HypotheticalChaining}
91 Overridden(?_roa) :- // a right of action is overridden if
92     ?_roa:RightOfAction[plaintiff->?_plaintiff, claim->?
93     _infringement, accused->?_accused], // there is a
94     right of action
95     Overridden(?_roa2), // another right of action is
96     overridden
97     ?_roa2:RightOfAction[claim->?_infringement2:HypoProduct[
98     real_sale->?_infringement]]. // the overridden right
99     of action's real sale is the infringement for the
100    first right of action.
101 \overrides(HypotheticalChaining,?-).
102
103 // LEGISLATIVE RULES
104
105 // Section 2.
106 // ...
107 // infringing means
108 // (a) in relation to a work in which copyright subsists, any copy
109 // , including any colourable imitation, made or dealt with in
110 // contravention of this Act,
111 //...
112 // The definition includes a copy that is imported in the
113 // circumstances set out in paragraph 27(2)(e) and section 27.1
114 // but does not otherwise include
115 // a copy made with the consent of the owner of the copyright in
116 // the country where the copy was made;
117
118 // Infringing Copies
119
120 //A valid assignment is made by the owner of the work.
121 ?X:ValidAssignment :-
122     ?X:Assignment[assignor->?_assignor,work->?_work],
123     ?_work[owner->?_assignor].

```

```

103
104 // A valid license is made by the owner of the work if there is no
      assignee, or the assignee if there is.
105 ?X:ValidLicense :-
106     ?X:License[licensor->?_licensor,work->?_work],
107     (
108         \naf ?_A:Assignment[work->?_work], // the
              ownership in the work has not been assigned,
              and
109         ?_work[owner->?_licensor] // the licensor is the
              owner of the work
110     ) \or
111     (
112         ?_B:Assignment[work->?_work,assignee->?_assignee],
              // there is an assignment of the work
113         ?X:License[licensor->?_assignee] // the license
              was granted by the assignee.
114     ).
115
116 // Note that our model excludes the possibility of an assignee sub-
      -assigning or a licensee sub-licensing, for simplicity.
117
118 // A person is authorized to copy a work if
119 ?X[authorized->?Y] :-
120     ?X:Work,
121     ?Y:Party,
122     (
123         \naf ?_L:ValidLicense[work->?X], // there are no
              valid licenses for the work
124         \naf ?_A:ValidAssignment[work->?X], // there are
              no valid assignments
125         ?X:Work[owner->?Y] // the person is the owner
126     ) \or (
127         ?_L:ValidLicense[work->?X,licensee->?Y] // the
              person is a valid licensee
128     ) \or (
129         \naf ?_L:ValidLicense[work->?X], // there are no
              valid licenses
130         ?_A:ValidAssignment[work->?X,assignee->?Y] // the
              person is a valid assignee
131     ).
132
133 // A product is an infringing product if it was manufactured by a
      person not authorized to manufacture it.
134 ?X:InfringingProduct :-
135     ?X:Product[manufacturer->?_manufacturer,original_work->?
      _work,origin->Canada], // the product has a
      manufacturer and a work
136     \naf ?_work:Work[authorized->?_manufacturer]. // the
      manufacturer is not authorized to make copies of the
      work.
137
138 // What constitutes an infringing sale?
139 ?X:InfringingSale :- // a thing is an infringing sale if

```

```

140     (
141         ?X: Sale [ seller ->?_seller , product_sold ->?_product :
                Product [ manufacturer ->?_manufacturer ] ] , // it
                is a sale of a product and
142         ?_product: InfringingProduct , // the product is an
                infringing product.
143         ?_seller = ?_manufacturer // the product is being
                sold by the person who manufactured it.
144     ).
145
146 // Section 27(2)(e)
147 // It is an infringement of copyright for any person to
148 //   a) sell or rent out
149 //   ...
150 //   e) import into Canada for the purpose of doing anything
                referred to in paragraphs (a) ...
151 // a copy of a work, ... that ... would infringe copyright
152 // if it had been made in Canada by the person who made it.
153
154 // A hypothetical product is a type of product
155 HypoProduct:: Product.
156 HypoProduct [ | real_sale => Sale | ].
157
158 // Create a hypothetical product for each product not made in
                Canada.
159 HypoProduct (?_product) : HypoProduct [ origin -> Canada , manufacturer ->?
                _manufacturer , original_work ->?_original , real_sale ->?_P ] :-
160     ?_P: Sale [ product_sold ->?_product : Product [ origin -> NotCanada
                , manufacturer ->?_manufacturer , original_work ->?
                _original ] ].
161
162 // A product not made in Canada is an infringing product if its
                hypothetical product made in Canada is infringing.
163 ?X: InfringingImportation :-
164     ?X: Sale , // X is a sale of a product
165     ?Y: HypoProduct [ manufacturer ->?_accused , real_sale ->?X ] , //
                Y is a matching hypothetical product for that sale.
166     ?Y: InfringingProduct . // and Y is an infringing product.
167
168 // Valid Assignees are given a property interest in the works
                assigned to them.
169 PR (?_assignee , ?_property) : PropertyRight [ property ->?_property , owner
                ->?_assignee ] :-
170     ?_Y: ValidAssignment [ assignee ->?_assignee , work ->?_property
                ].
171
172 @{ OwnerAssignedPropertyRights } // A valid assignment deprives the
                owner of property right.
173 \neg ?_X: PropertyRight [ property ->?_property , owner ->?_workowner ] :-
174     ?_Y: ValidAssignment [ work ->?_property , assignor ->?_workowner
                ].
175 \overrides ( OwnerAssignedPropertyRights , OwnersPropertyRight ) . //
                This rule overrides the default rule that owners have property
                rights.

```

```

176
177 // A person with property rights has a right of action for
      infringement .
178 @{PropertyOwnerRightOfAction}
179 RoA(? _plaintiff ,? _infringement ,? _accused):RightOfAction [plaintiff
      ->?_plaintiff ,claim->?_infringement , accused->?_accused] :- //
      the plaintiff has a right of claim for an infringement if
180     ( // either
181         ?_infringement:InfringingSale [seller->?_accused ,
            product_sold->?_product:Product [original_work
            ->?_work]]; // the work was infringed by a
            sale , or
182         ?_infringement:InfringingProduct [manufacturer->?
            _accused , original_work->?_work] // the work
            was infringed by a product
183     ) , // and
184     ?_PR:PropertyRight [owner->?_plaintiff ,property->?_work] ,
            // the plaintiff has a property right in the work .
185     ?_plaintiff != ?_accused . // the plaintiff and the accused
            are not the same party .
186
187 // A licensee has a RightOfAction for infringement .
188 @{LicenseeRightOfAction}
189 RoA(? _plaintiff ,? _infringement ,? _accused):RightOfAction [plaintiff
      ->?_plaintiff ,claim->?_infringement ,accused->?_accused] :- //
      the plaintiff has a right of claim for an infringement if
190     ?_L:ValidLicense [licensee->?_plaintiff ,work->?_work] , //
            the plaintiff is a licensee of the work and
191     ( //either
192         ?_infringement:InfringingSale [seller->?_accused ,
            product_sold->?_product:Product [scenario->?
            _same ,original_work->?_work]]; // The work has
            been infringed by sale , or
193         ?_infringement:InfringingProduct [manufacturer->?
            _accused , original_work->?_work] // the work
            has been infringed by a product
194     ) ,
195     ?_plaintiff != ?_accused . // the plaintiff and the
            accused are not the same party .
196
197 // TESTING DATA
198
199 // Specifying the Fact Situation in Euro Excellence v. Kraft
200 // This section is commented out when running the experiment .
201
202 // Kraft has given an exclusive license to KraftCanada for the
      Toblerone copyright which Kraft Owns .
203 //LicenseToKraftCanada:License [licensor->Kraft:Party , licensee->
      KraftCanada:Party ,work->Toblerone:Work [owner->Kraft]] .
204 // Euro Excellence has sold a Toblerone that was made by Kraft not
      In Canada
205 //SaleOfImportedToblerone:Sale [seller->EuroExcellence:Party ,
      product_sold->ImportedToblerone:Product [manufacturer->Kraft ,
      origin->NotCanada ,original_work->Toblerone]] .

```

```

206
207 // OUR EXPERIMENT
208
209 // This predicate will be used to search for bugs in the
    automatically-generated scenarios.
210
211 isabug(?_I) :-
212     ?_I:Infringement, // there is an infringement
213     ( // either
214         ?_I:InfringingSale[seller->?_S,product_sold->?_P:
            Product[original_work->?_W]], // the
            infringement is an infringing sale
215         \naf ?_P:HypoProduct, // the product sold is not
            hypothetical.
216         ?_W[authorized->?_A], // there is a person
            authorized with regard to the work
217         ( // either
218             ?_R:RightOfAction[plaintiff->?_A,claim->?
                _I], // the right of action has been
                overridden
219             Overridden(?_R),
220             ?bug = ?_R // in which case the right of
                action is the bug
221         ) \or (
222             \neg ?_R:RightOfAction[plaintiff->?_A,
                claim->?_I],// it never existed
223             ?_A != ?_S // the impugned seller is not
                also the plaintiff
224         )
225     ) \or (
226         ?_P:InfringingProduct:Product[manufacturer->?_M,
            original_work->?_W], // the infringement is an
            infringing product
227         \naf ?_P:HypoProduct, // the infringing product is
            not hypothetical.
228         ?_W[authorized->?_A], // there is a person
            authorized with regard to the work
229         ( // either
230             ?_R:RightOfAction[plaintiff->?_A,claim->?
                _P], // the right of actions has been
                overridden
231             Overridden(?_R),
232             ?bug = ?_R // in which case the right of
                action is the bug
233         ) \or (
234             \neg ?_R:RightOfAction[plaintiff->?_A,
                claim->?_P],// it never existed
235             ?_A != ?_M // the impugned manufacturer
                is not also the plaintiff
236         )
237     ).
238
239
240

```

```

241 // UTILITY CODE
242
243 // This section of code generates only unique sets of individuals
    in more than 1 role .
244
245 start : mynum [ size ->1, pos (1) ->1].
246 {1,2,3,4,5,6,7} : NumVal .
247 {A,B,C,D,E,F,G} : CaseParty .
248 CaseParty :: Party .
249 A [ num ->1 ].
250 B [ num ->2 ].
251 C [ num ->3 ].
252 D [ num ->4 ].
253 E [ num ->5 ].
254 F [ num ->6 ].
255 G [ num ->7 ].
256 ?X : mynum [ char (?Y) ->?Z ] :- ?X : mynum [ pos (?Y) ->?V ], ?Z : Party [ num ->?V ] .
    // creates a char (?X) predicate for each pos (?X) predicate .
257 ?X [ pos (?Y) ->?Z ] :- ?X [ prior ->?O [ pos (?Y) ->?Z ] ] . // mynums inherit
    the values of their originals .
258 ?-      insertall { \#newnum (?O, ?N, ?V) : mynum [ size ->?N, pos (?N) ->?V,
    prior ->?O ]
259         |
260         (?O : mynum [ size ->?S, pos (? -) ->?P ] ;
261         (?O : mynum [ size ->?S ], ?V=1) ) ,
262         ?N \is ?S + 1,
263         ?V : NumVal,
264         ?V =< ?N,
265         ?P \is ?V - 1,
266         ?S = 1 } .
267 ?-      insertall { \#newnum (?O, ?N, ?V) : mynum [ size ->?N, pos (?N) ->?V,
    prior ->?O ]
268         |
269         (?O : mynum [ size ->?S, pos (? -) ->?P ] ;
270         (?O : mynum [ size ->?S ], ?V=1) ) ,
271         ?N \is ?S + 1,
272         ?V : NumVal,
273         ?V =< ?N,
274         ?P \is ?V - 1,
275         ?S = 2 } .
276 ?-      insertall { \#newnum (?O, ?N, ?V) : mynum [ size ->?N, pos (?N) ->?V,
    prior ->?O ]
277         |
278         (?O : mynum [ size ->?S, pos (? -) ->?P ] ;
279         (?O : mynum [ size ->?S ], ?V=1) ) ,
280         ?N \is ?S + 1,
281         ?V : NumVal,
282         ?V =< ?N,
283         ?P \is ?V - 1,
284         ?S = 3 } .
285 ?-      insertall { \#newnum (?O, ?N, ?V) : mynum [ size ->?N, pos (?N) ->?V,
    prior ->?O ]
286         |
287         (?O : mynum [ size ->?S, pos (? -) ->?P ] ;

```



```

288     (?O:mynum[ size ->?S ],?V=1) ),
289     ?N \is ?S + 1,
290     ?V:NumVal,
291     ?V=<?N,
292     ?P \is ?V-1,
293     ?S=4}.
294 ?-    insertall{\#newnum(?O,?N,?V) :mynum[ size ->?N, pos (?N)->?V,
        prior ->?O]
295     |
296     (?O:mynum[ size ->?S, pos (? -)->?P ];
297     (?O:mynum[ size ->?S ],?V=1) ),
298     ?N \is ?S + 1,
299     ?V:NumVal,
300     ?V=<?N,
301     ?P \is ?V-1,
302     ?S=5}.
303 ?-    insertall{\#newnum(?O,?N,?V) :mynum[ size ->?N, pos (?N)->?V,
        prior ->?O]
304     |
305     (?O:mynum[ size ->?S, pos (? -)->?P ];
306     (?O:mynum[ size ->?S ],?V=1) ),
307     ?N \is ?S + 1,
308     ?V:NumVal,
309     ?V=<?N,
310     ?P \is ?V-1,
311     ?S=6}.
312
313 // QUERIES
314
315 // Case 1 - no assignments or leases.
316
317 /*
318 ?- \while (?_X:mynum[ size ->3,char (1)->?A, char (2)->?B, char (3)->?C
    ],?P:PlaceOfManufacture) \do
319     (\while (insert {
320         Sale(?A,?B,?C,?P) : Sale [ seller ->?C, product_sold->
            Prod(?A,?B,?C,?P) : Product ] ,
321         Prod(?A,?B,?C,?P) : Product [ manufacturer ->?B, origin
            ->?P, original_work ->Work(?A,?B,?C,?P) : Work ] ,
322         Work(?A,?B,?C,?P) : Work [ owner ->?A]
323     },
324     writeln(='Testing ' || ?A || ', ' || ?B || ', ' || ?C || ', and ' || ?
        P)@\io)
325     \do
326         (
327             (\while isabug(?X)
328             \do writeln(='Bug Found ' || ?X)@\io) ,
329             delete {
330                 Sale(?A,?B,?C,?P) : Sale [ seller ->?C,
                    product_sold->Prod(?A,?B,?C,?P) :
                    Product ] ,
331                 Prod(?A,?B,?C,?P) : Product [ manufacturer ->?B
                    , origin ->?P, original_work ->Work(?A,?B
                    ,?C,?P) : Work ] ,

```

```

332             Work(?A,?B,?C,?P) : Work[owner->?A]
333         }
334     )
335 ).
336 */
337
338 // Case 2 - Lease Only
339
340
341 ?- \while (?_X:mynum[ size ->5,char(1)->?A, char(2)->?B, char(3)->?C,
342         char(4)->?D, char(5)->?E],?P:PlaceOfManufacture)
343     \do
344         (\while (insert {
345             Sale(?A,?B,?C,?D,?E,?P) : Sale [ seller ->?C,
346                 product_sold ->Prod(?A,?B,?C,?D,?E,?P) :
347                 Product ],
348             Prod(?A,?B,?C,?D,?E,?P) : Product [
349                 manufacturer ->?B, origin ->?P,
350                 original_work ->Work(?A,?B,?C,?D,?E,?P)
351                 :Work ],
352             Work(?A,?B,?C,?D,?E,?P) : Work[owner->?A] ,
353             License(?A,?B,?C,?D,?E,?P) : License [
354                 licensor ->?D, licensee ->?E, work->Work(?
355                 A,?B,?C,?D,?E,?P) ]
356         }, writeln(='Testing License Scenario '|| ?A|| ',
357                 '|| ?B|| ', '||?C|| ', '||?D|| ', '||?E|| ' and
358                 '||?P)@\io)
359     \do
360         (
361         \while isabug(?X)
362         \do writeln(='Bug Found ' || ?X)@\io
363         ),
364         delete {
365             Sale(?A,?B,?C,?D,?E,?P) : Sale [
366                 seller ->?C, product_sold ->Prod
367                 (?A,?B,?C,?D,?E,?P) : Product ],
368             Prod(?A,?B,?C,?D,?E,?P) : Product [
369                 manufacturer ->?B, origin ->?P,
370                 original_work ->Work(?A,?B,?C,?
371                 D,?E,?P) :Work ],
372             Work(?A,?B,?C,?D,?E,?P) : Work[owner
373                 ->?A] ,
374             License(?A,?B,?C,?D,?E,?P) : License
375             [ licensor ->?D, licensee ->?E,
376                 work->Work(?A,?B,?C,?D,?E,?P) ]
377         }
378     ).
379
380 // Case 3 - Assignment And Lease
381 // Note that this search takes at least several hours on the
382 // tested hardware, and has never been successfully completed.
383 // There may be bugs in the code below, or running the code below
384 // may reveal bugs in the code above (or in the law!).

```

```

366 |
367 | /*
368 | ?- \while (?_X:mynum[ size ->7,char (1)->?A, char (2)->?B, char (3)->?C,
    | char (4)->?D, char (5)->?E, char (6)->?F, char (7)->?G ],?P:
    | PlaceOfManufacture) \do
369 |     (\while (insert {
370 |         Sale(?A,?B,?C,?D,?E,?F,?G,?P) : Sale [ seller ->?C,
    |         product_sold ->Prod(?A,?B,?C,?D,?E,?F,?G,?P) :
    |         Product ],
371 |         Prod(?A,?B,?C,?D,?E,?F,?G,?P) : Product [ manufacturer
    |         ->?B, origin ->?P, original_work ->Work(?A,?B,?C,?
    |         D,?E,?F,?G,?P) : Work ],
372 |         Work(?A,?B,?C,?D,?E,?F,?G,?P) : Work [ owner ->?A ] ,
373 |         Assignment (?A,?B,?C,?D,?E,?F,?G,?P) : Assignment [
    |         assignor ->?D, assignee ->?E, work ->Work(?A,?B,?C
    |         ,?D,?E,?F,?G,?P) ] ,
374 |         License (?A,?B,?C,?D,?E,?F,?G,?P) : License [ licensor
    |         ->?F, licensee ->?G, work ->Work(?A,?B,?C,?D,?E,?F
    |         ,?G,?P) ]
375 |     }, writeln(='Testing '|| ?A||', '|| ?B||', '||?C||', '|| ?
    | D||', '|| ?E||', '||?F||', '|| ?G||', and '||?P)@\io)
    | \do
376 |         (
377 |         \while isabug(?X)
378 |         \do writeln(='Bug Found ' || ?X)@\io
379 |         ),
380 |     delete {
381 |         Sale(?A,?B,?C,?D,?E,?F,?G,?P) : Sale [ seller
    |         ->?C, product_sold ->Prod(?A,?B,?C,?D,?E
    |         ,?F,?G,?P) : Product ],
382 |         Prod(?A,?B,?C,?D,?E,?F,?G,?P) : Product [
    |         manufacturer ->?B, origin ->?P,
    |         original_work ->Work(?A,?B,?C,?D,?E,?F
    |         ,?G,?P) : Work ],
383 |         Work(?A,?B,?C,?D,?E,?F,?G,?P) : Work [ owner
    |         ->?A ] ,
384 |         Assignment (?A,?B,?C,?D,?E,?F,?G,?P) :
    |         Assignment [ assignor ->?D, assignee ->?E,
    |         work ->Work(?A,?B,?C,?D,?E,?F,?G,?P) ] ,
385 |         License (?A,?B,?C,?D,?E,?F,?G,?P) : License [
    |         licensor ->?F, licensee ->?G, work ->Work(?
    |         A,?B,?C,?D,?E,?F,?G,?P) ]
386 |     }
387 | ).
388 | */
389 |
390 | // Case 4 - Assignment Only
391 |
392 | /*
393 | ?- \while (?_X:mynum[ size ->5,char (1)->?A, char (2)->?B, char (3)->?C,
    | char (4)->?D, char (5)->?E ],?P: PlaceOfManufacture)
394 |     \do
395 |         (\while (insert {

```

```

396         Sale(?A,?B,?C,?D,?E,?P): Sale [ seller ->?C,
           product_sold ->Prod(?A,?B,?C,?D,?E,?P) :
397         Product ],
           Prod(?A,?B,?C,?D,?E,?P): Product [
           manufacturer ->?B, origin ->?P,
           original_work ->Work(?A,?B,?C,?D,?E,?P)
           :Work ],
398         Work(?A,?B,?C,?D,?E,?P): Work [ owner ->?A ],
399         Assignment(?A,?B,?C,?D,?E,?P): Assignment [
           assignor ->?D, assignee ->?E, work ->Work(?
           A,?B,?C,?D,?E,?P) ]
400     }, writeln(='Testing License Scenario ' || ?A || ',
           ' || ?B || ', ' || ?C || ', ' || ?D || ', ' || ?E || ' and
           ' || ?P || '@\io)
401     \do
402         (
403         \while isabug(?X)
404         \do writeln(='Bug Found ' || ?X || '@\io
405         ),
406         delete {
407             Sale(?A,?B,?C,?D,?E,?P): Sale [
           seller ->?C, product_sold ->Prod
           (?A,?B,?C,?D,?E,?P): Product ],
408             Prod(?A,?B,?C,?D,?E,?P): Product [
           manufacturer ->?B, origin ->?P,
           original_work ->Work(?A,?B,?C,?
           D,?E,?P): Work ],
409             Work(?A,?B,?C,?D,?E,?P): Work [ owner
           ->?A ],
410             Assignment(?A,?B,?C,?D,?E,?P):
           Assignment [ assignor ->?D,
           assignee ->?E, work ->Work(?A,?B
           ,?C,?D,?E,?P) ]
411         }
412     ).
413 */
414
415 // KNOWN PROBLEMS
416 // 1. The while clauses with ?_X:mynum... are returning twice as
           many results as expected. Each scenario is being tested twice
           .
417 // 2. The scenario generating code above should be looped rather
           than run explicitly 6 times.
418 // 3. The isabug() predicate occasionally finds bugs which are "
           undefined" in the code, I have not had the opportunity to
           clarify why, or to fix that. The scenarios are AAAAB Canada,
           AABAC Canada, and AABAB Canada.

```